Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 271817, 8 pages
http://dx.doi.org/10.1155/2014/271817

Research Article

SCondi: A Smart Context Distribution Framework Based on a Messaging Service for the Internet of Things

Jongmoon Park and Myung-Joon Lee

Department of Electrical/Electronic and Computer Engineering, University of Ulsan, 93 Daehak-ro, Nam-gu, Ulsan 680-749, Republic of Korea

Correspondence should be addressed to Myung-Joon Lee; mjlee@ulsan.ac.kr

Received 25 March 2014; Accepted 6 May 2014; Published 26 August 2014

Academic Editor: Young-Sik Jeong

When developing IoT (Internet of Things) applications, context distribution is a key feature to support effective delivery of related contextual data obtained from things to all interested entities. With the advent of the IoT era, multiple billion devices can generate huge amounts of data that might be used in IoT applications. In this paper, we present a context distribution framework named SCondi utilizing the messaging service which supports MQTT—an OASIS standard IoT messaging protocol. SCondi provides the notion of *context channel* as a core feature to support efficient and reliable mechanism for distributing huge context information in the IoT environment. The context channel provides a pluggable filter mechanism that supports effective extraction, tailoring, authentication, and security of information.

1. Introduction

Context-aware computing, as a component of a ubiquitous computing [1–4], is a core technology that supports human-centric intelligent service using contextual information in real situations. As machine-centric world (M2M, machine-to-machine) [5, 6] is coming, the notion of context awareness supporting improved response ability plays an important role in the IoT since things connected to the internet have awareness/sensing ability in many cases [7–9].

The IoT which provides useful services combined with various IT technologies is considered to have the potential to change our world. The IoT is a future internet environment defined as a dynamic global network infrastructure [10]. Things with identities and intelligent interfaces can be active participants wherever they are enabled to interact and communicate. To support useful services based on the massive information related to these things, several technologies such as REST (representational state transfer) [11], MQTT (message queuing telemetry transport) [12, 13], XMPP (extensible messaging and presence protocol) [14], and CoAP (constrained application protocol) [15] are being studied for information delivery, message process, and communication protocol. Recently, the MQTT protocol has been adopted by OASIS (organization for the advancement of structured information standards) as a standard messaging protocol for the IoT.

With the advent of the IoT era, multiple billion devices can generate huge amounts of data that might be used in IoT applications. However, it is not an easy challenge to deliver these tremendous data at the right time, to the right place, and with the right quality. As one of the important components in context-aware computing, context distribution can be a key feature to support effective delivery of contextual data in the IoT environment [16]. As of now, although many research works on context distribution has been conducted, there is no standard mechanism for reliable and sophisticated context distribution in the IoT environment.

In this paper, we propose a context distribution framework named SCondi utilizing the messaging service which supports MQTT. SCondi provides the notion of *context channel* as a core feature to support efficient and reliable mechanism for distributing huge context information in the IoT environment. The context channel is an abstract communication channel which can reliably tailor and disseminate a collection of information to service providers. Based on the MQTT messaging service, the context channel provides a pluggable filter mechanism that supports effective extraction,

tailoring, authentication, and security of information. In short, the context channel provides higher level abstract mechanism for information delivery as a transport medium of context-aware systems in the IoT environment.

2. Backgrounds

IoT is a future internet environment that focuses on machine to machine communication, referring to uniquely identifiable objects and their virtual representations. MQTT is a standard Internet of Things connectivity protocol that is designed as an extremely lightweight publish/subscribe messaging transport, considering limited computing power and poor network connectivity. The MQTT protocol is developed by IBM and chosen as standard by OASIS.

As HTTP has made a web to be an infrastructure that share information over the internet, MQTT is expected to be a key infrastructure that makes billions of embedded low price devices to go online. It is already widely used in a lot of embedded systems.

Mosquitto is an open source (BSD licensed) message broker that implements the MQTT 3.1, providing a lightweight method of carrying out messaging using a publish/subscribe model [17, 18]. Mosquitto is designed to fit messaging among machines such as low-power sensors, mobile devices, embedded computers, and microcontrollers, supporting various OS platforms such as Microsoft's Windows, Apple's OS X, and Linux family.

Context-aware computing is a core technology that supports human-centric intelligent service using contextual information in real situations. Context refers to a variety of information that can define the state of the real world's entities, generally consisting of information such as entity identity, activity, status, time, and location [19]. Since many devices which can sense contextual information are getting connected to the internet, the context-aware computing plays an important role in the IoT.

