Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 720861, 17 pages
http://dx.doi.org/10.1155/2015/720861

Research Article

CSN: The Conceptually Manageable Sensor Network

Woojin Joe,[1] Jonghyun Lee,[2] and Karpjoo Jeong[3]

[1]*Department of Advanced Technology Fusion, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 143-701, Republic of Korea*
[2]*Department of Computer Engineering, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 143-701, Republic of Korea*
[3]*Department of Internet and Multimedia Engineering, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 143-701, Republic of Korea*

Correspondence should be addressed to Karpjoo Jeong; jeongk@konkuk.ac.kr

Received 19 October 2014; Accepted 9 February 2015

Academic Editor: Erik Buchmann

For the last decade, computer science and information technology have been rapidly expanding their application areas from computation and data processing inside computers to the real time monitoring and management of the real world outside computers. For those emerging applications such as Internet of Things, the flexible, scalable, and interoperable, collaborating sensor networks are crucial. In this paper, we present a sensor network system called the conceptually manageable sensor network (CSN). CSN is intended to support the conceptual management and integration of sensor networks and to provide well-defined and logical APIs for the facilitation of application development. The CSN design is based on the simple and intuitive conceptual model: sets and message queues. In order to minimize the system development efforts and to inherit the system quality of production level open source software, the CSN system is intentionally implemented as a set of extensions to the open source messaging system called ActiveMQ. We conducted some preliminary usability and performance tests for the current CSN implementation. For the usability test, we used data sets from a real world project for the energy-efficient management of Indoor Air Quality in subway stations. Both usability and performance tests showed promising results.

1. Introduction

Recent advances in information and communication technology enable a variety of devices, instruments, and appliances to evolve into "computer-like" systems with wireless communication and local computing. These days, those devices and instruments such as sensors, actuators, smart phones, robots, and home appliances can communicate with both each other and traditional computer systems such as web servers. Such interconnection of those devices and instruments is generally called *Internet of Things* (IoT) [1]. Major IoT applications include Environmental Monitoring, Infrastructure Management, Industrial Applications, Medical and Healthcare Systems, Building Management, and Transport Systems.

In such IoT applications where billion devices will be connected to the Internet in the future [2], an important class of devices is *sensing devices (sensors)* and *sensor networks* are a technology for managing those sensors [3, 4]. Sensor networks are assumed to be a network of sensor nodes. A sensor node is usually a system to include system software (e.g., TinyOS [5]), sensors, and a platform device (e.g., Raspberry Pi) with computing and communication capabilities. (In this paper, a sensor node is usually called just a sensor unless the explicit distinction is needed for technical clarity.) There has been a great deal of research effort on sensor networks, but most research work has been focused on *wireless sensor networks* (WSNs) [6]. WSN generally assumes a sensor node to be a small system with low battery power and limited wireless communication capabilities (e.g., short range radio communication).

Because of such hardware limitations, most research work on WSN has been aimed at optimizing network communication and on-board computation, especially with respect to energy consumption [7–9]. Due to such optimizations, WSN systems are usually aimed at and customized for specific application domains or technologies [10–13]. However, these conventional WSN systems fail to address the *ease of application development*, the *intuitive management of sensor*

networks, and the *integration of sensor networks* [12–14]. They are crucial issues in future IoT applications where a large number of heterogeneous sensor networks are developed and integrated for various applications.

In this paper, we propose a sensor network system called the *Conceptually Manageable Sensor Network System* (CSN). CSN is a novel approach to sensor networks designed to address those challenging issues in future IoT applications. However, CSN is not intended to replace conventional WSN systems, but to collaborate with them. The CSN approach to sensor networks is as follows.

(i) *Conceptual Approach.* CSN intends the application developer or the system administrator to perceive and to manage sensors and sensor networks conceptually. In CSN, a sensor network is modeled and in fact managed simply as a set of sensor networks where a sensor is considered to be just a singleton sensor network. In addition, a sensor network is a logical entity to produce a stream of data records. This conceptual design approach facilitates the management and integration of sensor networks.

(ii) *Application-Centric Approach.* Instead of forcing the application to deal with system-specific characteristics or requirements of sensor nodes or WSN software, CSN provides the application with logical and intuitive APIs that hide those characteristics or requirements from the application. In CSN, a sensor network (also, a sensor) is *logically modeled and in fact, physically managed as a message queue* (in fact, the messaging model such as JMS) [15]. This application-centric design approach facilitates the development of applications.

(iii) *Lightweight Approach.* We design the CSN system to be *a set of extensions to a general purpose messaging system*, instead of implementing the CSN system from scratch. The set of extensions can be designed to be independent of specific messaging system products because such messaging systems are generally based on a well-defined messaging model and specification such as JMS [16]. The current CSN system uses an open source free messaging system called ActiveMQ [17]. This lightweight design approach not only minimizes the development work but also inherits the high performance and reliability of production-level open source free system software.

This CSN project is not intended for the development of a conceptual model or system model for sensor networks but aimed at the development of sensor network middleware to be used for real world applications. In fact, we are currently planning to apply the current CSN system to real world applications such as the WISE project [18]. The WISE project is aimed at the development of technology and infrastructure for urban meteorological information services.

The rest of the paper is organized as follows. In Section 2, we explain the conceptual design of CSN. Sections 3 and 4 describe the system design and the implementation of CSN, respectively. In Section 5, we present experiments and show their usability and performance test results. These experiments are carried out with sensor data from a real world monitoring application. Section 6 discusses related work. In Section 7, we finally conclude this paper and talk about future work.

2. Conceptual Design

The CSN design focus and approach are significantly different from those of traditional sensor networks in that CSN is focused on the conceptual and application-oriented management. In this section, we explain the conceptual design of CSN in two ways: *Organization* and *Data Delivery*. The system design is presented in Section 3.

2.1. Sensor Network Organization Model. In CSN, a sensor and a sensor network are conceptually defined as follows.

(i) *Sensor.* A sensor is assumed to generate a stream of data records. For each sensor, the data format is assumed to be fixed, identical, and known. Each sensor has a unique ID that is assigned on the time of registration. Each sensor is assumed to have a network communication capability.

(ii) *Sensor Network.* A sensor network is *a set of sensor networks* (which are called network members). A single sensor itself is also considered to be a sensor network (which is called a *singleton sensor network*). A sensor network merges multiple data streams from the network members into a single data stream. As with a sensor, each sensor network has a unique ID that is assigned on registration.

(iii) *Semantic Annotation of Sensor Networks.* In CSN, a sensor network can be associated with multiple *semantic concepts (semantic tags)* such as CO_2. Semantic tags can also be symbolic names or IDs for arbitrary objects or concepts in the real world. For example, the symbolic name for a specific subway station (e.g., "Union Square") can be used as a semantic tag. A sensor network can be annotated with multiple semantic tags. In this paper, the set of semantic tags for a sensor network is represented as the list enclosed by angled brackets such as "[CO2, Tunnel, STA-101]."

Note that for a sensor network, its semantic annotation does not rule or constrain its membership in CSN. Semantic annotation is used to describe the characteristics or properties of sensor networks. For example, a sensor network with "CO2" as a semantic tag does not necessarily include all the CO_2 sensors. In CSN, semantic tags are mainly used to support the searching and administration for sensor networks by semantic tags. For example, the user can search sensor networks with the following query: "*find all the sensor networks with PM10 and 'STA-101' as their semantic tags*."

The CSN conceptual organization of sensor networks is illustrated in Figure 1 where we apply CSN sensor networks to an example from a real world IoT application for the energy-efficient management of Indoor Air Quality (IAQ) in subways [19]. The energy-efficient IAQ management requires the real

