Fully Distributed Ubiquitous Information Sharing on a Global Scale for the Internet-of-Things

Victor Kardeby, Stefan Forsström, Patrik Österberg, and Ulf Jennehag

Department of Information and Communication Systems
Mid Sweden University
Sundsvall SE-85170, Sweden
Email: victor.kardeby@miun.se, stefan.forsstrom@miun.se, patrik.osterberg@miun.se, ulf.jennehag@miun.se

***Abstract*—The Internet-of-Things will require ubiquitous information sharing between connected things on a global scale, which existing systems do not offer. Most current efforts focus on solutions for information dissemination, which induce single points of failure and introduce unnecessary communication delays. To this end we propose the SensibleThings platform, which is a fully distributed open source architecture for Internet-of-Things based applications. This article describes the major problems that Internet-of-Things platforms must address, our technical solution to these problems, and an evaluation thereof. We also present the current progress and a series of demonstrators, which show the wide range of applications enabled by the platform. Finally, we present how the platform will be used in our future research and potential spin off companies.**

Keywords–overlay;sensors;actuators;internet-of-things.

I. Introduction

This journal article is an extension of [1] and [2], where [1] is a best paper awarded publication at the 2014 International Conference on Digital Telecommunications (ICDT).

Today we can observe a large interest in applications that can utilize information from sensors attached to different things in order to provide more personalized, automatized, or even intelligent behavior. These are commonly referred to as Machine-to-Machine (M2M) applications [3] or Internet-of-Things (IoT) applications [4]. The IoT can be seen as a natural evolution of computer networking and communicating devices, from simple direct communication between computers, via globally connected computers, to small devices such as smartphones that are ubiquitously connected to the Internet. Together these form a worldwide network of interconnected everyday objects. Through the IoT, applications will display context-aware behavior [5] and even be able to have social interactions between themselves [6]. These applications may address a variety of areas, such as environmental monitoring (pollution, earth quake, flooding, forest fire), energy conservation (optimization), security (traffic, fire, surveillance), safety (health care, elderly care), and enhancement of social interactions. Furthermore, the IoT is surprisingly close to Mark Weiser's predictions made in 1991 on the computers of the 21st century [7].

There is also an interesting relationship between the IoT and big data [8], since all of the connected things will produce and consume large amounts of data. Current estimations are in the order of 50 billion connected devices year 2020 [9]. Thus, IoT applications will probably have a big impact on how we interact with people, things, and the entire world in the future. But in order to enable a widespread proliferation of IoT services there should be a common platform for dissemination of sensor and actuator information on a global scale. This is however a very difficult goal to achieve, because there is a large number of practical difficulties that must be solved. Therefore, the purpose of this article is to explore how to enable ubiquitous information sharing on a global scale for the IoT, using highly scalable fully distributed solutions. We present a realized solution called the SensibleThings platform, which is verified through a series of demonstrator applications and performance measurements.

The remainder of this article is outlined in the following way: Section II presents a list of requirements to evaluate potential platforms. Section III surveys current IoT architectures. Section IV presents our approach to address the requirements. Section V describes the SensibleThings platform which is our implementation of the approach. Section VI presents the verification of the platform and measurement results. Section VII discusses future IoT services. Finally, Section VIII presents the conclusion and future research.

II. Platform Requirements

To aid in evaluating IoT platforms, a list of application requirements has been constructed. This list is derived from previous and related work, for example, [2], [4], [10].

IoT applications spread very diverse areas, but common among many of them is the focus on many small devices on a global scale, low response times, reliable operation, and interoperability to support different hardware and features depending on the scenario. Therefore, we state that the majority of applications on the IoT will require the following from an underlying platform:

1) Capability of signaling between end points with low latency, without any unnecessary relaying of information.
2) Reliably handle transient nodes joining and leaving with high churn rates. Also avoiding choke points with significantly higher utilization than the rest of the system.

3) Ability to run on devices with limited computational and data-storage capacity.
4) Be extensive and adaptive to conform with a wide range of applications, devices, and future scenarios with currently unforeseen demands.
5) Easy to adopt and free to use in commercial products without restrictions in terms of software licenses and fees.

Some requirements have been left out of scope in this article because we focus on the open sharing of information where problems regarding security and privacy can be addressed at a later stage. In order to evaluate if a platform achieves our requirements we have also determined the following concrete and measurable metrics.

A. Evaluating Requirement 1

The first requirement on low response times cannot be measured using only the raw response times between the source and sink because they highly depend on the infrastructure in between. Since all IoT platforms are based on the Internet, all communication is inherently made using a best effort system. This makes it impossible to compare the raw response times from two different platforms. Hence, we evaluate the response time based on the ratio between the retrieval time compared to an ideal communication case. The ratio is calculated in (1), where $R_{latency}$ is the average ratio between the retrieval time $t^{retrieve}$ and the ideal round trip time t^{rtt} for each device i of the total devices N. Where $t^{retrieve}$ is the measured time to sends a sensor value from source to sink and t^{rtt} is the measured ideal round trip time, in this case ping.

$$R_{latency} = \frac{1}{N}\sum_{i=1}^{N}\frac{t_i^{retrieve}}{t_i^{rtt}} \tag{1}$$

B. Evaluating Requirement 2

The second requirement on reliability and scalability can be measured by the communication load on all ingoing devices. To calculate this we need to find the function $f_{avg}(N)$ in (2), of how the number of messages m handled per device per minute i changes as the number of devices N increase in the system. However, for most platforms the average number of messages per node and minute will logarithmically increase and converge to a constant.

$$f_{avg}(N) = \frac{1}{N}\sum_{i=1}^{N} m_i \tag{2}$$

The standard deviation of the amount of messages per device is also interesting for the scalability metric, since it can indicate central points of failure. The standard deviation as a function of the total number of devices N can be seen in (3). A constant or decreasing standard deviation function indicates an even system, where the new devices handle an even share of the workload, whereas an increasing standard deviation means that there is an uneven workload among the devices.

$$f_{stdev}(N) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(m_i - f_{avg}(N))^2} \tag{3}$$

C. Evaluating Requirement 3

The third requirement on resource efficiency can be measured in different aspects, such as energy consumption, processing time requirements, storage space requirements, and network load. The resource efficiency is difficult to calculate for a whole system, because it highly depends on the hardware present on the devices. Therefore, we define the resource efficiency as the maximal value U in the set of all resource utilizations u for different available resources j when the platform is running. See (4) and (5). The resource utilization for each resource is calculated by the amount of a resource used by the platform u_{used} divided by the available amount of the resource u_{avail}. If any resource utilization on any device in the system is close to 100%, the resources have been saturated and the system is no more resource efficient.

$$u = \frac{u_{used}}{u_{avail}} \tag{4}$$

$$U = max(\{u_1, u_2, ..., u_j\}) \tag{5}$$

D. Evaluating Requirement 4 and 5

The fourth and fifth requirement cannot be measured, only qualitatively evaluated. To be future proof the platform must have the ability to add new features without costly updating all ingoing devices individually. To study the cost, one has investigated the terms of usage for the platform to evaluate the type of costs that are related to the platform. These costs can be either traditional fees or resources such as bandwidth and computational power.

III. Related Work

There currently exist a vast amount of platforms which claim to enable an IoT, far more than can be listed in this paper. There are also standardization efforts being made by standardization bodies such as the Internet Engineering Task Force (IETF), the European Telecommunications Standards Institute (ETSI), and the Institute of Electrical and Electronics Engineers (IEEE). In general, most of these are focused on problems such as enabling Internet Protocol (IP) over radio communication and connecting different hardware to the Internet. Therefore, the standardization efforts are much less extensive in the layers closer to the application, which is what this article focuses on. As an example, the IETF are discussing standardization from the physical and Media Access Control (MAC) layer, covering the Internet Protocol Version 6 (IPv6) adaptation layer, the routing layer, and finally an application protocol [11]. The application protocol that they propose is, however, the Constrained Application Protocol (CoAP) which only provides simple transfer of sensor and actuator information directly between two known IP addresses. Meanwhile, the IEEE IoT standard P2413 [12] is still in its early stages, since their first working group meeting was in July 2014. Hence, no real results have so far been presented other than that the plan is to utilize existing technologies for the lower layer communication and that they are investigating which application level protocols to use in their standard.

