IoT Device State Management System to Inform Defect of Third Party Devices

Takafumi Sakai
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
fumi@pluslab.org

Kenshin Hata
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
kenshin@pluslab.org

Kensuke Suzuki
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
kens11@pluslab.org

Katsuhiro Naito
Aichi Institute of Technology
Toyota, Aichi 470-0392, Japan
naito@pluslab.org

***Abstract*—We are developing an IoT platform called InfoFlow. InfoFlow allows users to run IoT services in their IoT devices without the knowledge of IoT and to receive, process, and publish real-time data from devices owned by third parties. However, since IoT devices are machines, there is a possibility that an anomaly may occur and the real-time data may fail to be published. As a solution to this problem, there is a concept of network monitoring, which checks for anomalies in the state of devices and surrounding networks. The primary purpose of conventional network monitoring systems is to monitor devices within the scope of individuals or devices within an organization and to notify users when an abnormality occurs. While it is possible to check the status of devices and applications, it isn't easy to implement functions to monitor the status of services running within applications and display the status to third-party devices.**

This paper proposes a device state management system in InfoFlow. The device state management system will manage user-owned devices and send notifications to users and devices according to the status of the devices they communicate with. The evaluation result shows that the proposed system can send the status of third-party devices without additional hardware and that the time to send the notification is at the tolerance level.

***Index Terms*—information flow, IoT, data stream, real-time processing, network monitoring**

I. INTRODUCTION

IoT is currently used in a variety of fields and is expected to enrich our lives by increasing the number of IoT devices [1], [2]. On the other hand, frequent communication between IoT devices and the cloud has increased the load on the cloud, resulting in cloud failures and communication delays [3], [4]. One solution to this problem has been proposed by information flow [5]. Information flow is a concept where information is not stored in a single location. Data will be processed as a stream and, in advance, utilized in various locations. However, its effectiveness has not been verified in the field of IoT.

The authors have been developing an IoT platform based on the idea of information flow called InfoFlow [6]. In InfoFlow, users can create IoT services without the knowledge of IoT. IoT services can be deployed to users' devices and may affect the real world using real-time data from the device sensor or third-party devices. The acquired real-time data can also be published for other devices to use. We expect the published data to create new values when integrating with other real-time data.

When using InfoFlow to publish real-time data in a device, the device may not be able to publish the data correctly for reasons such as network errors and device failures [7]. In this case, there is a possibility that the services of the device receiving the target real-time data and the device receiving the processed data will be affected.

There is a concept of network monitoring, which checks that the server and its surrounding network are working properly. In network monitoring, it is possible to monitor hardware aspects such as whether the server itself is working properly. In advance, software aspects such as whether applications are down can also be checked [8].

Network monitoring can be performed by using dedicated software or Open Source Software such as Datadog and Zabbix [9], [10]. In recent years, there are also software packages that have the function of network monitoring in the execution software itself, for example, the LivenessProbe function in Kubernetes [11]. Network monitoring software mainly consists of a management server and an agent that monitors each server, and the agent is often a dedicated hardware outside the server. The agents monitor the status of the servers and applications by sending pings to the servers and monitoring the application ports, and in case of an abnormality, they send information to the monitoring software, which displays the abnormality on the browser for administrators to check.

Existing technologies are designed to detect only errors in applications and servers, and their primary purpose is to display such information to users. In InfoFlow, it is necessary