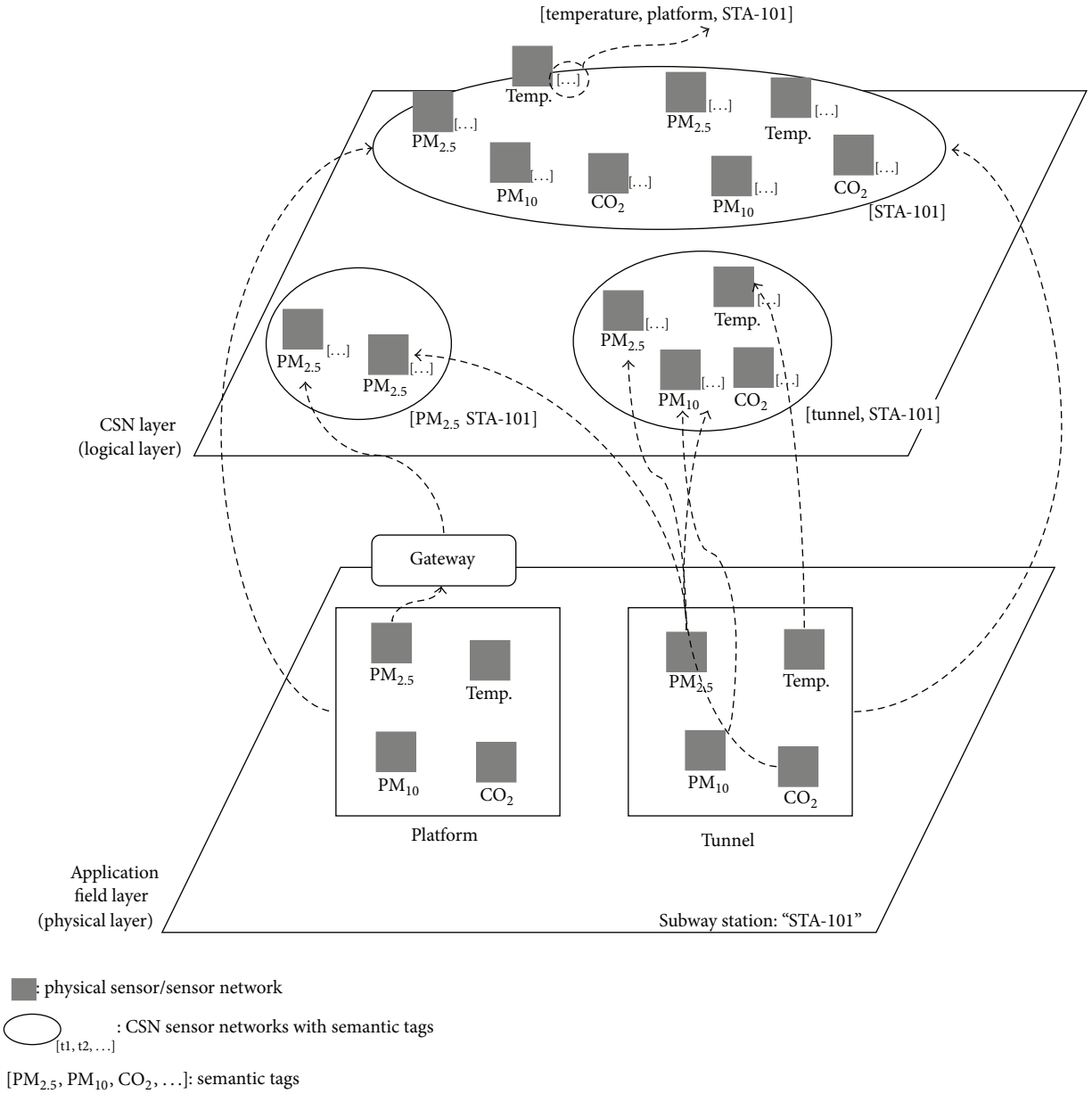


Figure 1: Sensor network organization model.

time monitoring of various air quality factors such as PM_{10} (particulate matter up to 10 micrometers in size), $PM_{2.5}$ (up to 2.5 micrometers in size), CO_2, Temperature, and Humidity.

In Figure 1, there are two layers: application field layer (physical layer) and CSN Layer (logical layer). The application field layer shows physical sensors in a subway station that consists of platforms and tunnels. In this figure, the sensors for PM_{10}, $PM_{2.5}$, CO_2, and temperature in the platform are formed into a WSN, and therefore, there is a gateway node. The tunnel also has PM_{10}, $PM_{2.5}$, CO_2, and temperature sensors, but those sensors are independent.

The CSN logical layer has three sensor networks. The first sensor network is intended for all the $PM_{2.5}$ sensors in the subway station. Its tags are "[PM2.5, STA-101]." The second sensor network is set up for all the sensors in the tunnel. Its tags are [Tunnel, STA-101]. The third sensor network is intended for all the sensors in the subway station. Its semantic tags are [STA-101].

The design rationales for this conceptual design are as follows. First, many sensor systems currently available can be configured to behave like the logical model of a sensor in CSN. In the real world, there are various kinds of hardware sensor devices and instruments being used. In addition, there are various *sensor-dependent* data loggers or DAQ systems that are usually developed and used only for particular sensors [20–22]. However, once those hardware sensors are integrated with data loggers or DAQ systems, most of the final integrated sensor systems to be deployed in the field are

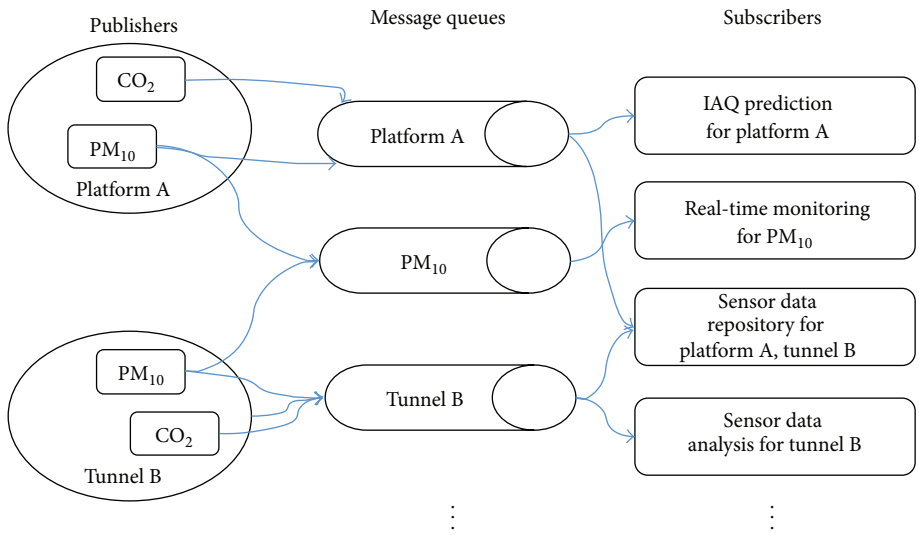


Figure 2: Sensor data delivery Model.

logically similar to the conceptual model of a sensor in CSN. That is, each of those systems can be considered to generate a stream of sensor data records and to send them to a remote information system by network communication.

Second, for sensors that do not have data loggers or DAQ systems, we think we can extend them to work as CSN-compatible sensors. There are a number of IoT application development platforms that facilitates the development of data loggers or DAQ systems. Such platforms include such as Qualcomm IoE Development Platform [23], Intel Galileo [24], Arduino [25], and Raspberry PI [26]. CSN supports a sensor agent (called CSN Sensor Agent) to interact with those sensors and to make them behave like CSN-compatible sensors. Such sensor agent can be easily run on those IoT platforms. More specific information about the CSN Sensor Agent will be explained in later sections.

In summary, CSN aims at the *software technology to build logical sensor networks from those available working sensor systems*. Furthermore, CSN is also intended to include not only just individual physical sensors but also other sensor network systems that can be modified to support our CSN conceptual model. Therefore, the conceptual design of CSN requires us to consider two layers of sensor networks: logical CSN layer and physical hardware sensor (or sensor network) layer.

2.2. Sensor Data Delivery Model. In the CSN conceptual model, a sensor network (or a sensor) is defined to generate a data stream. Furthermore, CSN also provides a conceptual model for sensor data delivery: how sensors send data out and applications retrieves data. The conceptual data delivery model is based on *the Publish/Subscribe Model* (e.g., JMS Messaging Model) [15]. The Publish/Subscribe Model is based on three concepts.

(i) *Message Queue. (In this paper, a message queue and a topic are interchangeable unless the explicit distinction is required.)* It is a communication or data exchange channel (or, queue) for delivering a stream of sensor data records. Therefore, the data stream from a sensor network can be considered as a message queue in the Publish/Subscribe Model. In CSN, a message queue is automatically created and assigned to a new sensor network.

(ii) *Publisher.* It sends a stream of data records into a message queue. For publication, it usually inserts a new data record into the message queue when the record is generated. In CSN, each sensor network is a publisher.

(iii) *Subscriber.* It receives a stream of data records from a message queue. In CSN, applications or CSN management systems such as Data Manager are subscribers.

Figure 2 illustrates how the CSN data delivery model based on message queues works. In this example, there are four sensors (singleton sensor networks) and two sensor networks with those sensors as network members. There is a message queue for each sensor network. The message queue has network members as its publisher. In this example, a message queue for sensor network "Platform A" has two publishers (sensors): CO_2 and PM_{10}. These sensors are also publishers for their own message queues.

The Publish/Subscribe Model (e.g., JMS) is a well-known model and is efficiently supported by messaging systems that are also called Message Oriented Middleware (MOM) [27]. There are currently a number of commercial or open source free messaging systems available [17, 28–30]. The Publish/Subscribe Model-based design of data delivery enables us to use a conventional messaging software system for CSN. This approach allows us to avoid the heavy implementation and performance optimization work for a data delivery system.

The Publish/Subscribe Model based data delivery is motivated by our observation: the data access to a sensor network

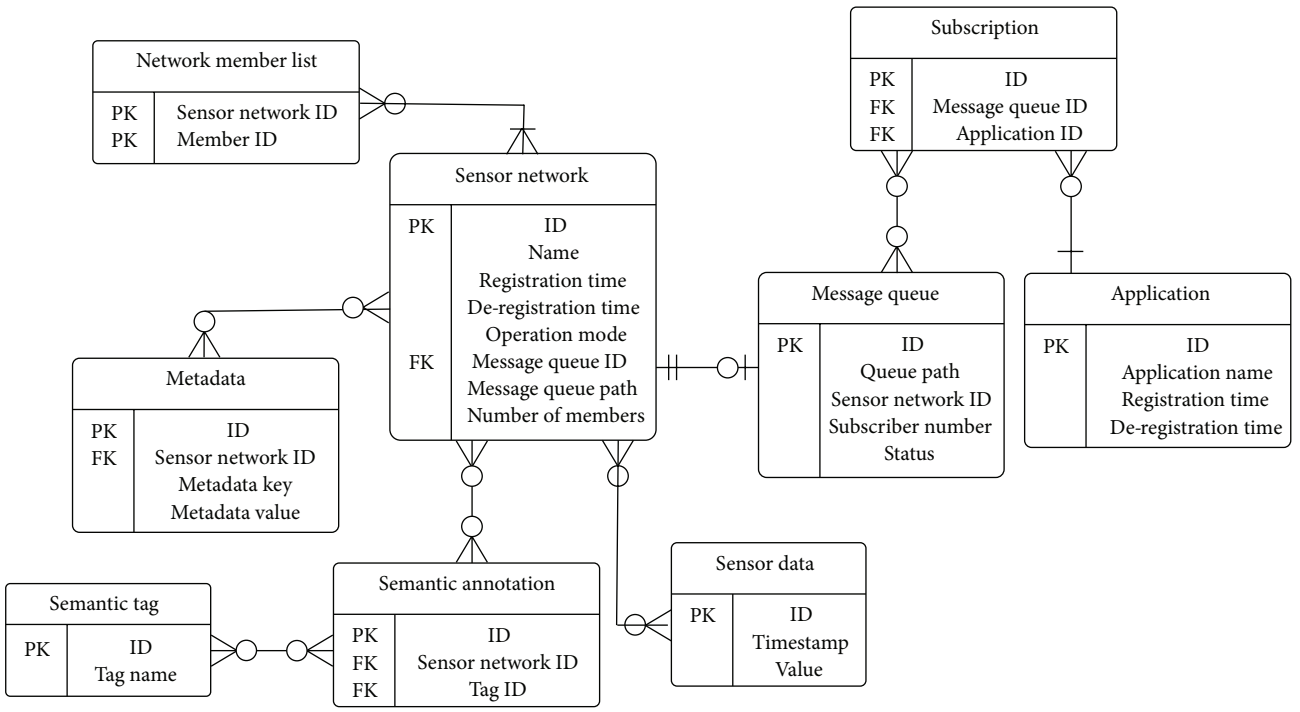


Figure 3: ER diagram of CSN system data model.

