CSDA: Rule-based Complex Sensor Data Aggregation System for M2M Gateway

Yuichi Nakamura
Hitachi Solutions, Ltd., Tokyo, Japan
Okayama University, Okayama, Japan
Email: yuichi.nakamura.fe@hitachi-solutions.com

Akira Moriguchi
Hitachi Solutions, Ltd., Tokyo, Japan
Email:
amoriguchi@hitachi-solutions.com

Toshihiro Yamauchi
Okayama University
Okayama, Japan
Email: yamauchi@cs.okayama-u.ac.jp

***Abstract*—To reduce the server load and communication cost of machine-to-machine (M2M) systems, sensor data are aggregated in M2M gateways. The C language is typically used for programming the aggregation logic, and the program is embedded into the firmware. However, developing aggregation programs is difficult for M2M service providers because it requires gateway-specific knowledge, and consideration must be given to CPU and memory resources. In addition, modifying aggregation logic requires firmware updates, which are risky. We propose a rule-based sensor data aggregation system, called the complex sensor data aggregator (CSDA) for M2M gateways. Data aggregation is categorized into filtering, statistical calculation, and concatenation. The proposed CSDA supports this aggregation process in three steps: the input, data processing, and output steps. The behaviors of these steps are configured by an XML based rule. The CSDA also supports update modules, which download and overwrite aggregation rules from the server when the modification of data aggregation logic is required. In this case, firmware updates are not necessary. The proposed system is evaluated in an M2M gateway experimental environment. Results show that developing CSDA configurations is much easier than using C because the configuration amount decreases by 10%. In addition, the performance evaluation demonstrates the proposed system's ability to operate on M2M gateways. CPU usage was less than 10%, even with a heavy load, and memory consumption was 128 Kbytes.**

I. Introduction

With the growth of machine-to-machine (M2M) technology, many companies have begun to provide M2M services, which provide new value by utilizing machine sensor data. For instance, construction and agricultural machine manufacturers provide remote monitoring services for their products by sending sensor and location data from products in the field to servers over the Internet [1][2][3]. These services reduce down time and prevent machinery theft. In M2M services, sensors connect to local networks, such as Zigbee [4] or the controller area network (CAN) [5]. As a result, they cannot directly upload data to the Internet. In order to create a bridge between sensors and the Internet, M2M service providers utilize devices called M2M gateways. An M2M gateway is able to communicate with both the sensor network protocol and the IP [6]. For example, when using a remote monitoring service for construction machinery, an M2M gateway is attached to the machine and it gathers data from the sensors via the CAN. Then it uploads data to a server via a wireless Internet connection. However, cost becomes a problem because wireless Internet contracts are usually measured rates and the server resource increases with an increase in data. In addition, the number of M2M gateways can be tens of thousands, sometimes millions, and the amount of data uploaded to the server also increases, substantially raising the cost. In order to reduce this cost, M2M service providers must reduce the amount of data transferred between the M2M gateway and the server.

There are three main approaches used to reduce the amount of data transferred between an M2M gateway and the server. The first approach reduces the protocol header overhead by using a lightweight protocol. The HTTP is a popular means of communication between the server and client. However, in M2M systems, HTTP overhead becomes a problem because the sensor data size is often about 100 bytes, but the HTTP header size is greater than 100 bytes. To reduce the overhead, lightweight protocols, such as MQTT [7] and CoAP [8], are efficient because their protocol header size is around 10 bytes [9]. In addition to this protocol-based approach, data can be reduced before being sent from the M2M gateway.

The second approach reduces the data size by processing the data on sensor nodes. TinyDB [10] processes a language similar to SQL on sensor nodes. SensorWare [11] also has a script language for sensor nodes. By using these systems, only necessary data is sent from the sensors, consequently reducing the amount of data sent from the M2M gateway. These technologies are suitable for controlling devices within sensor networks because their response times are fast. When a sensor detects an event, an action is immediately issued from the sensor. However, these technologies are not suitable for reducing the amount of data transferred between an M2M gateway and the server for two reasons. First, these systems are not intended for processing data from multiple sensors. It is efficient to gather a sensor's data depending on another sensor value. However, these technologies do not support such complex processes. Second, the installation and management cost is substantial. Data processing middleware and configuration need to be installed on all sensor nodes. If there are 100 sensors per machine and 10,000 machines, the number of required sensors is one million. Installing and managing software for this many machines is difficult. In addition, because sensor nodes are connected to the sensor network and not to the IP network, the two networks must be bridged in order to manage the sensor nodes from the server.

