electronics

MDPI

Article

Evaluation of a Smart Intercom Microservice System Based on the Cloud of Things

Hsin-Yu Huang [1,2], Yong-Yi Fanjiang [2,3,*], Chi-Huang Hung [2,4], Hsing-Yu Tsai [3] and Bing-Hong Lin [3]

[1] Department of Fashion Business Management, Lee-Ming Institute of Technology, New Taipei 243086, Taiwan; hyhuang@mail.lit.edu.tw
[2] Graduate Institute of Applied Science and Engineering, Fu Jen Catholic University, New Taipei 242062, Taiwan; allen@mail.lit.edu.tw
[3] Department of Computer Science and Information Engineering, Fu Jen Catholic University, New Taipei 242062, Taiwan; 400085264@m365.fju.edu.tw (H.-Y.T.); 408085143@m365.fju.edu.tw (B.-H.L.)
[4] Department of Information Technology, Lee-Ming Institute of Technology, New Taipei 243086, Taiwan
* Correspondence: yyfanj@csie.fju.edu.tw; Tel.: +886-2-2905-2444

Abstract: This research migrates a monolithic smart intercom system to a microservice architecture, making the system more secure, stable, and scalable. The security mechanisms of the instant messaging platform are combined with microservices to improve the security of the system. The stability and performance of microservices are shown to be better than those of monolithic services through experimental tests. Residents can use different instant messaging software instead of the handset to improve the convenience of using this system. This system also implements community broadcasts, platform broadcasts, unit broadcasts, and family broadcasts to exchange messages across different instant messaging platforms. This paper proposes OpenAPI for other smart intercoms to integrate this system's services and resources. In addition, this study deploys two microservice architectures using a native load balancer with a kube proxy and a service mesh load balancer with an istio proxy. Experiments were conducted during which residents tested the kube proxy and the istio proxy using stress testing tools on two microservice architectures, and the results showed that the kube proxy was slightly better than the istio proxy.

Keywords: microservice; OpenAPI; instant messaging software; smart intercom

check for updates

Citation: Huang, H.-Y.; Fanjiang, Y.-Y.; Hung, C.-H.; Tsai, H.-Y.; Lin, B.-H. Evaluation of a Smart Intercom Microservice System Based on the Cloud of Things. *Electronics* **2023**, *12*, 2406. https://doi.org/10.3390/electronics12112406

Academic Editor: George A. Tsihrintzis

Received: 23 April 2023
Revised: 22 May 2023
Accepted: 22 May 2023
Published: 25 May 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Software development typically uses structured analysis and design methods to break down business processes into modules with a hierarchical structure. Suitable modules with high cohesiveness and low coupling are designed to enhance the speed and quality of system development. Due to the popularity of the internet, more web applications are using web services, which usually consist of application programming interfaces (APIs) to enhance the speed of software development by executing service requests submitted by clients through remote servers on the internet.

With the maturity of cloud computing and containerization technology, many system developers have adopted microservice architecture. Therefore, more monolithic systems are migrating to microservice architecture. As the scale of a monolithic system increases, it takes a long time to compile, deploy, and start up the system. As a result, the dependency on various services increases rapidly. The system's stability may break down due to the problems of individual modules, which is difficult for new developers and maintainers to understand, resulting in a software development crisis and operator nightmare [1].

In the microservice architecture, each function can operate independently. An exception to one microservice does not affect others. If a microservice is overloaded, it can also be adjusted by resources to meet the utilization requirements. How many resources are used will be paid to the cloud platform owner, which can eliminate the work of management

and maintenance of the server side. Meanwhile, the integration of cloud services and the Internet of Things (IoT) technology has recently become an important issue, so the development of the Cloud of Things (CoT) is also being emphasized. This study proposes a microservice system architecture based on a monolithic smart intercom system [2]. A schematic diagram of the CoT-based microservice system is shown in Figure 1.