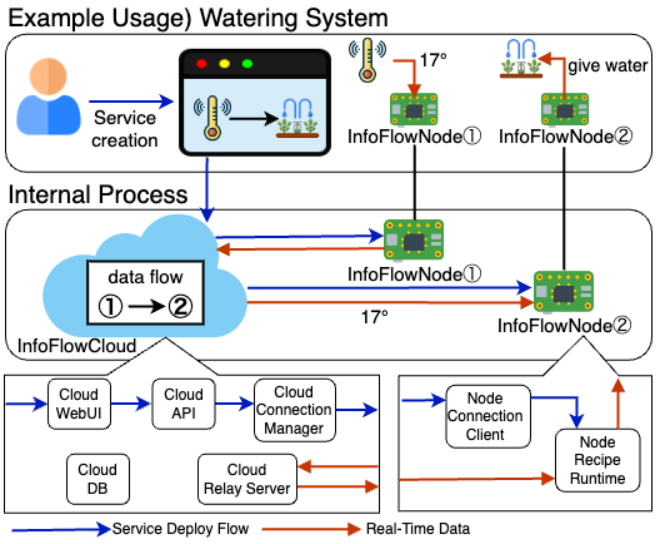

Fig. 1. Overview of InfoFlow.

to determine the state of the device, applications, and services inside the application. It is necessary to have a mechanism to inform the information to other devices in case of an error and recovery. It isn't easy to implement monitoring of services in applications and transmission of such information to other devices.

Therefore, this paper proposes a device state management system for InfoFlow. The proposed system informs the device receiving the data from the defective device in the event of device malfunction. The prototype implementation shows that the proposed system can send notifications, and the time taken to send the notification does not pose a problem even when the number of connected devices increases.

II. Proposed System

A. Info Flow

Figure 1 shows the overview of InfoFlow and the proposed system. The purpose of InfoFlow is to verify the effectiveness of the information flow and to enable users without IoT knowledge to create IoT services across IoT devices without being aware of the communication and roles of the devices. Using InfoFlow, users can create IoT services that utilize real-time data from third-party devices. We expect that new value will be created by communicating with third-party devices. To use InfoFlow, users first prepare an IoT device, launch the InfoFlow program, and register the device. Then, the user creates an IoT service on the browser and deploys it to the registered device. The service is created from a recipe, which is an object that serves as a design document of the service.

InfoFlow consists of two components: InfoFlowCloud and InfoFlowNode. InfoFlowNode is a component that utilizes real-time data. It can receive and publish real-time data from or to other devices depending on the recipe. This paper defines the device that publishes information in a service as a publisher node and the device that receives real-time data as a subscriber node. The publisher node and subscriber node depend on services. Furthermore, InfoFlowNode consists of the following two components.

- NodeConnectionClient
  Constantly connects to InfoFlowCloud to receive information necessary to build services and passes to the NodeRecipeRuntime.
- NodeRecipeRuntime
  Execution environment of the IoT service created by users.

InfoFlowCloud is a component for managing InfoFlow and services. InfoFlowCloud consists of the following five components.

- CloudWebUI
  Provides UI which can be operated from a browser to users.
- CloudAPI
  Provides API for CloudWebUI to retrieve data necessary for displaying pages.
- CloudDB
  Database for CloudAPI and CloudConnectionManager to persist data.
- CloudConnectionManager
  Provides constant connection with InfoFlowNode and sends information necessary for building IoT services to the appropriate devices.
- CloudRelayServer
  Used by InfoFlowNode to send and receive real-time data in the publish/subscribe pattern. InfoFlowNodes can publish or subscribe to data through topics.

B. Concept of The Proposed System

By using InfoFlow, it is possible to utilize real-time data from other devices, including third-party devices. On the other hand, a device may fail to send or receive data due to communication failures or other reasons. In this case, there is a possibility that one device's failure may cause secondary or tertiary problems for multiple related devices, such as devices subscribing to the real-time data of the device that failed in communication or devices receiving the processed data. To prevent such a situation, it is necessary to have a mechanism that enables the service to confirm that the device receiving the data is working properly, and to change the processing contents of the service if it is not working properly.