For such reason, M2M service providers take third approach, i.e. data aggregation on gateway, such as taking histogram and average from sensors on M2M gateway [12][13][14]. In this approach, the installation of aggregation logic is restricted to the M2M gateway, which is easier than

installing the logic on the sensors. Moreover, standardized technologies, such as TR-069 [15], used for managing software and configuration contribute to a reduction in installation and maintenance costs. M2M service providers need to write programs for the sensor data aggregation, but there are two issues with this. First, there is a problem with developing programs on firmware. Since M2M gateways are resource constrained, the programming language is usually C. Programming in C requires troublesome memory management, which is hidden in higher-level languages. In addition, M2M service providers need to learn the development environment, such as the compiler and tool chains, which strongly depends on the gateway because M2M gateways are not standardized. Attention also needs to be given to CPU and memory usage of the aggregation program. Consequently, it is difficult for M2M service providers to write aggregation programs on M2M gateways. The second problem arises when changing the data aggregation logic on the firmware. To change data aggregation logic, the firmware has to be updated. However, firmware updates are risky because the gateway may not work if the updates fail.

We propose the application of a complex sensor data aggregator (CSDA) for M2M gateways. The CSDA enables M2M service providers to develop data aggregation logic without needing to program in C. The CSDA defines the framework that supports the sensor data aggregation. It splits the aggregation process into input, data processing, and output steps, whose behaviors are controlled by a configuration file. The aggregation logic can also be changed by simply updating the configuration. This paper presents the design of the CSDA and evaluates the proposed solution using an M2M gateway.

II. The Problem with sensor data aggregation

After categorizing sensor data aggregation for M2M gateways, the problems with writing the aggregation logic are described in the following sections.

A. Categorizing sensor data aggregation on M2M gateway

Before the sensor data aggregation is categorized, the process of inputting sensor data into the M2M gateway must be described. An overview of an M2M system is shown in Fig.1. Each sensor node in the sensor network is connected to sensor chips through a circuit. Sensor nodes are composed of a physical sensor network interface and a CPU, in which logic fetches a value from the sensor chips and sends that value to the sensor network with the node's ID. The node ID identifies that node in the sensor network. The arbitration ID for the CAN protocol and the IEEE address for Zigbee are examples of node IDs. Note that multiple sensors may be connected to the same sensor node. In this case, multiple values are packed with that node's ID. The means by which fetched values are packed with node IDs, i.e. short, long, little, or big endian, depend on the sensor node vendor. The M2M gateway and sensors are connected by various types of networks, and data are inputted into the gateway. For example, because the control of vehicles, industrial machinery, and medical equipment requires real-time communication, the CAN protocol, which has priority control features, is often used. When measuring temperature and humidity in a large area, such as a plant or building, the wireless protocol Zigbee

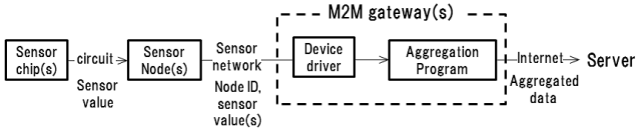

Fig. 1. Overview of M2M system

is used in order to remove the cost of installing wires. M2M gateways have physical interfaces and device drivers in order to communicate with the sensor network protocol. They receive data frames, which include the previously described node IDs and sensor values.

The sensor data aggregation process runs on an M2M gateway, using node IDs and sensor values as input. The process can be categorized into: (1) statistical calculations to reduce data size, (2) filtering to reduce data frequency, and (3) concatenation to reduce communication overhead.

1) Statistical calculations: Data size is reduced by calculating statistical values, such as average and histogram, from multiple data within a given time frame. For example, temperature sensor data for the cooling of water and oil within construction machinery are summarized to sum, average, minimum, maximum, and histogram. Then the processed data is sent to the server [12]. Similarly, engine speed and temperature sensor data within farm machinery are condensed to the average, minimum and maximum, and then sent to server [14]. Furthermore, statistical calculations can be performed multiple times, and multiple sensor data can be combined. When evaluating deviation from the stable state of construction machinery, variances are calculated for multiple temperatures and pressure sensor values. Then the variances are summed [13].