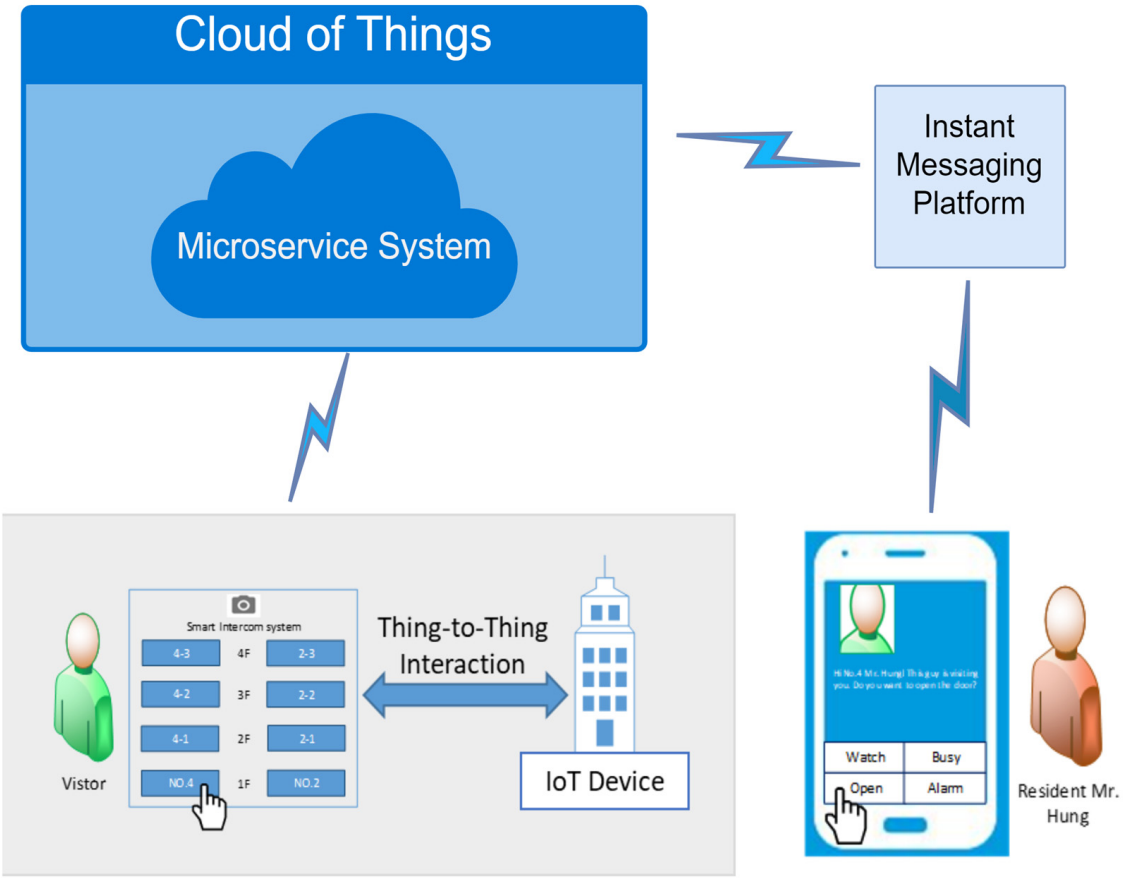


Figure 1. Schematic diagram of the CoT-based microservice system.

Furthermore, it is common to use a service mesh mechanism to improve the performance of microservices due to the complex dependencies and communication between their pods [3]. However, there are no complex dependencies, and there is a large amount of data traffic between the pods of microservices in this system. This study used a stress test tool to test the remote CoT device to confirm that the microservice architecture performs better than the monolithic system. At the same time, two load-balancing microservice architectures, kube proxy and istio proxy deployments, were implemented under the microservice architecture, and the simulations showed that the kube proxy was relatively more stable. The objectives and unique advantages achieved in this study are listed below.

- The migration of the monolithic smart intercom system to a microservice architecture was completed, while the security mechanisms of the instant messaging platform and the microservice architecture were combined to improve the security of this system.
- By extending the system functionality to provide more community and resident access, residents can use different instant messaging software to replace the handset of the intercom. The system enables community broadcast, platform broadcast, unit broadcast, and family broadcast through instant messaging microservices to enable residents to exchange messages with each other, even if they use different instant messaging software.
- Any smart intercom can be combined to use the features and resources of this system by providing OpenAPI. Residents are able to communicate with visitors through

instant messaging software and this microservice system while being able to control remote CoT devices.

- Simulation tests were conducted to compare two different load-balancing microservice architectures, with the kube proxy slightly better than the istio proxy. As a result of the stress tests conducted in this study, microservice systems are more stable and have better performance than the monolithic system.

The remainder of this article is organized as follows. Section 2 is a review of related work. Section 3 describes the proposed microservices' system development. The testing configuration is described in Section 4. Section 5 presents the testing standard and results, and the discussion is presented in Section 6. Finally, the conclusion is given in Section 7.

2. Related Research

A monolithic application can operate independently of other applications and perform all the steps required for a particular function. However, when a monolithic application grows to a certain size, it takes too long to compile, deploy, and start up, and the complexity of the dependencies between codes increases rapidly. The entire system may fail if a single function does not work, affecting other functions [1]. The independent deployment of microservices simplifies continuous integration and shortens delivery time, making them the most suitable technology for unified services. Microservice architectures can be deployed independently without restarting the entire application, and failure of a single microservice does not affect other microservices, each of which can scale independently using pools, clusters, and grids. The flexibility of the cloud architecture is well-suited for microservices, which are packaged to allow for the more flexible use of new frameworks, libraries, data sources, and other resources [4]. As microservice systems are more reliable than monolithic systems, an increasing number of monolithic systems are migrating to microservice architectures. Most of the focus is on the more fault-tolerant and resilient nature of microservice architectures, as well as their considerable maturity and availability [1,5,6].

An IP phone intercom system is an IP network system with a SIP server and many SIP clients that allows users to have voice, video, and SMS conversations and provides basic access to control devices [7]. However, the IP telephony software must be installed on the user's smartphone for them to be able to use it at any time during a call. There is also a mobile smart intercom system that can provide users with videos of each other. Users can select a channel and can also invite other online users to a temporary channel for intercom communication [8]. The main reasons for door lock systems are to avoid intrusion by unauthorized persons and to facilitate user access. Various door lock systems use IoT devices to provide remote access to control unauthorized access, and users use smartphones with integrated IoT devices to control door locks [9–11]. Many studies have been conducted to control home appliances through apps that use IoT technology [12–14]. Users can control home appliances through instant messaging software, reducing the need to install smartphone apps [15]. IoT-based smart home systems connect various household appliances to Internet, allowing users to easily check the status of their home and household appliances by using a server or connecting to an efficient API for control [16,17]. A prototype system for controlling air conditioners using an embedded system has been developed [18], and household appliances can also be controlled by IoT-based voice commands or through gesture detection and control [19–22]. In the smart intercom system designed by previous research, residents control the IoT device with instant messaging software without installing new applications and learning. Residents can check the presence of visitors and the status of their doors through their smartphones anytime, anywhere [23].

Although migrating from monolithic applications to microservices is the current trend, the migration process can be challenging for developers [1,4,24]. The intelligent building management system uses the Intelligent Building Management Data Processing Framework (IBFRAME) through the kube proxy to collect real-time data from IoT devices, but this also requires data analysis to monitor the building environment to support various smart building applications [25]. Integrating environmentally assisted living systems through

microservice architectures and the Internet of Medical Things (IoMT) sensors enables stakeholders and software architects to select and evaluate application frameworks and platforms to implement microservice-based systems [26]. Cloud computing enables organizations to share a single resource regardless of location, with IoT devices on embedded systems collaborating to bridge the gap between software and hardware. Cloud computing and serverless capabilities can be used to control and monitor physical hardware components and integrate IoT devices into the cloud using restful APIs [27]. Embedded systems are deployed on microservices using Node-RED's Workflow Manager, which provides tools for developing IoT systems such as the edge computing paradigm [28].

The smart intercom replaces the traditional intercom, and a resident can communicate with a visitor using the instant messaging software on their smartphone. In addition, residents can communicate effectively with visitors through web services on the Web of Things (WoT) at different network nodes and monitor the home to send alarms when necessary [2]. The WoT-based logical sensor architecture (WLSA) reduces data processing requirements through virtual WoT sensors. Deploying microservices to data streams at the edge optimizes the allocation of resources at the edge and in the cloud through dynamic linkage [29].