is difficult to model as the conventional Client/Server Model [13, 31]. In the conventional client/server model, a client is assumed to know the well-defined service interface of its server and to interact with its server in a synchronous way (e.g., in a blocking and request/response mode). However, sensors generate data on their own, independently of applications, and do not provide well-defined service interfaces. Therefore, the blocking mode based request/response interaction in the Client/Server Model is not effective for applications to use sensors.

3. System Design

3.1. System Data Model for CSN. As explained in Section 2, CSN is based on the explicit conceptual model that facilitates the system understanding, the effective application development, and flexible system extensions. In CSN, the actual system design is intended to be as consistent with the conceptual model as possible. Such consistency usually makes the system structure simpler and more logical. Our strategy for such consistency is to manage the states of major system components *explicitly in system data tables*: sensor network, message queue, application, sensor data, and semantic annotation. The lists of members for sensor networks are maintained as separate data table network member list, although the data table is logically a part of the Sensor Network table.

These system data tables contain a data record for metadata and status information about each sensor network, each message queue, each application or each semantic tag, respectively. The ER diagram for those system data tables is given in Figure 3. These data fields in those tables are self-explanatory in their names and we do not explain them explicitly.

3.2. System Architecture. Major system components (or modules) of CSN are as follows.

(i) *Sensor Agent.* The CSN runtime system has a CSN Sensor Agent for every hardware sensor. The Sensor Agent runs inside the CSN runtime system or is embedded into the data logger or DAQ system for the sensor. It hides system-specific sensor characteristics and provides the CSN conceptual model-based interface between the hardware sensor system and the other CSN system components. The Sensor Agent supports the distributed data delivery mode.

(ii) *Data Deliverer.* The CSN runtime system delivers all the sensor data by message queues. In fact, the CSN runtime system is designed to use a conventional messaging system for data delivery. ActiveMQ [17] is currently used as the Data Deliverer.

(iii) *Sensor Network Manager.* For sensor networks including a sensor, the CSN Sensor Network Manager supports registration/deregistration, semantic annotation with tags, and the management of the runtime state.

(iv) *Data Manager.* The CSN Data Manager supports three features: Data Logging, Data Searching and Centralized Data Delivery.

(v) *Message Queue Manager.* Message queues are created and managed by the Data Deliverer (which is in fact an independent messaging system), but the CSN

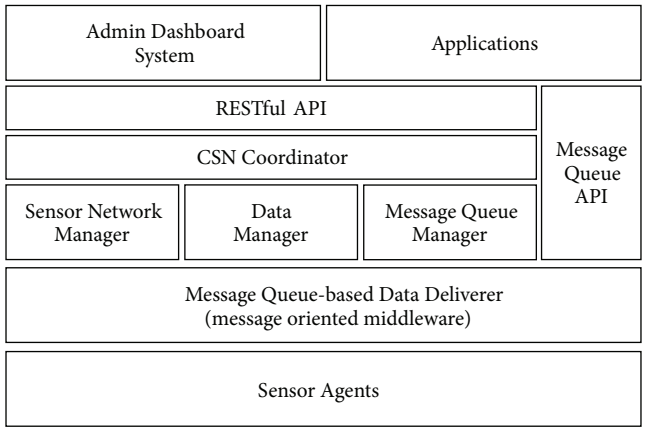


Figure 4: System Architecture.

Message Queue Manager decides and invokes management services in the Data Deliverer and maintains the current state about message queues.

(vi) *CSN Coordinator.* The CSN Coordinator coordinates CSN system components to handle all requests for configuration or administration from users, applications or the system administrator.

(vii) *APIs (RESTful and Message Queue Based).* The CSN runtime system provides two types of APIs: one for Sensor Data Publication/Subscription (Message Queue API) and the other for System Administration (RESTful API).

(viii) *Admin Dashboard.* The CSN Runtime System supports a simple admin system in a dashboard style.

Figure 4 shows the architecture of CSN.

3.2.1. Sensor Agent. As explained in Section 2, a sensor is modeled as a logical entity and therefore various real sensor nodes or WSN systems must be extended to operate according to the logical sensor model in CSN. For the development of such extensions, CSN provides the CSN Sensor Agent. The Sensor Agent can be used in two ways: *Local Embedded Mode* and *Remote Agent Mode*. In the local embedded mode, the CSN Sensor Agent code is installed inside the communication modem integrated into a sensor hardware system. Such conventional modems support network communication and usually have an embedded system such as Linux, Java Virtual Machine, and Android.

On the other hand, in the remote agent mode, the CSN Sensor Agent runs as a separate process or thread in the CSN runtime system. The remote agent mode assumes a kind of gateway node in sensor nodes or sensor networks that can communicate with the CSN Sensor Agent. In this case, the code in the Sensor-specific Sensor Library is added to interact with specific gateway nodes. Figure 5 shows how both the local embedded mode and the remote agent mode are organized.

In CSN, the Sensor Agent is intended to hide system-specific sensor hardware or software details from the other CSN system components and to provide the CSN logical sensor model-based interface to real sensors. The Sensor Agent code consists of two components: *Sensor-specific Sensor Library* and *Uniform Sensor Library*.

The CSN Uniform Sensor Library is the code libraries designed to provide the CSN logical sensor model-based interface for the other CSN system components such as the CSN Data Manager. The CSN Sensor-specific Sensor Library is a collection of the code libraries each of which is designed to implement the Uniform Access Interface for a certain specific sensor system.

For the Sensor Agent, we currently have a core architecture design, but do not have a full system design and supports for various sensor nodes or sensor networks. The Sensor Agent requires separate analysis and development work for each individual sensor nodes or sensor networks. This will be one of our major future work.

3.2.2. Data Deliverer. In CSN, data delivery is the most important operation with respect to both functionality and performance because the main role of sensor networks is to delivery data from sensors to applications. Therefore, it is crucial to design the CSN Data Deliverer to be scalable and reliable.

In CSN, we model the delivery of sensor data *exactly as the standard messaging model*, that is, Java Message Service (JMS) [16]. Therefore, any system software to support the standard messaging model can be used for sensor data delivery in CSN. Such messaging system is also called Message Oriented Middleware (MOM). There are a number of both commercial and open-source free messaging systems [17, 28–30]. These systems already support those features that the CSN system requires.

Therefore, we design the Data Deliverer to be just a wrapper for a messaging system (ActiveMQ in the current implementation). This way, we can use the existing performance and reliable features in the messaging system and at the same time keep the CSN system independent of any particular messaging software product. Thus, any available messaging system that supports the JMS protocol can be used as the Data Deliverer, later.

3.2.3. Sensor Network Manager. The CSN Sensor Network Manager supports the *administrative services for sensor networks* and maintains *their runtime states*. It provides the following features.

(i) *Registering a Sensor.* All sensors must be registered before they are serviced. When a sensor is registered, it is regarded as a singleton sensor network and a message queue (topic) is automatically created and assigned to it.

(ii) *Adding Metadata about a Sensor.* Additional information about a sensor (e.g., symbolic product model info, sensor type, and measurement unit) can be added for later management. Such metadata is stored as a key-value form.

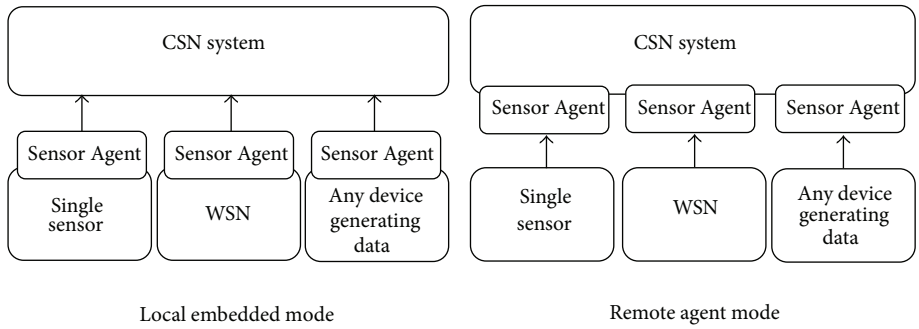


Figure 5: Operation modes of the sensor agent.

(iii) *Creating a Sensor Network.* The creation of a new sensor network involves selecting a collection of sensors (single-sensor networks) and registering the collection as a sensor network in CSN. The registering process is the same as for a singleton sensor network.

(iv) *Changing the Membership of a Sensor Network.* In CSN, the membership of any sensor network can be easily modified with respect to system management. However, since such membership changes in a sensor network affect the delivery of sensor data during runtime, the user or the administrator must consider that issue when changing the membership of a sensor network. This issue is also addressed when we discuss a performance test.

