Hindawi
Journal of Sensors
Volume 2019, Article ID 5107457, 9 pages
https://doi.org/10.1155/2019/5107457

Research Article

Management of IoT Sensor Data Using a Fog Computing Node

Gunjae Yoon,[1] **Donghwa Choi**,[2] **Jeongjin Lee**,[2] **and Hoon Choi**[2]

[1]*Gurum Networks, Seoul, Republic of Korea*
[2]*Department of Computer Science & Engineering, Chungnam National University, Daejeon, Republic of Korea*

Correspondence should be addressed to Hoon Choi; hc@cnu.ac.kr

Received 14 September 2018; Revised 10 December 2018; Accepted 24 December 2018; Published 19 February 2019

Academic Editor: Grigore Stamatescu

As IoT systems spread, transmissions of all data from various sensing devices to a remote OM (Operation and Management) server through the Internet can lead to many problems, such as an explosion of network traffic and delayed responses to data. Fog computing is a good means of resolving these problems in an IoT system environment. In this paper, a management method for sensor data in a fog computing node is proposed. The monitoring node monitors data from sensor devices using a data pattern from the OM server, which dynamically generates and updates the pattern. The monitoring node reports only the data beyond the normal range of the pattern to the OM server rather than sending all data to the OM server. The monitoring node can control the operations of sensor devices remotely according to the requests of the OM server.

1. Introduction

Currently, the Internet of Things (IoT) [1] is used in intelligent homes, smart buildings, factory automation systems, intelligent transportation systems, and autonomous car management systems. An IoT system consists of a number of sensors, actuators, and computing nodes. Generally, these leaf-node devices do not have enough computing resources to supply intelligent services. Therefore, one or more servers must undertake the processing, storing, and classifying of the data. Though a server or a cluster of servers is able to process all data generated by leaf-node devices, network problems such as traffic explosions or delayed transmissions can occur if the number of IoT systems increases. To alleviate these problems, the concept of fog computing [2–5] has been introduced. Fog computing does not store massive amounts of data on a large data server; instead, the data is stored near the data source. Fog computing has the advantage of mitigating network traffic that is concentrated on the server.

Traditional fog computing nodes simply performed filtered data transmission using a filtering criterion provided by a human administrator. A smart gateway [6] reduces the communication overhead of the core network and reduces the burden on the cloud. The human administrator of the smart gateway must change the filtering criterion when necessary. On the other hand, the system proposed in this paper automatically establishes and updates the filtering criterion without the administrator's intervention. Adaptive monitoring (AdaM) [7] refers to a framework that reduces the volume of generated data as well as the network traffic between IoT devices and data management endpoints. AdaM uses both adaptive sampling and adaptive filtering algorithms and dynamically adjusts the monitoring intensity and the amount of data disseminated through the network based on the current metric evolution. Unlike the monitoring node proposed in this paper, AdaM runs on an IoT device itself. Therefore, it can be applied to IoT nodes with certain types of computing hardware. The cognitive IoT gateway [8] is a type of fog node. The cognitive IoT gateway reduces the network traffic between IoT devices and cloud servers by migrating server applications to the gateway. It uses a machine learning algorithm to determine where to execute IoT applications between a fog node and a cloud server based on hardware information pertaining to the gateway, such as the CPU utilization, memory usage, and network bandwidth values. The cognitive IoT gateway differs from the proposed monitoring node in that it deals with the deployment of applications. Research from another point of view suggests a multi-core-based edge server type of architecture to improve the performance of the edge (fog) computing. An

edge server which corresponds to a fog node is a server with limited capabilities, though it must handle large amounts of data in real time. Another study suggested the use of many core systems to offer high-performance computing and caching services for these types of limited servers [9]. Using many-core systems for each edge server can be costly in a smaller IoT system. An edge-side company storage framework was also proposed to store IoT data [10]. Edge servers also usually have limited resources and small amounts of storage. When an edge server lacks a storage space, the edge-side company storage framework shares the storage of other near-edge servers to avoid long delays when offloading data to a cloud server. While the aforementioned study [10] concerns the efficient management of the storage space, this paper focuses on the data filtering issue.

A better means of reducing network traffic and the server workloads in an IoT system environment was proposed by the authors [11]. The filtering criteria in that study are automatically generated by the OM server using raw data from the sensor devices. The functional architecture of the method is described but detailed schemes for the collection, management, and transmission of sensor data are not included.

This paper expands on the earlier work [11] and includes a detailed communication scheme between monitoring nodes and sensor devices, a method for the adaptive generation of data patterns and the monitoring of sensor data, and a method by which abnormal data are transmitted to an OM server in an IoT system environment. The implementation of a prototype of an IoT system is undertaken to demonstrate the feasibility of the proposed method and to measure the response time of the fog node.

