Design and Evaluation of Mission-Oriented Sensing Platform with Military Analogy

Inomoto, Hikaru
Saiki, Sachio
Nakamura, Masahide
Matsumoto, Shinsuke

(Citation)
International Journal of Pervasive Computing and Communications,13(1):76-91

(Issue Date)
2017

(Resource Type)
journal article

(Version)
Accepted Manuscript

(URL)
https://hdl.handle.net/20.500.14094/90005925

Design and Evaluation of Mission-Oriented Sensing Platform with Military Analogy

Hikaru Inomoto, Sachio Saiki, Masahide Nakamura
Graduate School of System Informatics, Kobe University, Japan

Abstract

Purpose

The purpose of this paper is to perform large-scale environmental sensing with a lot of IoT devices, as typically seen in Smart City, efficiently and for multiple applications. In this paper, we propose a novel sensing method, called mission oriented sensing, which accepts multiple and dynamic sensing purposes on a single infrastructure.

Design/methodology/approach

The proposed method achieves the purpose by dealing sensing configuration (application's purpose) as mission. It realizes sharing single infrastructure by accepting multiple missions in parallel, and it accepts missions' update anytime. In addition, the sensing platform based on military analogy can command and control a lot of IoT devices in good order, and this realizes mission oriented sensing above.

Findings

Introducing mission oriented sensing, multiple purpose large-scale sensing can be conducted efficiently. The experimental evaluation with a prototype platform shows the practical feasibility. In addition, the result shows that it is effective to update sensing configuration dynamically.

Research limitations/implications

The proposed method focuses aggregating environmental sensor value from a lot of devices, and thus it can treat stream data such as video or audio or control a specific device directly.

Originality/value

In proposed method, a single sensing infrastructure can be used by multiple applications, and it admits heterogeneous devices in a single infrastructure. In addition, the proposed method has less technical restriction and developers can implement actual platform with technologies for context.

Keywords

Mission-oriented sensing, Large-scale environment sensing, Military Analogy, Context-aware sensing, IoT

1 Introduction

Large-scale environment sensing using IoT devices is attracting a lot of attention, due to the rapid progress of IoT (Internet of Things)(Miorandi et al., 2012)(Atzori et al., 2010) and Cloud Computing. The large-scale environment sensing is promising for various applications. For example, *Smart City* (Hollands, 2008) collects environmental sensor data from whole city and provides value-added services to residents using the data. *Smart Agriculture* (Lazarescu, 2013) measures states of cultivation and crops in broad farms. *Smart Mobility* (Yu et al., 2012) monitors various data from road traffic networks.

To achieve such the large-scale environment sensing, this paper especially focuses on the following three requirements:

Requirement R1 (Shared sensing infrastructure)
To perform the large-scale sensing, the administrator has to deploy a huge number of sensing devices as the sensing infrastructure, and maintains them properly. From the viewpoint of cost and efficiency, the sensing infrastructure must be *shared among multiple applications*, instead of being dedicated to a single application.

Requirement R2 (Dynamic sensing configuration)
For some applications, it is effective to *update the sensing configuration dynamically*, depending on a situation such as day of week, time of day, sensing locations. For example, the road traffic monitoring gathers high-density data in commuter rush hours, while it steps down the sampling rate in light-traffic hours. This dynamic configuration can reduce the data volume without declining the quality of information [1].

Requirement R3 (Admitting heterogeneous devices) The sensing infrastructure should *accommodate heterogeneous devices*, in order for multiple applications to use the shared infrastructure for various purposes. In general, each application has own interesting environmental attributes (e.g., temperature, humidity, brightness, sound level, vibration, human presence). Even for the same attributes, sensors with different performance may be required for different purposes.

The latest IoT devices and Cloud services can be used as a means to implement large-scale environment sensing. However, there is no systematic methodology considering the above requirements R1, R2, R3 together, as far as we know. To fulfill the requirements, we need a clever method that can *command and control many heterogeneous devices in good order*, for different purposes and situations.

To cope with the requirements, we present a novel platform of large-scale environment sensing by borrowing an *analogy of military system*. Typically, a military consists of many soldiers controlled by military system. A military has a *mission*, and each soldier performs concrete actions according to the mission. The mission can be updated dynamically, and soldier's actions are changed accordingly by the mission update. In the proposed method, we regard individual devices in the large-scale sensing infrastructure as soldiers. Also, we consider the sensing tasks given by various applications as missions. Then, we propose *mission-oriented sensing platform*, which commands and controls many devices in good order for dynamically-updated missions.

For Requirement R1, we allow every application to request *missions* to the sensing platform. A mission involves sensing configuration information to be specified based on the purpose of the application. Upon receiving a mission, the sensing platform collects requested data from

[1] The sensing method that changes behaviors depending on a situation is generally called *context-aware sensing*(Perera et al., 2015).

appropriate sensors, and stores the data on a designated database. For requirement R2, we design the platform so as to accept dynamic update of running missions. Based on structured chain of commands, the platform delivers the mission update to proper devices, and forms a new sensing configuration to the devices. For requirement R3, we abstract the heterogeneous sensing devices as uniform soldiers with different equipment. The platform delegates all the device-proprietary operations to the soldier.

To implement the above mechanisms, we construct a *military hierarchy* within the sensing platform, consisting of three ranks: *(Soldier)* a sensing device measuring sensor values, *(Leader)* an edge device controlling a group of soldiers within a division of the target area, and *(Commander)* a regional server managing commands and controls for all the devices. Using the mission-oriented sensing based on this hierarchy, we aim to implement large-scale environment sensing.

In this paper, we especially focus on the concept design of the mission-oriented sensing, and discuss an implementation method based on the military analogy. We also develop a prototype system, and conduct an experimental evaluation in a real environment using the prototype.

2 Preliminaries

2.1 Large-Scale Environment Sensing

We use a term *environment sensing* to refer to any activities that measure and collect environmental data with one or more environmental sensors. Typical environmental data include temperature, humidity, brightness, sound volume, vibration, gas pressure, and human presence. Thanks to the latest IoT technology, the measured sensor data can be delivered via the internet. The proper combination of IoT and Cloud services enables large-scale and broad-area environment sensing with reasonable cost.