2) Filtering: Data from an M2M gateway can be reduced by filtering the low priority data. Filtering is subcategorized into two types: input filtering, using node IDs, and threshold filtering.

- Node ID filtering: To obtain important sensor values, the M2M gateway only takes the data frame, which includes the specific node ID. For example, since engine and transmission sensor values in construction machines are the ones used to detect failure, only these sensor values are gathered and sent to the server [12].
- Threshold filtering: The purpose of this filtering is to gather only the values that imply trouble. For example, when a statistical value calculated from a construction machine's sensor data exceeds some threshold, the calculated value is sent to the server. Otherwise, no data is sent [13].

3) Concatenation: When data is sent to the server, protocol header information is attached. To reduce the overhead, multiple sensor data are joined and sent at once. The timing of this depends on the type of data being collected and the reason for its collection. Referring again to construction machinery, for example, a summary of operation logs are sent daily, and the engine sensor data indicating failure is sent immediately [16].

B. Problems with sensor data aggregation using firmware programming

The sensor data aggregation process previously described is programmed into the firmware of the M2M gateway. However, there are problems in developing and updating programs in firmware.

1) Developing firmware: There are two problems with developing aggregation programs on firmware. The first problem concerns the CPU and memory resource of the M2M gateway. Cost and quality requirements for M2M gateways are strict because they often operate in extreme environments and the number of deployed devices is large. As a result, memory and CPU resources are much more constrained than enterprise servers. A CPU clock for an M2M gateway is usually around 200-600 MHz, and the RAM size is about 1-64 MB. Specifications of M2M gateways are typically not disclosed to the public, but an example of these types of CPU and memory resources can be found in an industrial communication board, whose CPU clock is 250 MHz and RAM size is 64 MB [17]. Because it is difficult to use higher-level language on this sort of hardware, the programming language is usually C. However, C is difficult to program, and its memory management is troublesome. Programmers also need to take into account memory and CPU usage. The second problem involves the development environment of the M2M gateway. SDKs for M2M gateways are different depending on the gateway device. The development process is typically composed of the following: writing aggregation logic in C on a PC, cross-compiling for the gateway, producing a firmware image, and transferring it to the M2M gateway's flash memory. All these processes are different depending on the M2M gateway because hardware, OS, and userland programs are not standardized.

2) Updating firmware: In order to modify aggregation logic after the M2M gateways are deployed, the firmware needs to be updated. However, there are difficulties in modifying programs, installing firmware, and recovering from a failure.

- Modifying programs: The C program needs to be modified in order to change the aggregation logic. There are problems similar to those previously described in the development of the C program. In addition, the C program requires repeat testing in order to avoid regression.
- Installing new firmware: After new firmware is developed, it must be installed on the devices. There are two ways of doing this. In the first method, field engineers perform the installation. Although this is the most reliable method, it is very expensive when the number of devices is 10,000 or 100,000. To reduce this human cost, the second method performs the updating through the network. However, there is still the problem of network cost in this scenario. Since firmware is usually implemented on read-only filesystems, entire filesystem images, which often exceed 10 MB, must be delivered via the mobile network, and the network cost for all the devices becomes impossible to ignore.
- Failure recovery: When writing firmware from the network fails due to a power outage during the update, the firmware is broken. To avoid this, a backup area, whose size is the same as the firmware's, is a necessary component of the flash ROM. In addition, the recovery procedure needs to be performed by an engineer.

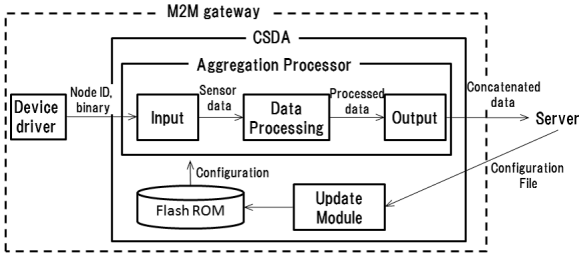

Fig. 2. Architecture of the CSDA

III. Proposal of Complex Sensor Data Aggregator

A. Basic design of the CSDA

As described in the previous section, there are problems with programming and updating sensor data aggregation logic in M2M gateway firmware. In this section, we propose a CSDA as a solution to these problems. An overview of the CSDA is shown in Fig. 2. The aggregation processor, whose behavior can be changed by a configuration file that removes programming, and the update module change the aggregation logic via the network. The design of the aggregator and configuration are described in the next sections.