In addition to the standardization efforts, there are multiple forums and alliances made of industry and academia participants to both promote the idea of an IoT and to establish

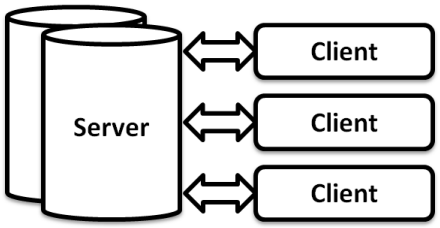

Figure 1. Overview of typical centralized architectures

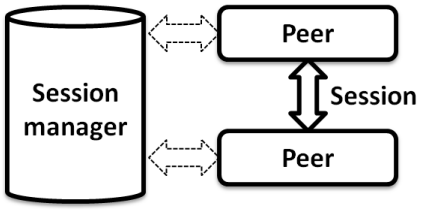

Figure 2. Overview of typical semi-distributed architectures

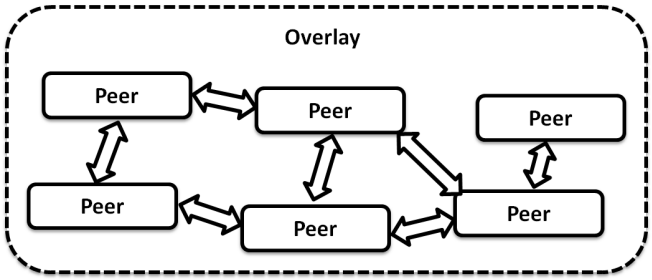

Figure 3. Overview of typical fully distributed architectures

some form of praxis and consensus. These include for example the AllSeen Alliance [13], European Research Cluster on the Internet of Things [14], HyperCat [15], Internet of Things – Architecture [16], the Internet of Things Council [17], the Internet of Things Europe [18], The Internet of Things Initiative [19], the IoT Forum [20], IP for Smart Objects Alliance [21], oneM2M [22], and Open interconnect [23].

Additionally, there are research [24] that investigates the possibilities and challenges for using software defined networking (SDN) [25] as an IoT platform, where [24] concludes that SDN will provide a unified method of managing network resources globally. In [26] the authors implements a vertical SDN controller that is used in conjunction with advanced network calculus- and genetic algorithm-techniques in order to improve data throughput, end-to-end delay and end-to-end jitter.

The many platforms found in related work can generally be categorized into three groups, centralized (or cloud distributed), semi-distributed, and fully distributed systems.

A. Centralized Systems

Most of the systems being released today focus on distributing the data on some form of cloud-based IoT architecture. The cloud is a concept that rise in popularity, but it is in many cases simply a new word for traditional web services. Very few commercial cloud-based systems explain how the distribution and synchronization is actually done inside their architecture, which type of virtualization they use, how many servers they have, etc. Either way, cloud-based systems can be considered as centralized systems since they always relay the sensor and actuator information through a centralized point, in this case a cloud (be it one or many connected servers). See Figure 1 for an overview of a centralized system. The main problems with these solutions are that they have difficulties achieving requirement 1 on direct communication between end devices, requirement 2 on no central points of failure, and requirement 5 on an open and free to use system, because they are based on large scale servers. Typical examples of these centralized or cloud-based architectures include: SicsthSense [27], ThingSpeak [28], Sen.Se [29], Nimbits [30], ThingSquare [31], EVRYTHNG [32], Paraimpu [33], Xively [34], XOBXOB [35], Thingworx [36], One Platform [37], Carriots [38], OpenIOT [39], SAP Internet-of-Things [40], and many more.

B. Semi-Distributed Systems

The semi-distributed systems are often based on session initiation protocols, whereas they afterward use direct communication between the connected devices. See Figure 2 for an overview of a semi-distributed system. Because of this, they usually contain a centralized point for coordinating the communication. Thus, semi-distributed systems are faster and to some extent easier to scale than centralized solutions, but they still have difficulties coping with requirement 2 and 5. Typical examples of these semi-distributed architectures include: ETSI M2M [41], SENSEI [42], ADAMANTIUM [43], and other platforms based on 3GPP IMS [44].

C. Fully Distributed Systems

Fully distributed systems operate in a peer-to-peer manner, where clients both store and administer the information locally on each entity without centralized components, see Figure 3. To achieve this, they often utilize hash tables to enable logarithmic scaling when the number of entities increases in magnitude. These systems do not contain any single point of failure and are thus more resilient, though the distribution itself often requires additional overhead in order to maintain an overlay. The main problem associated with fully distributed systems is, however, that they place a larger responsibility on the end devices, and thus have difficult to achieve requirement 3. Examples of such systems are the Global Sensor Networks (GSN) [45], the RELOAD architecture [46], and MediaSense [2].

IV. Technical Platform

Because the existing systems are unable to address the five requirements to enable a large scale IoT stated in Section II to enable a large scale IoT, a new solution is needed. Therefore, we have developed a platform based on how well the different categories achieve the requirements. An overview of this platform can be seen in Figure 4. In short, the platform is based on connecting sensor and actuators to form an IoT using current IP networking, fully distributed systems, peer-to-peer communication, and distributed hash tables (DHT). DHTs are distributed systems, which enable the storing of key-value pairs that can be utilized as a distributed storage service. A DHT

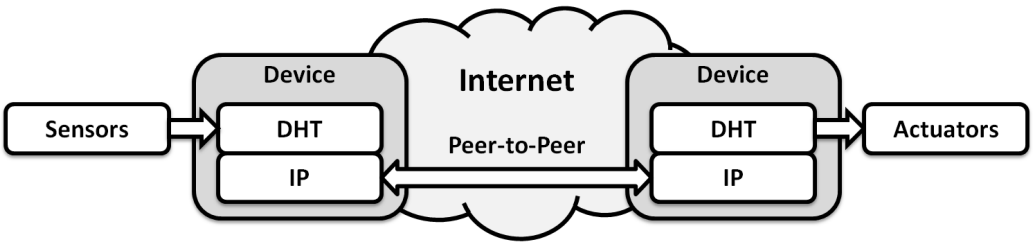

Figure 4. Overview of the proposed approach.

can therefore be seen as a form of DNS system but without the centralized servers. The DHT can be used to organize all connected entities in a logical structure which collaboratively stores data over distributed entities, where anyone can retrieve the data.

In detail, the platform should be IPv4 and IPv6 compatible which should make it future proof of many years to come, since the proliferation of the IPv6 standard is still not widespread. The platform should be scalable and avoid central points of failure by employing a fully distributed architecture, with no centralized points and a communication load that scales at worst logarithmically with the number of connected devices. The platform should employ peer-to-peer communication between the connected devices. As this type of communication is made directly between the data sources and the sinks without any unnecessary proxying of the data. Thus, only the normal Internet routing is added as delay, which is as optimal as one can achieve on the Internet. The platform should employ DHTs as an overlay network in order to become stable. Seamlessness in fully distributed systems is difficult to achieve, but there exists quite prominent NAT penetration techniques, which can be employed to solve most cases. Furthermore, constructing the ingoing protocols for lightweightness is a whole optimization problem in its own. But the idea is that the chosen DHT protocol should be lightweight by itself. Lastly, extensibility can be achieved by, for example, implementation specific solutions such as smart redistribution protocols, automatic updates, and distributed dynamic loading of components in runtime.

V. The SensibleThings Platform

In order to address the stated requirements we have created the SensibleThings platform, which is implemented as a layered architecture. A layered structure is chosen in order to more easily exchange modules as needed in both research and commercial applications. The layers and their default modules are explained in detail in the original article, but they are summarized here as well. The actual SensibleThings code is based on a fork of the MediaSense platform from 2013, which is a realization and implementation of the MediaSense architecture explained in [2].

A. Conceptual solution

The SensibleThings platform can be applied to a wide range of scenarios because of its versatile features. For example, how to perform low latency communication and finding information within the platform. This section presents these features and the corresponding conceptual solutions inside the platform.

1) Communication: Peer-to-peer communication is employed as the solution for the communication in the SensibleThings platform. The reasons behind this are many, but most important are the performance benefits. Peer-to-peer communication has an inherently low source to sink delay, because all communication is made with as few intermediate steps as possible. A fully distributed peer-to-peer system has no central points of failure and should thus be resilient to infrastructure failures. In a peer-to-peer system all participants provide their own network capacity, which is important because there is a significant investment involved in maintaining an IoT architecture. However, this is also a drawback since it causes vulnerability to denial of service attacks and an overlay must therefore be maintained to achieve good functionality and reliability. The are also many different transport layer protocols that could be utilized to send the peer-to-peer communication. Where one might strive for certain capabilities of the protocols, such as the flow control in TCP and the encryption of SSL. However, the conceptual idea in the SensibleThings platform is to enable the communication with as low overhead as possible, but still maintaining some type of reliability of the communication.