In this paper, we propose a state management system for IoT devices. The proposed system analyzes the real-time data flow of the service. It notifies the device of receiving the data when there is an anomaly in the device publishing the data. By using the notifications, the service can seamlessly change its

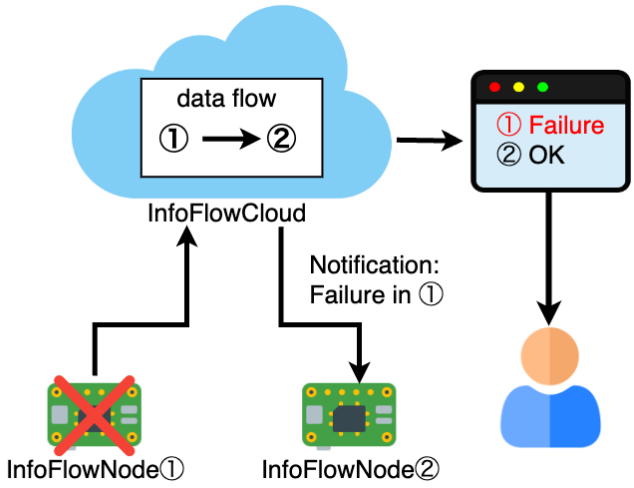

Fig. 2. Overview of the Proposed System.

TABLE I
DEVICE NOTIFICATION STATUS

DEAD	Publish device dead
RESTARTED	Publish device restored
UNKNOWN	Any other status occurred

processing, and the service can be executed. The proposed system enables us to check the status of applications and devices, which has been possible with conventional network monitoring technologies, and also to check the status of services, which has been challenging to check.

The proposed system consists of a device status notification function and a device management function. Figure 2 shows a schematic diagram of this research. In the proposed system, InfoFlowCloud detects device anomalies, analyzes the data flow, and sends notifications to the devices receiving data from the anomalous device. The device receiving the notification can perform error handling by processing the information in the notification, such as switching the process. In addition, the browser user can know the device's abnormality. Although the proposed system also restarts applications when an error occurs, users still need to touch the device directly in case of hardware failures.

C. Device Status NotificationFunction

The device status notification function is a function to send a notification to InfoFlowNodes based on the status of the publisher node used in the IoT service. The proposed system analyzes the recipe and stores the publisher node used in the service to CloudDB during the service deployment flow. Notifications will be sent depending on the state of the publisher node, regardless of the owner of the publisher node and the subscriber node. To send a notification to devices, the proposed system must know which device gets real-time data from which device.

Table I shows the list of statuses of the notification the proposed system can send. These systems are common and unaffected by services running on the device. DEAD will be sent when the publisher node disconnects from InfoFlowCloud, and RESTARTED will be sent when InfoFlowCloud detects the connection. UNKNOWN is a status code that represents another status of the device. In addition to the status code, the date and time the error occurred and the device's id is sent. Notifications will be sent to the NodeConnectionClient through the streaming between NodeConnectionClient and the CloudConnectionManager without additional hardware. NodeConnectionClient will transfer the notification data to JSON format and send it to NodeRecipeRuntime. Notifications can be used in NodeRecipeRuntime. When a notification is received, it can be used in various ways, depending on the user's perspective, such as publishing data to the device's subscribe node that the service is running in an incomplete state, performing alternative processing, or receiving data from a different device.

Fig. 3 shows the deployment flow using the proposed system. CloudAPI analyzes the recipe in the proposed system and determines the publisher node. Storing the information from the publisher node allows the proposed system to send notifications. In advance, the proposed system checks the device connection and stores the service information. The information can be seen on the device management dashboard. Since the recipe is in JSON format with objects in an array, the system can detect the publish nodes by searching for specific key-value pairs.

Fig. 4 shows the sequence of how the proposed system sends a notification to devices. There are two main reasons notices are sent. When the publisher node starts or the publisher node disconnected from the InfoFlowCloud caused by the device's defect. The notification process when the publish node starts works as follows.

1) The publisher node starts and checks if it can connect to InfoFlowCloud. If the connection is confirmed, it sends a server-side streaming request to CloudConnectionManager in InfoFlowCloud. The publisher node sends the device ID as information when sending the request.
2) When the CloudConnectionManager accepts the server-side streaming request, it will first update the device status stored in the CloudDB.
3) After updating the device information, the server-side streaming between the publisher node and the CloudConnectionManager will start.
4) CloudConnectionManager searches for the subscriber node of the connected publisher node. If there is a connected subscriber node, it will send a notification with the status RESTARTED.