3. Design of SCondi

In this section, we introduce a context distribution framework named SCondi which is based on a messaging service for the IoT, supporting smart and effective dissemination of context data. We begin by mentioning the core requirements for the context data distribution proposed by Bellavista et al. [20]. To satisfy the requirements, the context channel is provided with filter mechanism as a key facility for reliable context data distribution.

3.1. Key Requirements of Context Data Distribution. Bellavista et al. mentioned that context data distribution needs to meet following 5 requirements for providing an effective context-aware service in IoT environment.

(1) Communication should be asynchronous and anonymous among context producers and consumers.

(2) To support mobile heterogeneous and wireless scenarios, the context data distribution has to promptly adapt to mobility and current available resources.

(3) The context data distribution must enforce visibility scopes of context data to avoid useless management overhead.

(4) The context data distribution has to enforce QoC-based constraints for timeliness and reliability guarantees of data delivery.

(5) The context data distribution has to handle data life cycle for self-control of the distribution process.

To satisfy the requirements, SCondi provides the following features. First, adopting MQTT as a messaging mechanism, our framework supports asynchronous and anonymous communication among message publishers and subscribers (satisfying (1)). Secondly, our framework provides effective mobility and reliable message delivery based on MQTT's QoS (Quality of Service) in limited network environments (satisfying (2)). Thirdly, the context channel in our framework provides filter chain mechanism for the QoC constraints such as context data management, resource access control, data validation, and timeliness (satisfying (3) and (4)). Finally, it also provides common filters for general usage and customized filters through predefined interfaces (satisfying (5)).

3.2. Architecture of SCondi. SCondi has two key components: context channel and channel selector. The context channel is a transmission facility used to convey a collection of contextual data specified by the channel creator. The channel selector receives raw data from external data providers (e.g., sensors, SNS data, calendar data, email, etc.), spreading each raw context data to all the context channels that need the contextual data as their constituent, using the MQTT messaging facility as shown in Figure 1. To receive the required data, context channels should subscribe to the channel selector. When receiving the collection of the associated contextual data, the context channel delivers the collection of data to each subscribers of the channel after processing the collection of data with the filter chain through the MQTT messaging facility. In this way, the MQTT message delivery mechanism is used twice to pass the associated data to channel subscribers.

A topic with unique namespace in MQTT is allocated to each context channel. The topic manages the associated contextual data as subtopic, forming a hierarchal structure. In other words, A channel ID is assigned as a main topic while each contextual data is assigned as a subtopic separated by a "/" following the channel ID. Based on the MQTT messaging facility, SCondi provides decoupled one-to-many pub/sub through the context channel which allows any contextual data to be published once and multiple consumers to receive the collection of the needed contextual data.

In SCondi, a context is composed of a set of context primitives (CPs), where a CP is a set of related data that are used as a practical unity in applications. For instance, most elevators in the modern era have been fitted with several safety devices such as overload sensor, door sensor, fire sensor, gas sensor, cable sensor, and fault diagnosis module as shown in Figure 2. A manufacturer needs information such as equipped sensors and location of installed elevators for