2) Finding information: The conceptual solution for finding information in the SensibleThings platform has been to employ a DHT. The SensibleThings platform associates the IP addresses of sensors with a Universal Context Identifier (UCI)[1] in the DHT. The UCIs can be seen as the unique identity of a sensor and looks like a combination between a URL and an E-mail address. To be specific, a UCI follows the structure of a Universal Resource Identifier according to RFC 3986[47] but we omit the scheme here to save space.

```
user@domain/path[?options]
```

Thus, a typical example a Celsius temperature sensors UCI owned by the user named Victor Kardeby is:

```
victor.kardeby@miun.se/temperature?unit=celsius
```

However, most DHT's only support the finding and resolving of an UCI to an IP address, they do not support searching for information. For example, the DHT can resolve a previously known UCI to an IP address, but it can not find the UCI's of sensors in a particular city. Hence, the conceptual solution for this is to employ distributed searching algorithms in the key space to enable intelligent searching of information and meta information in the system.

3) Publish and Subscribe: Most retrieval of information on the IoT is based on the publish and subscribe paradigm, hence the SensibleThings platform should support this. The conceptual solution for subscriptions in the Sensiblethings

platform is to create a distributed subscription system where each device handle its own subscribers. Thus, notifying them whenever the sensor value is updated.

4) Security: The information on the IoT often originates from sensors, which have sensed their surroundings. Therefore, the information can possibly contain private information, which could be utilized by malicious users. Hence, there is a need for security in the SensibleThings platform, both in the form of encryption of the data and authentication of those who may access it. Security is out of scope, but our conceptual solution is to encrypt the peer-to-peer communication, which can be done in a number of different ways. But it is important to remember that the encryption should be done without the use of a centralized authority and as lightweight as possible, because of the devices with limited hardware resources. After the communication has been encrypted, the authentication problem can be addressed. However, authentication in the platform should also be constructed in a fully distributed manner with low overhead. Thus, some type of distributed authentication and trust system should be employed.

5) Mobility: Another big problem is the mobility and connectivity of devices. In the course of seconds a device might change its IP address multiple times as it switches between a home WiFi and 3G/4G connectivity when a person leaves a house. This spontaneous change of IP address makes traditional mobility services ineffective[48] due to frequent updates to third party support. The lookup system could become unreliable as it might have obsolete information in the DHT. The conceptual solution to this is to employ different fully distributed solutions to reduce the convergence time of the DHT's IP address records and to estimate a persons behavior in order to predict the change and update the DHT beforehand.

6) Persistence: To store large amounts of information in a fully distributed peer-to-peer system is difficult, since each node might have limited capabilities. Furthermore, because each node is responsible for its own sensors and is the only source of its information, they are responsible for the persistence of their own information. The conceptual solution to create persistence in the SensibleThings platform is to introduce cloud based persistence for storing of important information and as an offloading system when a node exceeds its ability to persist its data. The cloud based persistence should only be used as a backup for extremely vital information, which should never be lost, such as history of medical sensors.

7) Reasoning: Intelligent reasoning is one of the final purposes of connecting sensors to together, to make applications alter their behavior depending on the context of the users. In order to create intelligent reasoning, both large statistical data and machine learning techniques can be used. These require computationally heavy calculations, data mining, storage space, and other resources. Generally, intelligence and reasoning is solved on the application layer at the respective endpoint and thus is not a part of the platform itself. Therefore, an application only has access to data that the user would be authenticated to access. However, data on the platform should of course be opened up to support more ubiquitous types of data mining. The SensibleThings platform has an extensive add-in system, which can add functionality, such as intelligent reasoning, as needed in runtime. Different types of intelligent reasoning engines have been applied on the data from the platform. For example, creating intelligence from continually changing user profiles called context schemas, which are based on relevancy of the information and derived from the algorithms in [49]. Furthermore, different types of machine learning techniques have been applied to estimate the context of a person, based on information from the IoT [50].

8) Interoperability: There exists a wide range of different sensor devices, which have different capabilities and support different protocols, both open and proprietary. This makes it difficult to build a platform that can utilize all the different hardware and to create interoperability between them. However, in the latest years there seem to have arrived a consensus between the different Wireless Sensor Network operating systems, since both major operating systems TinyOS [51] and Contiki [52] support the Constrained Application Protocol (CoAP) [53]. Therefore, the SensibleThings platform also support sensors and actuators, which expose CoAP interfaces. Furthermore, as the SensibleThings platform is far from the only platform for enabling IoT applications, there is a need to create interoperability between platforms as well. The conceptual solution to this in the SensibleThings platform is to create add-ins that can bridge between different platform technologies. For example, create an add in for bridging sensors connected to different REST-based [54] cloud services.

B. Architecture Layers

The SensibleThings platform is divided into five different layers, which can be seen in Figure 5. The interface layer exposes the platform's Application Programming Interface (API) to the applications, the add-in layer makes it possible to extend the platform with additional functionality, the dissemination layer addresses finding and retrieving data from other entities, the networking layer handles the IP connectivity, and lastly the sensor and actuator layer that connects sensor and actuators. Thus, each layer focuses on specific problems and the remainder of this section will describe the layers and their implementation.

1) Interface Layer: The interface layer is the public interface that applications use to interact with the SensibleThings platform. It includes a single component, the SensibleThings application interface, which is a generic API for developers to build their own applications on top of. The main problem that the interface layer addresses is related to requirement 5 on easy usage, namely how to make the platform easy to understand and easy to implement applications with. Different approaches were explored, but since almost all communication on the platform is done asynchronously, the listener Java pattern is typically used in the application interface. Hence, almost all interface access with the platform is done through normal function calls, whereas the values are returned in event listeners. The sensor location address scheme is abstracted such that they are represented by objects acquired by resolving UCIs, as defined in Section V-A2. In short, the SensibleThings platform has the following basic interface:

- **SensibleThingsPlatform(Listener)** a constructor that joins the distributed system and sets the listeners to use

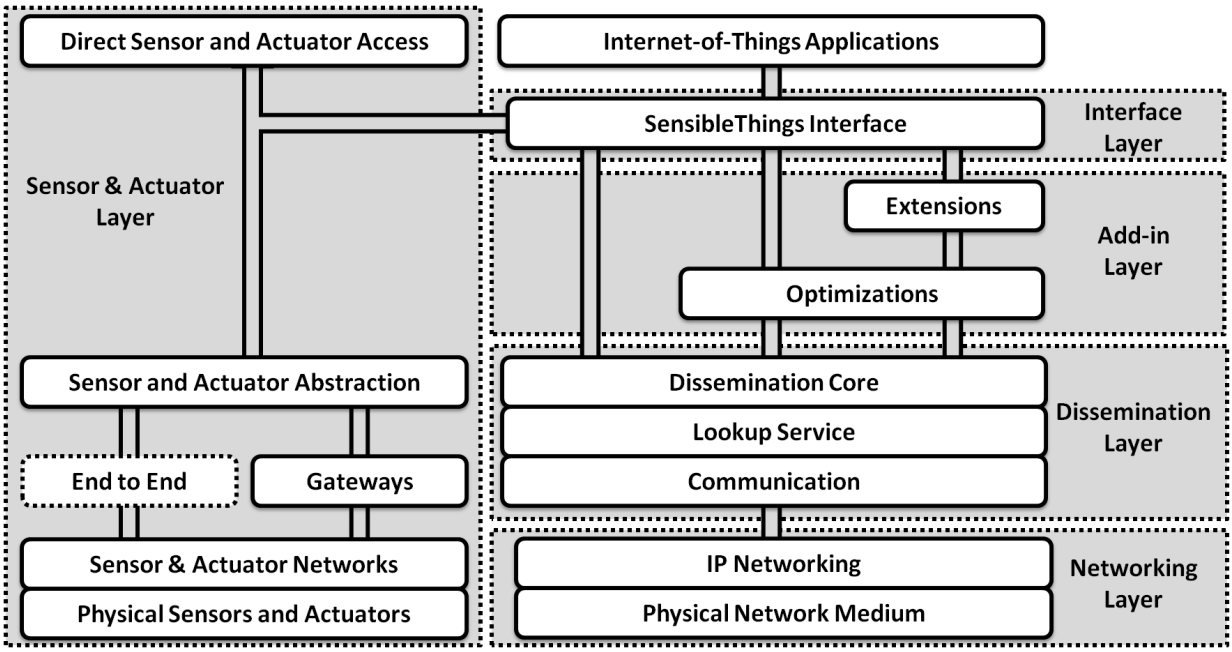

Figure 5. Overview of the SensibleThings platform's architecture

- **Register(UCI)** registers the specified UCI in the system
- **Resolve(UCI)** resolves a UCI to an node address
- **Get(UCI, Node)** retrieves the value of sensor, given its UCI and resolved node.
- **Set(UCI, Node, Value)** sets an actuator, given its UCI, resolved node, and the value to set
- **Shutdown()** performs a graceful leave from the distributed system