Integrating the IoT and cloud computing is becoming increasingly important and referred to as the CoT. Integrating IoT and cloud computing is not straightforward and has several key issues. While CoT can create many business opportunities, the privilege of accessing virtual resources and storage capacity in the cloud, as well as the protection of identity, security, and privacy, becomes very important. CoTs are built in public clouds, and IoT devices are often deployed in resource-limited areas, which further complicates the system [30,31]. The BCoT ecosystem can be seen as a blockchain as a service (BaaS) integrated with cloud computing to develop and deploy blockchains for cloud IoT applications. BaaS provides the infrastructure and technology to ensure the robust and efficient operation of the service [32]. The three scenarios based on fog-only, cloud-only, and fog–cloud collaborative scenarios depend on the requirements of the user and the application [33].

The Hipster Store eCommerce application in a Kubernetes cluster demonstrates the use of the service mesh istio to monitor communication between microservices and develop automated testing and recovery capabilities [34]. The bookstore application uses the service mesh istio to inject sidecar proxies into each microservice and dynamically load balance between services through the istio control plane application service-specific routing. The experimental results show that the istio-based system maintains stability and consistency by maintaining responsiveness and consumes fewer resources [35].

Table 1 shows eight related studies to this paper, of which the first three are different monolithic intercom systems, and the last five are applications in microservice architectures. The IoT type is classified as IoT, WoT, and CoT. Two different load-balancing microservice architectures are implemented in this paper and the bookstore application. However, as the bookstore application does not access IoT devices, the istio proxy was tested to perform better.

A typical load-balancing solution combines HTTP and message queues to support microservice communication. The load of the successor microservice depends on the load assigned by the predecessor microservice [36–38]. Based on the ideas of container technology, microservice architecture, and service mesh structure, an application architecture for an IoT platform based on the service mesh structure is designed. After experience and testing, the results proved that the architecture design has good performance [39,40]. The length of data is contained in HTTP reactions using the istio default log message. An algorithm is used for calculating the application error identification metrics, using items such as JSON formatting in HTTP reactions as the metrics. Experimental measurements add the log message length and response time. The average log message length increased by 216 words, and the average response time increased by 7% compared to istio's default log format [41].

Table 1. Intercom-related research properties list.

Type	Non-Extra Application	Deploy Chatbot	Monolithic/Microservice	IoT Type	Load Balance Proxy	Provide OpenAPI	Fields of Application
IP Phone Intercom [7]	-	-	Monolithic	IoT	-	-	Intercom
Mobile Intercom [8]	-	-	Monolithic	IoT	-	-	Intercom
Smart Intercom [2]	V	Single IM Software	Monolithic	WoT	-	-	Intercom
Intelligent Building [25]	-	-	Microservice	IoT	Kube Proxy	-	Building
WLSA [29]	-	-	Microservice	WoT	Istio Proxy	-	Logical Sensor Architecture
Hipster Application [34]	-	-	Microservice	-	Istio Proxy	-	eCommerce
Bookstore application [35]	-	-	Microservice	-	Kube Proxy/Istio Proxy	-	Bookstore
Smart Intercom Microservice	V	Multiple IM Softwares	Microservice	CoT	Kube Proxy/Istio Proxy	V	Intercom

-: Not supportive. V: Supportive.

One study developed a chatbot system (MsdoBot) that integrates microservices to develop and operate many supplementary tools to provide the required information on various usage scenarios and for monitoring and maintaining real-time system status reports [42]. An information robot provides users with multiple channels of information delivery through web subscriptions. It is based on an event-based microservice architecture to realize and provide real-time data delivery [43]. Factory employees can use Telegram Messenger to remotely control their working electronic devices with the help of an artificial intelligence chatbot without the need for manual switching and to support energy savings [44]. Using the Facebook Messenger application, a smart security system and a home automation system were implemented, allowing residents to control the device even when they are not at home and use the chatbot to lock or unlock the door and check who is there. The chatbot is used to alert staff if they are allowed to turn equipment on or off, such that the chatbot notifies staff to turn on fans when the temperature is high [45].

In this paper, the monolithic smart intercom system is migrated to a microservice architecture that allows residents to use different instant messaging software to replace the intercom handset and exchange messages with each other. The deployed Cloud of Things framework can provide residents with the ability to communicate with visitors through instant messaging software and microservices, and any smart intercom can join the system with the OpenAPI.

Although microservice architectures have proven to be more advantageous than monolithic architectures, monolithic architectures are easier to develop. They are a good choice when the system load is low and there are fewer integration, connectivity, and configuration issues. A microservice architecture is more efficient if an application needs to handle more requests. The choice between these two architectures depends entirely on the type of problem to be solved [46]. There is a need to compare monolithic and microservice architectures with reference to relevant studies [1,40,41,47]. Therefore, this study will also use stress testing tools to simulate and compare two different load-balancing microservice architectures and monolithic systems by activating CoT or IoT devices.

3. Microservice System Development

3.1. Monolithic to Microservice

In this study, a monolithic smart intercom system is used as the basis for deploying a microservice architecture using microservice and IoT technologies to enhance functionality and make the system more secure, stable, scalable, and convenient for more communities and residents.

The monolithic smart intercom system is an integration of an embedded system and a chatbot server that provides residents with a menu of instant messaging software to communicate directly with visitors through various services and to perform functions of door-related devices. When a resident clicks one of the “Open door”, “Monitor”, “Alarm”, or “Busy” buttons, the instant messaging software sends a message through the messaging API, which then uses a webhook to trigger an event to be sent to a chatbot server on the embedded system to perform a specific function based on the message. The software framework of the monolithic smart intercom system is shown in Figure 2.

In addition to migrating the above functions to the microservice, this study establishes an instant messaging microservice that enables residents to receive messages from the smart intercom and even communicate with each other when residents choose to use different instant messaging software. On the other hand, these bot functions of the chatbot server on the embedded system can be migrated to the microservice. Residents who use different instant messaging software can receive messages, control the CoT device, and communicate with each other through the instant messaging microservice.