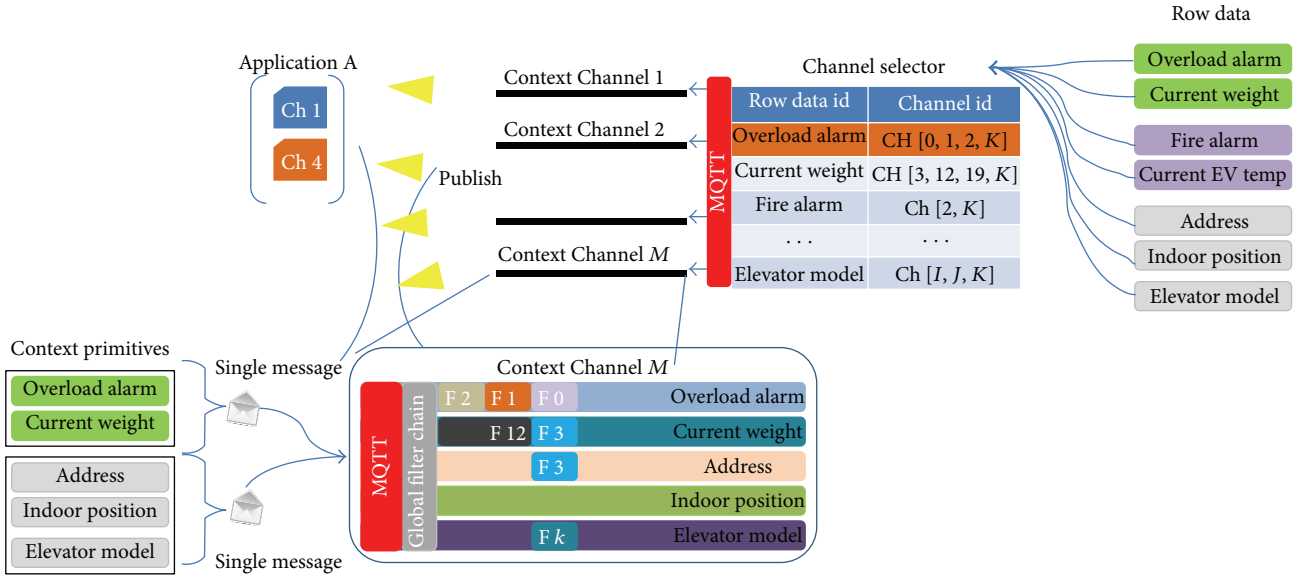

Figure 1: Overall structure of SCondi.

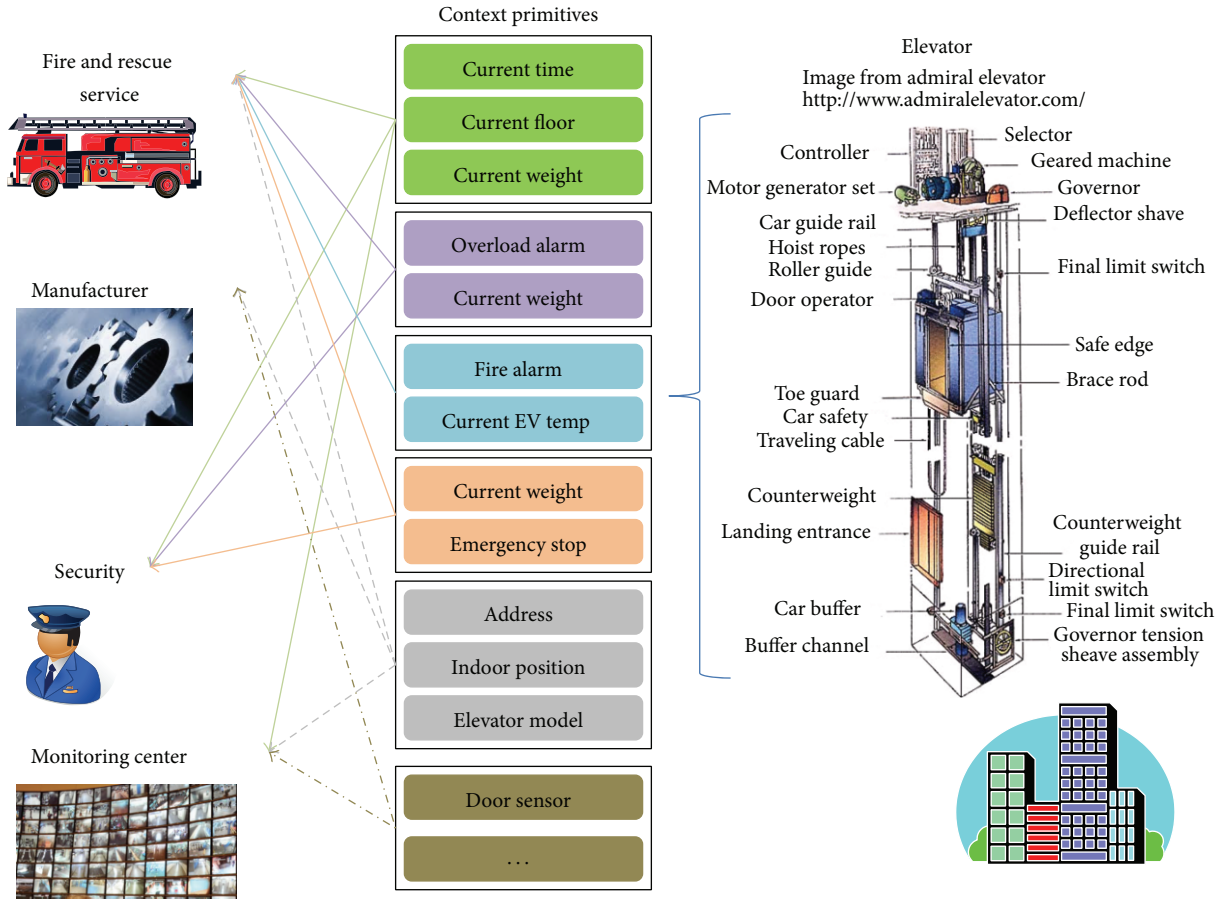

Figure 2: Example of context primitive.