This paper proceeds as follows. Section 2 briefly introduces the concept of fog computing and explains the proposed operational process of our system. Section 3 describes the functions of the monitoring node to support the operations presented in Section 2. After the implementation of the prototype, the response time measurements and a discussion of the traffic reductions which can be expected when using this system are presented in Section 4. The concluding remarks are given in Section 5.

2. Fog Computing

2.1. Architecture. Figure 1 shows a typical architecture of a fog computing system, which has three hierarchical systems. The monitoring nodes, also known as fog nodes, connect several IoT systems to the OM server. A monitoring node is a proxy of the OM server, and it manages the IoT system. Sensed data generated by IoT devices are transmitted to and processed by the monitoring node instead of the OM server. The monitoring node manages received data and reports only abnormal situations which arise to the OM server. As a result, the network traffic and workload of the OM server are reduced.

2.2. Proposed Operational Process. The aim of this study is to provide the following operations to the IoT system using the monitoring nodes of fog computing and the OM server. (1) When a new sensor device is connected to the monitoring node, the monitoring node recognizes it and reports it to the OM server in real time. (2) The sensor device transmits data to the connected monitoring node. The monitoring node stores the data received from the sensor device. (3) When the monitoring node receives the data in step (2), it also checks whether a data pattern exists for this device. If a data pattern does not exist, the monitoring node transmits the received data to the OM server to generate a data pattern. (4) The OM server acquires the data from the monitoring node and generates a pattern for this specific sensor device, which is a collection of sets of values and generation times of data from the device. This pattern informs us of fluctuations of normal data values with respect to the time of day. (5) When a pattern is generated with sufficient data, such as data collected over several hours, the OM server sends the data pattern to the monitoring node. These operations can be called the pattern generation process, and they are depicted in Figure 2. We assume that an IoT device is connected to only one monitoring node. One monitoring node may have multiple IoT devices, and one OM server may have multiple monitoring nodes.

After the data pattern is generated, monitoring operations can take place. (1) Sensor devices transmit data to the monitoring node. The monitoring node stores the received data. (2) The monitoring node also compares the IoT data with the data patterns. (3) If the value lies within the acceptable deviation range, steps (1) and (2) are repeated. This range may be set by the administrator. For example, a 5% range means that ±5% of the difference between the received data value and the pattern value is accepted as a normal case. (4) If the value is beyond the normal data pattern range, the monitoring node records this in a log file. The monitoring node takes a proper action to control this, and it also reports this abnormal value to the OM server with the corresponding time information. (5) The OM server recognizing the abnormal situation can analyze the data and alert the administrator or may reflect the data when updating the corresponding data pattern. (6) The server may request the monitoring node of the stored data in a specific time interval for operation and management purposes.

The advantages of the proposed method compared to other fog computing mechanisms include the fact that data filtering is performed on a per-device basis given a data pattern for each sensor device. Unlike other mechanisms, our method does not require any human effort to provide data filtering criteria (normal data patterns), and data patterns are created systematically and provided by the server. The data patterns can dynamically adapt to appropriate situations; for example, different versions of data patterns can be made with respect to the time of day or considering seasonal or climate changes.

3. Management of Sensor Data in a Monitoring Node

For the operations of the previous section, the monitoring node was designed to have the functions shown in Figure 3.

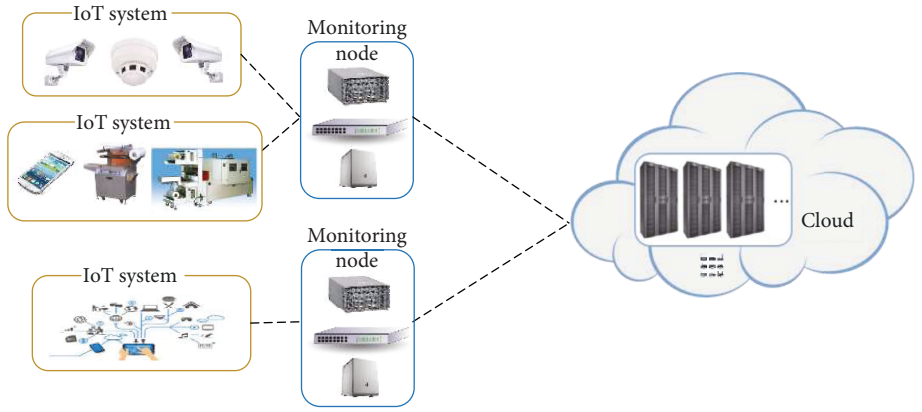

Figure 1: Fog computing architecture.

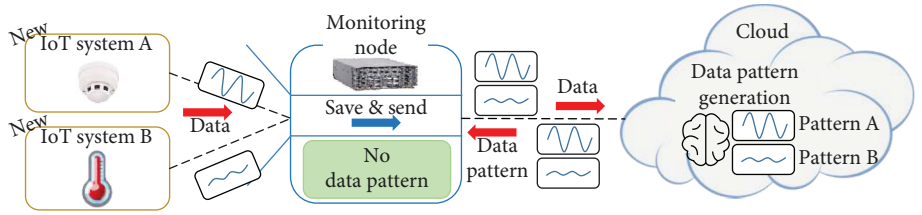

Figure 2: Data pattern generation process.