In addition, this system develops an OpenAPI for smart intercoms, enabling this system to provide services to integrated smart intercoms. Any smart intercom will be able to access the features and resources of this system to provide services through OpenAPI.

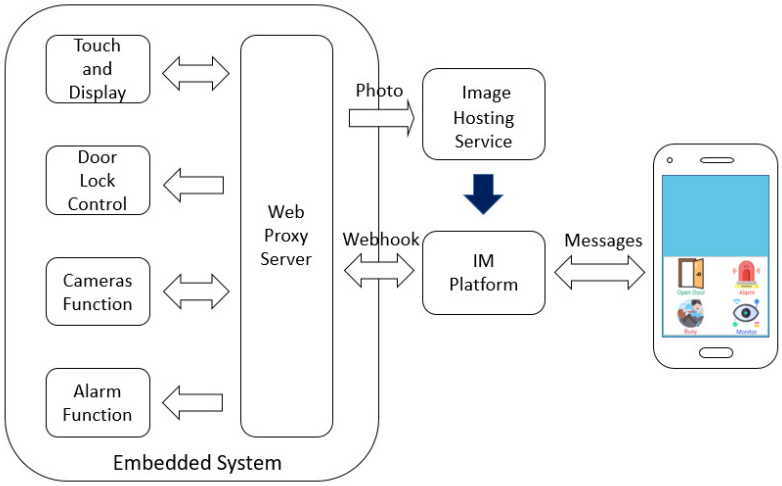


Figure 2. The software framework of the monolithic smart intercom system.

3.2. Microservice Architecture

This system consists of an intercom microservice, an instant messaging microservice, and an administration microservice. Residents can use instant messaging software to receive visitor messages and click on the menu of the instant messaging software to activate the system functions. The software framework of this system is shown in Figure 3.

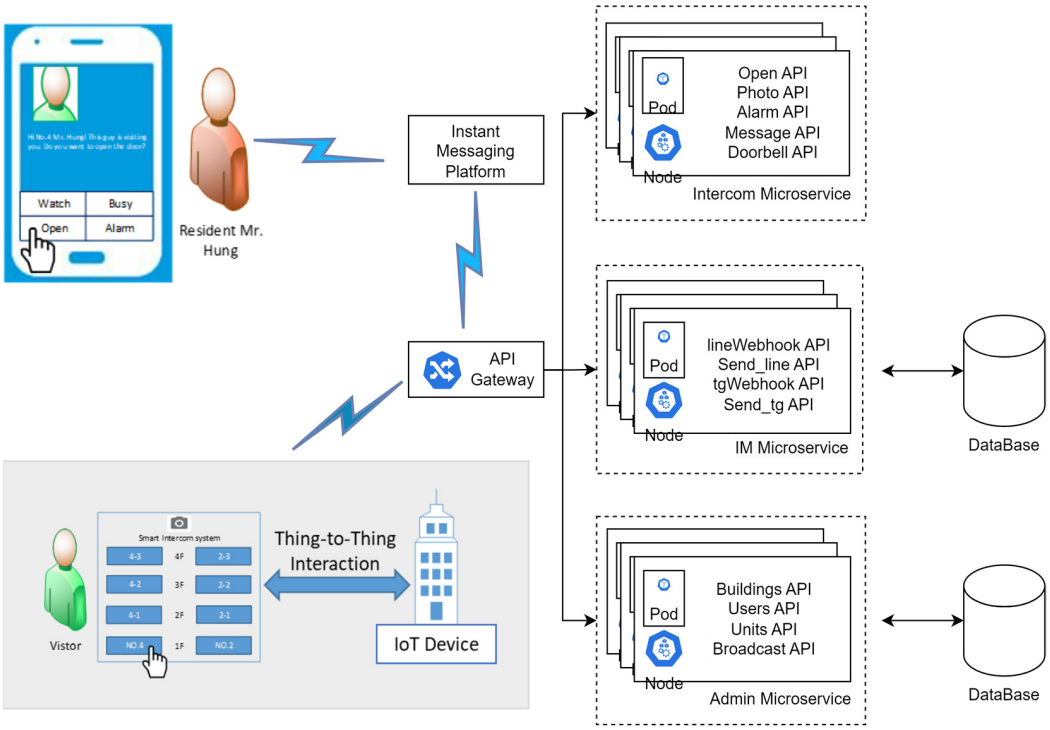


Figure 3. Software framework of this system.

The system provides a microservice for managing and controlling the smart intercom called the "intercom microservice", which contains the following APIs.

- Open API (for one-way intercom delivery).
- Photo API (for one-way intercom delivery).
- Alarm API (for intercom unidirectional delivery).
- Message API (for intercom send/receive).
- Doorbell API (for intercom).

- Smart Intercom Management: intercomsList API, Regintercom API, intercomsReads API, intercomsModify API, and intercomsDelete API.

In addition, the "instant messaging microservice" is a webhook mechanism triggered by messages sent from different instant messaging platforms. The related APIs are activated according to different events, and the results are sent to the users of the messaging software. The related APIs are listed below.

- lineWebhook API.
- Send_line API.
- tgWebhook API.
- Send_tg API.

To allow more communities and residents to use the system, "management microservices" are divided into community building management, unit management, user management, and broadcast management APIs to provide related services, and the APIs of each category are listed below.

- Community Building Management: buildingsList API, buildingsAdd API, buildingsRead API, buildingsModify API, and buildingsDelete API.
- Unit Management: unitsList API, unitsAdd API, unitsRead API, unitsModify API, and unitsDelete API.
- User Management: usersList API, usersAdd API, usersRead API, usersModify API, and usersDelete API.
- Broadcast Management: platformBroadcast API, buildingBroadcast API, and unitBroadcast API.

Suppose the resident presses the "Open" button. In that case, the message will be sent to the instant messaging software platform, which will use the webhook mechanism to execute the "Open API" of the intercom microservice through the API gateway. The "Open API" controls the remote GPIO to switch on the CoT device to execute the "Open Door" action through the internet.