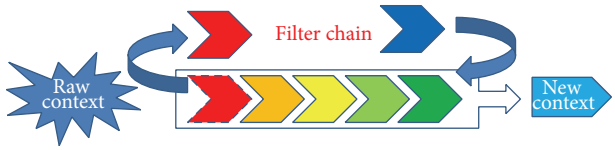

Figure 3: Pluggable ordered filter chain mechanism.

maintenance all together. In emergency situation, information such as floor, time, and location are required for the rescue service at the same time. As above, CP can be used effectively when information is commonly used or needed to be provided as a meaningful unit.

SCondi provides an authentication and authorization mechanism to manage access levels for context channels. To access a context channel, a subscriber should have a retrieval authority for the channel. For convenience of users, the framework supports three types of context channel according to the purpose of the application: *open*, *access-limited*, and *group channel*. An open channel allows everyone to access while an access-limited channel accepts the qualified subscriber who has the related access key. A group channel accepts the users authorized by the authentication module for the related group only.

To process contextual data with different characteristics depending on the purposes of context channels, we provide context channels with the *filter chain* mechanism. A filter chain is a collection of ordered filters. As a key feature to determine the characteristics of a context channel, the filter chain of the channel provides predefined filter interfaces to process contextual data. In addition, it allows *pluggable filter adaptation* which supports dynamic filter insertion/removal without interruption of service during run-time. For the effective run-time channel configuration, the framework uses the concept of ownership to context channels.

3.3. Context Filter for Quality of Context. QoC (Quality of Context) is a very important factor to be considered for the context data distribution efficiency and reliability. Traditionally QoC has focused on the quality of data only. Recently, to ensure the availability of data with the right quality, in the right place and at the right time, many related studies concentrate on complex characteristics such as data transmission time, data reliability, data accessibility, data refreshment/up-to-date, and data precision. In other words, QoC must be considered depending on the purpose of the services. To reflect these aspects, SCondi provides filter chain mechanism that allows filters to be applied for supporting QoC in context channels.

A filter changes original contextual data to qualified data according to QoC criteria. Also, as shown in Figure 3, filters can be logically combined to create more complex subscriptions patterns depending on the characteristics of context channels. In other words, filter chain mechanism helps to reduce bandwidth and to enhance scalability and to increase QoC for context distribution in the IoT environment. Our framework provides common filters that can be commonly used in context channels for QoC constraints such as value_range, time, value_changed, and average filter.

Table 1: Filter interface.

Interface	Description
execute(context)	Changes original contextual data to qualified data according to the filter logic
nextFilter(context)	After execute(), passes context to next filter in the filter chain

Table 2: Filter chain interface.

Interface	Description
addLast(filter)	Adds the specified filter at the beginning of this chain
addFirst(filter)	Adds the specified filter at the end of this chain
addBefore(filter, base filter)	Adds the specified filter before the base filter in this chain
addAfter(filter, base filter)	Adds the specified filter after the base filter in this chain
invoke(context)	Invokes the filters of the chain in order
delete(filter)	Deletes the specified filter in this chain
replace(base filter, new filter)	Replace the specified filter with new filter

(1) The value_range filter passes values only within a certain range.

(2) The time filter passes values within a specific time period.

(3) The value_changed filter passes values different from previous values.

(4) The average filter calculates the average value with the specified condition, transmitting the calculated value.

In addition, SCondi provides predefined interfaces to create a custom filter and to manage filter chain apart from the common filters.