1) Aggregation Processor: To cover the aggregation process categorized in section II-A, the aggregation processor handles sensor data in three flow steps: the input, data processing, and output steps. In each step, necessary logics are embedded into the aggregation processor in advance. These logics are defined by the configuration file. An overview of these steps is described as follows.

- Step1 Input: Node IDs and binary data, including sensor values, are passed from the device driver, and the node IDs are filtered. Then sensor values are extracted from the binary code.
- Step2 Data processing: Statistical calculations, such as average, sum, minimum, maximum, and histogram, are performed on the extracted sensor values. Threshold filtering is also performed on the sensor and calculated values. The two tasks, statistical calculation and threshold filtering, can be combined.
- Step3 Output: Multiple output data are produced in the data processing step. These data are concatenated and periodically sent to the server or immediately sent if it is urgent.

2) Configuration Language: The aggregation processor generates a single output from multiple data, as shown in Fig. 3. This figure displays the case in which engine temperature data is gathered when the engine load is heavy. Three sensor data are extracted during the input step. If the engine frequency exceeds a threshold, environmental temperature is subtracted

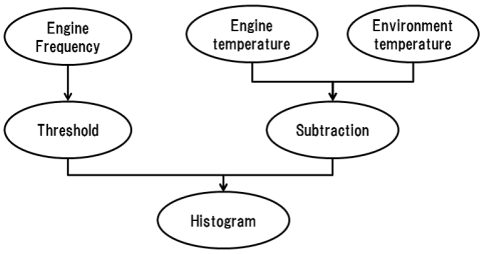

Fig. 3. An example of flow in aggregation processor

from the engine temperature to remove the environment's effect. Finally, a histogram for this difference is taken. The flow structure is a tree in which a single output is generated from multiple sensor data. Therefore, XML, which is also a tree structure, is adopted as the configuration language.

3) Update Module: This module downloads the configuration from the network and rewrites it. Failure recovery must be considered similar to firmware update. To achieve this, aggregation processors and update modules run as separate processes. The update is performed as follows:

- Step 1: The update module downloads the configuration file onto the RAM, and a copy of the old configuration file is created in the flash ROM as a backup.
- Step 2: The update module writes a new configuration to the flash ROM.
- Step 3: The update module restarts the aggregation processor
- Step 4: The aggregation processor loads new configuration from the flash ROM

Even if writing the configuration fails in Step 2 because of an unexpected power outage, the update can be resumed from Step 1, and the old configuration can be used, regardless of whether the resuming download fails. The flash ROM size required for the backup is at most the size of the configuration. Details of the aggregation processor's configuration language and its implementation on resource-constrained M2M gateways are described in the following subsections.

B. Configuration language

The configuration language is composed of the steps, input, data processing and output, corresponding to those of the aggregation processor. Following is an explanation of this configuration and some examples.

1) Input step: In this step, the node ID filtering and target sensor values are described. Fig. 4 shows an example of this configuration in the *extract* tag in line 3. Here, the *rawId* attribute refers to the node ID filtering. Only the input data, whose node ID is described in rawID, is processed. In this example, only the data frames with node IDs of 10 are processed. The *offset* and *len* attributes correspond with the data extracted from the data frame payload. Between zero and seven bits of payload are extracted and passed to the next step. In order to identify the extracted data, an *id* attribute is used in the *rule* tag. Here, the extracted data are identified by *50*.

```
1  <definition process="Selection">
2      <rule id="50">
3          <extract rawId="10" offset="0" len="8"/>
4      </rule>
5  </definition>
```

Fig. 4. Part of configuration for input step

```
1  <definition process="Calculation">
2      <rule id="100">
3          <if operator="greater" threshold="100">
4              <value>
5                  <seldata id="50"/>
6              </value>
7              <action>
8                  <calc method="histogram">
9                      <arg key="max" value="100"/>
10                     <arg key="min" value="0"/>
11                     <arg key="series" value="20"/>
12                     <calc method="subtraction">
13                         <seldata id="51" index="1"/>
14                         <seldata id="52" index="2"/>
15                     </calc>
16                 </calc>
17             </action>
18         </if>
19     </rule>
20 </definition>
```