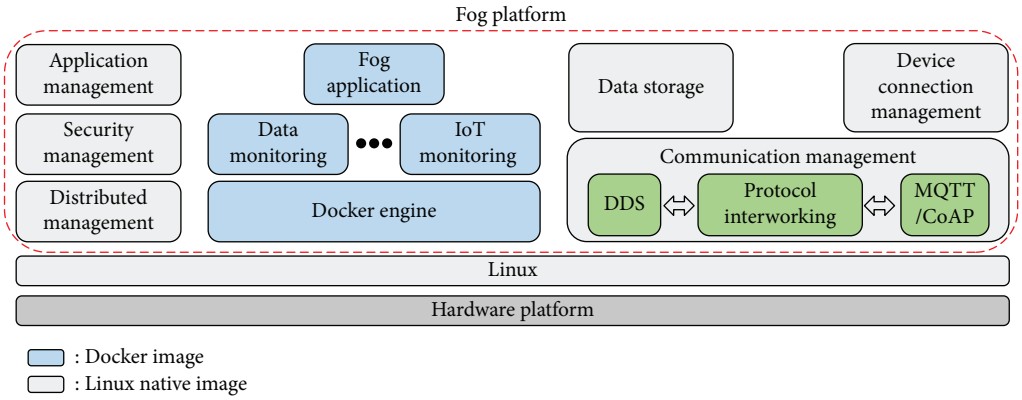

Figure 3: Functional architecture of the monitoring node.

3.1. Data Storage. The sensor devices of an IoT system can generate data frequently or periodically. The data storage component stores the data in a compressed form to reduce the storage capacity. When unexpected or abnormal data is received, the data storage component must keep logs for system management and failure tracking.

The data storage component requires an efficient data management scheme to prevent performance degradations. The monitoring node manages data in a block by block manner in order to improve the efficiency of data search and transmission processes. A data set includes data values, identifiers of the device and sensor along with a timestamp to distinguish which IoT device generated the data as shown in Figure 4. The data storage component also responds to requests from the OM server. When the OM server requests specific data or data generated during a given time interval, this component searches for and retrieves the data from storage and sends the data to the server.

3.2. Communication Management. This component transmits data from/to IoT devices to/from the OM server. Each IoT system can use different communication protocols, such as Modbus [12], MQTT [13], or CoAP [14]. IoT systems cannot be accessed by an OM server unless this protocol difference is resolved. By converting legacy communication protocols between IoT devices and the monitoring node to a common standard Internet protocol between the monitoring nodes and OM servers, the monitoring node can connect different IoT environments and therefore achieve scalability. The Data Distribution Service (DDS) of the OMG (Object

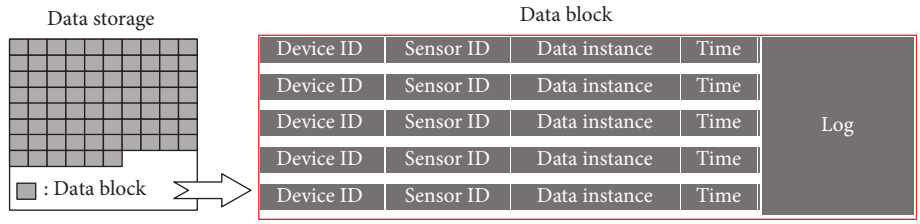

Figure 4: Data block structure.

Management Group) [15, 16] can be used as the common Internet-side protocol.

We explain the communication management component with respect to the Modbus protocol. The Modbus protocol is a master/slave type communication protocol. The master sends a request to the slave and waits for a response, and the slave sends a response to the master (Figure 5). Typically, Modbus includes the Modbus RTU based on serial communication and the Modbus TCP based on socket communication. We used the Modbus TCP in this research.

The Modbus TCP protocol sets up necessary information (e.g., IP address and port number) for socket communication and attempts to make a connection from the master to the slave using the information.

After a connection is made, the master requests data from the slave according to the Modbus frame format shown in Figure 6, and the slave sends a response. Modbus defines six function codes, as shown in Table 1, for requests to the slave from the master.

We set the monitoring node as the master and each IoT device as a slave. Figure 7 shows the message exchange diagram. The monitoring node requests data from a slave by sending a message with function code 4 to the IoT device, and the IoT device transmits a response matching the requested function code. The data in the reply from the IoT system are sent to the data storage component via IPC (Inter-Process Communication).

The communication management component uses DDS to communicate with the OM server. Therefore, the monitoring node converts legacy protocols to DDS to send IoT data to the OM server. To support high flexibility in protocol conversions and adaptations to monitoring data updates, the monitoring node applies DDS-XTYPES, which are extensible types for DDS.