4. Implementation of SCondi

In this section, we explain the implementation of context distribution framework to support reliable delivery of context data. SCondi is implemented with Java program language, using Mosquitto as the MQTT message broker to support efficient and reliable messaging. The context channel provides a filter chain mechanism through the filter interface and the filter chain interface. The *filter interface* provides *execute()* to process contextual data through a specific filter as shown in Table 1. To pass the processed contextual data to the next filter in the filter chain, it provides *nextFilter()*. Additionally, as shown in Table 2, it provides the filter chain interface to add, delete, and replace a filter in a chain. The *filter chain interface* provides *addFirst()*, *addLast()*, *addBefore()*, and *addAfter()* to add filters at a specific position in a chain. It also gives *delete()* to remove filters and *replace()* to replace filter with a new one.

```
<Context>
  <Context Primitive1>
    <Overload Alarm>OFF</Overload Alarm>
    <Current Weight>120 KG</Current Weight>
  </Context Primitive1>
  <Context Primitive2>
    <Fire Alarm>OFF</Fire Alarm>
    <Current EV Temp>24°C</... >
  </Context Primitive 2>
  <Context Primitive 3>
    <Address> ......
  </Context Primitive 3>
</Context>
```

Algorithm 1: Context definition.

Lastly, It provides *invoke()* to execute the filters of the chain in order. In this way, the context channel can be configured to have various characteristics depending on the ordering of filters. A context is defined by a set of CPs as described in Algorithm 1.

Algorithm 2 shows an example of using a filter chain. In this case, the associated context channel executes Average-Filter for calculating average value of 10 recent weights of the underlying elevator. Then, ChangedValueFilter compares the calculated value with the previous average value. If the calculated value is not equal to the previous one, the context channel delivers the calculated value to subscribed applications.

According to the rapidly increasing number of devices in the IoT, both context channels and the channel selector can impose heavy overloads on a single MQTT message broker. Thus, SCondi uses 2 message brokers for the channel selector and context channels, respectively. A topic of the first Mosquitto message broker (for channel selection) is assigned to each source of contextual data which is provided by the external context adapter. The channel selector manages a contextual data and the associated channels through the context-channel mapping table, publishing the data to the message broker through the assigned topic. To subscribe a specific contextual data, each context channel should request a subscription of the associated topic to the message broker. After acquiring the approval from subscription authorization, the context channel can receive the interested data from the message broker by subscribing to the associated topic.

After processing the received data through its filter chain, the context channel publishes the specified set of data to the second message broker (for end user delivery). To receive the set of interested contextual data from the context channel, end user applications should subscribe to the context channel. Our framework also provides the management facility for subscription permissions on context channels. In addition, the channel provides the context-filter mapping table to manage each context data and its related filters. The channel supports 2 types of filter: *global filter* and *local filter*. Whole context data in the channel is affected by the global filter while local filter is applied only to the specific context data based on the table.

```
...
FilterChain =
        Channel.getFilterChain(CurrentWeight)
FilterChain.addFirst(new AverageFilter(10));
FilterChain.addLast(new ChangedValueFilter());
...

class AverageFilter
implemented Filter
{
......
    Object execute(Object c){
      sum += c;
      if (checkCount()){
        result = sum/avgCount;
        init();
        return nextFilter.execute(result);
         }
......
}

class ChangedValueFilter
implemented Filter{
......
    Object execute(Object c){
      old = cur;
      cur = c;
      if (isChanged() == true)
        return nextFilter.execute(cur);
      ...
}
```

Algorithm 2: Example of filter chain.

5. Performance Analysis

Since SCondi provides higher level abstract mechanism for information delivery in the IoT environment, the qualitative effectiveness of our framework is very clear. So, we focus on the effectiveness in terms of quantity of delivered messages as illustrated in Figure 4. Whereas each data should be transmitted in an original MQTT message broker, all data in a CP can be passed as a single message through our context channel with appropriate setting. To show effectiveness of our framework, we compare the quantity of delivered messages in a formal way. For this, let us begin by defining the following terms:

t_d: total number of source data;

t_p: total number of CPs;

m_{pd}: average number of source data in a single CP.