3.3. OpenAPI of Intercom

OpenAPI is a standard format for defining APIs and supports the Restful API specification. This study uses a third-format library to support the OpenAPI 2.0 specification, which can generate help files, test server programs, and test client programs directly from the comments written in the code.

The following is an example of OpenAPI for smart intercoms. The name, parameters, and response format of OpenAPI for smart intercoms can be defined by adding the following comments to the code shown in Listing 1.

Listing 1. Example of comments in the Regintercom API.

```
    //swagger:route POST/intercom Regintercom
//
//Register new intercom for specific building
//
//Returns Intercom Information
//
//Responses:
//200:  IntercomInfo
//500:
```

The first swagger: the route is used to define an OpenAPI, "POST" is the method used for HTTP calls, the next intercom is the OpenAPI category, and the last Regintercom is the name of this API. The following four lines are the description block of this OpenAPI. The last defines the status code and the format of the https response. The status code is defined in the HTTP/1.1 standard as the meaning of each number. The example above defines two responses: one is 200 for a successful API response, and the accompanying response format is IntercomInfo. The other response is 500 for an internal server error, and the blank after 500 means that the error message does not carry any response.

In addition, this Regintercom API defines the structure of the parameters and the IntercomInfo data structure. The `swagger:parameters Regintercom` is defined here as

API parameters and `swagger:model IntercomInfo` is defined as the `IntercomInfo` data structure of the API response shown in Listing 2.

Parameters of Regintercom.

The swagger:parameters are defined here as API parameters, and the RegisterParam is used to specify the corresponding API://swagger:parameters

Listing 2. Regintercom parameters and the IntercomInfo data structure.

```
    //swagger:parameters Regintercom
type Regintercom struct {
//required:  true
//in:  body
BuildingID int64
//required:  true
//in:  body
AuthCode int64
}

IntercomInfo data structure://swagger:model
//swagger:model Intercom
type Intercom struct {
ID int64 ‘json:  ‘‘id’’‘
Title string ‘json:  ‘‘title’’‘
BuildingID int64 ‘json:  ‘‘buildingID’’‘
}
```

After completing the above Regintercom API configuration, the following command can be run to generate the swagger.json file, which is the file of this Regintercom API shown in Listing 3.

Listing 3. The command used to generate the swagger.json file.

```
swagger generate spec -m -o swagger.json
```

To make the Regintercom OpenAPI file more readable, running the command (shown in Listing 4) generates a highly readable web page, as shown in Figure 4.

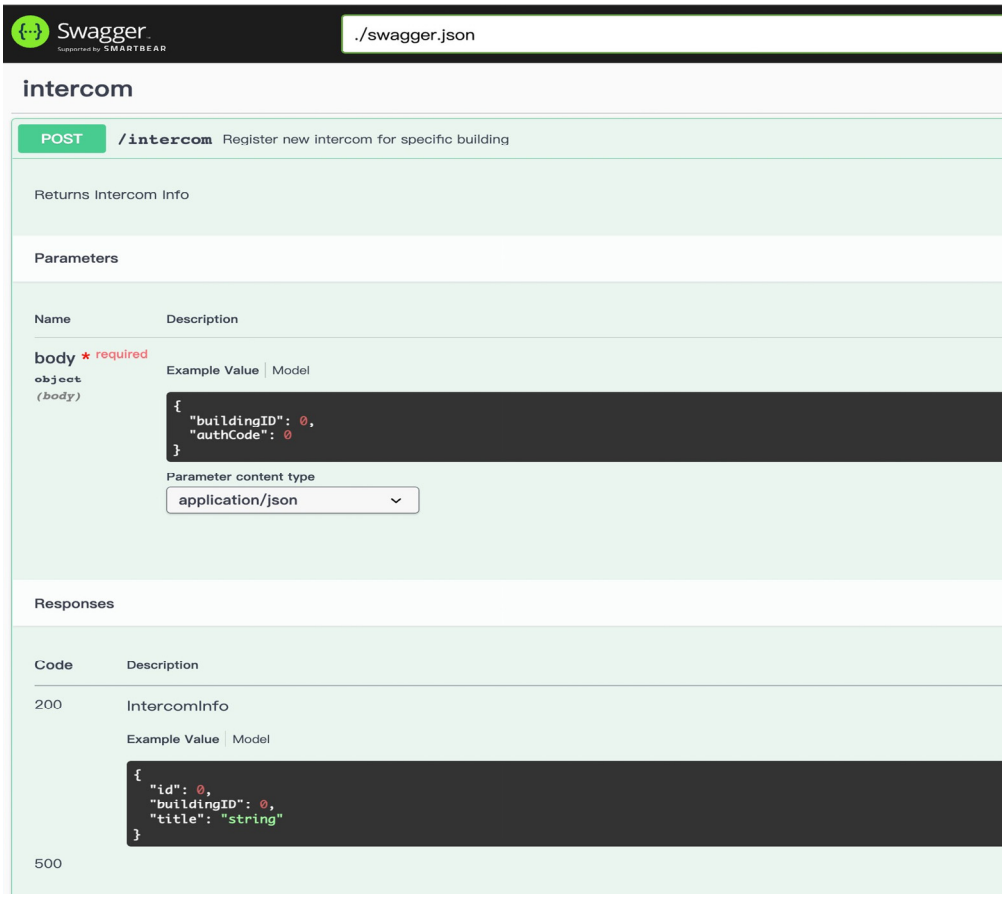


Figure 4. Regintercom OpenAPI is a highly readable web page.

Listing 4. The command used to generate the highly readable web page.

```
    docker run -p 80:8080 -e SWAGGER_JSON=/foo/swagger.json -v
server:/foo swaggerapi/swagger-ui
```

3.4. Cloud of Things

The monolithic system uses a web service approach to directly control the IoT device with a program from the chatbot server on the embedded system. As the system migrates, the associated bot functions are also migrated to the microservice, which connects directly to the embedded system through an IP address to remotely control the GPIO pins to turn on the CoT device. The CoT is a cloud-based platform for high-performance IoT applications that connect our devices and machines, allowing for the remote monitoring and management of CoT devices. The system integrates the security mechanisms of the instant messaging microservice and the instant messaging platform, allowing both parties to exchange messages through a secure authentication process. To ensure that control of remote CoT devices is legal, the system only allows access to the CoT device settings through strict security procedures.