DDS represents data in a structure known as a topic. Figure 8 shows the structure of a topic. Topics are composed of a name and a type. A topic name refers to the identification of data in a DDS domain network. The topic type information also includes the type name and its data structure. The type information must be defined before creating topics.

OMG defined the extensible types for DDS specifications to make the topic creation and definition processes flexible. Figure 9 shows the topic representations of DDS-XTYPES. DDS-XTYPES specifies how to express the structure and the attributes of a DDS topic. When DDS undertakes the endpoint discovery process, the participating DomainParticipant can forward the structure information, expressed in the DDS-XTYPES form, of the endpoint discovery messages.

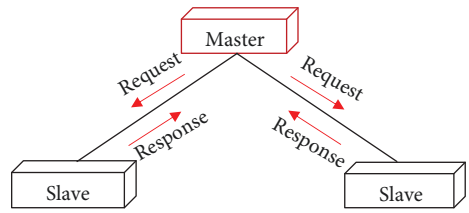

Figure 5: Communication topology of Modbus.

When DDS DomainParticipant receives endpoint discovery messages including DDS-XTYPES information, the DomainParticipant can understand and use the topic even if the information of the topic was unknown to the DomainParticipant.

The proposed communication technique can undertake flexible communication and monitoring using DDS-XTYPES. In order to apply DDS-XTYPES, the monitoring node must know the type of data transmitted by a legacy protocol. The monitoring node has IDL (Interface Definition Language) files representing these data structures, and the node converts IoT data to a DDS topic using the IDL information. These IDL files are written by device manufacturers and are downloaded from the OM server. When the IDL information or topics are updated, these changes are propagated through DDS's endpoint discovery processes with DDS-XTYPES. Examples of such cases include those when existing monitoring information is changed or when new monitoring information is added by a new sensor node or a monitoring system. Due to these DDS propagation processes, the monitoring nodes and OM servers can adapt to these changes during runtime without any modification or compilation processes.

3.3. IoT Monitoring. This component monitors the connection states and operational states of connected IoT devices. An IoT device may fail or may sometimes be detached. If the communication channel between the device and the monitoring node is unreliable, communication may fail. The monitoring node continuously checks the state of the channel and the device when possible.

The monitoring node attempts to collect the device information of each IoT device, such as the manufacturer/model name, hardware information, address, and other parameters. If this profile information is embedded in the device and if the communication protocol between the device and the monitoring node is capable of delivering the profile, the monitoring node can obtain the device information. For example, when a new IoT device is plugged into the IoT

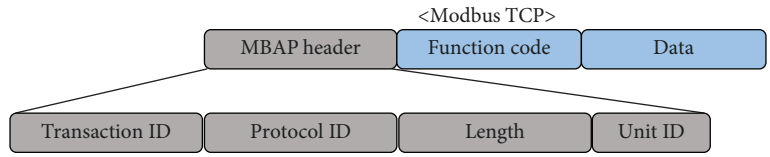

Figure 6: Modbus TCP frame structure.

Table 1: Modbus function code.

Function type		Function name	Function code
Data access	Bit access	Read discrete inputs	2
		Read coils	1
		Write single coil	5
	16-bit (word) access	Read input register	4
		Read holding registers	3
		Write single register	6

system, this component of the monitoring node may automatically detect the device type and status. It then reports this information to the OM server in real time.

The device is also controllable through the monitoring node if necessary. This IoT monitoring component controls the IoT device by transferring messages with function code 5 and the device ID and sensor ID to the slave, as described in Figure 6. For example, if the monitoring node recognizes that the value of a temperature sensor exceeds a certain threshold after reading the temperature, the monitoring node turns on the air conditioner by transmitting the message with the function code "Write" and changing the registry value controlling the air conditioning power of the IoT device.

3.4. Data Monitoring. This component analyzes the collected data. The monitoring node receives the data pattern from the OM server to judge the normality of the received data. The data storage component saves the data received from a sensor. The values of the received data are then compared with the data pattern. If the data is not in the normal range, the monitoring node records this in a log file and reports the occurrence of abnormal data to the OM server. An example of such a pattern is given in Table 2.

The OM server generates a data pattern for the monitoring node to detect abnormal data from IoT devices. There are many technologies for generating patterns, but a simple primitive method is used as the initial first trial. As shown in Figure 10, the OM server divides 24 hours into 144 sections of ten-minute periods and calculates the average of the sensor data values collected from the monitoring node during each time section. The OM server maintains the sequence of the average values as the data pattern for a specific IoT device.

The OM server sends the generated data pattern to the monitoring server by putting in a DDS topic, as shown in Figure 11.

The OM server must update the data pattern to adapt to change of normal values due to environmental issues, such as seasonality changes. The monitoring node reports to the OM server when a value beyond the acceptable deviation range is received or when data are continuously received from a device whose values are close to the maximum/minimum range of the data pattern.