Then, in case that every CP is used once by application, the amount of source messages can be written as

$$t_p * m_{pd}. \quad (1)$$

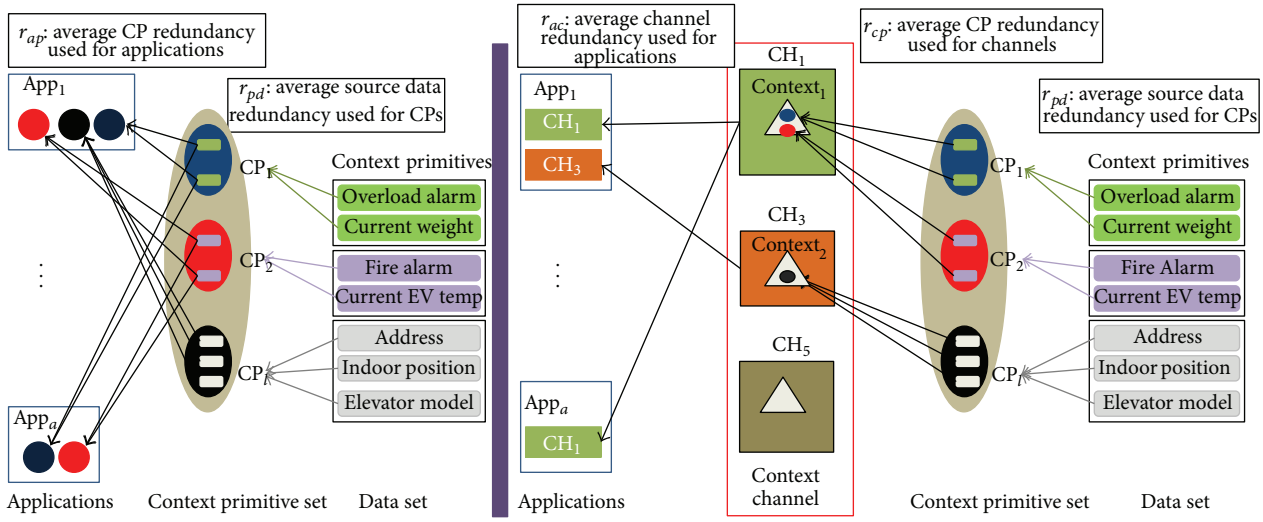

Figure 4: Context distribution with/without context channel.

In general, a source data can belong to multiple CPs. So we define r_{pd} denoting the average source data redundancy used for CPs:

$$r_{pd} = t_p * \frac{m_{pd}}{t_d} \quad \left(r_{pd} > 0\right). \tag{2}$$

Let t_a denote the total number of applications and m_{ap} denote average number of CPs in a single application.

For simplicity, assume that every application receives its needed data once for a certain period of time p. Then, the total number of CPs used during time interval p is

$$t_a * m_{ap}. \tag{3}$$

Since a CP is generally required by multiple applications,we definethe average CP redundancy used for applications denoted by r_{ap}:

$$r_{ap} = t_a * \frac{m_{ap}}{t_p} \quad \left(r_{ap} > 0\right). \tag{4}$$

Now, we can calculate the amount of all messages (denoted by S_b) passing through the original MQTT message broker:

$$S_b = t_a * m_{ap} * m_{pd}. \tag{5}$$

Applying (2) and (4) to (5), we have the following:

$$S_b = r_{ap} * r_{pd} * t_d. \tag{6}$$

Now, we measure the amount of all messages in our SCondi. We start by new terms t_c and m_{cp} that denote the total number of channels in SCondi and the average number of CPs used in a single channel, respectively. We assume that every context channel receives its needed CP once for a certain period of time p_1. Then, the total number of CPs passed during time interval p_1 can be written as

$$t_c * m_{cp}. \tag{7}$$

Since a CP generally belongs to multiple context channels, r_{cp} which is the average CP redundancy used for channels can be defined by

$$r_{cp} = t_c * \frac{m_{cp}}{t_p} \quad \left(r_{cp} > 0\right). \tag{8}$$

We assume that every application receives its needed CP once from the related context channels for a certain period of time p_2. Let m_{ac} denote the average number of channels required in a single application. Since a context channel can belong to multiple applications, let us define r_{ac} meaning the average channel redundancy used for applications by

$$r_{ac} = t_a * \frac{m_{ac}}{t_c} \quad \left(r_{ac} > 0\right). \tag{9}$$

Recall that each data should be transmitted in an original MQTT message broker while all data in a CP can be passed as a single message from context channels with appropriate setting. Now, we can count the amount of all messages (denoted by S_c) passing through SCondi during time period $(p_1 + p_2)$:

$$S_c = t_a * m_{ac} * m_{cp} + t_c * m_{cp} * m_{pd}. \tag{10}$$

Applying (2), (8), and (9) to (10), we have the following:

$$\begin{aligned} S_c &= t_c * r_{ac} * m_{cp} + t_p * r_{cp} * m_{pd} \\ &= r_{ac} * r_{cp} * t_c + r_{cp} * r_{pd} * t_d. \end{aligned} \tag{11}$$