Fig. 5. Part of the configuration for data processing step

```
1  <definition process="Sending">
2      <rule id="1">
3          <cat>
4              <header data="device001"/>
5              <caldata id="100"/>
6              <caldata id="101"/>
7          </cat>
8          <send method="periodic"
               address="192.168.1.150" port="8081">
9              <arg key="unit" value="minute"/>
10             <arg key="interval" value="30"/>
11         </send>
12     </rule>
13 </definition>
```

Fig. 6. Part of the configuration for output step

2) Data processing step: Statistical calculations, threshold filtering, and a combination of the two are described in this step. An example configuration is shown in Fig. 5. In order to refer data from the input step, a *seldata* tag is used. Line 13 assigns the data from the input step an identification of *51*. The statistical calculation is described by the *method* attribute in the *calc* tag. Lines 12-14 show the process for subtracting the data with IDs of 52 from the data with IDs of 51. Lines 8-16 demonstrate the process for generating a histogram from lines 12-14 with a maximum of 100, a minimum of 10, and a total of 20 in the series. Logics corresponding to *method* are embedded in the CSDA in advance. Threshold filtering is described by the *if* tag. Lines 3-6 can be written as: if data identified as *50* are greater than 100, then proceed to the next step. The resulting output data is identified in the *rule* tag and identified as *100*.

3) Output step: An example configuration for the output step is shown in Fig. 6. The process of concatenating and sending data to the server is described. In lines 4-7, data concatenated from the previous steps are identified as *100* and *101*, and the header data is labelled *device001*. In the *send* tag, the timing for sending concatenated data is written. In this example, data is sent every 30 minutes.

C. Implementation for M2M gateway

To save CPU resource and memory, the CSDA supports sensor data buffering and binary configuration.

1) Sensor data buffering: The maximum speed of a CAN network, which is widely used in industrial machines, is 1 Mbps, and the frame size for one datum is between 64 and 128 bits. Therefore, the frequency of input data is about 100 μs at peak time. If the CSDA processes data every 100 μs during its peak, there are function call overheads and referring configurations every 100 μs. Such overheads cause problems when a CPU's resource is constrained. To reduce the number of overheads, the CSDA buffers input data and processes multiple data. In addition, in order to limit memory usage the buffer is statically allocated. Real-time processing is lost through the buffering, but this is not an issue because the data processed by the CSDA is utilized in the server and the network delay between the server and the M2M gateway is more than 1 msec. The real-time feature is already lost here.

2) Binary configuration: It is difficult to parse an XML-based text configuration file on an M2M gateway because the XML parser usually consumes memory resource. To save memory, an XML-based configuration is encoded into a binary format in advance. The binary format can be interpreted from the top in order to prevent the performance loss that occurs because of the random memory access. For example, the configuration execution sequence in Fig. 5 can be found in lines 3, 12, and 8. In binary format, it is sorted according to the execution sequence.

IV. Evaluation

In the evaluation, the developmental process of aggregation logic on an M2M gateway is evaluated first. Then its performance is measured against an experimental environment similar to an M2M gateway.

A. Development of the aggregation process

1) Experiment environment: The productivity of C and CSDA when developing an aggregation process is compared. The following aggregation process is used for evaluation.

- Input process step: Take two kinds of sensor value for Sensor A and Sensor B
- Data processing step: When the 30 second average value for Sensor A is greater than a threshold, create histogram for Sensor B.
- Output step: The histogram is sent once each day.

In order to compare the amount of memory required by the two methods, the number of steps is calculated for the C code and the number of tag pairs is counted for the CSDA configuration.

2) Results and considerations: The number of steps required in the C program is 575, and the number of tag pairs in the CSDA configuration is 36. The semantics of C and XML are different, but the amount of configuration is much less than in the C code. In addition, there are following three difficulties in writing C language code.