2) Add-in Layer: The add-in layer enables developers to add optional functionality and optimization algorithms to the platform beyond the basic primitives offered by the interface layer. For example, add-ins can help the platform meet specific application requirements, such as handling the available capacity in regards to computational power and bandwidth. The add-in layer deals with requirement 4 on being extensive. It manages different extensible and pluggable add-ins, which can be loaded and unloaded in runtime when needed. These add-ins are divided into optimization and extension components, but the platform can include any number of them at the same time. The add-in layer therefore handles how the add-ins should be managed, loaded, and the API's chain of command. In the current platform, there are still some limitations as these issues have not been prioritized. For example, some add-ins hijack functionality of the platform when enabled. Thus, some add-ins become mutually exclusive and will not function properly together. Examples of currently implemented add-ins are: caching, buffering, publish/subscribe, streaming, and password authentication.

3) Dissemination Layer: The dissemination layer enables sharing of information between all entities that participate in the system and are connected to the platform. A variant of the Distributed Context eXchange Protocol (DCXP) [55] is used, which offers communication among entities that have joined the distributed system, enabling exchange of context or sensor information with low response times. The operation of the DCXP includes first resolving of UCIs and subsequently transferring information directly between the peers. Therefore, the dissemination layer includes three components, a dissemination core, a lookup service, and a communication system. The dissemination core exposes the primitive functions provided by DCXP, the lookup service stores and resolves UCIs within the system, and the communication component abstracts transport layer communication. In short, the dissemination layer enables registration of sensors in the platform, resolving the location of a sensor in order to find it, and the communication to retrieve the actual sensor values.

The main design choices faced when developing the dissemination layer was regarding the choice of lookup service that supports requirements 1, 2, and 3, namely quick dissemination, being reliable with good scalability, and lightweight operation. There exists a number of DHTs that the platform could use, where we have chosen to focus on three prominent ones, namely Chord [56], Kelips [57], and P-Grid [58]. All three choices have their separate advantages and disadvantages. Chord uses a ring structure, which is difficult to maintain and has a logarithmic lookup time $O(log(N))$. Kelips uses affinity groups with a much simpler synchronization scheme and has fixed lookup time of $O(1)$, but it does not scale as well and has a larger overhead. P-Grid has a trie based structure with a logarithmic lookup time of $O(log(N))$ with load balancing and extra features such as range queries, but is also quite complex and difficult to maintain. In the current platform, both Chord and Kelips are completely reimplemented to operate within the platform and with the same license as the rest of the code. We have also experimented with the currently available P-Grid code, but since that is using multiple source code licenses (including propagating open source licenses), it cannot be a part of the SensibleThings code

TABLE I
SERIALIZATION METHOD COMPARISON.

Message Type	Binary	Java	Java compressed
Get	75 bytes	157 bytes	137 bytes
Kelips sync	856 bytes	2991 bytes	1252 bytes

at this stage. Currently, the platform defaults to the Kelips DHT implementation, simply because that code is more stable than the Chord implementation when nodes join and leave rapidly.

Another important design choice faced in the dissemination layer was the choice of communication protocol. Requirement 1 states that the communication should be fast with low overhead. Therefore, the aim was to have the useful payload data already in the first packet. Because of this, a variant of a Reliable User Datagram Protocol (RUDP) [59] is utilized as the default protocol. The problem with RUDP is however that the packets are sent in clear text, but to support industry applications the platform must provide the possibility of encryption. There exists several approaches for enabling this, such as different key exchange schemes with varying degrees of security and overhead. In the end, the decision was to support standard Secure Sockets Layer (SSL) encryption through the Java Secure Socket Extension to make it possible to encrypt the data if needed. The encryption is however only useful to prevent eavesdropping, not man in the middle attacks, because all certificates will be self signed by the end devices. There is also a significant overhead related to SSL, and since there is an initial handshake the useful data will no longer arrive in the first packet.

We have also made design choices in relation to the serialization of messages, namely how the messages are coded when sent over the Internet (before any encryption). Furthermore, minimizing the message size is tightly coupled to requirement 1 on fast dissemination and requirement 3 lightweight operation. For example, a binary serialization format is most suitable from a performance perspective, but a text based format is most usable from a human-readable perspective and a code-specific format is the easiest to program. In the end, the choice was to support all different serialization formats but the default is set to Java's object serialization with added GZIP [60] compression, to make it easier to develop new extensions. Likely, the Java serializer will be replaced in the future, in order to make transitions to other platforms and programming languages feasible. Table I shows a comparison of the message sizes depending on some of the different options for serialization. Where the Get message is a typical small message and the Kelips sync message is a typical large message.

4) Networking Layer: The networking layer enables communication between different entities over current IIP based infrastructure, such as fiber optic networks or wireless and mobile networks. Hence, the networking layer is separated into two inner components, an IP network and the physical network medium. In short, the networking layer thus abstracts any underlying IP-based network architecture. The problem faced in the networking layer was related to requirement 2 on stability and seamless communication. The first versions of the platform did not take NAT and firewalls into consideration, it only worked if all devices was on the public Internet. However, today almost all consumer devices are connected to the Internet through either NAT or some type of firewall, either in their home or at their work, but also on the mobile phone networks. The NAT and firewall problem is however only a question of configuration if the user is allowed to enable features such as port forwarding on the NAT routers. But that this rarely the case.

Multiple approaches were considered, ranging from IPv6 solutions, to Universal Plug and Play (UPnP), different hole punching techniques, and finally simple proxy solutions. The chosen solution first tries the normal approaches, such as direct connections and UPnP. If this fails, it instead utilizes distributed proxy nodes in the system. However, the proxy solution stands in direct contradiction to requirement 1 on real-time communication without unnecessary relaying of information and requirement 2 on no central points of failure. Because of this, it is only used as a last resort when there are no other possible options. There also exists problems related to the capacity of the Internet connections and the network delay, but since the SensibleThings platform is built on top of the existing Internet architecture, it cannot affect these parameters. Therefore, as long as the useful payload data is sent in the first packet, as in the RUDP implementation, it is considered to be transmitted as fast as possible by the underlying network infrastructure.

5) Sensor and Actuator Layer: The sensor and actuator layer enables different sensors and actuators to connect into the platform. The sensors and actuators can vary greatly and the platform therefore offers two options to connect them. Firstly, they can be connected directly if they are accessible from the application code, such as in the case of smartphone sensors. Secondly, the sensors and actuators can connect through the sensor and actuator abstraction. The abstraction enables connectivity either directly to wireless sensor networks or via more powerful gateways. Hence, the sensor and actuator layer is separated into five components: the directly accessible sensors and actuators, an abstraction component, different sensor and actuator networks, sensor and actuator gateways, and the physical sensors and actuators.

In the sensor and actuator layer, there were problems with the actual sensor hardware platforms that is available today, especially in regards to requirement 3 on being lightweight. Different vendors of sensors have different platforms that the sensors run on, especially when it comes to connecting large Wireless Sensor Networks (WSN). Typically, cheap analog sensors can be connected directly to a more powerful device, such as a smartphone or a Raspberry Pi [61]. But to connect traditional WSN architectures such as TinyOS [51] or Contiki [52], the platform must communicate via CoAP [53] or other lightweight protocols that they can handle. Therefore, in most of the examples we have utilized either smartphones with sensors already built in, or Raspberry Pi devices with attached sensors. However, any device that can run the Java code for the platform can be a part of the system, and any low end device that can communicate via CoAP can easily be connected via a more capable device.