The *large-scale environment sensing* we focus on in this paper refers to an environment sensing in a broad (indoor or outdoor) area, in which a large number of sensors measure the area collaboratively, send the data via network, and store the data in a designated database (in a cloud, for instance). The large-scale environment sensing is characterized by the vast sensing area, a large number and wide variety of sensors, and the large volume and density of measured data. The infrastructure for such large-scale environment sensing is especially promising for cyber-physical smart systems, such as smart city and smart agriculture.

The behavior of the large-scale environment sensing is determined by *sensing configuration*, which specifies how the sensing is conducted. The sensing configuration should describe from where the platform collects data, what the target environment attributes are, to which database the collected data are stored, and other parameters. The sensing configuration usually varies from one application to another. So we assume that the configuration is derived from the purpose of individual application.

2.2 Challenges

Challenges of the large-scale environment sensing lie in its scale.

The first challenge is due to the cost of sensing infrastructure. The large-scale environment sensing requires the provision, installation, operation and maintenance of a large number of sensing devices within a vast area, which yields huge cost and effort. Because of that, it is unrealistic that each application has its own infrastructure. Considering increasing demand of cyber-physical smart systems, the sensing infrastructure must be shared among multiple applications. This justifies Requirement R1 in Section 1.

The second challenge is due to resource requirements. The large-scale sensing can generate a large volume of data quite easily. Handling these big data requires a large amount of resources such as database storage, network infrastructure, and computing resource. To achieve efficient use of the resources, it is preferable to change the sensing configuration dynamically depending on situation. Similar to the road monitoring example in Section 1, an advertising application in a shopping mall may want dense sensing in a certain floor where a special sale event is taking place. Also, the shopping mall may want to stop sensing after closing to save disk usage. This justifies Requirement R2 in Section 1.

The third challenge is from the variety of sensing devices in the infrastructure. Preferably, the large-scale environment sensing measures a wide variety of environment data for various applications. So the infrastructure must accommodate heterogeneous sensing devices. For this, it is unrealistic to force every application developer to manage all device-dependent configuration. This severely declines the usability of the infrastructure. Also, the portability of the configuration across different areas is not guaranteed. Therefore, the sensing platform should accommodate heterogeneous devices with isolating device-dependent operations and application-defined sensing configuration. This makes application developers free from proprietary device knowledge. This justifies to Requirement R3 in Section 1.

2.3 Scope of Paper

To clarify the scope of this paper, we put the following assumptions in the large-scale environment sensing dealt with the proposed method.

- The target sensing area satisfies environmental conditions under which all devices work correctly.
- All the devices for the environment sensing are installed at fixed locations, and they don't move.
- Stable power and network connectivity are supplied to every device.

It can be seen from these assumptions that our target is large-scale sensing with relatively mild environment constraints. We do not assume severe environment with, for instance, gale wind, heavy rain, or ultra-hot gas, under which the devices may be broken. Also, we do not assume mobile sensors with limited power and unstable network.

Under the above assumptions, we try to propose a method that implements large-scale environment sensing, fulfilling Requirement R1-R3 in Section 1.

3 Mission-Oriented Large-Scale Environment Sensing

In order to address the challenges in the previous section, we propose a novel platform of large-scale environment sensing, called *mission-oriented environment sensing*.

3.1 Key Idea

The mission-oriented environment sensing is a method of IoT sensing, which defines every sensing configuration as a *mission*. Intuitively, a mission is a requirement of an application, which characterizes the sensing configuration. A mission includes environment attributes to be collected, locations where the data is measured, a sampling rate, an address of database to which the data is stored, and so on.

Table 1: An instance of requested mission

Parameter	Value
requirement	[barometer, brightness, humidity, temperature]
place	1F corridor, Building of system informatics
trigger	{interval:10}
destination	mongodb://xxx.xx.xxx.xx/sensing
supervisor	KobeUniv-CSBuilding-Platform01
purpose	{app: TestApp, mode: env-monitoring-default}

Each application creates a mission based on its own purpose, and then requests to the sensing platform. The platform interprets the mission, and tells concrete instructions to relevant sensing devices. Based on the instruction, each device measures specified data. The data is finally stored in a designated database. A mission can be dynamically added, updated or deleted. Also, the platform can accept multiple mission simultaneously.

Since the proposed platform can accept multiple missions in parallel, it can be shared by multiple applications, which addresses Requirement R1. A mission can be updated dynamically depending on a context, which addresses Requirement R2. A mission is device-independent description to be interpreted by the platform, which addresses Requirement R3.

3.2 Mission

A mission must contain all necessary information for the platform to execute the large-scale environment sensing. Through an interrogative analysis (i.e., WHAT, WHERE, WHEN, HOW, WHO, WHY), we derive the following six parameters to be involved in a mission.

(WHAT) `requirement:` This parameter specifies *what* data should be collected in the mission. It is defined by a set of environment attributes, such as temperature, brightness, humidity, motion, gas pressure, etc.

(WHERE) `place:` This parameter specifies *where* the data should be measured in the mission. It is defined by a set of identifiers of places within the target data.

(WHEN) `trigger:` This parameter specifies *when* the data should be measured in the mission. It is defined by a sampling rate (e.g. every 10 sec.) or a sampling condition (e.g. record when the value is greater than 28).

(HOW) `destination:` This parameter specifies *how* the data should be stored in the mission. It is defined by an address of a database.

(WHO) `supervisor:` This parameter specifies *who* is responsible for the mission. It is defined by an identifier of a region server that manages all sensors within the area.

(WHY) `purpose:` This parameter specifies *why* the mission is requested. It is defined by an application ID and its execution mode. These are used for distinguishing the corresponding mission from multiple missions.

We assume, in the above parameters, that `requirement`, `place`, `trigger`, `description` are given by an application that uses the proposed sensing platform. On the other hand, `supervisor` and `purpose` are assigned by the platform when the mission is accepted.