3.5. System Security

A monolithic system provides permission checking by storing user information in a session at login and retrieving user information from there on subsequent accesses. This study migrates a monolithic system into a microservice architecture and integrates the security mechanisms of an instant messaging platform and microservices. The instant messaging software uses the webhook service to connect to the Token-based authentication mechanism of the microservice. The Token is generated by the authentication server built on the IM microservice after the user has been authenticated using the user ID of instant messaging software as the user ID when entering the system. The Token is used as a pass for users to access system resources during system usage, instead of the instant messaging software's user ID, so that users' privacy and security will be fully protected.

The system provides a process for administrators to review users joining the system. Upon receipt of the administrator's approval of the user application, the system will initiate the user registration and authentication process. In addition to completing the registration process of the system, the user will also be required to agree to join the instant messaging software official account of the system through security authentication by the instant messaging platform. Therefore, the system provides a security mechanism between user registration and usage to fully protect the privacy and security of users. The security mechanism of the system is shown in Figure 5.

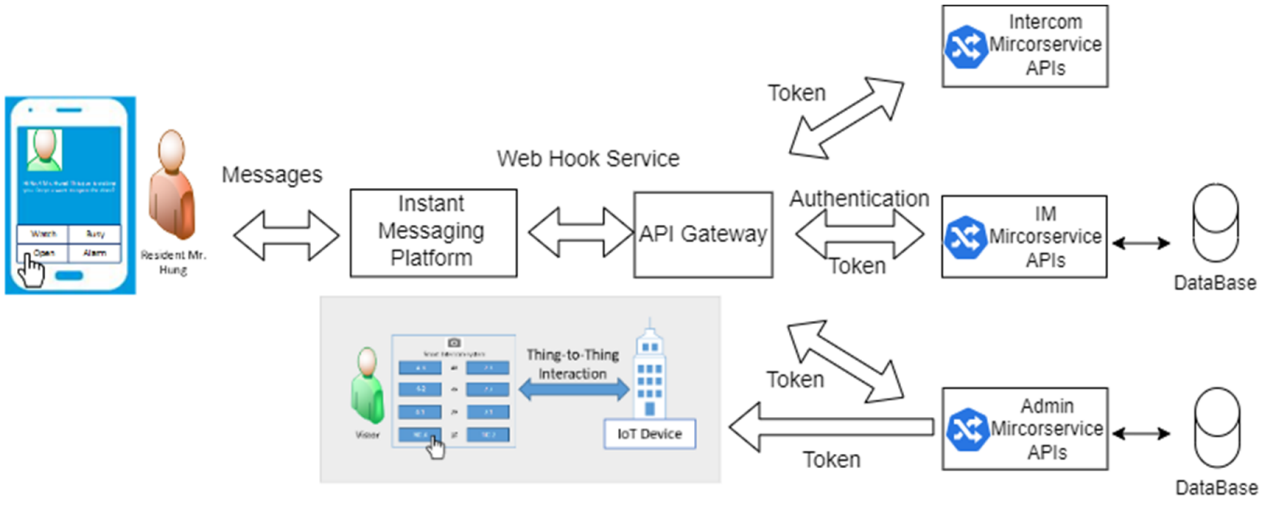


Figure 5. The security mechanism of this system.

Unlike a session ID, a Token is not just a key; it usually contains information about the user, and an authenticated Token can be used to identify the user. The API gateway in this system is the entry point for accessing API privilege control, combined with the authorization server on the IM microservice built into this system. After a validation check with the user ID of the instant messaging software, the authorization server returns an access Token with which access to system resources is indicated as authorized by the system. The API gateway or service-to-service communication goes through a two-way authentication and encryption channel to establish a baseline for normal service and to detect abnormal intrusion. The Token is in JSON format and is used to access resources such as protected APIs, CoT devices, etc. Before accessing these system resources, the Token can then be checked for authenticity using an authentication mechanism, or the Token can be checked for permissions use a pair of public and private keys. It also prevents containers from being directly connected and accessed by a fixed IP location and ensures that containers are secure on the network through proper network environment settings.

The system also allows both parties to pass messages through a secure authentication process by combining the instant messaging microservice with the security mechanisms of the instant messaging platform. When the system handles messages sent from different instant messaging platforms, they are also authenticated by the instant messaging platform. In other words, users from different instant messaging platforms must go through the security authentication process of the instant messaging platform and the security authentication mechanism on the microservice system before they can access the corresponding API.

No user can access user information on other instant messaging platforms through the system. The system provides effective security mechanisms to protect the privacy and security of users. The CoT connection settings are also available to the system through a rigorous security check process as described above, ensuring that the controlling CoT device is system approved.

4. Testing Configuration

In this system, the related functions are broken down into each independent API, and the services are provided by each API in the microservice architecture. When the intercom microservice receives an "alarm" message from the resident through a webhook on an instant messaging platform, it will directly find the "Alarm API" through the API gateway to control the CoT device. The schematic diagram of this system receiving alarm message processing is shown in Figure 6.

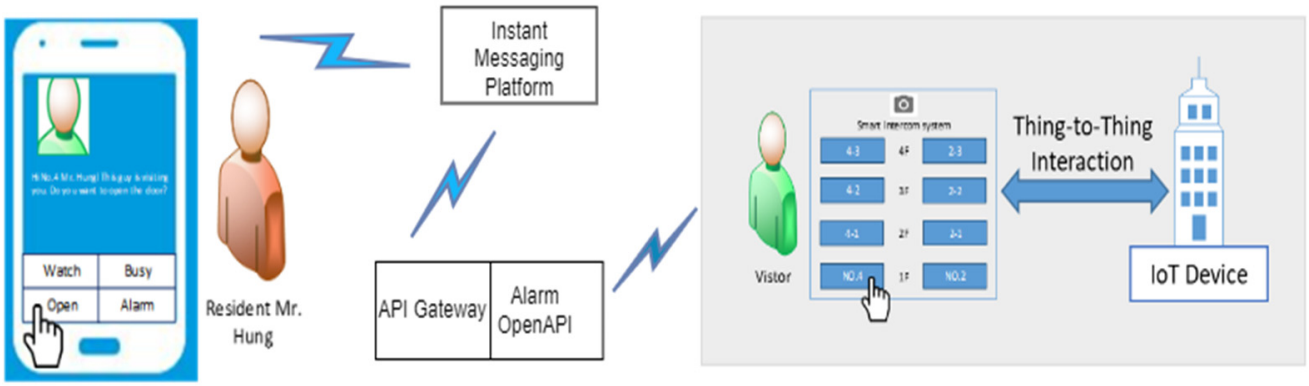


Figure 6. Schematic diagram of this system receiving alarm message processing.