C. Source Code License

One purpose of the platform is to make it available for industry partners to develop their own applications and

then commercialize the products, see requirement 5. This requirement made it impossible to use a strict and propagating open source license such as GNU General Public License (GPL). The amount of external code should also be kept to a minimum in order to enable an open source community centered around the platform with the power and possibility to easily incorporate changes into the code without navigating multiple licenses. In the end, the decision was to use the GNU Lesser General Public License (LGPL) that allows companies to make commercial products on top of the platform, without forcing their products to be open source as well, while forcing changes to the core to propagate back to the platform and allow the creation of an open source community centered on the platform.

VI. Results

The current results include launching our new development website for the SensibleThings platform (www.sensiblethings.se). This website will act as a portal for all developers who want to utilize the platform in their applications. For example, the website has developer packages to download, the complete source code, example applications, exercises, tutorials, and a wiki. All the code and resources is provided free and under the LGPL version 3 open source license. The remainder of this section will present measurements made on the platform, an evaluation of the platform in relation to the requirements, and some of the demonstrator applications, which have been built using the platform.

A. Measurements

Initial testing, demonstration, and evaluation of the platform has been conducted using a testbed with fixed and mobile access to the Internet. In terms of performance we have measured the platform to be on par with UDP traffic. This was made in two steps, first general expressions were created for the amount of messages sent in the platform, and secondly actual measurements of these expressions were performed in the testbed. The expressions were derived from the life cycle of a device within the system. This life cycle can be seen in Figure 6, which shows the external join rate λ_{join} and the amount of messages per join m_{join}. The rate of resolve operations in an application $\lambda_{resolve}$ and the amount of messages per resolve $m_{resolve}$. The rate of get operations in an application λ_{get} and the amount of get messages per get operation m_{get}. The rate a device has to perform maintenance λ_{maint} and the amount of messages per maintenance run m_{maint}. Finally, the leave rate of devices in the testbed λ_{leave} and the amount of messages for a single device to leave m_{leave}. General expressions for the total amount of messages sent within in the testbed can be seen in (6) to (9). In these, M_{join} represents the total amount of join messages being sent within the platform, which is expressed as the rate of new devices λ_{join} times the amount of messages for a single join operation m_{join}. In similarity, M_{leave} represents the total amount of leave messages being sent within the system, which is expressed as the rate of devices leaving λ_{leave} times the amount of messages for a single leave operation m_{leave}. The variable M_{maint} represents the total amount of maintenance messages being sent within the system. It is expressed as the rate of performing maintenance λ_{maint}, times

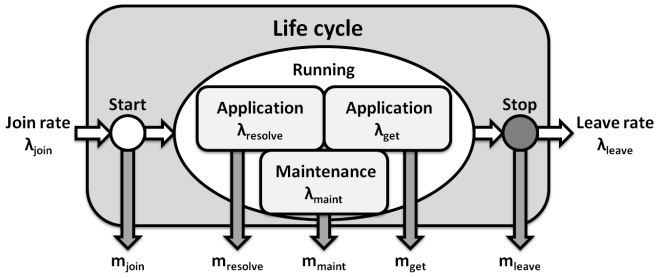

Figure 6. The life cycle of a device.

the amount of messages for a single maintenance operation m_{join} per device, times the total amount of devices in the testbed N. The variable M_{appl} represents the total amount of application messages being sent within the system. It is expressed as the rate of the resolve operations $\lambda_{resolve}$ times the amount of resolve messages for a single resolve operation $m_{resolve}$ and the rate of the get operations λ_{get} times the amount of get messages for a single get operation m_{get}. By summarizing these operations M_{join}, M_{leave}, M_{maint}, and M_{appl} we get the total amount of messages being sent within the whole system M_{total}

$$M_{join} = \lambda_{join} * m_{join} \quad (6)$$

$$M_{leave} = \lambda_{leave} * m_{leave} \quad (7)$$

$$M_{maint} = N * \lambda_{maint} * m_{maint} \quad (8)$$

$$M_{appl} = \lambda_{resolve} * m_{resolve} + \lambda_{get} * m_{get} \quad (9)$$

$$M_{total} = M_{join} + M_{leave} + M_{maint} + M_{appl} \quad (10)$$

We have also formalized expressions for the latencies in the testbed, which can be seen in (11) to (14). In these, L_{join} represents the latency for a device to join the testbed, which is expressed as the processing delay induced by the devices $d_{processing}$ and the amount of messages required for joining m_{join} times the network delay $d_{network}$. In similarity $L_{resolve}$ represents the latency for a device to perform the resolve operation in the testbed, which is expressed as the processing delay and the amount of messages for a single resolve operation $m_{resolve}$ times the network delay. Furthermore, L_{leave} represents the latency for a device to gracefully shutdown and leave the testbed, which is expressed as the processing delays and amount of messages for the graceful shutdown m_{leave} times the network delay. Lastly, L_{get} represents the latency for a device to get a sensor value from another device in the testbed, which is expressed as the amount of messages required for retrieving the data m_{get} times the network delay and any processing delays.

$$L_{join} = m_{join} * d_{network} + d_{processing} \quad (11)$$

$$L_{resolve} = m_{resolve} * d_{network} + d_{processing} \quad (12)$$

$$L_{leave} = m_{leave} * d_{network} + d_{processing} \quad (13)$$

$$L_{get} = m_{get} * d_{network} + d_{processing} \quad (14)$$

We can practically measure (8), (11), (12), (13), and (14) directly in the testbed environment. All other expressions are too highly dependent on the churn rates and the specific

TABLE II
LATENCY MEASUREMENTS FOR JOIN, RESOLVE, AND LEAVE OPERATIONS

Operation \ Device	Dell 780		Nexus 4	
	μ	σ	μ	σ
Join (L_{join})	464 ms	7.67 ms	2580 ms	1320 ms
Resolve ($L_{resolve}$)	36.8 ms	13.5 ms	423 ms	175 ms
Leave (L_{leave})	1.87 ms	0.129 ms	21.9 ms	12.6 ms

TABLE III
LATENCY MEASUREMENTS FOR THE GET OPERATION (L_{get})

Source \ Sink	Dell 780		Nexus 4	
	μ	σ	μ	σ
Fujitsu TX100	4.77 ms	0.145 ms	154 ms	43.5 ms
Padphone infinity	280 ms	136 ms	437 ms	92.8 ms
Pi_1 (Telia)	41.4 ms	2.08 ms	248 ms	40.6 ms
Pi_2 (Servanet)	47.1 ms	11.7 ms	209 ms	38.3 ms
Pi_3 (BBB)	76.3 ms	11.5 ms	277 ms	64.9 ms
Pi_4 (SUNET)	94.7 ms	61.2 ms	303 ms	68.9 ms
Pi_5 (Bahnhof)	106 ms	24.9 ms	289 ms	97,9 ms
Pi_6 (Acreo)	111 ms	36.7 ms	317 ms	123 ms
Pi_7 (Telia)	197 ms	24.6 ms	449 ms	59.7 ms
Pi_8 (Telia)	213 ms	35.1 ms	643 ms	98.2 ms
Pi_9 (Telenor)	370 ms	89.8 ms	1260 ms	937 ms

application scenarios, for any practical and useful measurements to be made. The measurements were performed using a total of 9 Raspberry Pi devices, 2 desktop computers, and 2 mobile device. All with heterogeneous connectivity and different network properties. The performance evaluation was performed on one of the mobile devices (a LG Nexus 4) and one of the desktop computers (a Dell OptiPlex 780), in order to evaluate the effects of the mobile Internet connection and its weaker hardware. Both the desktop computers was connected directly to the campus network with a public IP address and the mobile devices was connected via Telia's mobile 3G/4G network available at the campus in Sundsvall. The maintenance overhead of the whole testbed M_{maint} in (8) with all 13 devices connected, was measured to be on average 82.6 messages per minute with a standard deviation of 3.75 messages. The latencies for the different operations L_{join}, $L_{resolve}$, L_{leave} can be seen in Table II. Where, μ stands for the numerical average and σ for the standard deviation of the measurements. Finally, we measured the latency for retrieving the actual sensor values L_{get} from all the other connected devices, namely a desktop computer (a Fujitsu Primergy TX100), another mobile device (a Padphone infinity), and 9 Raspberry Pi devices. The results from these measurements can be seen in Table III. Where the Dell desktop computer and the Nexus 4 was used as measuring sinks and all the other devices acted as sensor sources.