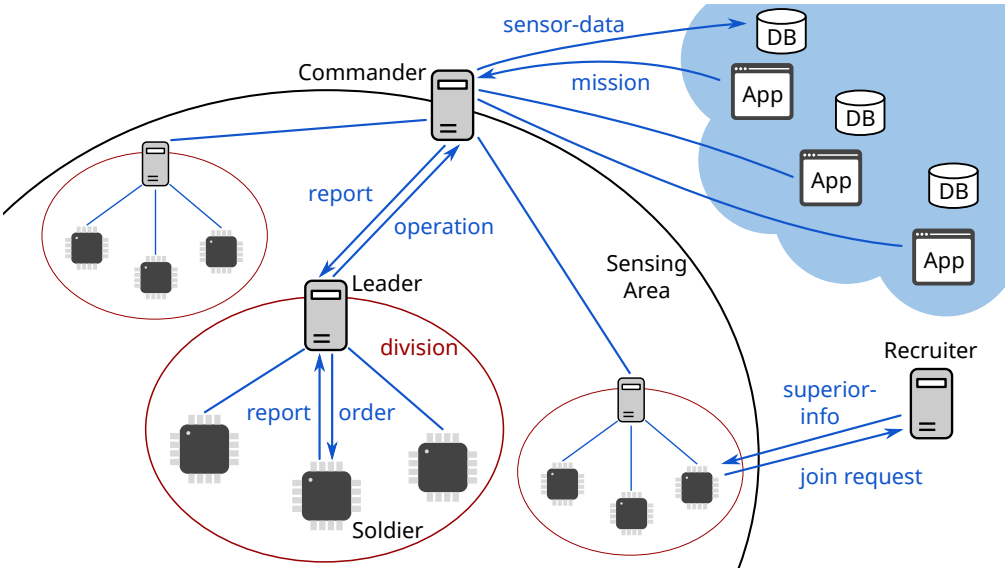

Figure 1: Architecture of proposed sensing platform

Table 1 shows an instance of a mission. This mission supposes a situation that an application (TestApp) measures environment data in a corridor of the first floor, building of system informatics of Kobe university. The mission requires collecting barometer, brightness, humidity, temperature with the interval of 10 seconds, and requests to store the data to sensing database of MongoDB.

3.3 Introducing Military Analogy

To execute multiple and dynamic missions consistently with the large number of sensing devices, it is essential to command and control these many sensors in good order. To achieve this, we introduce an *analogy of military system*. Intuitively, all devices in the infrastructure are regarded as *soldiers*, which collaboratively work to accomplish given missions. From various concepts in the military field, we particularly focus on the following three concepts:

3.3.1 Hierarchy for Divide and Conquer

A military generally adopts a hierarchical system to manage many soldiers orderly in a divide and conquer manner. We make full use of this *hierarchy* to construct an architecture of the proposed mission-oriented sensing, which is as shown in Fig. 1. In the proposed architecture, we classify devices in the infrastructure into the following three classes:

Soldier: A *soldier* is a sensor node that actually measures environment values in the bottom of the hierarchy. According to an *order* given by his superior (Leader, see below), a soldier measures data and sends the data to the superior as a *report*. Normally, a soldier corresponds to an IoT device having one or more environment sensors. In this case, the sensors are regarded as equipment of the soldier.

Leader: A *leader* is an edge device (or edge server) that manages a group of soldiers in a certain *division* within the target area. A leader interprets an *operation* given by his superior

(Commander, see below), and sends concrete orders to his subordinates (his soldiers). A leader also receives and summarizes reports from his soldiers, and sends the summary reports to his commander. We assume that a leader captains several to a dozen of soldiers within the same division (e.g., one room, one floor, etc.). Hence, each division of the target area is governed by a leader and his soldiers. As a result, `place` in the mission can be mapped into leaders that govern relevant divisions.

Commander: A *commander* is a regional server that manages all the leaders in the target area, and works as a mediator between the sensing platform and applications. A commander interprets a *mission* requested by an application, creates operations for the mission, and sends them to relevant leaders. In addition, a commander gathers reports from leaders, and sends them to the designated database.

Besides these military men who are directly involved with sensing, there is another role that *maintains* the hierarchy.

Recruiter: A *recruiter* is a registry server that maintains configuration information of the military hierarchy. The recruiter supports a freshman (i.e., a new device) to join the hierarchy, by providing superior's contact information. It also keeps track of which and when a device joins (or leaves) the hierarchy.

3.3.2 Stepwise Refinement of Mission

In our design thought, a mission is defined by the six device-independent parameters (See Section 3.2). So the applications do not have to understand a variety of device-proprietary operations and configurations. In fact, however, every device needs concrete device operation and configuration. To fill the gap, the proposed platform lets the commander, the leader, and the soldiers transform a mission into a more concrete one through *stepwise refinement*.

More specifically, a mission can have the following three levels of abstraction, as appeared in Section 3.3.1.

Mission: As defined in Section 3.2, a *mission* is device-independent sensing requirement, defined by individual applications using the six parameters. A mission is at the highest abstraction level.

Operation: An *operation* is a set of tasks to be performed by a group of soldiers in a division under a leader for accomplishing the mission. Based on a given mission, a commander generates a set of operations, each of which is sent to a leader.

Order: An *order* is a concrete instruction given to a soldier for accomplishing the operation. Based on a requested operation, a leader generates a set of orders, each of which is delivered to a soldier.

3.3.3 Communication with Order–Report Protocol

As in a military, every communication in the proposed platform must be performed based on a pair of order and report. That is, every device receives an order from his superior, then works for the order, and finally reports the result to the superior. In this order–report protocol, the device cannot work for anything else, or cannot communicate with irrelevant devices.

The proposed platform executes multiple missions in parallel. Hence, a *mission identifier* is assigned to every order and report. The mission identifier is generated from the `purpose` parameter of a mission. It is used to identify which mission a given order (or report) relies on.