Figure 12 is a diagram showing a data pattern of a sensor device. Data values within the range of section (a) are regarded as normal; hence, the monitoring node stores the data in this case without further action. Data values in section (b) are also regarded normal, and the monitoring node stores the data. In this case, the frequency of data of section (b) is also counted, and the monitoring node requests an update of the data pattern if the frequency is high. Because the data in section (c) exceeds the normal range, the monitoring node transmits the data to the OM server.

The OM server is able to reconfigure the monitoring of IoT systems by updating the data patterns of IoT devices and downloading them onto the monitoring node. The monitoring node uses the Docker platform, which allows the node to install new software onto the monitoring node. For example, the monitoring node can download and install a new algorithm to detect abnormal data from the OM server.

4. Performance Evaluation

We implemented a prototype system to demonstrate the feasibility of the monitoring node in this study. The middle part in Figure 13 represents the sensors and an actuator. A DHT-22 temperature-humidity sensor, an IR sensor, an MQ-135 gas sensor, and a small fan are used with Arduino. We used an Apple MacBook Pro 2017 (Intel dual-core i5-7360u, 3.1 GHz) for the monitoring node and a Gigabyte AERO (i7-7700HQ, 2.8 GHz, Ubuntu 18.04) for the OM server.

Advantages of the proposed monitoring node include the fact that the monitoring node can control IoT devices more quickly than the OM server and the monitoring node can reduce network traffic.

First, we show how quickly the monitoring node controls IoT devices. We positioned a heat source near the

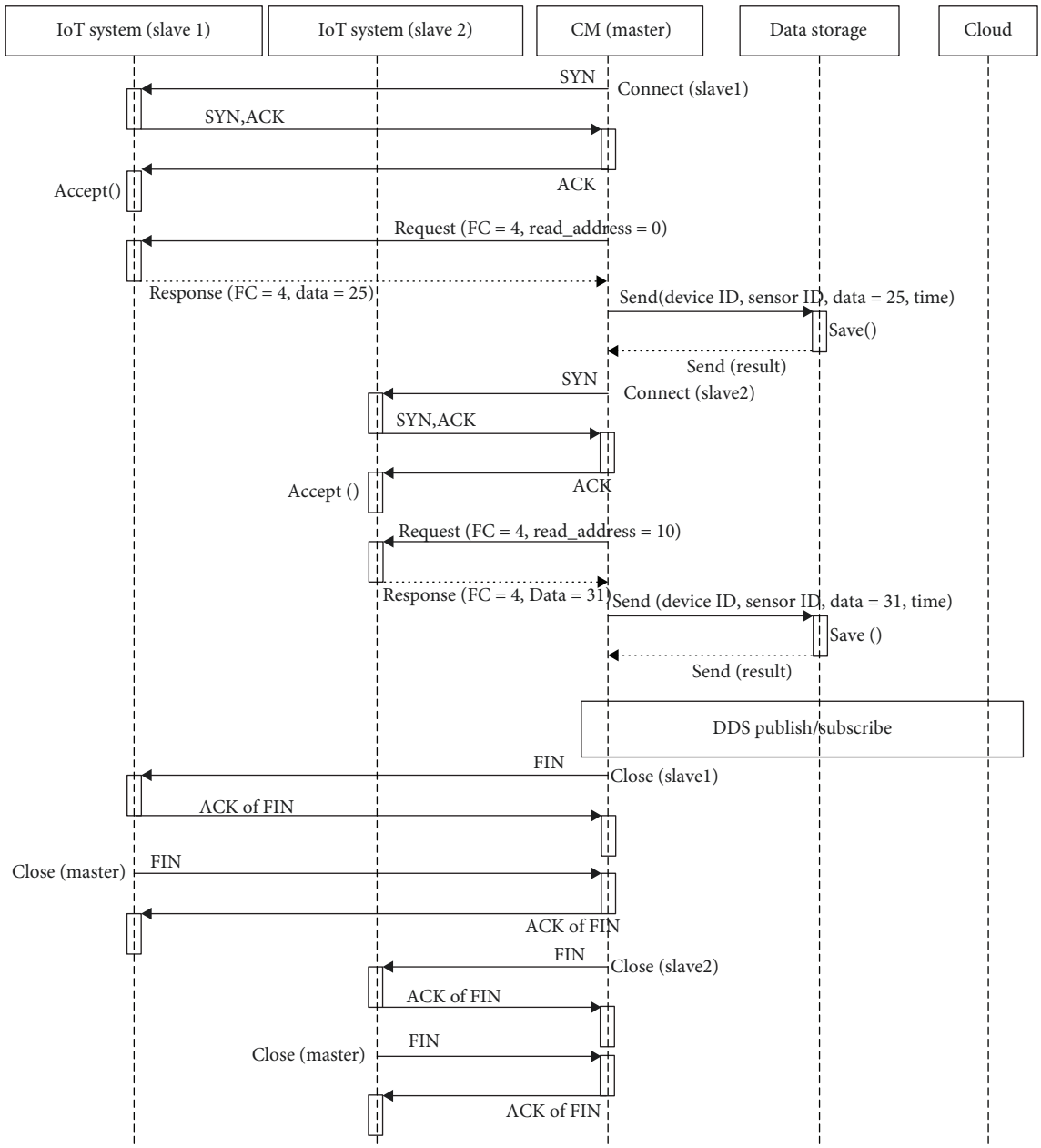

Figure 7: Modbus connection configuration and information transfer procedure. CM: communication management; FC: function code.