Since the time required to transmit messages is fairly smaller than the time interval that applications require source data, we can assume $p = p_1 + p_2$ without loss of generality. Now, we discuss the condition over which the amount of all messages passing through SCondi is less than the amount of

all messages passing through original MQTT message broker. Consider

$$S_b > S_c = r_{ap} * r_{pd} * t_d > r_{ac} * r_{cp} * t_p + r_{cp} * r_{pd} * t_d \quad (\text{applying } (6), (11)). \tag{12}$$

Note that the condition

$$r_{ap} > r_{cp} \tag{13}$$

should always satisfy to hold $S_b > S_c$, since $r_{ac} * r_{cp} * t_p$ is always greater than zero. Consider

$$\left(r_{ap} - r_{cp}\right) * r_{pd} * t_d > r_{ac} * r_{cp} * t_p. \tag{14}$$

In case that $r_{ap} > r_{cp}$, (14) is equivalent to

$$r_{pd} * \frac{t_d}{t_p} > r_{ac} * \frac{r_{cp}}{r_{ap} - r_{cp}}. \tag{15}$$

Since $r_{pd} = t_p * m_{pd}/t_d$ (by (2)), we finally have the following:

$$m_{pd} > \frac{r_{ac} * r_{cp}}{\left(r_{ap} - r_{cp}\right)}. \tag{16}$$

To capture the meaning of conditions for $S_b > S_c$, using a practical data, let us calculate the amount of total delivered messages in SCondi and that of MQTT broker. We assume that $t_d = 100000$, $t_p = 80000$, $t_a = 5000$, and $m_{pd} = 3$. For MQTT broker case, we assume the average number of CPs in a single application is 200. In SCondi, we assume that the total number of channels is 2000, the average number of CPs in a single channel is 50 and the average number of channels in a single application is 4. Table 3 shows the calculation of the total delivered messages in both cases. As a result, S_b is 3000000 and S_c is 1300000 when m_{pd} is 3. In this case, both conditions (13) and (16) "$r_{ap}(= 12.5) > r_{cp}(= 1.25)$" and "$m_{pd}(= 3) > r_{ap} * r_{cp}/(r_{ap} - r_{cp})(= 1.11)$" are satisfied, resulting in $S_c < S_b$.

Figure 5 depicts the total amount of messages according to the value of m_{pd} under the same conditions. Considering the result, even in the aspect of the amount of total messages passing through, the concept of context channel is very useful and effective when CP is composed of two or more source data.

Table 3: Example of delivered message comparison with practical data.

Common conditions $t_d = 100000, t_p = 80000, t_a = 5000, m_{pd} = 3$	
MQTT broker case	SCondi case
Let $m_{ap} = 200$	Let $t_c = 2000, m_{ac} = 4, m_{cp} = 50$
$S_b = t_a * m_{ap} * m_{pd}$ $= 5000 * 200 * 3$ $= 3000000$	$S_c = t_a * m_{ac} * m_{cp}$ $+ t_c * m_{cp} * m_{pd}$ $= 5000 * 4 * 50$ $+ 2000 * 50 * 3$ $= 1300000$
$r_{pd} = t_p * m_{pd}/t_d$ $= 80000 * 3/100000$ $= 2.4$	$r_{cp} = t_c * m_{cp}/t_p$ $= 2000 * 50/80000$ $= 1.25$
$r_{ap} = t_a * m_{ap}/t_p$ $= 5000 * 200/80000$ $= 12.5$	$r_{ac} = t_a * m_{ac}/t_c$ $= 5000 * 4/2000$ $= 10$
$m_{pd} > r_{ac} * r_{cp}/(r_{ap} - r_{cp})$ $> 10 * 1.25/(12.5 - 1.25)$ > 1.11	