First is bit operation. In the input step, sensor data is extracted from the binary frame. In order to handle binary communication, bit operations, which are difficult to read, are used, as shown Fig. 7. Eight valuables and three arrays are used in a mere three lines. On the other hand, data extraction can be described in one simple line using the CSDA. Second difficulty is memory management. As is well known, failure to handle C arrays causes many problems. If there is a bug in the boundary handling, the memory may be destroyed, and this weakness can be exploited in a buffer overflow attack. The CSDA hides such memory management. Third one is thread management: The thread needs to be separated for the input process and data processing in order to enable data receiving during the data processing. Thread programming depends on the OS, and programmers need to learn this programming. Moreover, the shared resource has to be handled carefully because failure can lead to resource destruction or a deadlock. The CSDA also hides this.

```
* Part of C code
for(idx2 = 0; idx2 < byteLength ;idx2++){
  if(offsetRem > rightShift){
       output[idx2] = (data[idx3]>>rightShift)
       & forAnd[leftShift];
* CSDA configuration
 <extract rawId="10" offset="0" len="8"/>
```

Fig. 7. Comparison of C and CSDA at input step

TABLE I. Embedded system used in the evaluation

	Specification
CPU	Freescale i.MX25(ARM926EJ-S) 400 MHz
RAM	LPDDR SDRAM 64 MB
Sensor network interface	ISO11898 compliant CAN interface
OS	Linux 2.6.26

B. Performance evaluation

1) Experimental setup: CPU clocks for M2M gateways are often around 200-600 MHz in size, and RAM sizes are around 1-64 MB, as was stated in section II-B1. We used the Armadillo 420 [18] as an evaluation device because its CPU power and RAM size are similar to these above specifications and its SDK is available to the public. Its specifications are shown in Table I. The CAN was selected as a sensor network because it is widely used in industrial devices.

To ensure the CSDA works during heavy sensor network traffic, data are sent to the evaluation device via the CAN at a rate of 1 Mbps, which is the maximum rate of the CAN, then the CPU and memory usage are measured. The detailed conditions of the sensor data and configuration of the CSDA are described as follows.

In order to simulate heavy traffic, data from 200 sensors are created. This number is large enough because the number of electronic control units (ECUs) in a vehicle is about 100 at most [19]. To increase the amount of sensor data as much as possible, multiple sensor data are packed in one CAN packet. A CAN frame is composed of a 64-bit header and 0-64 bits of data. Four 16-bit random sensor data are packed in the data field. Since there are 200 sensors, 50 kinds of CAN data frames are prepared.

As in a typical data aggregation process, threshold filtering and histogram are combined in the configuration of CSDA.

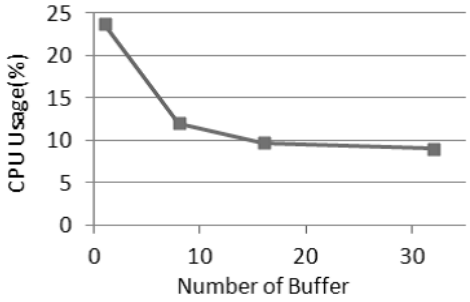

Fig. 8. CPU usage of CSDA

During the input process, four kinds of sensor values are extracted from each of the 50 kinds of CAN data frames and saved in the buffer. During the data processing step, when one sensor data exceeds a threshold, 199 histograms are created from the remaining 199 sensor data. The output step is omitted in this evaluation because the frequency is assumed to be low, minutes or hours, and its effect on the resource consumption is also low.

2) Result: CPU usage results are shown in Fig. 8. CPU usage decreases as the number of buffers increases. A 16-bit sensor value is stored in each buffer. When the number of buffers exceeds 16, CPU usage is less than 10%. In this evaluation environment, the CPU clock is 400 MHz. If the clock is 200 MHz, which is the lowest possible value for an M2M gateway, then it would appear that the usage is less than 20% at peak time. The CSDA's RAM usage is 128 Kbytes. The number of buffers is 16, where the improved CPU usage can be seen in Fig. 8. RAM usage is less than 1%. Even if the RAM size is 1 MB, which is the least amount possible for an M2M gateway, the RAM consumption is less than 13%.

V. Related works

A data stream management system (DSMS) [20] partially removes data aggregation coding. It processes the input data stream based on the rule called query. However, because it is not originally intended for sensor data aggregation, the rule language does not cover the input and output steps. It is primarily intended for real-time processes, so it requires enough CPU and memory resource. There is a DSMS for resource-constrained devices [21], but its rules are static and a firmware update is necessary to change them. AirSenseWare [22] processes data using sensor nodes and a rule-based server. Its focus is the framework for distributed data processing and not aggregation on a gateway.