3.4 Workflow of Environment Sensing

We here illustrate the workflow of the proposed large-scale environment sensing. The workflow consists of three phases: joining to hierarchy, notifying missions, aggregating values.

(Phase 1) Joining to Hierarchy

When a new device is deployed in a sensing area, it asks a recruiter who the superior of the new device is. The recruiter knows the device structure of the whole hierarchy in advance. So, the recruiter responds the new device's superior that is derived from the structure information. Upon receiving the response, the new device sends a joining message to the superior. Finally, the superior accepts the new device as a subordinate, which completes the process of joining to hierarchy.

(Phase 2) Notifying Missions

An application creates (or updates) a mission based on the application's purpose, and submits the mission to a commander. Next, on receiving a mission, the commander finds leaders who are responsible for the area specified in `place` of the mission. For each leader found, the commander creates an operation and sends it to the leader. The operation includes `requirement`, `trigger`, and `purpose`. They are refined from the mission, so that the leader can understand necessary sensor attributes, measurement schedule, and a mission identifier, respectively. Then, on receiving an operation, a leader finds soldiers in his subordinates that have appropriate sensors specified in `requirement`. For each soldier found, the leader creates an order and sends it to the soldier. The order includes `requirement`, `trigger`, and `purpose`. They are refined from the operation, so that the soldier can perform appropriate environment sensing.

(Phase 3) Aggregating Values

On receiving an order, a soldier initiates environment sensing according to the order. The soldier measures environment attributes described in `requirement` with a specified timing in `trigger`. The soldier sends the values as a report to his superior (the leader) with a mission ID in `purpose`, a soldier ID, and time stamp. Next, the leader receives reports from his subordinates, and stores the reports on hand with location information. At regular time intervals. The leader sends the stored reports to his superior (the commander). Then, the commander receives the data from the leader. Finally, the commander sends the data to a database specified in `destination`.

A whole troop is constructed by repeating the *joining to hierarchy* while there is a military man who doesn't join, and a new mission or changing mission is notified on the *notifying missions* flow. Then, sensor values are collected by executing the *aggregating values* according to accepted missions.

4 Implementation

To evaluate the proposed method, we implemented a prototype of the platform.

All the software program of the soldier, the leader, and the commander are written in the Python language. Each program was deployed as a Web service, to which external program can access by JSON/HTTP protocol. The communication was basically from a subordinate to a superior, where an order is *pulled* and a report is *pushed*.

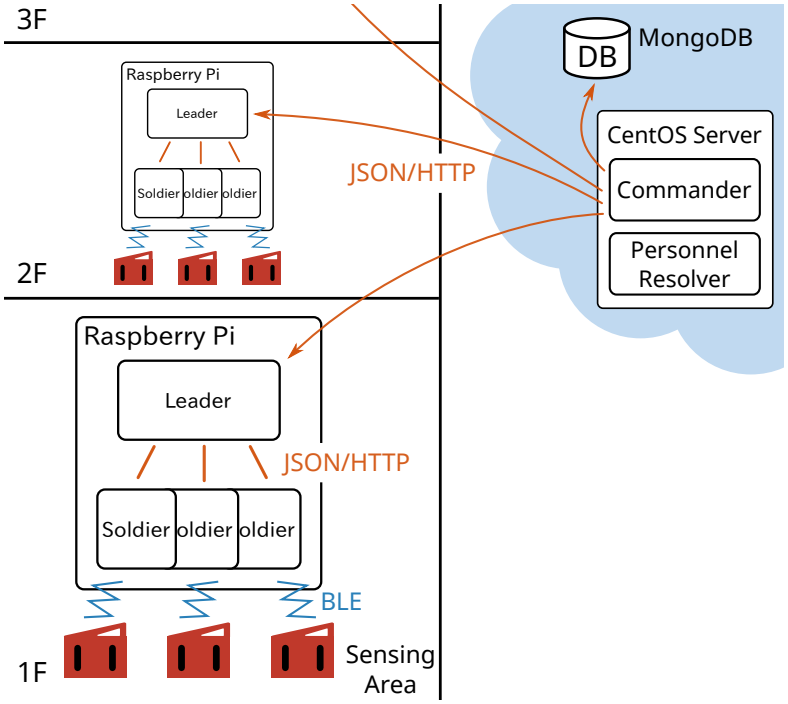

Figure 2: System configuration of the prototype

We also implemented *personnel resolver* as a Web service, with which each personnel (i.e., program in the military hierarchy) can resolve his superior and division based on his own ID. When starting up, each personnel accesses the personnel resolver to identify under whom he works.

The hardware configuration of the prototype is shown in Fig. 2. We used *SensorTag* [I] (a product of Texas Instruments Incorporated) for the sensing devices. SensorTag contains multiple environment sensors (barometer, temperature, humidity, brightness, gyroscope, accelerometer, magnetometer). These values can be obtained via Bluetooth Low Energy (BLE). *Raspberry Pi 3* was used as the execution platform of a leader and his subordinates, which were as described in Fig. 2. For each SensorTag, one soldier process is allocated. Each soldier obtains designated sensor values from the corresponding SensorTag via BLE. The devices used in the prototype are shown in Fig. 3. A commander process and the personnel resolver program were installed on a CentOS server.

5 Experimental Evaluation

5.1 Purpose of Experiment

To evaluate the proposed method, we conduct an experiment in a real environment using the prototype. The purpose of the experiment is to check basic features of the proposed platform. Especially, we confirm the parallel execution of multiple missions as well as dynamic mission

[I] http://www.ti.com/sensortag/

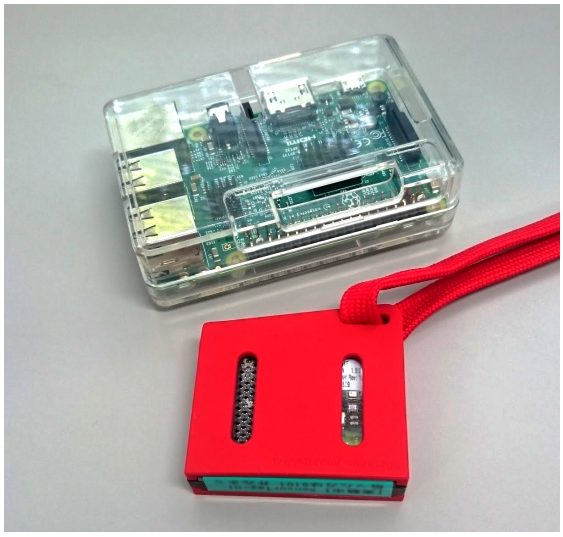

Figure 3: Raspberry Pi 3 and SensorTag

updates. We conduct the experiment in a real environment, instead of software simulation. This is to evaluate feasibility and limitations of the proposed method in practical setting.