All the measurements was setup with the two desktop computers and Pi_1, Pi_2, and Pi_6 connected directly with public IP to the platform, while the two mobile devices, Pi_3 to Pi_5, and Pi_7 to Pi_9 was connected via different NAT solutions. An overview of the measurement setup can be seen in Figure 7. To be specific, both desktop computers were connected via Mid Sweden University's SUNET 100/100 Mbps network with public IP addresses. The Nexus 4 was connected via Telia 3G 10/2 Mbps connection behind carrier grade NAT. The Padphone infinity was connected via Telia 4G 20/2 Mbps connection, also behind carrier grade NAT. Pi_1 was connected via Telia ADSL 30/12 Mbps connection with public IP. Pi_2 was connected via ServaNet 100/100 Mbps connection with

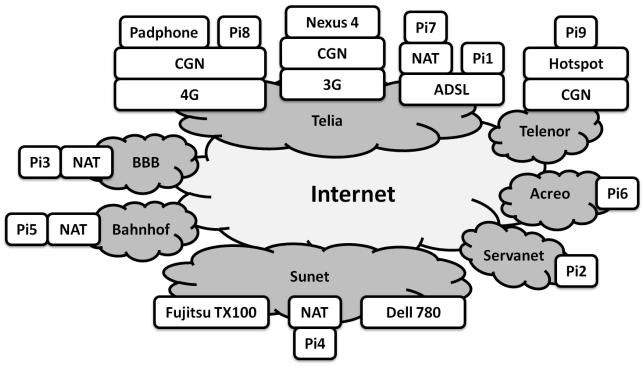

Figure 7. The measurement setup.

a public IP address. Pi_3 was connected via Bredbandbolaget (BBB) 100/10 Mbps connection behind a home NAT router. Pi_4 was connected via Mid Sweden Universities SUNET 100/100 Mbps network behind an enterprise NAT router. Pi_5 was connected via Bahnhof 100/100 Mbps connection behind a home NAT router. Pi_6 was connected via Acreo 100/100 Mbps connection with public IP. Pi_7 was connected via Telia ADSL 8/1 Mbps connection behind a home NAT router. Pi_8 was connected via Telia 4G 40/2 Mbps connection behind a mobile Internet router. Pi_9 was connected via Telenor 3G 10/2 Mbps connection behind both carrier grade NAT and mobile hotspot NAT.

B. Evaluation

In order to ensure that the platform achieves the requirements from Section II and evaluation of them have been made in the currently available platform using the Kelips lookup and RUDP communication. The first requirement on being fast was evaluated using (1), which states how efficient the communication is, as a percentage of the ideal case i.e., ping. To evaluate this, a subset of the setup as the previous measurements were used. We found that the $R_{latency}$ ratio of our platform is calculated to be 5.09 or around 500% using equation (1). This mean that the response times of our platform is 5 times worse than the most optimal ideal case of the shortest response time made possible by the underlying networks. Hence, there is room for improvement in terms of low latency, which is understandable because some of the nodes in the measurements were behind NAT and thus had to be tunneled.

The second requirement was evaluated using two metrics. The first was how the function for the average amount of messages per node changes as the number of nodes increase. Basically the communication load on each node in the system. This was measured in an emulated environment, where a single powerful computer could run multiple instances the platform nodes. Thus, the amount of nodes was incremented between 10 to 100 nodes in steps of 10 for each run. In each of these runs, the total number of messages was counted during a 10 minute period, after which the total was averaged down to messages per minute and then averaged in according to (2) with the number of nodes. The results from these measurements can bee seen in Figure 8, which shows that the current Kelips lookup has a decreasing average communication load, which converges to a constant about 11 messages. This is

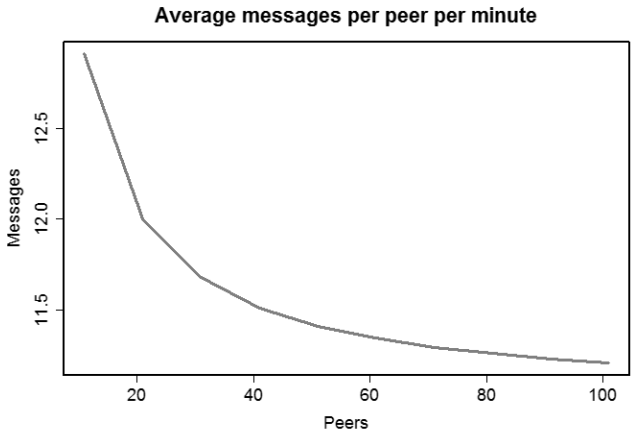

Figure 8. A graph of the average communication in the platform.

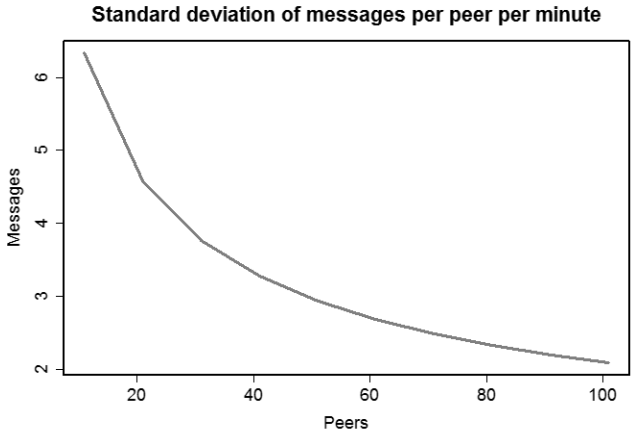

Figure 9. A graph of the standard deviation of the communication load in the platform.

because each additional node only adds a fixed amount of new messages into the system. This show that the Sensiblethings platform scales well, since each new device does not induce a heavier communication load for the ingoing devices. The reason for the higher average with few nodes and decreasing average is because the bootstrap device sends 21 extra messages compared to the other nodes.

In relation to requirement 2, the standard deviation of these values are interesting as well. As it indicated how even the system is and an increasing standard deviation indicates potential central points of failure. A graph of this can be seen in Figure 9, which shows that the standard deviation is decreasing toward a constant around 2 messages. Hence, the system becomes more evenly loaded as new nodes join the platform and there does not seem to exist any central points of failure. In similarity with the average, the only reason for the high standard deviation with few nodes is because the bootstrap device sends more messages compared to the other nodes.

In relation to requirement 3 on being lightweight, the resource utilization has to be evaluated. To evaluate this we tested three different devices running the platform, a Dell 780 workstation, a Raspberry Pi, and a Nexus 4 Android phone. In detail, we measured the memory usage, the processor utilization, and the network utilization of each of these devices when they run live platform. The memory usage and processor utilization measurements was taken directly from the device. But to determine the network utilization, first the maximum network capacity had to be determined. This was determined by using online broadband test tools [62], which show the practical maximum network capacity. This was done because the devices can not achieve their theoretical network speeds because of hardware limitations. For example, the Raspberry Pi has a 100/100 mbit network card but can practically only come up to around 70/30 mbit in actual throughput. Table IV show the results from all the utilization measurements, and thus the utilization metric was measured to be 26.7% in the platform, because that was the utilization on the most heavily loaded device property in our test. Namely, the processor load on the Nexus 4 device.

TABLE IV
UTILIZATION MEASUREMENTS.

Device	Dell 780	Raspberry Pi	Nexus 4
Processor	0.258%	11.0%	26.7%
Memory	1.029%	5.27%	2.10%
Network download	0.0194%	0.0552%	0.209%
Network upload	0.0200%	0.114%	2.10%

Requirement 4 on extensibility can be consider achieved because of the many add-ins developed for the platforms, as well as its component based nature. Where, for example, the lookup service can be exchanged in the future for a more suitable one. Thus, the platform is extensible and future proof because changes can be made to the platform as new standards and paradigms emerge. Lastly, requirement 5 on being easy to adopt and free to use, can be considered achieved since there exists simple tutorials and example code on the website. There is also no monetary cost directly involved in the platform because each node brings their own resources to the system and it has an open source code license that allows commercialization.