(v) *Semantic Annotation of a Sensor Network.* CSN allows the application or the user to annotate sensor networks with semantic tags. Examples of semantic tag are CO_2, "Room#816," and "Year 2010." The application or the user can search sensor networks of interest by semantic tags.

3.2.4. Data Manager. The CSN Data Manager supports three features: *Data Logging*, *Data Searching*, and *Centralized Data Distribution*. In the Data Logging mode that is by default, every sensor data is automatically stored into a permanent storage system and then can be queried and retrieved by the application or the user, later. For the performance reason, the Data Logging feature can be turned off. The CSN permanent storage system for sensor data is designed to be based on stream databases such as MongoDB [32], AURORA [33], NiagaraCQ [34], StreamInsight [35], and Infosphere Streams [36].

Currently, we use MongoDB as our permanent storage system. Table 1 describes the format of a sensor data record to be stored in the permanent storage system. An actual sensor data is represented in JavaScript Object Notation (JSON). JSON is a simple data exchange standard that is text-based and widely used [37]. The basic format of JSON is a collection of key-value pairs where keys and values are specified as strings. An example of a sensor data record is as follows:

{"ID": "8712," "Timestamp": "2014-07-07 17:35:08," "Value": "28.45"}.

Table 1: Data format for sensor data records.

Attributes	Data type	Contents
ID	Integer	Sensor network ID assigned by CSN system
Timestamp	Datetime	Timestamp when the sensor data is created
Value	String	Measured value from sensor

In the format, we currently assume the data type of Field *Value* to be a single-valued text string. In the current design of CSN, the management of the sensor data type is assumed to be the application developer's responsibility. That is, the application is expected to parse the string in the *Value* field. The metadata in the system data table (i.e., Sensor Network) can be used to contain type information for the Value field that can be JSON.

For *Data Searching*, the Data Manager allows the user or the applications to query and retrieve a certain part of the data stream for a sensor network. CSN currently support simple queries. A search query consists of a sensor network ID (data stream) and a search condition. The search condition can be either an interval of time or the number of the latest records.

The interval of time can be specific minutes, hours, or dates (e.g., "May 1, 2014"). It can also be a pair of start time and end time. For example, if the search condition is "May 1, 2014," then the Data Manager returns the stream of sensor data records to be published within the specific data, May 1, 2014. In addition, the Data Manager also supports the centralized data delivery mode that will be explained in Section 3.2.9.

3.2.5. Message Queue Manager. In CSN, the Data Deliverer (in fact, a messaging system) provides message queues for the delivery of sensor data. However, the Message Queue Manager supports the administration of those message queues as follows.

(i) *Creating a Message Queue.* In CSN, every sensor network must have a message queue. Therefore, when a new sensor network is registered, the Message Queue Manager requests the Data Deliverer (in fact, only a messaging system) to create a new message queue for the sensor network.

Table 2: Semantic tag based query API for sensor network searching.

Description	Search the sensor network IDs which have semantic tags specified in the query
Resource URL structure	/csn/search
Method	POST
Parameter	Target (required): networks or sensors.
Request body parameter	Operations: AND, OR, or NOT. Tags: semantic tags for searching sensor networks.

Request body sample	{
	"operation": "and",
	"leftOp": "Tag1",
	"rightOp": {
	"operation": "and",
	"leftOp": "Tag2",
	"rightOp": "Tag3"
	}
	}

Returns	The ID list of sensor networks matching the searching condition

(ii) *Removing a Message Queue.* When a sensor network is deleted, the Message Queue Manager asks the Data Deliverer to delete its corresponding message queue.

(iii) *Getting Enqueued/Dequeued Message Counts.* The Message Queue Manager maintains the information about how many messages have been enqueued and dequeued.

(iv) *Getting Subscriber List.* This list is basically used to maintain the list of the current subscribers. This information also determines that message queues can be removed or not. Only while a message queue (topic) has no subscribers, then the message queue cannot be removed.

3.2.6. CSN Coordinator. The CSN Coordinator is responsible for configuring and coordinating all the other system components. Its major features are as follows.

(i) *Initialization.* The Coordinator initializes and starts all the system components.

(ii) *Termination.* The Coordinator first saves critical system information into the permanent system storage and then closes all the other system components.

(iii) *System Backup and Restore.* The Coordinator performs the entire system-level backup and restores operations.

(iv) *System Configuration.* The Coordinator supports the system configuration services.

3.2.7. RESTful APIs. In CSN, we design the API to the management services to be RESTful [38]. The motivations for the RESTful design of the CSN APIs are as follows. First, the RESTful APIs are simple and intuitive to understand and to use. Second, the APIs are very easy to implement and require simple runtime Web infrastructure such as a Web server and a Servlet Engine. Finally, the RESTful APIs facilitate the design and implementation of interactions between external application clients and the CSN runtime system components.

Currently, CSN supports three classes of RESTful APIs: *Sensor Network Management, Data Management, and Coordination*. First, the Sensor Network Management API is intended for the management services for sensor networks. Sensor networks, message queue names, metadata, and semantic tags are treated as RESTful resources.

In the Sensor Network Management API, for a certain sensor network, the information about which sensor members it has is meaningful for checking the configuration of the sensor network. For example, the GET request with URI "/csn/networks/8712" asks for the network resource whose ID is 8712, then it returns the network data such as network name, member count, topic id and so on.

Second, The Data Management API supports two types of queries: *Semantic tag based query for sensor network searching* and *Time interval based query for data stream access*. The semantic tag-based query processing allows the user or the application to search sensor networks by a logical expression of multiple semantic tags (e.g., "PM10 & Platform"). The logical expression supports the three logical operators: AND, OR, and Negation. Table 2 presents the syntax of semantic tag based query for sensor network searching.

The time interval based query processing allows the user or the application to retrieve a part of the data stream for a sensor network by specifying a time interval. The time interval can be specified either by callable intervals (e.g., September 10, 2014) or a pair of start and end time points (e.g., from 00:00:00 AM, September 10, 2014, to 00:00:00 AM, September 11, 2014). An example of the time interval based query is expressed as follows in the RESTful API:

"/csn/search?target=network&id=8712&from=2014-09-10T18:00:00?to=2014-09-12T18:00:00."

This query requests a subset of the data stream of a sensor network (whose ID is 8712) that was collected from 6 PM, September 10, 2014, to 6 PM, September 12, 2014. The syntax of time interval based query API is shown on Table 3.

Finally, the RESTful API for the CSN Coordinator is mainly intended for administration services in the CSN runtime system. It is usually used by the CSN Admin Dashboard. The syntax for the API are presented in see Appendix B in Supplementary Material available online at http://dx.doi.org/10.1155/2015/720861).

3.2.8. Message Queue APIs. The user and the client application can receive a data stream from a sensor network by subscribing the message queue (topic). They use the Message Queue API to subscribe a message queue. How to use the API is almost the same as how to use a conventional messaging system. The API provides four methods: *EstablishConnection, RegisterCallbackMethod, SubscribeTopic, and UnsubscribeTopic*.

First, the client application uses the EstablishConnection method to set up a connection to a message queue in the CSN Data Deliverer (in fact, an open source messaging system). At that point, the client application needs information such as the URL of the CSN runtime system, the client application ID (subscriber ID), and the message queue ID (topic ID).

Second, the client application registers a message callback method for the message queue that will be used to receive data records sent from the message queue. Third, the client application subscribes the message queue. Finally, the client application unsubscribes the message queue when it does not need to receive data from the sensor any longer. Detailed descriptions of Message Queue APIs are on Table 4.

3.2.9. Data Delivery Mode. In CSN, there are three data delivery modes: *Centralized*, *Distributed with membership caching*, *and Distributed with no membership caching*. In the centralized data delivery mode, every sensor sends data records to the Data Manager and then the Data Manager forwards those data records to sensor networks according to the memberships of sensor networks. Figure 6 illustrates the centralized data delivery mode. The centralized mode is intended for small sensors with limited communication capabilities. On the other hand, the distributed modes are intended for applications with a large number of sensors. Membership caching can be used for applications with rare membership changes.

In the centralized mode, the communication between a sensor and the Data Manager is not necessarily required to use a message queue. In order to send data to the Data Manager, the sensor may use various network communication methods such as a TCP/IP socket. Those various network communication methods allow us to support a variety of sensors.

In the distributed mode, a sensor publishes data records to all the message queues of the sensor networks that the sensor belongs. In this mode, each sensor must maintain information about what sensor networks have it as their members: sensor network membership. The membership information can be cached or must be read on every sensor data publication, depending on whether we support the update to sensor network membership at runtime. Figure 7 shows the distributed data delivery mode.

Table 3: Time interval based query API for data stream searching.

Description	Searching a part of sensor data stream with searching query
Resource URL structure	/csn/search?target=data
Method	GET
Parameters	Id (optional): sensor or network ids From (optional): start datetime To (optional): end datetime Duration (optional): time interval Unit (optional): the unit of time interval Num (optional): the number to limit the searching results
Returns	A searched subset of the sensor data stream

Table 4: Message Queue API.

Number	Interface signatures
1	Public void establishConnection (String connURL, String appName, String topicPath)
2	Public void registerCallbackMethod (MessageCallback callback)
3	Public void subscribeTopic ()
4	Public void unsubscribeTopic ()

4. Implementation

4.1. Implementation Approach and Environments. We developed a prototype system for CSN. The main motivation for this prototype system is to prove that the concept and design of CSN is efficiently implementable and a viable solution for real world applications. Our implementation approach to the current CSN system is twofold: *Centralized* and *Minimal*.

CSN can be implemented in either a centralized or a distributed way. The centralized implementation that has a single server for management and data delivery is simple and requires a less amount of work. However, the distributed implementation with a set of multiple servers supports scalability more efficiently and is better suited for a large number of high frequency sampling sensors. For the ease of implementation, the current CSN system is based on the *centralized design*.