5.2 Experiment Setting

The experiment has been conducted in corridors of the building of system informatics, Kobe university. The building has five floors. In every floor, one RaspberryPi and two SensorTags (east and west) were installed. For example, Fig. 4 shows a floor plan of the first floor, showing positions of devices deployed.

The experiment was conducted for three days from 14:30 of July 9th, 2016, to 14:30 of July 11th, 2016. Three missions M1, M2, and M3 were prepared to perform the environment sensing in parallel within the three days. The three missions respectively correspond to context-aware sensing (M1), high-density sensing (M2), low-density sensing (M3). The missions were configured by the same setting for `requirement, place, destination, supervisor`, as shown in Table 2. Only `trigger` was changed to make the comparison clearer, which is as shown in Table 3. Note in the table that M2 and M3 have static intervals of 10 and 120 seconds, respectively. As for M1, the sampling intervals are dynamically changed for night and day.

5.3 Result

Through the three-day environment sensing by 3 missions in 10 locations, and 7 kinds of sensors, the total 210 sets of time-series sensor data were collected. For example, the brightness in west-side of third floor (3F-west) and accelerometer in east-side of fifth floor (5F-east) are shown in Fig. 5 and Fig. 6, respectively. Around 2 a.m. of July 10th, processes of leaders of 4F and 5F were accidentally stopped by a server problem. The processes were restarted 9 a.m. of the same day. We can see in Fig. 6 that the data lacked during the failed term.

As a comparison of data with different intervals, the brightness in 1F-east with intervals of 10 seconds (M2) and 120 seconds (M3) are shown in Fig. 7 and Fig. 8. The two time-series data were collected by missions M2 and M3 at the same location and time. We can see from the graphs that two missions simultaneously measured the same brightness with different resolutions.

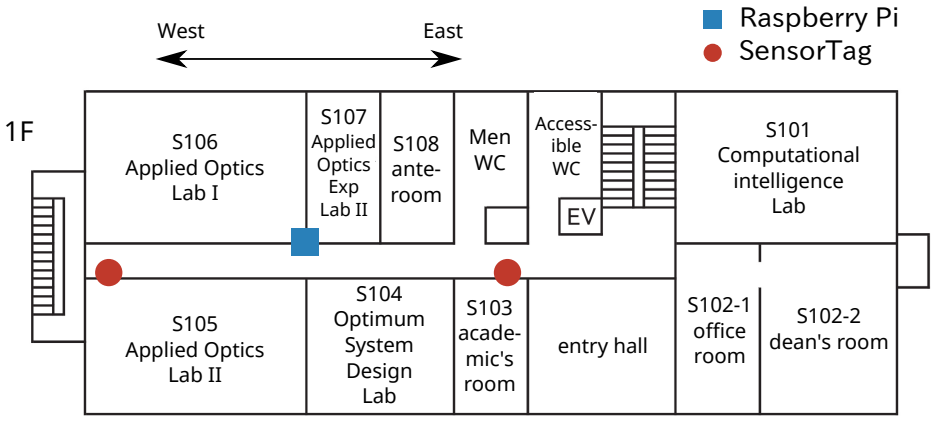

Figure 4: Positions of installed devices (1F)

Table 2: Common parameters of three missions

Parameter	Value
`requirement`	[barometer, brightness, humidity, temperature, accelerometer, gyroscope, magnetometer]
`place`	All five floors
`destination`	mongodb://dbserv/sensing
`supervisor`	Commander001

Table 3: Configuration of `trigger`

day	time slot	interval		
		M1	M2	M3
day 1	14:30 - 21:00	10 sec		
	21:00 - 9:00	120 sec		
day 2	9:00 - 21:00	10 sec	10 sec	120 sec
	21:00 - 9:00	120 sec		
day 3	9:00 - 14:30	10 sec		

Compared to the west side (see Fig. 5), the brightness of the east side is low even in day time, because the west side is near a sunburst window.

Fig. 9 shows the humidity measured by mission M1, where the sampling interval was changed at 9 p.m. It can be seen that the density of data was changed at the time. It means that the dynamic mission update was achieved successfully.

Finally, we evaluate the data loss by counting the number of data actually measured. The result is shown in Table 4. Theoretically, mission M2 should count $1,209,600$ data points, since 6 points per minute $\times$ 60 minutes $\times$ 48 hours $\times$ 10 devices $\times$ 7 sensor attributes. Mission M3 should count $100,800$ points, since it is one-twelfth of those of M2. Mission M1 switched its sampling interval between 10 and 120 seconds every half a day. Therefore M1 should count the half of those of M2 and the half of those of M3, which is $655,200$ in total. As shown in Table 4, the data loss rate within the three-day sensing was around 7% for M1, and 10% for M2 and M3.