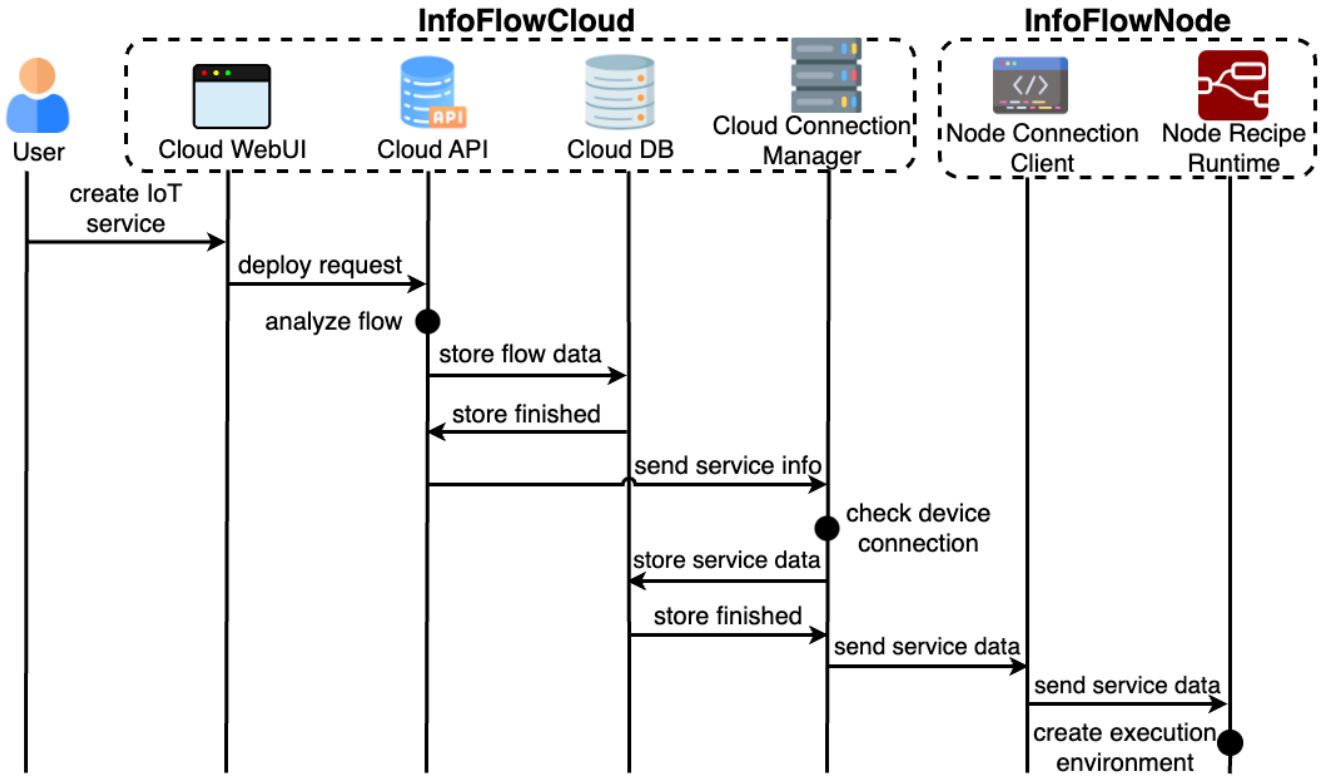

Fig. 3. InfoFlow Service Deploy flow.

The notification process when the publish node is disconnected from InfoFlowCloud works as follows.

1) When the CloudConnectionManager and publisher node are connected, it will keep the connection by sending the Keep-Alive signal. If the publisher node gracefully ends the program, the node sends the stream end request. Otherwise, CloudConnectionManager will determine that the node is not connected to InfoFlowCloud when the Keep-Alive request fails. Regardless of the termination method, the process is the same after this.
2) CloudConnectionManager deletes the information from the device stored when streaming.
3) Updates the device status stored in the CloudDB as disconnected.
4) Query the subscribing node of the disconnected device.
5) Send a notification with the status DEAD.