In CSN, main design features are exactly based on the Publish/Subscribe Model (often known as the JMS standard for Java). Those features are already efficiently and reliably supported by many conventional messaging systems including a number of free open source systems. For this reason, we intended to *minimize our implementation efforts for CSN* by developing CSN on top of a well-known open-source messaging system called Apache ActiveMQ [17] and using

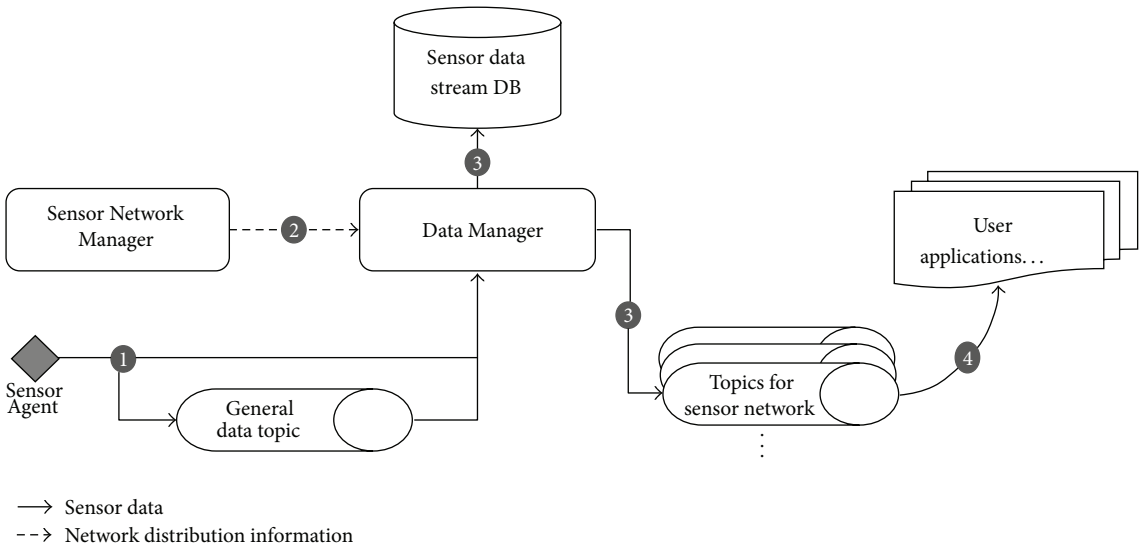


Figure 6: Centralized delivery mode.

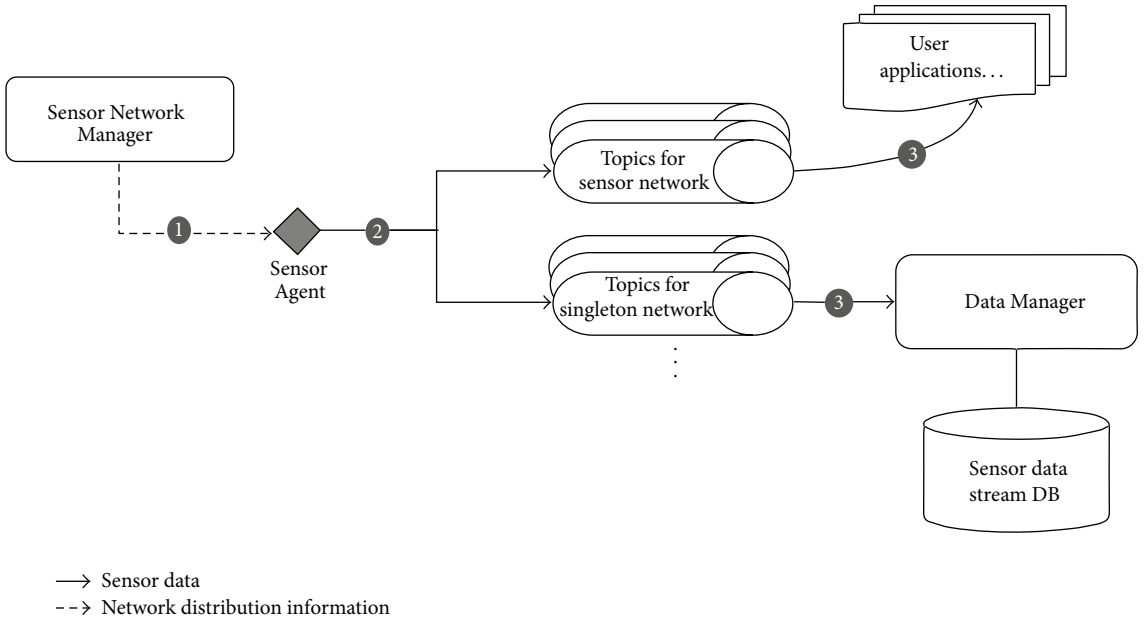


Figure 7: Distributed delivery mode.

functions already available in ActiveMQ as it is and as much as possible.

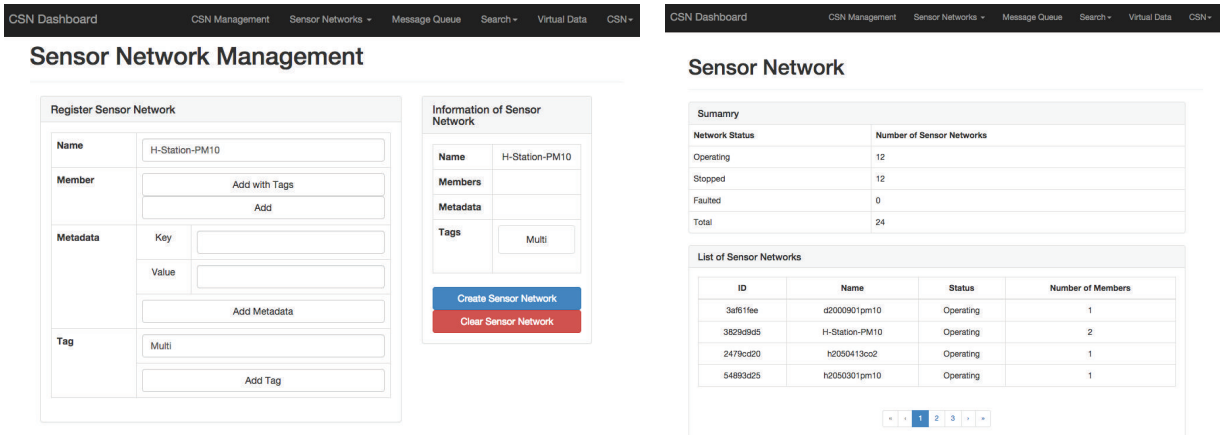
In consequence, the current CSN implementation can be considered to be *a set of extensions to a conventional messaging system*. This implementation approach has two important advantages. First, we have avoided a significant amount of unnecessary implementation work and simplified the CSN runtime system. Second, the CSN runtime system has inherited powerful functionality, performance, and reliability from the widely used open source system ActiveMQ for free.

The main implementation tools and environments are summarized in Table 5.

4.2. Admin Dashboard. The current CSN implementation provides a simple administration service that is designed as a web based dashboard. The current Admin Dashboard is not intended for sophisticated production-level administration but for simple administration aimed at system testing and evaluation. Therefore, we plan to re-implement a new version of admin tools in the future. The Admin Dashboard provides four classes of functions: (1) the CSN Runtime System Administration, (2) Sensor Network Management, (3) Semantic Tag based Sensor Network Searching, and (4) Message Queue (Topic) Management. The Admin Dashboard uses the RESTful APIs of the CSN runtime system explained in Section 3.2.7.

Table 5: Implementation environments.

Contents	Technologies
Implementation language	Java Development Kit 1.7.0
System data management	MySQL 5.1.73
Sensor data management	MongoDB 2.6.5
RESTful API	Jersey 2.5.1, Tomcat 7.0.55, Jackson JSON 2.3.3
Message queue management	ActiveMQ 5.8.0, Eclipse Paho 0.4.0 [39]
Admin dashboard	AngularJS 1.2.16, Twitter bootstrap 3.2.0
Messaging protocol	JMS, MQTT [40]



(a) Sensor network registration

(b) Sensor Network Status

Figure 8: Sensor network management windows.

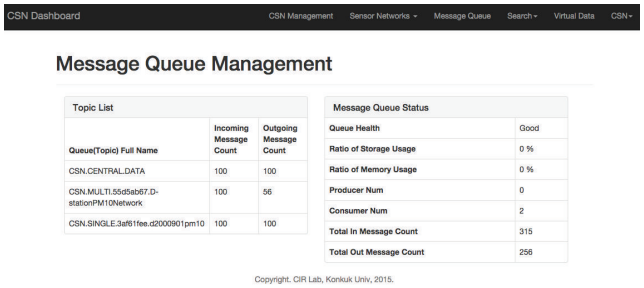


Figure 9: Message deliverer management window.

4.2.1. Sensor Network Management. The sensor network management related features are accessed in the Sensor network management page on Figure 8. This page allows the user to work with the Sensor Network Manager. Currently, the sensor network management page provides the registered sensor network list and its counts. It also supports the registration of a new sensor network (or also single sensor) and to annotate semantic tags to sensor networks for later searching like the right side of Figure 8.

4.2.2. Message Queue Management. Message Queue (and its topics) management page is available in the Admin Dashboard shown on Figure 9. It allows the user only to view the status of message queues. It displays the list of topics and the number of messages transmitted through each message queue and gives the internal system status information such as memory usage and message broker status.

4.3. Messaging Connectivity Protocol. In CSN, almost every network communication and interaction take place through message queues. There are a number of connectivity protocols for messaging systems [40–42]. Among them, the CSN runtime system supports *Message Queue Telemetry Transport (MQTT)* [40] because MQTT is a light-weight messaging protocol optimized in IoT based sensor networks. The CSN runtime system uses the Eclipse Paho library [39] that supports the client API for MQTT.