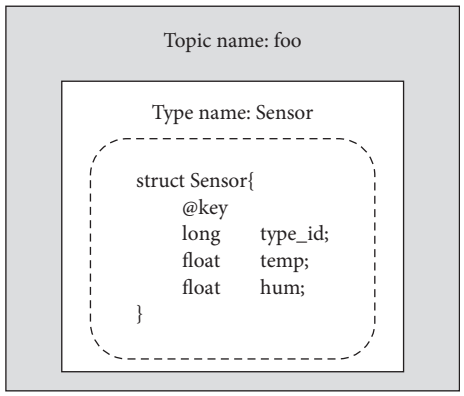

Figure 8: DDS type composition.

temperature-humidity sensor on purpose and measured the response time, i.e., the delay from the time the IoT device, Arduino, sends the measured high-temperature value to the monitoring node up until the time the Arduino device receives a control message to turn on the fan from the monitoring node. The average of 36 measurements was found to be 30.220 milliseconds. If the monitoring node is not used, the sensed value is sent to the remote OM server and the server sends back a control message. The server's response time will approximate the monitoring node's response time plus the network round-trip delay. The network delay to the OM server depends on the distance and the number of router hops from the IoT system to the OM server. In this experiment, we used

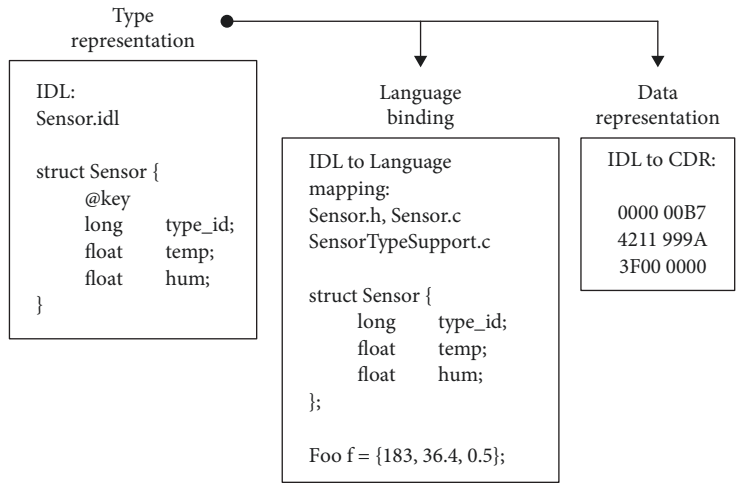

Figure 9: Topic representation of DDS-XTYPES.

Table 2: An example of a data pattern for an IoT device.

Date	Time	Mean normal value	Acceptable deviation
2018-01-01	12:00:00	20.05	5%
2018-01-01	12:00:30	20.05	5%
2018-01-01	12:01:00	21.00	5%
2018-01-01	12:01:30	21.00	5%
:	:	:	:

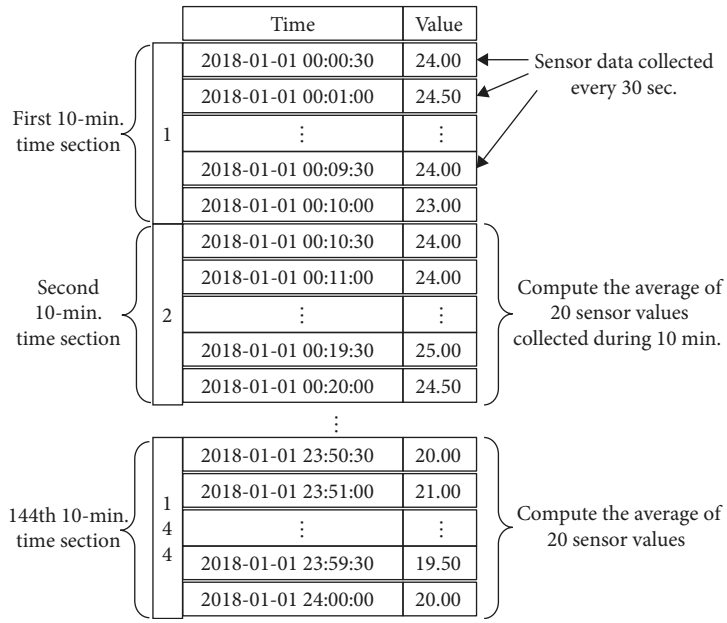

Figure 10: 144 sections of data.

the average roundtrip delay of the North America region, which is 36.737 milliseconds [17].