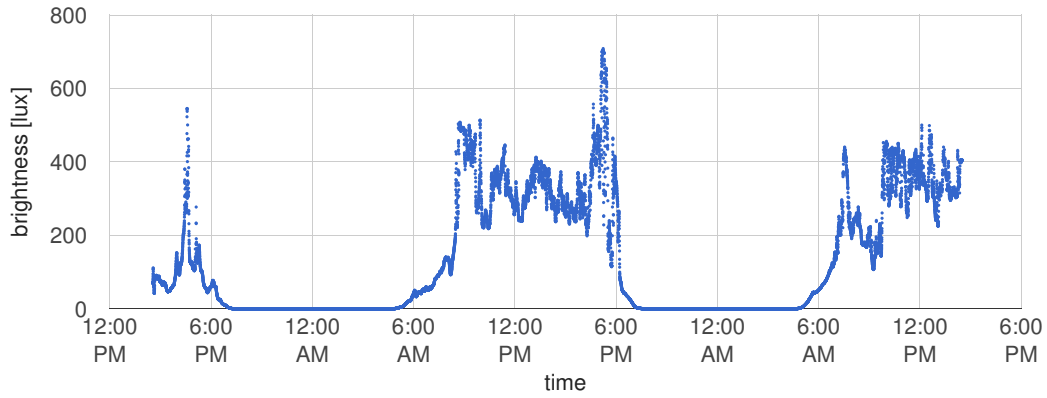

Figure 5: brightness in 3F-west (M2)

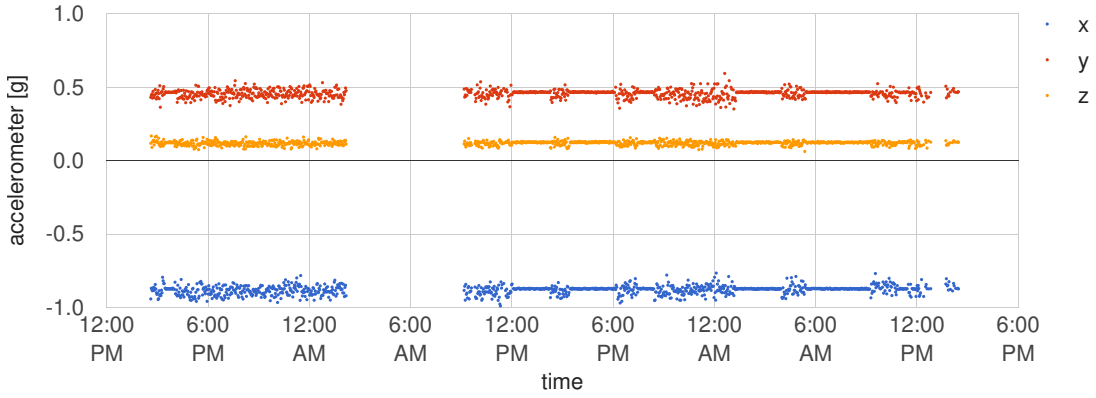

Figure 6: accelerometer in 5F-east (M3)

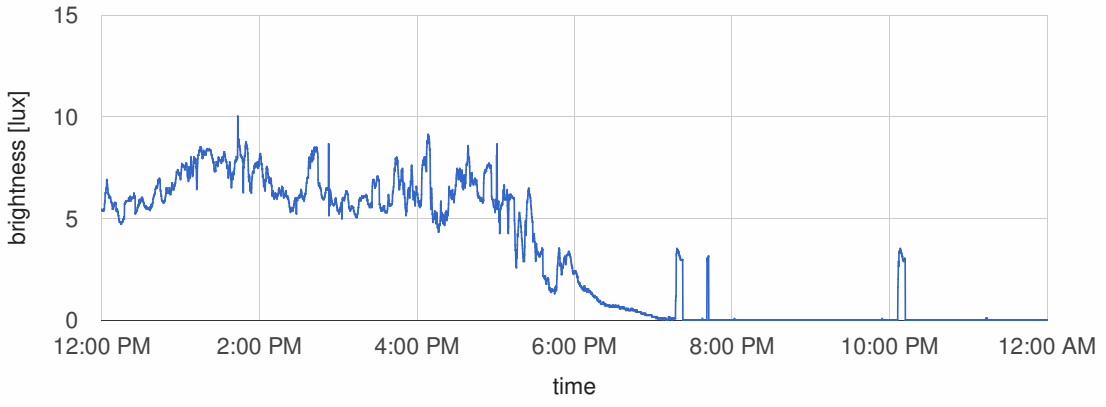

Figure 7: brightness in 1F-east (M2)

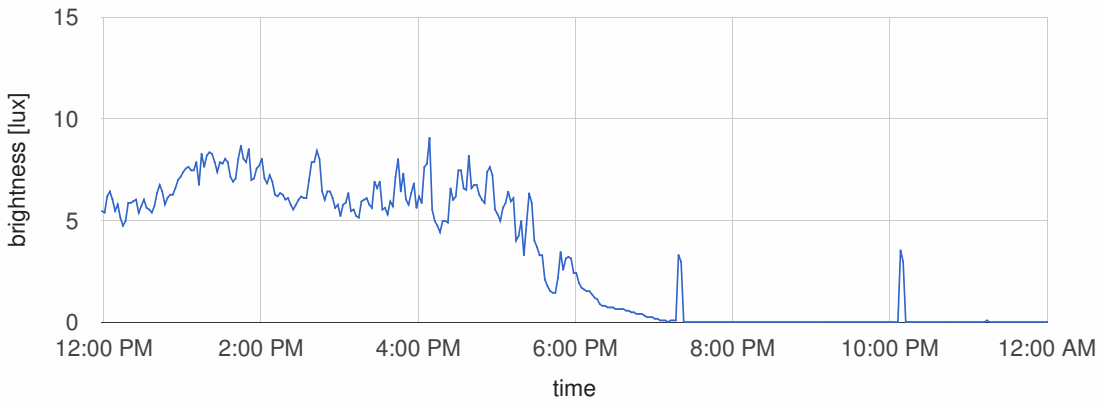

Figure 8: brightness in 1F-east (M3)