5. Experiments

5.1. Experimental Plan and Setup. We conducted two types of experiments with the current CSN runtime system: usability and performance. The usability tests were designed to evaluate common use cases of sensor networks: Sensor Registration, Sensor Network Creation, and Sensor Data Access. The performance tests were focused on the evaluation of the performance of sensor data delivery operations because the efficiency of data delivery is crucial for the overall performance of sensor networks.

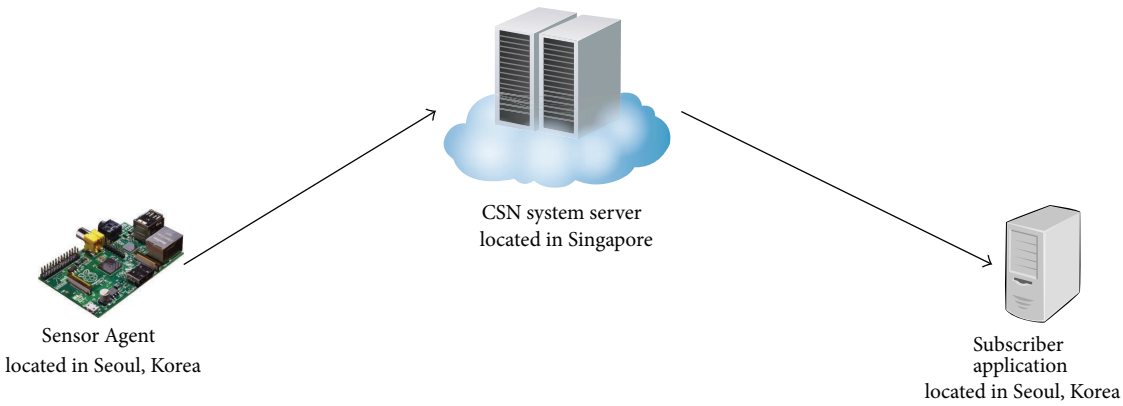


Figure 10: Experimental setup.

For our performance tests, we set up the experiment environment as shown Figure 10. In this experimental setup, a CSN Sensor Agent generates sensor data and then publishes them to a message queue in the CSN system. The Sensor Agent process runs on the Raspberry Pi [26]. Raspberry Pi is a widely used IoT hardware platform that supports various sensors or actuators.

An application subscribes sensor data from the message queue. Both the CSN Sensor Agent process and the application run on the same PC in Seoul, Korea. The CSN runtime system is deployed on a cloud server located in Singapore. Therefore, both publishing and subscribing operations require a long distance network communication operation. This setup was intended for geographically large scale monitoring applications in IoT.

5.2. Usability Test. We conducted some usability tests on the current CSN runtime system with sensor data sets from a real world application project called SubAir [19, 43]. The SubAir is a project (1) to monitor both Indoor Air Quality (IAQ) of subway platforms/tunnels and energy consumption of ventilation systems in real time manner, (2) to control ventilation system in an energy-efficient way. SubAir has an IAQ and energy consumption database collected by 194 sensors for IAQs at three subway stations and their tunnels in Line 3 of Seoul Metro (http://www.seoulmetro.co.kr/) during two years. There are various types of IAQ sensors such as PM_{10}, $PM_{2.5}$, CO_2, Temperature, Humidity, and Watt-hour meter.

For the usability tests, we took the sensor data sets for a single day (March 1, 2012) from the IAQ database and stored them into a test database. Then, we implemented Sensor Agents using CSN Message API to read sensor data from the test database, instead of real sensors. However, the CSN runtime system dealt with sensor data streams exactly in the same way as it works with real sensors, except the modified Sensor Agents.

We developed a few simple application programs intended for the energy-efficient IAQ management for subway stations. These programs were designed to perform simple statistical computations (e.g., averages) for IAQs of

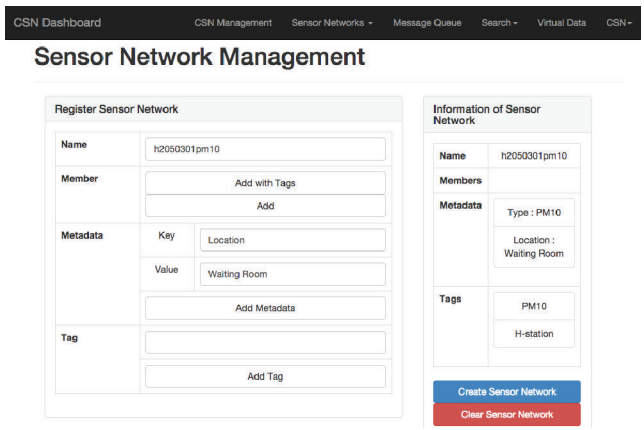


Figure 11: Sensor network registration window.

platforms or tunnels and to compare IAQs (PM_{10} and CO_2) at platforms with those at tunnels. Such computation and comparison are widely used in many environmental research projects.

First, we registered sensors by the CSN Admin Dashboard. The Admin Dashboard allows the user to register a sensor in a simple and easy way. Figure 11 shows a snapshot of the sensor network registration window. In CSN, since a sensor is considered to be a singleton sensor network, the same registration window is used for both sensors and sensor networks.

For a new sensor, the user simply enters a symbolic name ("h2050301pm10" in this example), a set of key-value pairs for metadata, and a set of semantic tags. The user enters metadata by entering a key-value pair in string and then clicking on the "Add Metadata" button repeatedly. Multiple semantic tags are also inserted in the same way. In this example, "PM10" and "H-Station" are given as semantic tags. The metadata is {"Type": "PM10"} and {"Location": "Waiting Room"}. The user can find the information entered for the sensor in the right-hand sensor network information table. The user completes the sensor registration operation by clicking on the "Create Sensor Network" button.

```
TopicSubscriber subs = new SubscriberFactory().getTopicSubscriber();
subs.setConnection("tcp://localhost:1883", "Usability-Test",
                   "CSN.MULTI.55d5ab67.H-stationPM10Network");
MessageCallback mc = new MessageCallback() {
        public void messageArrived(String data) {
                System.out.println("Subscribed:" +data);
        }
        public void deliveryComplete(IMqttDeliveryToken token) {}
        public void connectionLost(Throwable cause) {}
};
subs.setMessageCallback(mc);
subs.subscribe();
```

Algorithm 1

In order to create a new sensor network, the user enters a symbolic name and adds members (sensors or sensor networks). Sensors or sensor networks can be added as members either by the semantic tag based searching or by entering sensor network IDs directly. For example, clicking on the "Add with Tags" button creates a pop-up window for the semantic tag based search and the user can search sensor networks of interest by specifying semantic tags. For sensor networks, metadata and semantic tags are also added in the same way as for a single sensor. Those added members are shown in the sensor network information table.

Once sensors are registered and sensor networks are created, writing application code to receive data from sensors is straightforward in CSN. The code requires two types of operations: topic subscription and the callback style of asynchronous data access. In CSN, each sensor or sensor network is represented as a topic in the messaging system (i.e., ActiveMQ). An application first subscribes to the topic and then receives sensor data by a callback function asynchronously when data is available.

The code given below shows a callback method and topic subscription operations. For the sake of readability, we include only the code related to the CSN runtime system. In the code, the *messageArrived* method of the *MessageCallback* class is a callback method. When sensor data is available on the topic, the CSN runtime system invokes the *messageArrived* method asynchronously with the current sensor data as an input argument of the string type.

An application program can subscribe to a topic by using a *TopicSubscriber* class object. The object is simply created by using a factory class and configured to have a domain name (or an IP) with a port number, a session ID, and a topic ID. The callback method is registered into the *TopicSubscriber* object (see Algorithm 1).

The usability tests showed that registering sensors and creating sensor networks can be easily carried out by using the CSN Admin Dashboard. Also, receiving sensor data can be implemented as a small number of lines of code. Furthermore, the code is almost generic so that it can be used for a variety of applications with a little amount of modification.

5.3. Performance Test. Since the current implementation of the CSN runtime system is largely based on the ActiveMQ runtime system, the performance and scalability of the CSN system are significantly dependent on those of the ActiveMQ system. Therefore, we did not intend to conduct intensive performance tests with complicated experimental setups because those tests would actually evaluate the ActiveMQ system, not the implementation of the CSN runtime system.

Instead, we focused on the evaluation of the performance difference between centralized and distributed data delivery modes. The centralized delivery mode requires the CSN Data Manager to route every sensor data record, but the distributed delivery mode allows individual sensors to publish data directly to the ActiveMQ server. Therefore, we intended to evaluate the overhead of the CSN runtime system (specifically, extensions to the ActiveMQ system) by the performance difference.

First, we tested and compared the performances of the centralized and distributed data delivery modes with 10,000 sensor data records. These data records were randomly generated with ten milliseconds interval. In this test, we used only one Sensor Agent and one application client.

For the distributed data delivery mode, we furthermore tested performance for two configuration options: *caching* of the sensor network membership and *no caching*. In the caching option, the Sensor Agent communicates with the Sensor Network Manager, only once at the beginning, to get information about what message queues it should publish sensor data. Afterwards, it uses the cached information to decide where to publish sensor data. In the no caching option, the Sensor Agent communicates with the Sensor Network Manager for every sensor data record, repeatedly.

The caching configuration provides more efficient data delivery but does not support dynamic changes to the sensor network membership at runtime. On the other hand, the no caching configuration causes significantly more runtime overhead, but allows dynamic changes to the sensor network membership at runtime.

The communication and processing overheads of three delivery modes are summarized as follows.