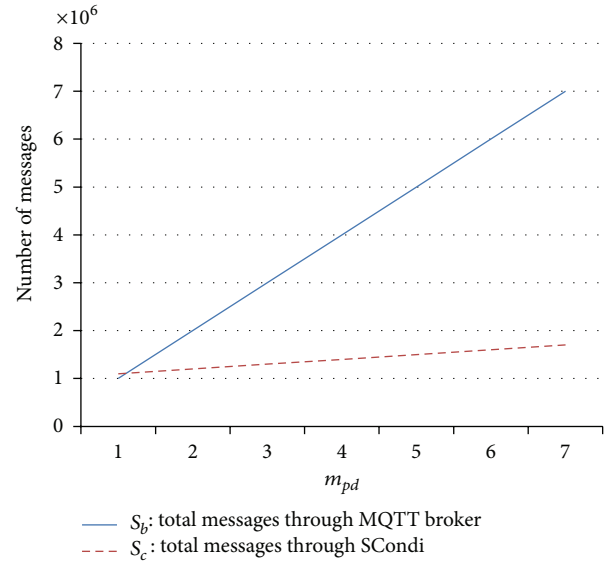

Figure 5: m_{pd} effect on total amount of messages.

6. Conclusion

In this paper, we introduced a context distribution framework named SCondi that supports effective dissemination of context data through context channels. SCondi is based on two major components: channel selector and context channel. The channel selector sends each raw context data to each of the context channels that requires the contextual data, using the MQTT messaging facility which has been adopted by OASIS as a standard messaging facility for the IoT. The context channel provides a filter chain mechanism that supports effective extraction, tailoring, authentication, and security of information through various types of filters. Based on the MQTT messaging facility again, when receiving the collection of the associated contextual data, the context channel delivers the collection of data to each subscriber of the channel after processing the collection of data with the filter chain. In addition, SCondi supports three types of

context channel according to the purpose of the application: open, access-limited, and group channel. We also showed that SCondi is very useful and effective both in quality and quantity of delivered messages. We believe that the novel concept of context channel and the presented framework can be very useful for context distribution in the IoT environment.

Conflict of Interests

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

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (no. 2013R1A1A4A01004459).

References

[1] B. N. Schilit, N. I. Adams, and R. Want, "Context-aware computing applications," in *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pp. 85–90, IEEE Computer Society, Santa Cruz, Calif, USA, December 1994.

[2] J. Park, H. Lee, and M. Lee, "JCOOLS: a toolkit for generating context-aware applications with JCAF and DROOLS," *Journal of Systems Architecture*, vol. 59, no. 9, pp. 759–766, 2013.

[3] D. Gallego and G. Huecas, "An empirical case of a context-aware mobile recommender system in a banking environment," in *Proceedings of the 3rd FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing (MUSIC '12)*, pp. 13–20, IEEE, Vancouver, Canada, June 2012.

[4] S. Oh, "Using an adaptive search tree to predict user location," *Journal of Information Processing Systems*, vol. 8, no. 3, pp. 437–444, 2012.

[5] "M2M, machine-to-machine," http://www.m2m.com/.

[6] ABI Research, *More Than 30 Billion Devices Will Wirelessly Connect to the Internet of Everything in 2020*, ABI Research, London, UK, 2013.

[7] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: a survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[8] N. Ahmed Surobhi and A. Jamalipour, "A context-aware M2M-based middleware for service selection in mobile ad-hoc networks," *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[9] V. Cristea, C. Dobre, and F. Pop, "Context-aware environments for the internet of things," in *Internet of Things and Intercooperative Computational Technologies for Collective Intelligence*, vol. 460 of *Studies in Computational Intelligence*, pp. 25–49, Springer, Berlin, Germany, 2013.

[10] "Internet of Things Strategic Research Roadmap," CERP- IoT, April 2011.

[11] REST, http://tools.ietf.org/html/rfc2616.

[12] MQTT, http://mqtt.org/.

[13] V. Lampkin, W. T. Leong, L. Olivera, S. Rawat, N. Subrahmanyam, and R. Xiang, *Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry*, IBM Redbooks, New York, NY, USA, 1st edition, 2012.

[14] XMPP, http://xmpp.org/.

[15] CoAP, https://datatracker.ietf.org/doc/draft-ietf-core-coap/.

[16] T. Teraoka, "Organization and exploration of heterogeneous personal data collected in daily life," *Human-Centric Computing and Information Sciences*, vol. 2, no. 1, pp. 1–15, 2012.

[17] Mosquitto, http://mosquitto.org/.

[18] MQTT Servers/Brokers, http://mqtt.org/wiki/doku.php/brokers.

[19] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing (HUC '99)*, vol. 1707 of *Lecture Notes in Computer Science*, 1999, pp. 304–307.

[20] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A survey of context data distribution for mobile ubiquitous systems," *ACM Computing Surveys*, vol. 44, no. 4, article 24, 2012.