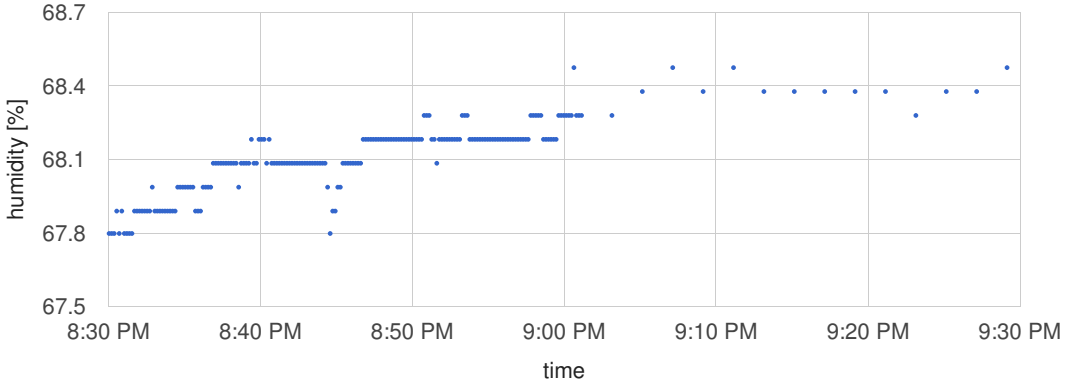

Figure 9: humidity in 1F-east (M1)

Table 4: Count of collected data

mission	actual count	expected count	loss rate
M1	608,118	655,200	7.18%
M2	1,090,733	1,209,600	9.82%
M3	89,194	100,800	11.52%

5.4 Evaluation

We can see in Fig. 5 that the brightness in the corridor was high in daytime and was low in nighttime. So it is confirmed that the environment sensing was properly performed according to the mission. In Fig. 7 and Fig. 8, the two time-series data have the same shape but different resolution. Thus it is confirmed that different missions M2 and M3 were performed in parallel at the same time. In Fig. 9, we can see that the density of data points changed at 9 p.m, which justifies that the dynamic update of mission M3 was successfully performed by the prototype.

As for the data loss shown in Table 4, a major cause of this is due to the failure of the leader processes in 4F and 5F. By this failure, environment sensing of 7 hours in the two floors was suspended. This was equivalent to 5.8% loss of the total sensing. Another cause of the data loss is due to unstable wireless communication between SensorTags and Raspberry Pi. We had to deploy Raspberry Pi inside an electric pipe shaft of the floor, where the power and wired network were available. Since the shaft was surrounded by walls and an iron door, BLE connections between SensorTags and RaspberryPi sometimes lost. This led to the data loss.

To cope with this problem, we have to implement a self-healing feature within individual soldier and leader to recover the connection, or a feature alerting problems so that a superior can detect the failure. In addition to the features, in case that a leader fails, we can deploy multiple leaders for same sensing area in advance, and let an operational leader take over the failed leader. The mechanism can be realized by a commander, in a way that the commander orders the take-over action when a failure of a leader is detected. Note, however, that a leader cannot always take over a failed leader due to compatibility of hardware / software or sensor device's specification.

In Fig. 7 and Fig. 8, we can observe short pulses of the brightness in the night time. The SensorTag measuring this brightness was installed beside a toilet as shown in Fig. 4. Therefore, the SensorTag captured automatic lighting of the toilet as a short pulse, which can infer the entrance of the toilet. Interestingly, we can observe two consecutive pulses around 8 p.m. in Fig. 7, but only one pulse in Fig. 8. This means that mission M3 missed one of the two toilet entrances. As seen in this example, decreasing sensing density can reduce the data size, but may cause loss of information. Therefore, each mission should be carefully designed based on its purpose, considering the trade-off between data size and information quality.

6 Discussion

In this section, we discuss relevant topics for future improvement of the proposed method.

6.1 Considering Data Freshness

In the proposed platform, measured sensor data are cached on a leader's hand and sent to a commander periodically, as described in Section 3.4. This means that the measured data may not be always *fresh* when the data arrives at the database.

The large-scale sensing infrastructure communicates large volume data, and gets heavy loads from a lot of data transfer's overheads. To reduce the load we reduce the number of transfer using the cache mechanism, and this leads to decreasing freshness of the data.

For some applications, however, high freshness of information is important. For instance, an application that monitors home situation online (e.g. abrupt increase of room air temperature, suddenly rising sound volume) needs fresh sensor data. On the other hand, the freshness is not important for an application that calculates average temperature every day with a batch process.

Fresh data can be collected by performing sensing without cache, although this increases the load of data transmission. In other words, information freshness and reducing the load on infrastructure are in a trade-off relation. To allow applications to choose freshness or reducing the load, we consider introducing freshness as a mission parameter, which adjusts the cache level.

6.2 Alleviating Workload of Commander

In the proposed platform, all sensor data measured in a sensing area are aggregated to upper layers of the hierarchy. They are eventually gathered to a database through a commander. A commander receives all leaders' reports and can be a bottleneck of the data flow.

There are several ways to cope with the bottleneck of the commander. First, we can try to reduce the number of transfer. This approach needs long-time cache as described in the previous subsection.

Second, we can compress the report data. This approach can reduce volume of information by summarizing cached data, or by compressing the data on communication protocols. In our prototype, changing data format from JSON to MessagePack [II] [2], compressing HTTP communication by gzip, or using MQTT[III] instead of HTTP can be promising methods.

Third, we can allow a leader to bypass the data without routing a commander. Each leader sends own reports to the database directly to eliminate workload of receiving reports from commanders. In this way, the commanders can focus entirely on managing their subordinates' state or making and sending operations. However, individual leaders have to manage right destination of every report, which may increase the total complexity of the system.

6.3 Handling Streaming Media

The proposed method does not require specific protocols or technology for data transmission between the devices. It requires at least the capability of sending small data chunks of sensor data or mission. There is no need of high bandwidth or fast latency. In addition, as far as every pair of superior and subordinate can communicate, different protocols can be used for different layers.

In return for the loose restriction in the protocols, the proposed platform is not good at dealing with stream media such as video or audio streams. The platform basically aggregates data in the middle of entire data flow. Therefore, it does not assume the use-case of pipeline, where the stream data is transmitted from the data source directly to the destination.

6.4 Choosing Communication Protocol

As previously discussed, any protocol can be chosen for data transmission between devices. Especially between leaders and soldiers, there is a variety of choices, each of which has own characteristic.