To understand the effectiveness of the smart intercom system in improving the service performance after migrating from a monolithic system to a microservice system, a stress testing tool is used to assess the server performance of HTTPs and web pages. The stress testing tool can simulate many users and record the simulated user test scenarios

individually. By analyzing the test results of the smart intercom system under different architectural conditions, we can understand which system architecture performs the best.

Stress testing is a performance test that investigates how many users the system can handle at a given time by activating simulated users to make various requests simultaneously. Load balancers are used to improve the overall network performance by distributing the request and traffic load among multiple resources.

In this paper, to test the monolithic system and two different load-balancing microservice systems, the monolithic system uses a fixed IP instead of an IP proxy to improve its operational performance. Stress test tools are used to simulate residents using instant messaging software to switch on alarm WoT devices for testing. The two different microservice smart intercom systems were simulated using instant messaging software by a resident pressing the "Alarm" button and using "AlarmOpenAPI" through ingress to trigger the remotely GPIO codes to turn on the buzzer. The flow of the "Alarm" message processing by the system is shown in Figure 7.

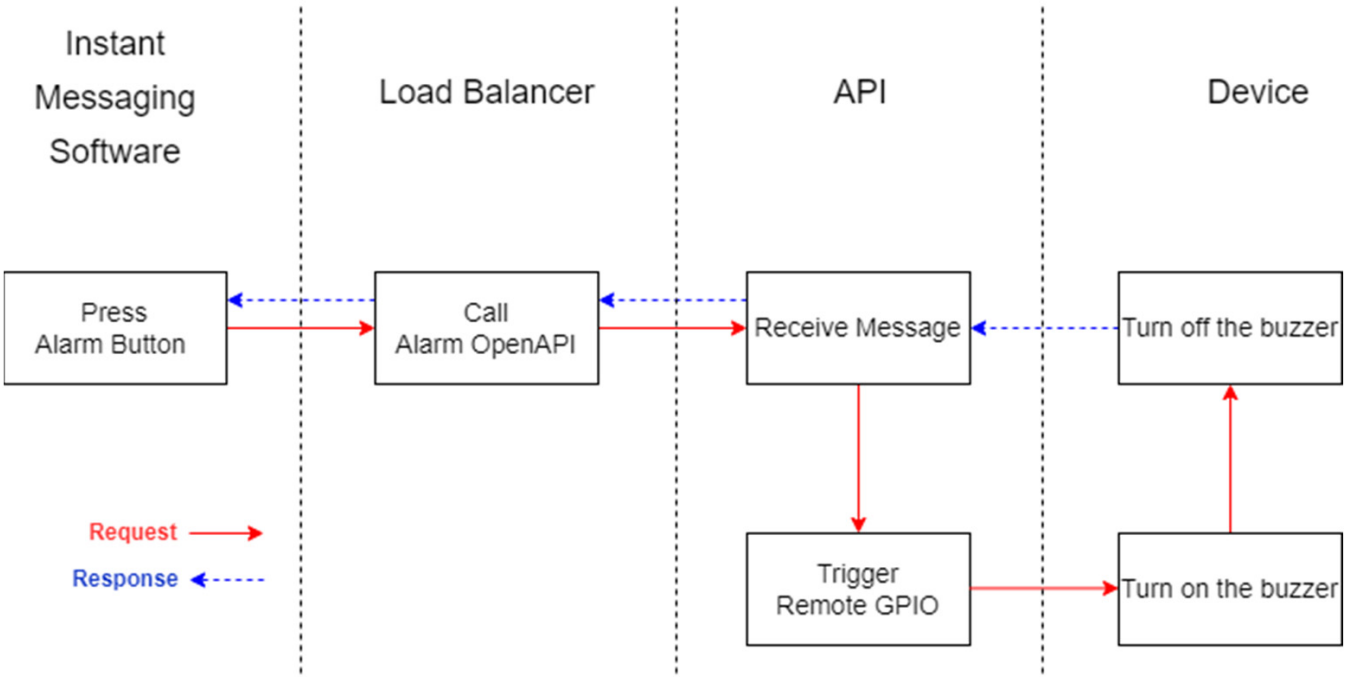


Figure 7. The flow of the "Alarm" message processing by the system.

Since the procedures for controlling the buzzer are the same for the two load-balancing microservice systems and the buzzer is an exclusive device, it is time-consuming to perform the switching operation. Therefore, when testing with the stress test tool, we simulated many residents using the same test script and collected the testing log from the stress test tool to decipher which load balancer performs the best.

This study deploys two load-balancing microservice systems using a native load balancer with a kube proxy and a service mesh load balancer with an istio proxy. The resident presses a button using the instant messaging software, and the instant messaging platform executes a webhook specifying URL that is an entry of Ingress. When Ingress sets up a native load balancer, Ingress finds the API of the dependent message in the corresponding pod for communication, and all pods share the kube proxy, as shown in Figure 8.