C. Demonstrator Applications

Proof-of-concept demonstrator applications have been built using many different devices, sensors, and actuators, in order to show the versatility of the SensibleThings platform. In most demonstrators we have utilized computers, Android smartphones, and Raspberry Pi devices to show the show that the platform works across different devices and network connections. In detail, this article will present functional demonstrators in three areas where the platform can applied to. The first type of application area is sensor reading applications, which simply retrieves sensor values over the platform to monitor things. The second application area is the intelligence home, which is a combination of sensors and actuators. Lastly, the third application, in which we show demonstrator applications of is surveillance applications, to monitor an area with images and video feeds. These are all typical IoT scenarios, but the platform itself is versatile enough to be applied to even more areas. We can foresee possible applications of the platform ranging from health-care, logistics, emergency response, tourism, and smart-grids, to more social-

Figure 10. Sensor reading example applications.

Figure 11. Intelligent home example application.

oriented applications such as crowd sourcing, dating services, and intelligent collaborative reasoning.

1) Sensor Reading Applications: The first demonstrator applications are simple sensor reading applications. In these demonstrators we have attached different types of sensors to Raspberry Pi devices and connected the to different types of networks. Figure 10 shows an example Raspberry Pi sensor device and two example android applications, which retrieves the sensor values in a ubiquitous real-time manner trough the SensibleThings platform. The left application show radon measurements from a Raspberry Pi device with an attached radon sensor, which, for example, a landlord or house owner can monitor their properties with. The right application show a graph of a temperature sensor attached to another Raspberry Pi device, which can be used, for example, to monitor the health status of sensitive objects in logistics, such as food or medicine.

2) Intelligent Home Applications: The second demonstrator application is a typical intelligent home scenario, where one wants to control appliances in a home. To this we have utilized an intelligent wall socket, which has built in sensor and actuator functionality. For example, it can turn and off each socket and measure the power consumption of the connected appliance, as well measuring the environmental temperature and humidity. Figure 11 shows this intelligent wall socket and a simple intelligent home application, both connected to the platform. The application can retrieve the sensor values from the socket, control the power, and measure the consumption of the appliances that are plugged in.

3) Surveillance Applications: The third demonstrator application is a typical surveillance scenario, where one wants

Figure 12. Surveillance example application.

to monitor a specific area. For example, to detect trespassers or to avoid collisions with obstacles. To this demonstrator application we have utilized Raspberry Pi devices with attached camera sensors, to then transfer the images over the platform. Figure 12 shows this Raspberry Pi camera device and an applications, which utilize the data from the camera. The application retrieves the camera images and performs image analysis to detect obstacles and anomalies. In this case, to detect if a person some other obstacle is present on a railroad track.

VII. Future Internet-of-Things Services

The IoT is often discussed [63], [64], [65], as one of the next steps in the evolution of everyday computing. However, the proliferation has so far only been marginal. Thus, the future of IoT services is still quite uncertain. Where the most prominent one is that there is still no real killer application for the IoT. The scenarios often discussed when advocating an IoT (logistics, health-care, intelligent home, surveillance, etc.) are simply not engaging enough to start the revolution. Furthermore, people are generally skeptical to sharing sensor information [66], which are tightly coupled to themselves, even though some people still share almost anything on social media such Facebook, Twitter, and Google+. Either way, we believe the open sharing of information is paramount to the proliferation of the IoT and without a killer application to really show the benefits to the common user, the proliferation will probably not happen.

The are also uncertainties in the business model [67], for example, the IoT will consume quite significant amount of network resources. This raise questions, such as who shall build this infrastructure and who shall stand for this cost. Basically, how can sensor manufacturer make money on IoT devices, how can operators make money on IoT traffic, and how can application developers make money from applications that utilize information from the IoT. There is also the question of ownership of the information, who owns the information on the IoT and thus determines the cost of using it. Furthermore, as we presented in Section III, there is still no single standard or commonly accepted solution for the IoT, there are still questions regarding the protocols and architectures to use. We believe that because of the vision of an open flow of information on the IoT, an open protocol, and free open source implementations has the most advantages, which is why we built the SensibleThings platform. But the standardization committees might have a different points of view on the openness of the information and protocols and the patent situation also plays a large role in the standardization efforts.

Because of all the uncertainties we believe that the true killer application of the IoT is caught in a form of catch 22. We believe that this killer application is the creation of truly intelligent behavior in all different applications, which can understand the situations and context of their users and react accordingly. The closest service available today is Google Now [68], which tries to achieve this behavior with the information Google has available. However, to truly create this type of functionality it will require large amounts sensor information, which is not yet available, since people do not have the hardware or are not sharing it yet. Furthermore, truly context aware applications also require large amounts of processor power to run different machine learning algorithms, which some one must pay the cost for. And since there is real business model yet, no one can take this cost responsibility for it. Hopefully, there will some type of consensus in the IoT research soon. So that the common user can start seeing the benefits what a global and ubiquitous IoT with open sharing of information can offer.

VIII. Conclusion and Future Work

In this paper, we presented the challenges we have encountered when researching and developing the SensibleThings platform. The first contribution is the identification of the requirements for a functional and fully distributed IoT platform in Section I. The second contribution is a short survey of existing IoT platforms, presented in Section III. The main contribution is however the SensibleThings platform itself, which is shown to fulfill all the stated requirements. The platform can disseminate information to end devices quickly with low overhead (requirement 1). It is reliable and operates without any central points of failure (requirement 2). The platform is lightweight and can run on mobile devices or Raspberry Pi devices (requirement 3). It is extensible (requirement 4) with all its add-ins and functionality. Finally, the platform is licensed under a well known and widely accepted open source license making it free to use and at the same time encouraging development of commercial a products (requirement 5). Therefore, our conclusion is that when compared to related work, the SensibleThings platform can be classified as a fully distributed solution, where the overhead and communication is kept as lightweight as possible. This is the only type of solution that will scale for billions of connected devices and still be able to disseminate sensor information with real-time demands.

Our future work are directed toward improving and optimizing the existing code, as well as investigating other possible choices of the DHT and communication protocol. We will also develop more extensions to satisfy specific applications demands, such as seamless integration with other IoT platforms and cloud infrastructures. Finally, we will investigate the possibility to create a formal open communication standard of the platform.

Acknowledgment

This research has been supported by grant 00163383 of the EU European Regional Development Fund, Mellersta Norrland, Sweden. The authors also want to thank our partners from industry and academia, in particular Acreo AB and Stockholm University.

References

[1] S. Forsström, V. Kardeby, P. Österberg, and U. Jennehag, “Challenges when realizing a fully distributed internet-of-things-how we created the sensiblethings platform,” in ICDT 2014, The Ninth International Conference on Digital Telecommunications, 2014, pp. 13–18.

[2] T. Kanter, S. Forsström, V. Kardeby, J. Walters, U. Jennehag, and P. Österberg, “Mediasense–an internet of things platform for scalable and decentralized context sharing and control,” in ICDT 2012, The Seventh International Conference on Digital Telecommunications, 2012, pp. 27–32.

[3] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. Johnson, “M2M: From mobile to embedded Internet,” IEEE Communications Magazine, vol. 49, 2011, pp. 36–43.

[4] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” Computer Networks, vol. 54, 2010, pp. 2787–2805.

[5] J. Hong, E. Suh, and S. Kim, “Context-Aware Systems: A Literature Review and Classification,” Expert Systems with Applications, vol. 36, 2009, pp. 8509–8522.

[6] L. Atzori, A. Iera, and G. Morabito, “From" smart objects" to" social objects": The next evolutionary step of the internet of things,” IEEE Communications Magazine, vol. 52, no. 1, 2014, pp. 97–105.

[7] M. Weiser, “The computer for the 21st century,” Scientific american, vol. 265, no. 3, 1991, pp. 94–104.

[8] D. E. O’Leary, “Big data, the internet of things and the internet of signs,” Intelligent Systems in Accounting, Finance and Management, vol. 20, 2013, pp. 53–65.

[9] L. Ericsson, “More than 50 billion connected devices,” 2011.

[10] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, Vision and challenges for realising the Internet of Things. EUR-OP, 2010.

[11] Z. Sheng, S. Yang, Y. Yu, A. V. Vasilakos, J. A. McCann, and K. K. Leung, “A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities,” IEEE Wireless Communications, vol. 20, 2013, pp. 91–98.