(i) *Centralized Delivery Mode.* This mode involves four major operations for each sensor data record: one

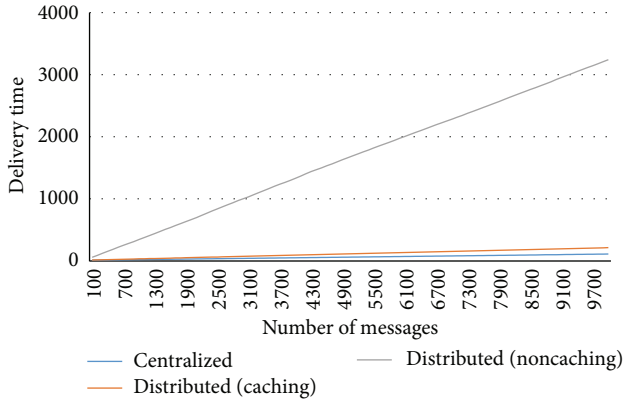


Figure 12: Performance test between three data delivery modes.

publication by the Sensor Agent, one subscription by the Data Manager, one publication by the Data Manager, and one subscription by the application client.

(ii) *Distributed Delivery Mode with Caching*. This mode involves two major operations for each sensor data record: one publication by the Sensor Agent and one subscription by the application client.

(iii) *Distributed Delivery Mode with No Caching*. This mode involves three major operations for each sensor data record: one communication between the Sensor Agent and the Sensor Network Manager, one publication by the Sensor Agent, and one subscription by the application client. In the current implementation, the communication between the Sensor Agent and the Sensor Network Manager is very costly because the Sensor Agent uses the RESTful API of the Sensor Network Manager that goes through a web server, every time.

Figure 12 shows the performance results of three data delivery modes. As expected, the distributed delivery mode with no caching showed the worst performance because of the overhead of the RESTful Web Services. But the centralized mode and the distributed delivery mode with caching showed very similar performance results. This result implies that the CSN Data Manager does not cause significant performance overheads even for high frequency sensor data.

In addition to the performance comparison of centralized and distributed data delivery modes, we also conducted the performance test of the data delivery operation with different numbers of application clients: one, five and ten, respectively. The test results are shown in Table 6. When the number of messages was small, the difference in delivery time was quite substantial among three cases, but the difference became insignificant, as the number of messages grew large. We think initialization operations for applications caused the performance difference for the test cases of small numbers of messages.

Throughout these performance tests, we concluded that the current CSN prototype system already shows high, reliable and scalable performance. The impressive performance results are mainly due to ActiveMQ that is proven to be really reliable, efficient and scalable.

Table 6: Performance results.

Number of Messages	1 application (sec)	5 applications (sec)	10 applications (sec)
50	13.199	20.862	37.163
100	15.641	23.477	39.660
1000	60.604	68.402	84.684
10000	510.807	518.589	534.762
20000	1010.913	1018.659	1034.897
30000	1511.133	1518.809	1534.995
40000	2011.411	2018.970	2035.077
50000	2511.751	2519.098	2535.190
60000	3011.860	3019.196	3035.287
70000	3512.017	3519.401	3535.547
80000	4012.244	4019.539	4035.790
90000	4512.464	4519.767	4536.214
100000	5012.626	5019.932	5036.830

6. Related Works

There has been a great deal of research effort on sensor networks [1, 3]. Most research work has centered on wireless sensor networks (WSN) and mainly focused on system software for a sensor node, network management, communication protocols and power management. Since sensors in traditional WSNs are assumed to have low computing power and battery lifetime, their research projects on WSN are usually aimed at the customization and optimization for characteristics of specific application domains in order to minimize computing overhead and energy consumption.

In addition, there have been a number of software development projects on sensor network OSs and middleware including TinyOS, Mate, Magnet, Impala, and Milan [4, 5, 13, 44–49]. Although these projects aim at middleware software for WSNs, they do not address issues in the conceptual management and integration of a large number of sensor networks.

As compared to these traditional WSN research projects, the CSN system is designed for the management and integration of sensor networks and independent of application-specific or sensor-specific characteristics. CSN allows the user or the application to manage sensor networks in a conceptual way. CSN provides well-defined, simple, intuitive and generic APIs for various applications that is based on both the messaging model [16] and the REST model [38].

Because of runtime overheads and requirements for reliable networks, the CSN system is not suitable to run directly on small hardware sensors and wireless networks in WSN, but traditional WSN systems can be integrated into the CSN system as underlying sensor networks. For this kind of integration, the CSN Sensor Agent needs to be customized to work with gateway nodes of WSN systems. In such integration, the CSN system and traditional WSN

systems are at different layers where the CSN system runs on top of traditional WSN systems and applications access sensors of those WSN systems through the CSN system.

There are some sensor network middleware based on a logical model. Such middleware is focused on facilitating the development of applications and usually designed to support a variety of applications in more general and standard way. In such approach, major projects include Global Sensor Network (GSN) [12], Mires [13], and Data Turbine [50, 51]. These projects have motivations, goals and design features similar to those of the CSN project.

GSN models sensors and sensor networks as virtual sensors that have multiple input data streams and one output data stream. In GSN, virtual sensors are hierarchically pipelined and organized into a tree or a graph structure. GSN supports efficient, simple, and intuitive data delivery from sensors to applications. In contrast, CSN models a sensor network as a set of sensor networks or sensors. Therefore, a new sensor network can be theoretically created by set operations such as union and intersection. The set-based sensor network model is more logical than virtual sensors and enables more sophisticated management of sensor networks.

Both Mires and Data Turbine explicitly support the messaging model (Publish/Subscribe Model) for sensor data delivery as in CSN. In both systems, sensors publish data to message queues where applications subscribe data. However, they do not support the explicit management of sensor networks although sensor networks can be implemented with message queues at the application level.

GSN, Mires and Data Turbine do not allow the user to manage sensors or sensor networks semantically. In CSN, sensor networks can be associated and searched with semantic tags. This semantic tag based management can facilitate the management of a large number of sensors and sensor networks significantly.

With respect to the system implementation, GSN, Mires, and Data Turbine developed their own data delivery systems, but CSN uses the open source messaging system called ActiveMQ as its data delivery system. We believe this implementation approach can minimize our development efforts significantly and take advantage of system upgrades and further developments in the open source development project.

Sensor OGC's Web Enablement (SWE) is an international research and development effort to develop data standards, open interfaces, and reference implementations for a variety of sensor related applications [14]. Although the SWE effort is comprehensive and addresses crucial technical issues in managing sensors and their data, SWE is a very heavyweight approach for many science and engineering applications. It requires application developers to understand a great deal of amount of technical knowledge and system administrators to manage a substantial scale of system infrastructure. On the other hand, the CSN system is a very lightweight approach that does not require much technical knowledge or complicated system infrastructure.

There are very active on-going research efforts to apply semantics technologies to sensor networks [52]. Semantic Sensor Web is an effort to extend Sensor Web with semantic technologies [52]. Ontologies for sensors and their data are actively being explored [53–56]. Although these current efforts are attempting to address important semantic issues in sensor management and are promising in the future, they do not seem mature enough to be used for real world applications for now.

In contrast, CSN supports a simple semantic annotation scheme with tags and their logical expressions. Although the scheme cannot handle complicated semantic management requirements, we believe it is efficient, intuitive and furthermore sufficient for many applications.

7. Conclusion and Future Work

Emerging technologies and applications such as pervasive computing, cyber-physical systems, and Internet of Things [57–60] require the effective management and integration of numerous heterogeneous sensors and sensor networks in a scalable, reliable and consistent way [14, 59]. However, most traditional research projects on sensor networks have been focused on sensor node-level system issues such as lightweight operating systems for sensor nodes, communication protocols, network management, and the optimization of energy consumption [1, 3]. Therefore, they fail to effectively address those high level system issues raised in emerging technologies and applications.

In this paper, we presented a sensor network system called the Conceptually Manageable Sensor Network (CSN). In the CSN project, we intended to address sensor network issues raised in those emerging applications such as Internet of Things. The main objective of the CSN system is to enable a variety of applications to manage, to integrate and to access a large number of heterogeneous sensor networks in a simple, intuitive and consistent way, regardless of application characteristics. In addition, we intended CSN to facilitate the development of applications by providing simple, intuitive, and well-defined APIs.

In CSN, sensor networks and the application access to them are explicitly managed by using message queues in a uniform and consistent way. Many core functions to be required by the CSN runtime system are already available in conventional messaging systems. Therefore, we implemented the CSN runtime system to be *a set of extensions* to the open source messaging system called ActiveMQ [61]. However, the CSN system implementation is still independent of ActiveMQ because the CSN system design is based on the standard messaging model and JMS specification.

This implementation approach has a few important advantages. First, we were able to reduce the system implementation work, substantially. Second, the CSN runtime system largely inherits the system attributes of ActiveMQ such as performance, reliability, and scalability. Therefore, we believe that the current CSN runtime system is a prototype implementation but can be easily improved to provide production-level services. Finally, the CSN system can also easily benefit from future system upgrades and improvements in ActiveMQ.

We conducted some usability and performance tests with the current CSN runtime system. In the usability test,

we found that the creation of sensor networks logically and the message queue-based access to sensor data streams were simple and intuitive to implement and to run. In the performance test, we found the CSN runtime system was able to handle a large number of sensor data records generated in a very high frequency (i.e., 100 milliseconds), efficiently and reliably without any system optimization.

The CSN project is at an early stage and therefore needs lots of future work. First, the CSN Sensor Agent will be re-implemented. Our future work will be focused on the Sensor-specific Sensor Library. The library consists of the Uniform Access Interface and a set of actual code for specific hardware sensors that implements the interface. Currently, the Uniform Access Interface includes only a few simple operations and there is little support for real hardware sensors. Second, we plan to add security supports to the CSN runtime system. The current CSN implementation has little support for security. Third, we are currently evaluating the effectiveness and practicality for sensor ontologies [62] to extend the current simple semantic tags in CSN. Finally, we plan to test the CSN runtime system for real world field monitoring in the project WISE [18].