The Wireless LAN (Wi-Fi) is the most popular protocol for PCs, smartphones or one-board PCs. On top of Wi-Fi, standard protocols such as TCP/IP and HTTP are available. Thus, choosing Wi-Fi facilitates software development of leaders and soldiers. However, it consumes much electricity, which cannot be applied to devices with energy constraints.

The Bluetooth Low Energy (BLE) is an emerging protocol suitable for battery-powered devices. It is used in the proposed prototype with SensorTag. BLE consumes much less energy

[II] http://msgpack.org/

[2] a data format expressing data structure likes JSON, as binary instead of text data. In general, it needs less data size than JSON.

[III] http://mqtt.org/

than Wi-Fi. However, tools or libraries are not widely spread yet. So, low-level programming is required for implementing data transmission.

For a protocol in the upper layer, there is a choice of HTTP, MQTT or WebSocket. Choosing HTTP facilitates software development, because a wide variety of existing knowledge and resources are available. However, since the information must be always *pulled*, HTTP is not good at delivering real-time events (e.g. mission alert from a superior, or joining message from a new soldier) [3]. MQTT is an emerging protocol for IoT that uses the publish / subscribe message communication. WebSocket is a standard protocol that supports push communication. Using MQTT (or WebSocket) allows two-way and real-time notification, where connection management is supported by standard features of the protocol.

7 Related Works

Perera et al. (Perera et al., 2015) proposed a mobile sensing platform for context-aware sensing in the IoT domain. Their approach measures sensor data only when a pre-defined context holds. It requires each sensing device (i.e., a soldier in our method) to install a middle-ware for the context evaluation. So it assumes relatively rich devices such as smartphones. Our approach differs in that the context reasoning is up to individual applications, and that the sensing platform dynamically changes the sensing configuration by mission update. In this sense, our method can work with cheaper and fixed devices, which is good for large-scale environment sensing.

Sakakibara et al. (Sakakibara et al., 2016) proposed an autonomous sensor box and management services for easy provisioning and management of IoT sensors.

In their system, each sensing device automatically retrieves its sensing configuration from a cloud service when booting. This reduces human effort of installation and configuration of a large number of devices.

This kind of mechanism is quite essential for our problem of the large-scale environment sensing. However, their system does not consider capabilities of dynamic configuration update or infrastructure sharing, which are focused in this paper.

Autefage et al. (Autefage et al., 2015) proposed a service discovery system for mobile swarm of UMS (Unmanned System). This system is similar to ours in that the system tries to discover necessary devices in a mission-oriented manner. The method considers to choose an optimal communication method, depending on the device mobility or the device network size.

We will consider these elements, when we extend our platform for mobile devices where unknown devices dynamically participate and leave from the sensing infrastructure.

Galache et al. (Galache et al., 2014) proposed a concept, ClouT, which manages large-scale resources within smart city. It prescribes unified services to abstract various computing resources (infrastructures, sensors, or actuators) as Cloud services.

This concept is relevant to ours in that it tries to use the large-scale sensor infrastructure as a shared platform among various applications. In ClouT, however, the concrete method for large-scale environment sensing is basically up to the software service layer, which is out of the scope.

[3] In the proposed method, the real-time notification is not a mandatory requirement. So, obtaining data based on polling communication is not a big problem.

8 Conclusion

In this paper, we have presented a novel platform for large-scale environment sensing that can be shared by multiple applications. The key idea to achieve the platform is the mission-oriented sensing, where application-specific sensing configurations are given by missions. The proposed method introduces three military analogies: hierarchy for divide and conquer, stepwise refinement of mission, communication with order–report protocol. They accomplish essential requirements of large-scale environment sensing, (R1) shared sensing infrastructure, (R2) dynamic sensing configuration, (R3) accommodation of heterogeneity.

We have also implemented the prototype, and conducted an experimental evaluation with the prototype. As a result, the proposed platform was feasible for practical environment sensing. Our future work includes development of self-healing mechanism for disconnection, as well as extension to allow dynamic change of hierarchy.

9 Acknowledgment

This research was partially supported by the Japan Ministry of Education, Science, Sports, and Culture [Grant-in-Aid for Scientific Research (B) (No.16H02908, No.15H02701, No.26280115), Young Scientists (B) (No.26730155), and Challenging Exploratory Research (15K12020)].

References

Atzori, L., Iera, A. and Morabito, G. (2010). "the internet of things: A survey", *Computer Networks* **54**(15): 2787–2805.

Autefage, V., Chaumette, S. and Magoni, D. (2015). "a mission-oriented service discovery mechanism for highly dynamic autonomous swarms of unmanned systems", *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pp. 31–40.

Galache, J. A., Yonezawa, T., Gurgen, L., Pavia, D., Grella, M. and Maeomichi, H. (2014). "clout: Leveraging cloud computing techniques for improving management of massive iot data", *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, pp. 324–327.

Hollands, R. G. (2008). Will the real smart city please stand up?, *"City: analysis of urban trends, culture, theory, policy, action"* **12**(3): 303–320.

Lazarescu, M. (2013). "design of a wsn platform for long-term environmental monitoring for iot applications", *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **3**(1): 45–54.

Miorandi, D., Sicari, S., Pellegrini, F. D. and Chlamtac, I. (2012). "internet of things: Vision, applications and research challenges", *Ad Hoc Networks* **10**(7): 1497–1516.

Perera, C., Talagala, D. S., Liu, C. H. and Estrella, J. C. (2015). "energy-efficient location and activity-aware on-demand mobile distributed sensing platform for sensing as a service in iot clouds", *IEEE Transactions on Computational Social Systems* **2**(4): 171–181.

Sakakibara, S., Saiki, S., Nakamura, M. and Matsumoto, S. (2016). "indoor environment sensing service in smart city using autonomous sensor box", *15th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2016)*, pp. 885–890. Okayama, Japan.

Yu, X., Sun, F. and Cheng, X. (2012). "intelligent urban traffic management system based on cloud computing and internet of things", *International Conference on Computer Science Service System (CSSS)*, pp. 2169–2172.