[12] P2413 Standard for an Architectural Framework for the Internet of Things (IoT). IEEE. [Online]. Available: http://iot.ieee.org/ [retrieved: December, 2014]

[13] AllSeen Alliance. [Online]. Available: https://allseenalliance.org/ [retrieved: December, 2014]

[14] European Research Cluster on the Internet of Things. [Online]. Available: http://www.internet-of-things-research.eu/ [retrieved: December, 2014]

[15] Hyper/Cat. [Online]. Available: http://www.hypercat.io/ [retrieved: December, 2014]

[16] Internet of Things Architecture (IoT-A). [Online]. Available: http://www.iot-a.eu/public [retrieved: December, 2014]

[17] Internet of Things Counsil. [Online]. Available: http://www.theinternetofthings.eu/ [retrieved: December, 2014]

[18] Internet of Things Europe. [Online]. Available: http://www.internet-of-things.eu/ [retrieved: December, 2014]

[19] Internet of Things Initiative (IoT-I). [Online]. Available: http://www.iot-i.eu/public [retrieved: December, 2014]

[20] IoT Forum. [Online]. Available: http://iotforum.org/ [retrieved: December, 2014]

[21] IP for Smart Objects (IPSO) Alliance. [Online]. Available: http://www.ipso-alliance.org/ [retrieved: December, 2014]

[22] oneM2M. [Online]. Available: http://www.onem2m.org/ [retrieved: December, 2014]

[23] Open interconnect. [Online]. Available: http://openinterconnect.org/ [retrieved: December, 2014]

[24] A. L. Valdivieso Caraguay, A. Benito Peral, L. I. Barona López, and L. J. García Villalba, “SDN: Evolution and Opportunities in the Development IoT Applications,” International Journal of Distributed Sensor Networks, vol. 2014, 2014, pp. 1–10.

[25] F. Hu, Q. Hao, and K. Bao, “A survey on software-defined network and openflow: From concept to implementation,” Communications Surveys Tutorials, IEEE, vol. 16, no. 4, Fourthquarter 2014, pp. 2181–2206.

[26] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian, "A Software Defined Networking architecture for the Internet-of-Things," in 2014 IEEE Network Operations and Management Symposium (NOMS). IEEE, May 2014, pp. 1–9.

[27] SicsthSense. [Online]. Available: http://sense.sics.se [retrieved: December, 2014]

[28] ThingSpeak. [Online]. Available: https://www.thingspeak.com [retrieved: December, 2014]

[29] Sen.se. [Online]. Available: http://open.sen.se [retrieved: December, 2014]

[30] Nimbits. [Online]. Available: http://www.nimbits.com [retrieved: December, 2014]

[31] Thingsquare. [Online]. Available: http://thingsquare.com [retrieved: December, 2014]

[32] EVRYTHNG. [Online]. Available: http://www.evrythng.com [retrieved: December, 2014]

[33] Paraimpu. [Online]. Available: http://paraimpu.crs4.it [retrieved: December, 2014]

[34] Xively. [Online]. Available: https://xively.com [retrieved: December, 2014]

[35] XOBXOB. [Online]. Available: http://www.xobxob.com [retrieved: December, 2014]

[36] ThingWorx. [Online]. Available: http://www.thingworx.com [retrieved: December, 2014]

[37] One Platform. [Online]. Available: http://exosite.com/products/onep [retrieved: December, 2014]

[38] Carriots. [Online]. Available: https://www.carriots.com [retrieved: December, 2014]

[39] Openiot: Open source solution for the internet of things into the cloud. [Online]. Available: http://openiot.eu [retrieved: July, 2014]

[40] Sap internet of things. [Online]. Available: http://www.sap.com/pc/tech/internet-of-things.html [retrieved: December, 2014]

[41] ETSI, "Machine-to-machine communications (m2m); functional architecture," (TS 102 690 V1.1.1), Tech. Rep., 2011.

[42] M. Presser, P. Barnaghi, M. Eurich, and C. Villalonga, "The SENSEI project: integrating the physical world with the digital world of the network of the future," IEEE Communications Magazine, vol. 47, no. 4, 2009, pp. 1–4.

[43] H. Koumaras, D. Negrou, F. Liberal, J. Arauz, and A. Kourtis, "ADAMANTIUM project: Enhancing IMS with a PQoS-aware multimedia content management system," International Conference on Automation, Quality and Testing, Robotics, vol. 1, 2008, pp. 358–363.

[44] G. Camarillo and M.-A. Garcia-Martin, The 3G IP multimedia subsystem (IMS): merging the Internet and the cellular worlds. Wiley, 2007.

[45] K. Aberer, M. Hauswirth, and A. Salehi, "A middleware for fast and flexible sensor network deployment," in Proceedings of the 32nd International Conference on Very Large Data bases, 2006, pp. 1199–1202.

[46] C. Jennings, B. Lowekamp, E. Rescorla, S. Baset, and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol," RFC 6940 (Proposed Standard), Internet Engineering Task Force, Jan. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc6940.txt

[47] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (INTERNET STANDARD), Internet Engineering Task Force, Jan. 2005, updated by RFCs 6874, 7320. [Online]. Available: http://www.ietf.org/rfc/rfc3986.txt

[48] G. Chellani and A. Kalla, "A review: Study of handover performance in mobile IP," CoRR, vol. abs/1312.2108, 2013, pp. 137–151.

[49] S. Forsström and T. Kanter, "Continuously changing information on a global scale and its impact for the internet-of-things," Mobile Networks and Applications, vol. 19, no. 1, 2014, pp. 33–44.

[50] S. Forsström and V. Kardeby, "Estimating contextual situations using indicators from smartphone sensor values," in 2014 IEEE International Conference on Internet of Things (iThings), 2014.

[51] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for sensor networks," in Ambient intelligence. Springer, 2005, pp. 115–148.

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

[53] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252 (Proposed Standard), Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://www.ietf.org/rfc/rfc7252.txt

[54] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," ACM Transactions on Internet Technology (TOIT), vol. 2, no. 2, 2002, pp. 115–150.

[55] T. Kanter, S. Pettersson, S. Forsstrom, V. Kardeby, R. Norling, J. Walters, and P. Osterberg, "Distributed context support for ubiquitous mobile awareness services," in Communications and Networking in China, 2009. ChinaCOM 2009. Fourth International Conference on, Aug. 2009, pp. 1–5.

[56] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in in Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, vol. 31. New Yourk, NY, USA: ACM Press, 2001, pp. 149–160.

[57] I. Gupta, K. Birman, P. Linga, A. Demers, and R. Van Renesse, "Kelips: Building an efficient and stable p2p dht through increased memory and background overhead," in Peer-to-Peer Systems II. Springer, 2003, pp. 160–169.

[58] K. Aberer, "P-grid: A self-organizing access structure for p2p information systems," in Cooperative Information Systems. Springer, 2001, pp. 179–194.

[59] T. Bova and T. Krivoruchka, "Reliable udp protocol," IETF, Tech. Rep., 1999.

[60] P. Deutsch, "GZIP file format specification version 4.3," RFC 1952 (Informational), Internet Engineering Task Force, May 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1952.txt

[61] Raspberry pi. [Online]. Available: http://www.raspberrypi.org [retrieved: December, 2014]

[62] L.-P. G. Hansen, I. C. Østhus, and P. Heegaard, "Online broadband test tools," in Proceedings of Norsk informatikkonferanse 2009, 2009.

[63] P. Balamuralidhara, P. Misra, and A. Pal, "Software platforms for internet of things and m2m," Journal of the Indian Institute of Science, vol. 93, no. 3, 2013, pp. 487–498.

[64] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," Future Generation Computer Systems, vol. 29, no. 7, Sep. 2013, pp. 1645–1660.

[65] S. Hachem, A. Pathak, and V. Issarny, "Service-Oriented Middleware for the Mobile Internet of Things: A Scalable Solution," in IEEE GLOBECOM: Global Communications Conference. Austin, United States: IEEE, Dec. 2014. [Online]. Available: https://hal.inria.fr/hal-01057530

[66] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," Computer Networks, vol. 57, no. 10, Jul. 2013, pp. 2266–2279.

[67] S. Turber, J. vom Brocke, O. Gassmann, and E. Fleisch, "Designing business models in the era of internet of things," in Advancing the Impact of Design Science: Moving from Theory to Practice. Springer, 2014, pp. 17–31.

[68] Google Now. [Online]. Available: http://www.google.com/landing/now/ [retrieved: December, 2014]