Disclosure

This paper was also written as part of Konkuk University's research support program for its faculty on sabbatical leave from 2013 to 2014.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] Gartner Report, 2014, http://www.gartner.com/newsroom/id/2636073.

[3] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[4] S. Hadim and N. Mohamed, "Middleware for wireless sensor networks: a survey," in *Proceedings of the 1st International Conference on Communication System Software and Middleware (Comsware '06)*, pp. 1–7, January 2006.

[5] P. Levis, S. Madden, J. Polastre et al., "TinyOS: an operating system for sensor networks," in *Ambient Intelligence*, pp. 115–148, Springer, Berlin, Germany, 2005.

[6] P. Rawat, K. D. Singh, H. Chaouchi, and J. M. Bonnin, "Wireless sensor networks: a survey on recent developments and potential synergies," *Journal of Supercomputing*, pp. 1–48, 2013.

[7] H. Lee, M. Jang, and J. Chang, "A new energy-efficient cluster-based routing protocol using a representative path in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2014, Article ID 527928, 12 pages, 2014.

[8] Y.-G. Ha, H. Kim, and Y.-C. Byun, "Energy-efficient fire monitoring over cluster-based wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2012, Article ID 460754, 11 pages, 2012.

[9] C. Li and Y. Liu, "ESMART: energy-efficient slice-mix-aggregate for wireless sensor network," *International Journal of Distributed Sensor Networks*, vol. 2013, Article ID 134509, 9 pages, 2013.

[10] K. Martinez, J. K. Hart, and R. Ong, "Environmental sensor networks," *Computer*, vol. 37, no. 8, pp. 50–56, 2004.

[11] L. Selavo, A. Wood, Q. Cao et al., "LUSTER: wireless sensor network for environmental research," in *Proceedings of the 5th ACM International Conference on Embedded Networked Sensor Systems (SenSys'07)*, pp. 103–116, November 2007.

[12] 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 (VLDB Endowment)*, pp. 1199–1201, 2006.

[13] E. Souto, G. Guimarães, G. Vasconcelos et al., "Mires: a publish/subscribe middleware for sensor networks," *Personal and Ubiquitous Computing*, vol. 10, no. 1, pp. 37–44, 2006.

[14] M. Botts, G. Percivall, C. Reed, and J. Davidson, "OGC sensor web enablement: overview and high level architecture," in *GeoSensor Networks*, pp. 175–190, Springer, Berlin, Germany, 2008.

[15] S. Tai, A. Totok, T. Mikalsen, and I. Rouvellou, "Message queuing patterns for middleware-mediated transactions," in *Software Engineering and Middleware*, vol. 2596 of *Lecture Notes in Computer Science*, pp. 174–186, Springer, Berlin, Germany, 2003.

[16] JMS Specification, https://java.net/projects/jms-spec/pages/Home.

[17] Apache ActiveMQ, 2014, http://activemq.apache.org/.

[18] The WISE Project, 2014, http://wise2020.org/eng/.

[19] I. C. Jeong, G. H. Li, N. G. Kim, and S. B. Lim, "An integrated monitoring and controlling system for subway system," in *Proceedings of the 2nd International Conference on Engineering and Industries (ICEI '11)*, pp. 1–3, Jeju-do, Republic of Korea, December 2011.

[20] M. Trakimas, S. Hwang, and S. Sonkusale, "Low power asynchronous data acquisition front end for wireless body sensor area network," in *Proceedings of the 24th International Conference on VLSI Design (VLSI Design '11)*, pp. 244–249, January 2011.

[21] W. Joe, M. Jiang, and K. Jeong, "Design and implementation of a M2M/IoT based programmable data logger for automatic environmental and ecological observation in real-time way," *Journal of KIISE: Computing Practices and Letters*, vol. 40, no. 1, pp. 11–13, 2013.

[22] Y. Sokha, K. Jeong, J. Lee, and W. Joe, "A complex event processing system approach to real time road traffic event detection," *Journal of Convergence Information Technology*, vol. 8, no. 15, pp. 142–148, 2013.

[23] Qualcomm IoE Development Platform, https://developer.qualcomm.com/mobile-development/development-devices/internet-everything-ioe-development-platform.

[24] Intel Galileo, http://www.intel.com/content/www/us/en/do-it-yourself/galileo-maker-quark-board.html.

[25] Arduino, http://www.arduino.cc/.

[26] Raspberry Pi, http://www.raspberrypi.org/.

[27] E. Curry, "Message-oriented middleware," Middleware for communications, 2004.

[28] IBM MQ, http://www-03.ibm.com/software/products/ko/ibm-mq.

[29] A. Videla and J. J. W. Williams, *RabbitMQ in Action*, Manning, 2012.

[30] H. Pieter, *ZeroMQ: Messaging for Many Applications*, O'Reilly Media, 2013.

[31] L. Zhai, C. Li, and L. Sun, "Research on the message-oriented middleware for wireless sensor networks," *Journal of Computers*, vol. 6, no. 5, pp. 1040–1046, 2011.

[32] MongoDB, http://www.mongodb.org/.

[33] D. J. Abadi, D. Carney, U. Çetintemel et al., "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, vol. 12, no. 2, pp. 120–139, 2003.

[34] J. Chen, D. J. DeWitt, F. Tian, and Y. Wan, "NiagaraCQ: a scalable continuous query system for internet databases," *ACM SIGMOD Record*, vol. 29, no. 2, pp. 379–390, 2000.

[35] T. Grabs, R. Schindlauer, R. Krishnan, J. Goldstein, and R. Fernández, "Introducing microsoft stream insight," Tech. Rep., 2009.

[36] A. Biem, E. Bouillet, H. Feng et al., "IBM infosphere streams for scalable, real-time, intelligent transportation services," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*, pp. 1093–1104, June 2010.

[37] JSON, 2014, http://json.org/.

[38] R. T. Fielding and R. N. Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, 2002.

[39] Eclipse Paho, 2014, http://www.eclipse.org/paho/.

[40] MQTT, 2014, http://mqtt.org/.

[41] STOM, 2014, http://stomp.github.io/.

[42] AMQP, 2014, http://www.amqp.org/.

[43] Y.-S. Kim, I.-W. Kim, J.-C. Kim, and C. Yoo, "Real-time multivariate monitoring and diagnosis of air pollutants in a subway station," in *Proceedings of the International Conference on Control, Automation and Systems (ICCAS '08)*, pp. 2610–2615, October 2008.

[44] K. Henricksen and R. Robinson, "A survey of middleware for sensor networks: state-of-the-art and future directions," in *Proceedings of the International Workshop on Middleware for Sensor Networks (MidSens '06)*, pp. 60–65, November 2006.

[45] M. M. Molla and S. I. Ahamed, "A survey of middleware for sensor network and challenges," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPP '06)*, pp. 223–228, August 2006.

[46] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 85–95, 2002.

[47] R. Barr, J. C. Bicket, D. S. Dantas et al., "On the need for system-level support for ad hoc and sensor networks," *SIGOPS Operating Systems Review*, vol. 36, no. 2, pp. 1–5, 2002.

[48] T. Liu and M. Martonosi, "Impala: a middleware system for managing autonomic, parallel sensor systems," *ACM SIGPLAN Notices*, vol. 38, no. 10, pp. 107–118, 2003.

[49] W. B. Heinzelman, A. L. Murphy, H. S. Carvalho, and M. A. Perillo, "Middleware to support sensor network applications," *IEEE Network*, vol. 18, no. 1, pp. 6–14, 2004.

[50] T. Fountain, S. Tilak, P. Hubbard, and P. Shin, "The open source DataTurbine initiative: streaming data middleware for environmental observing systems," in *Proceedings of the International Symposium on Remote Sensing of Environment*, 2009.

[51] S. Tilak, P. Hubbard, M. Miller, and T. Fountain, "The ring buffer network bus (RBNB) dataturbine streaming data middleware for environmental observing systems," in *Proceedings of the 3rd IEEE International Conference on E-Science and Grid Computing (E-Science '07)*, pp. 125–133, Bengaluru, India, December 2007.

[52] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic sensor web," *IEEE Internet Computing*, vol. 12, no. 4, pp. 78–83, 2008.

[53] K. Taylor, A. Ayyagari, and D. De Roure, "Research topics in semantic sensor networks," in *Proceedings of the Semantic Sensor Network Workshop*, 2006.

[54] M. Compton, C. Henson, L. Lefort, H. Neuhaus, and A. Sheth, "A survey of the semantic specification of sensors," in *Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN '09)*, pp. 17–32, October 2009.

[55] V. Huang and M. K. Javed, "Semantic sensor information description and processing," in *Proceedings of the 2nd International Conference on Sensor Technologies and Applications (SENSORCOMM'08)*, pp. 456–461, Cap Esterel, France, August 2008.

[56] L. M. Ni, Y. Zhu, J. Ma et al., "Semantic sensor net: an extensible framework," in *Networking and Mobile Computing*, vol. 3619 of *Lecture Notes in Computer Science*, pp. 1144–1153, Springer, Berlin, Germany, 2005.

[57] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference (DAC '10)*, pp. 731–736, June 2010.

[58] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," *Computer*, vol. 36, no. 3, pp. 25–31, 2003.

[59] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

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

[61] B. Snyder, D. Bosnanac, and R. Davies, *ActiveMQ in Action*, Manning, 2011.

[62] M. Compton, P. Barnaghi, L. Bermudez et al., "The SSN ontology of the W3C semantic sensor network incubator group," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 17, pp. 25–32, 2012.