D. Device Management Function

In the conventional InfoFlow, the device register function was implemented. However, users must identify their devices using the device ID determined by the system, which cannot be changed. To check if the device was running correctly, users had to check the device directly. As a solution, the device management function allows users to manage their devices easily. This feature can be divided into two main functions.

First, we provide the device management dashboard to simplify device management by UI. Through the dashboard, the user can see the status of their owned devices and whether it is connected to the cloud or not. In the dashboard, user can identify their devices by device names. The device name can be given during device registration and modified through the CloudWebUI or in the device settings. Moreover, information on the running service can be confirmed.

Second, to reduce the frequency with which users directly touch the device, the proposed system provides the function to restart the program in a run-time error automatically. Since the program restarts on run time errors, the frequency of direct contact with a device can decrease. InfoFlowNode sends Keep-Alive signals between the device and the cloud to periodically check the device status to maintain the connection between the device and the cloud.

III. Implementation

We implemented the device management function and device status notification function. Since CloudWebUI and CloudAPI are implemented using TypeScript, and CloudConnectionManager and NodeConnectionClient are implemented using Golang, the proposed system is also implemented using TypeScript and Golang. Since MySQL is used in CloudDB for data persistence, tables, and relations were added to the existing implementation. The execution environment of

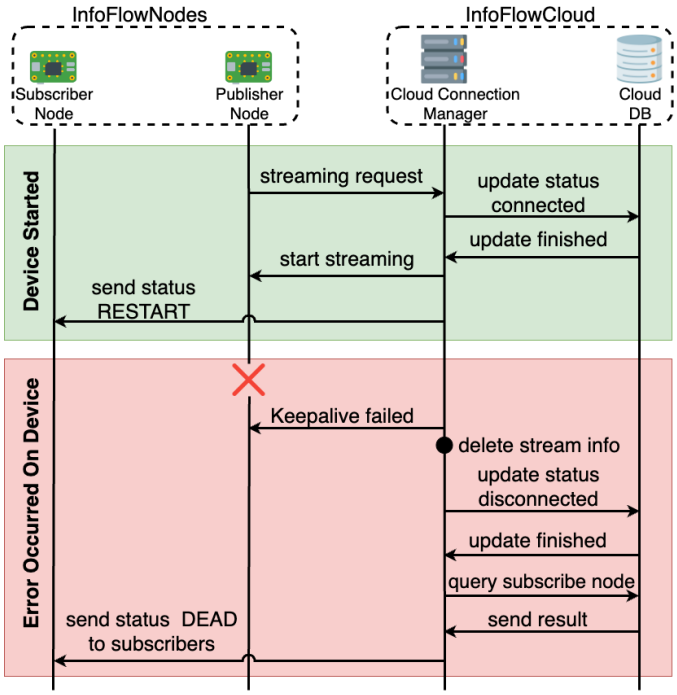

Fig. 4. InfoFlow Notification flow.

InfoFlowNode is built using Docker to reduce the differences in the environment of IoT devices. The implementation of NodeConnectionClient is modified so that the program terminates abnormally when an error occurs. When the program terminates abnormally, the container, which is the execution environment, also terminates. Since Docker is configured to perform the container health checks, the container is restarted even if terminated.

The InfoFlowNode notification mechanism is implemented by extending the existing service deployment function. The current system is implemented using server-side streaming of gRPC. In extending this function, both recipes and notifications are received in the same RPC, and the processing contents are determined based on the type of data sent. NodeRecipeRuntime in InfoFlowNode is based on Node-RED, enabling applications to be created in low code. We implemented a custom node for Node-RED to receive notifications from InfoFlowCloud and utilize them in NodeRecipeRuntime. The custom node is a built-in feature of Node-RED and does not require dedicated hardware. The custom node releases an endpoint in NodeRecipeRuntime accessible only from within the device. The custom node does not accept any information and has only one output to send the received notifications. The user can use the notification information by constructing a flow from a single work. Since this endpoint is for receiving errors, it is implemented as a POST method endpoint named /error. When NodeConnectionClient gets a notification, it converts it into JSON format and sends the information to this endpoint, which can be used in NodeRecipeRuntime.