VI. Conclusion

To reduce the server load and communication cost of M2M systems, sensor data are aggregated in an M2M gateway. The aggregation logic is usually programmed in C rather than higher-level programming languages because the CPU and memory resource is constrained. However, there are difficulties with programming in C and with updating the programs. Data aggregation methods have been categorized, and the proposed complex sensor data aggregator (CSDA) enables sensor data aggregation in M2M gateways without the need for programming. This CSDA supports the categorized data aggregation methods in three steps: the input, data processing, and output steps. In each step, behavior is configured using XML-based rules. Experimental results show that developing CSDA configurations is much easier than programming in C because the configuration amount is less than 10% of the comparable C code. In addition, the performance evaluation shows it works with M2M gateways because the CPU usage with the CSDA is less than 10%, even given a heavy load, and its memory consumption is 128 Kbytes.

References

[1] Komatsu: Komtrax: http://www.komatsuamerica.com/komtrax

[2] Hitachi Construction Machinery: Global-eService: http://www.hitachi-c-m.com/global/businesses/products/global_e-service.html

[3] Yanmmar: Yanmar's Advanced Technology Gives Customers a SmartAssist, 2013: http://yanmar.com/news/contents/105278.php

[4] Zigbee Alliance, "Interconnecting Zigbee & M2M Networks" http://docbox.etsi.org/workshop/2011/201110_m2mworkshop/03_m2mcooperation/zigbee_taylor.pdf

[5] CiA : Controller Area Network http://www.can-cia.org/index.php?id=can

[6] ETSI TS 102 690 V1.1.1, "Machine-to-Machine communications(M2M) functional architecture," 2011

[7] MQtt: http://mqtt.org

[8] Z. Shelby, Sensinode and K. Hartke, "Request for comments 7252:Constrained Application Protocol(CoAP)," IETF, 2014

[9] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight Internet protocols for web enablement of sensors using constrained gateway devices", In proceedings of 2013 International Conference on Computing, Networking and Communications(ICNC2013), pp.334-340, 2013

[10] S. Madden, M. Franklin, J. Hellerstein, W. Hong: "TinyDB:An Acquisitional Query Processing System for Sensor Networks, ACM Transactions on Database Systems," Vol.30, No.1, 2005

[11] A. Boulis, C. Han, R. Shea, M. Srivastava, "SensorWare:Programming sensor network beyond code update and querying," Pervasive and Mobile Computing, Vol.3, No.4, pp.386-412, 2007

[12] T. Murakami, T. Saigo, Y. Ohkura, Y. Okawa and T. Taninaga, "Development of Vehicle Health Monitoring System(VHMS/WebCARE) for Large-Sized Construction Machine," Komatsu Tech Rep, Vol.48 No.150, pp.15-21, 2003

[13] Hitachi Construction Machinery Co., Ltd, "Device for collection construction machine operation data," WIPO Patent, WO2013077309 A1, 2013

[14] Yanmar Co., Ltd, "Remote monitoring terminal device for traveling work machine or ship," WIPO patent, WO2013080712 A1, 2013

[15] BroadBand Forum, "TR-069 Issue 1 Amendment 2": http://www.broadband-forum.org/technical/download/TR-069_Amendment-2.pdf

[16] Y. Takishita, K. Murakami, K. Seki and K. Morishita, "Application of ICT to Lifecycle Support for Construction Machinery," Hitachi Review, Vol.62, No.2, pp.107-112, 2013

[17] Communication module for industrial devices, http://www.hitachi-ul.co.jp/system/cmodule/index.html(In Japanese)

[18] Atmark Techno, Inc., Armadillo-420, http://armadillo.atmark-techno.com/armadillo-420

[19] C. Ebert and C. Jones, "Embedded Software: Facts, Figures, and Future," Computer, IEEE Computer Society Press, Vol.42 Issue 4, pp.42-52, 2009

[20] The STREAM Group, STREAM: The Stanford Stream Data Manager IEEE Data Engineering Bulletin, 2003

[21] S. Katsunuma, S. Honda, K. Sato and H. Tanaka, "The Static Scheduling Method in Data Stream Management for Automotive Embedded Systems," IPSJ Journal Database, Vol.5. No.3, pp.36-50, 2012

[22] K. Muro, T. Urano, T. Odaka and K. Suzuki, "AirsenseWare: Sensor-Network Middleware for Information Sharing," In proceedings of 3rd International Conference on Intelligent Sensors, Sensor Networks and Information 2007(ISSNIP2007), pp.497-502, 2007