As shown in Table 3, the response time of the monitoring node is shorter than that of the cloud server. If we consider the heavy workload at the server due to the considerable amount of traffic from many IoT devices, the server's response may become even slower.

We performed a simple evaluation to depict the second advantage of the monitoring node. Figure 14 shows the shapes of the network traffic changes. From times 0 to T1,

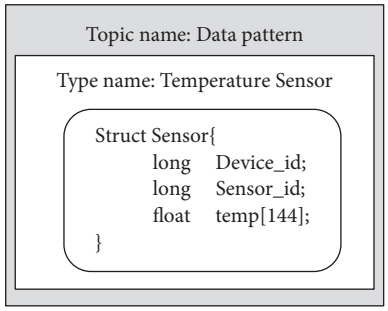

Figure 11: Topic for a data pattern.

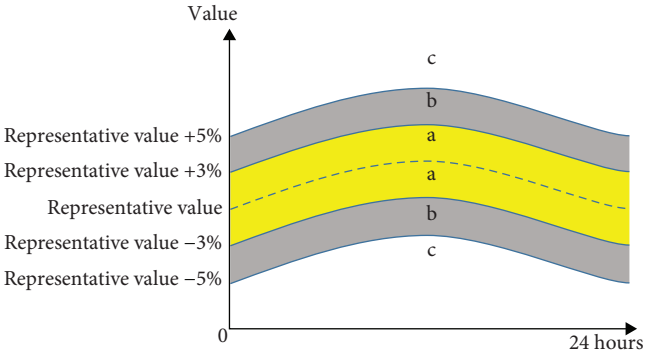

Figure 12: Comparison of received data with the data pattern.

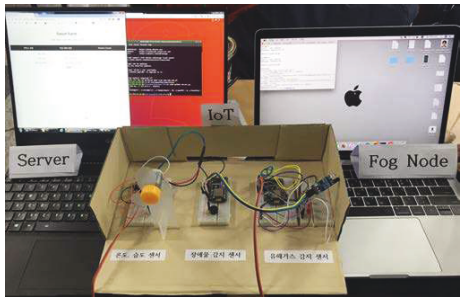

Figure 13: A prototype implementation of a monitoring node and an IoT system.

the monitoring node discovers a new sensor device and establishes a connection with it. Upon reaching time T1, the discovering and connecting processes end. From times T1 to T2, the monitoring node transmits all of the data to the OM server on the Internet because the monitoring node does not yet have a data pattern for this device. Therefore, the network traffic reaches its maximum level, and the workload of the OM server is high. During this period, the OM server learns and generates a pattern from the received data. At time T2, the OM server completes the generation of the data pattern and sends it to the monitoring node. At time T3, the monitoring node receives and installs the data pattern, after which it begins to detect an abnormality. The monitoring node reports only abnormal data to the OM server after T3.

Table 3: Comparison of response times (unit: ms).

Average response time of the monitoring node	30.220
Average response time of the cloud server	30.220 + 36.737 = 66.957

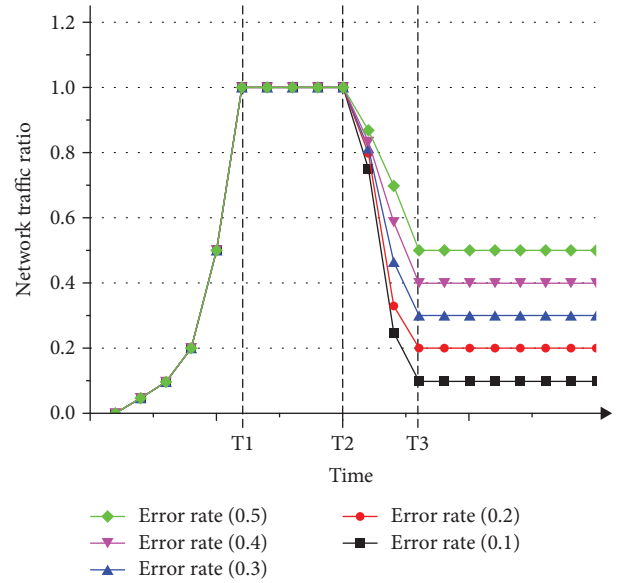

Figure 14: Network traffic reduction over time.

Network traffic arises in proportion to the assumed error rate, which is the rate of the occurrence of abnormal data from some sensor device. We simulated different error rates and calculated the corresponding traffic rates. If we assume an error rate of 0.5, half of the received data is normal and is not sent to the OM server after T3. With an error rate of 0.2, we do not need to send 80% of the data to the OM server. Therefore, the proposed method can decrease the network traffic considerably. In a typical IoT system without a monitoring node, all of the sensor data are directly sent to the OM server; the network traffic ratio will always be 1 after T1.