TABLE II
SPECIFICATIONS OF THE MEASURING DEVICES

PC	
OS	Ubuntu 20.04.1 LTS
CPU	Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz
Memory	64GB RAM
Raspberry Pi 4 Model B	
OS	Raspbian GNU/Linux 11.0 (Bullseye)
CPU	Quad Core Cortex-A72 (ARM v8) 64-bit
Memory	8GB RAM

TABLE III
CPU USAGE WHEN RECEIVING NOTIFICATION

Number of notifications	CPU usage
0 (no notification)	5.2%
1	5.7%
3	5.9%
5	6.8%

IV. EVALUATION

Fig. 5 shows the evaluation environment for this evaluation. We prepared InfoFlowCloud, Publisher Node, and Subscriber Nodes for this verification. Table. II shows the devices' specifications used in this verification. Since Publisher Nodes and Subscriber Nodes are reproduced in a virtual environment, the numbers can be dynamically changed.

To demonstrate the appropriateness of the proposed system, we perform the operation test of the proposed system. We set up six devices, each owned by a different user, and each runs a different service. As a test environment, we prepared two Publisher Nodes and three Subscriber Nodes, each receiving data from one Publisher Node. We repeatedly started and stopped the program of one Publisher Node and confirmed that notifications were sent to the Subscriber Nodes that received data from the relevant Publisher Node. As a result of the verification, we demonstrated that users could receive notifications from third-party Publisher Nodes without additional hardware. Furthermore, we confirmed that the device management dashboard reflects the state of the connection between the device and the cloud.

We measured the load on the nodes when receiving notifications. We used Raspberry Pi 4 for this evaluation. Fig. III shows the results of the verification. The values in the table show the CPU usage of the task running by the InfoFlowNode and the Node-RED, and they are an average of 10 times for each condition. Since it is difficult to expect more than one notification at a time, we estimated many notifications this time. We verified up to five notifications at the same time. As a result of the verification, we confirmed that the notifications on the device side are not too large and do not significantly affect the service operation of the device.

We also measured the time the cloud takes to send a notification to the Subscriber Node after it detects a device failure. We categorized the time until InfoFlowCloud detects

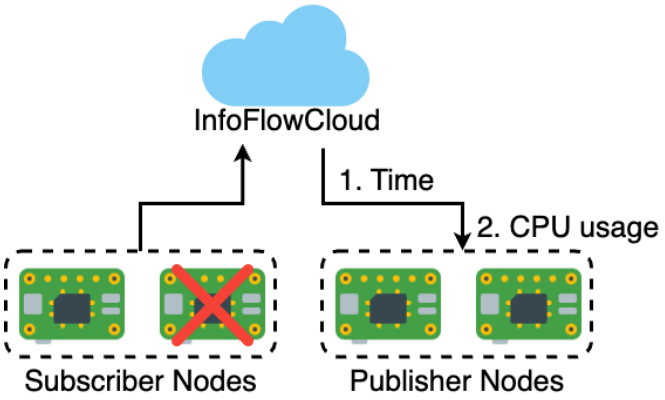

Fig. 5. Evaluation Environment.

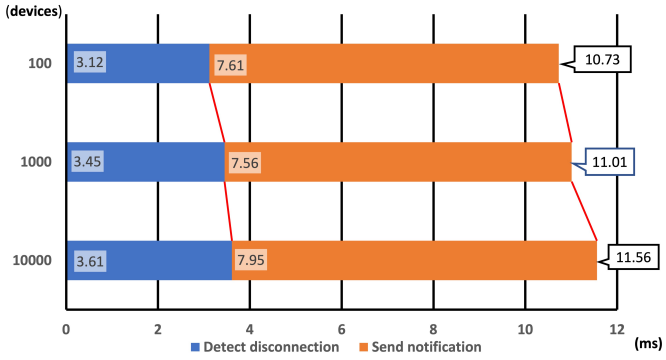

Fig. 6. Time taken to send notifications.

a device failure and the time until InfoFlowCloud processes the detected information and sends a notification to the Subscriber Node. Fig. 6 shows the average of 30 measurements taken under each condition. Since we expect that the number of real-time data used by one service is about five for many devices, each InfoFlowNode is configured to publish and subscribe to five other devices. As a result, we found that sending a notification for each device disconnection detection took almost twice as long. We believe this is because complex queries are made to CloudDB when sending notifications. We also found that the connection detection time and the notification time do not increase significantly even when the number of devices increases. From these results, we confirmed that the proposed system is acceptable in terms of speed, even when the number of devices increases, and there is no significant increase in the time required for notification when a problem occurs in the publisher node of the service.

V. Conclusion

In this paper, we have proposed a device management function and a device status notification function that can be used in InfoFlow. In the proposed system, we added a function to send a notification when a device used by an IoT service deployed on an InfoFlowNode disconnects from the InfoFlowCloud. The InfoFlowNode can change the processing of the service based on the received notifications and can also indicate to other devices that the service is running inadequately. We also added a function to restart the device management dashboard automatically and InfoFlowNode when they are terminated for some reason so that users can easily manage many IoT devices. The verification results show that the proposed system can tolerate the time required for InfoFlowNode to detect disconnection and send a notification even when the number of connected devices in the InfoFlowCloud increases.

Acknowledgment

This work is supported in part by Grant-in-Aid for Scientific Research (C)(21K11877), Japan Society for the Promotion of Science (JSPS).

References

[1] M. Bansal, I. Chana, and S. Clarke, “A Survey on IoT Big Data: Current Status, 13 V’s Challenges, and Future Directions,” *ACM Computing Surveys*, vol. 53, no. 6, pp. 1–59, Nov. 2021.

[2] M. Aledhari, R. Razzak, B. Qolomany, A. Al-Fuqaha, and F. Saeed, “Biomedical IoT: Enabling Technologies, Architectural Elements, Challenges, and Future Directions,” *IEEE Access*, vol. 10, pp. 31 306–31 339, 2022.

[3] M. M. Sadeeq, N. M. Abdulkareem, S. R. M. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, “IoT and Cloud Computing Issues, Challenges and Opportunities: A Review,” *Qubahan Academic Journal*, vol. 1, no. 2, pp. 1–7, Mar. 2021.

[4] M. Laroui, B. Nour, H. Moungla, M. A. Cherif, H. Afifi, and M. Guizani, “Edge and fog computing for IoT: A survey on current research activities & future directions,” *Computer Communications*, vol. 180, pp. 210–231, Dec. 2021.

[5] K. Yasumoto, H. Yamaguchi, and H. Shigeno, “Survey of Real-time Processing Technologies of IoT Data Streams,” *Journal of Information Processing*, vol. 24, no. 2, pp. 195–202, 2016.

[6] T. Sakai, K. Hata, T. Wada, and K. Naito, “IoT platform using information flow to reduce load on cloud,” in *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 1211–1216, Jun. 2022.

[7] M. Aboubakar, M. Kellil, and P. Roux, “A review of IoT network management: Current status and perspectives,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 7, pp. 4163–4176, Jul. 2022.

[8] X. Hu, Y. Xiang, Y. Li, B. Qiu, K. Wang, and J. Li, “Trident: Efficient and practical software network monitoring,” *Tsinghua Science and Technology*, vol. 26, no. 4, pp. 452–463, Aug. 2021.

[9] Datadog, “Cloud Monitoring as a Service — Datadog,” https://docs.datadoghq.com, February 2023.

[10] Zabbix, “Zabbix :: The enterprise-class open source network monitoring solution,” https://www.zabbix.com, February 2023.

[11] D. McIntosh, “The Utility of Service Oriented Architectures (SOA) and Microservice Architectures in Naval Systems.”