The values of traffic rates during 0~T1 and T2~T3 were set to certain intermediate values simply to show a gradual increase/decrease during these transient time spans. The purpose of this graph is to show the differences in the traffic rates during the T1~T2 time span and the time span after T3.

5. Conclusion

The purpose of the monitoring node in this paper is to control IoT devices quickly and to reduce network traffic and the server's workload in the IoT system environment. The monitoring of the normality of data from each sensor device is based on the corresponding data pattern provided by the OM server. The monitoring node with the data patterns reports only the data outside of the normal data range to the OM server rather than sending all the data to the OM server.

Generating a proper data pattern is an important subject. The OM server adaptively generates data patterns and provides them to the monitoring nodes for each sensor device. Data patterns should be generated from a large set of data collected by each sensor device. Though we described a simple method of pattern generation, we can improve it further. It can be replaced by another method, such as one based on a deep learning technique.

Other future research subjects may include a quick method to detect abnormal sensor values at the monitoring node. The capability of the monitoring node can be further extended to search and download the profile information of IoT devices from an external server such as the OM server or the manufacturer's server. Performance evaluations of the fog computing with IoT systems on a larger scale are also challenging.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

The funding sponsors had no role in the design of the study, in the collection, analyses, or interpretation of the data, in the writing of the manuscript, and in the decision to publish the results.

Conflicts of Interest

The authors declare no conflict of interest.

Authors' Contributions

G. Yoon and D. Choi equally contributed to this work.

Acknowledgments

This work was supported by the National Research Foundation of Korea grant funded by the Korean government, MSIT (No. NRF-2017R1D1A1B03029262). D. Choi was partly supported by Korea Electric Power Corporation (Grant number: R18XA05).

References

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

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM*, pp. 13–16, Helsinki, Finland, 2012.

[3] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *Wireless Communications and Mobile Computing*, vol. 2018, Article ID 7476201, 23 pages, 2018.

[4] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. S. Goren, and C. Mahmoudi, "Fog computing conceptual model," Tech. Rep. 500-325, NIST (National Institute of Standards and Technology), 2018, November 2018, https://www.nist.gov/publications/fog-computing-conceptual-model.

[5] A. Paul, H. Pinjari, W.-H. Hong, H. C. Seo, and S. Rho, "Fog computing-based IoT for health monitoring system," *Journal of Sensors*, vol. 2018, Article ID 1386470, 7 pages, 2018.

[6] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *Future Internet of Things and Cloud (FiCloud), 2014 International Conference*, pp. 464–470, Barcelona, Spain, 2014.

[7] D. Trihinas, G. Pallis, and M. D. Dikaiakos, "AdaM: an adaptive monitoring framework for sampling and filtering on IoT devices," in *2015 IEEE International Conference on Big Data (Big Data)*, pp. 717–726, Santa Clara, CA, USA, 2015.

[8] F. Jalali, O. J. Smith, T. Lynar, and F. Suits, "Cognitive IoT gateways: automatic task sharing and switching between cloud and edge/fog computing," in *SIGCOMM Posters and Demos '17 Proceedings of the SIGCOMM Posters and Demos*, pp. 121–123, Los Angeles, CA, USA, 2017.

[9] S.-J. C. Ramneek, S. H. Jeon, Y. J. Jeong, J. M. Kim, S. Jung, and S. Pack, "Boosting edge computing performance through heterogeneous manycore systems," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 922–924, Jeju, South Korea, 2018.

[10] Y. Li, J. Luo, J. Jin, R. Xiong, and F. Dong, "An effective model for edge-side collaborative storage in data-intensive edge computing," in *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))*, pp. 92–97, Nanjing, China, 2018.

[11] J. Lee, G. Yoon, and H. Choi, "Monitoring of IoT data for reducing network traffic," in *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pp. 395–397, Prague, Czech Republic, 2018.

[12] Modicon, Inc., "Modicon Modbus Protocol Reference Guide, PI-MBUS-300 Rev. J," 1996, http://modbus.org/docs/PI_MBUS_300.pdf.

[13] S.-J. Kim, K.-W. Cho, M.-E. Lee, and O. Chang-Heon, "A study on adaptive QoS control system based on MQTT for reducing network traffic," *International Conference on Future Information & Communication Engineering*, vol. 9, no. 1, pp. 223–226, 2017.

[14] Z. Shelby, K. Hartke, and C. Bormann, *The Constrained Application Protocol (CoAP)*, Internet Engineering Task Force (IETF), 2014.

[15] Object Management Group (OMG), *Data Distribution Service for Real-Time Systems, Version 1.2*, Object Management Group (OMG), 2007.

[16] Object Management Group (OMG), *The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol, Version 2.1*, Object Management Group (OMG), 2009.

[17] Verizon Digital Media Services, Inc, *IP Latency Statistics,* December 2018, https://enterprise.verizon.com/terms/latency/.