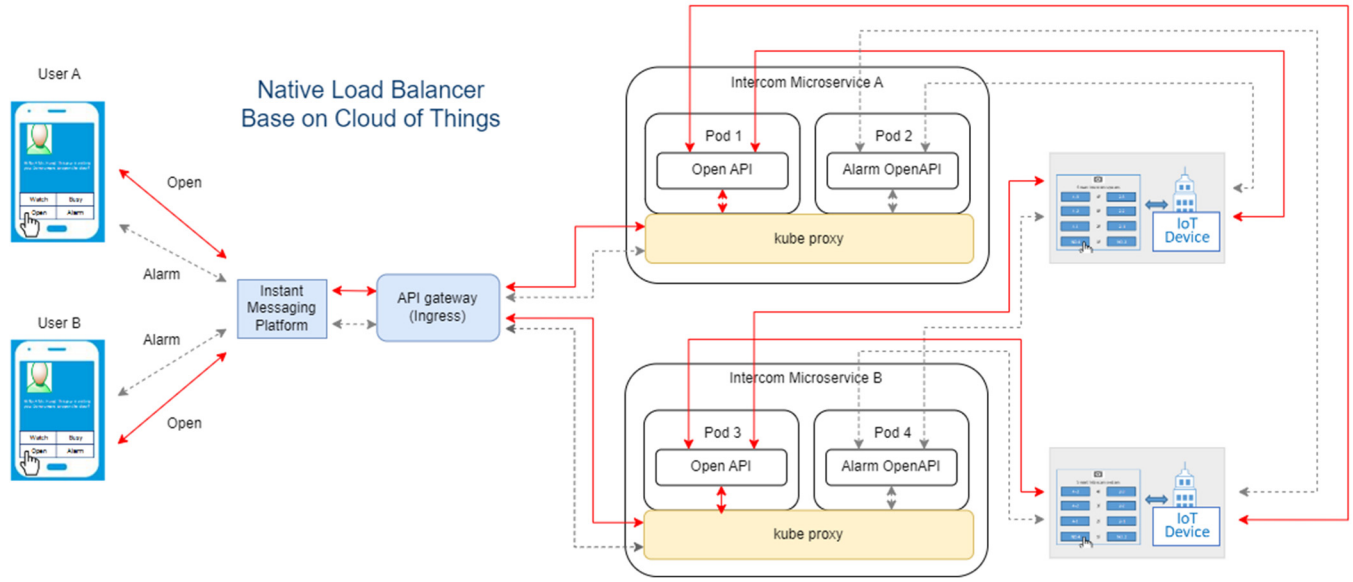


Figure 8. Schematic diagram of the native load balancer.

When Ingress sets up a service mesh load balancer, Ingress finds the API of the dependent message in the corresponding pod for communication, each pod is attached with an istio proxy, and the traffic between istio proxies can be communicated and encrypted, as shown in Figure 9.

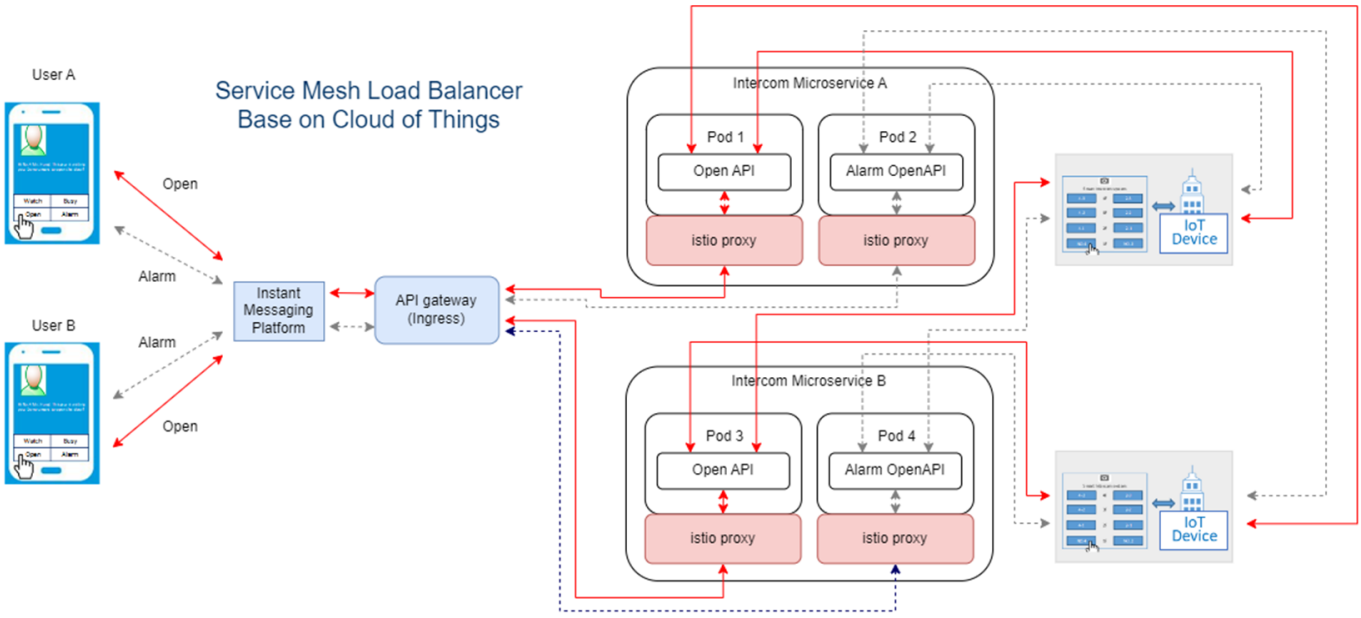


Figure 9. Schematic diagram of service mesh load balancer.

In these two load-balancing microservice architectures, connecting to two smart intercoms with fixed IP addresses makes it possible to access the CoT device.

5. Testing Standards and Results

5.1. Testing Limitation

As the testing environment for this study needs to be conducted in the real world, the following factors may lead to the increased complexity of the testing operations, and therefore trade-offs and limitations are necessary.

- The control of the intercom system by the resident using instant messaging software on the smartphone is often affected by the transmission speed of the network. This test simulates the execution of a stress test command by obtaining the execution script through the instant messaging platform through the actions of a resident. The test was conducted without actually sending the commands from the resident using instant messaging software on the smartphone, and the results were not sent back to the resident.
- The speed of the network transmission of the cloud service platform and the intercom also affected the results. However, the workload on the cloud service platform cannot be controlled. Therefore, this study can only select the test period with more stable data execution based on over 100 experiments. As much as possible, only the smart intercom used the network segment during the test to minimize the impact on the test operation.
- This study used a variety of IP configuration methods for smart intercoms, such as the Ngrok, Node-RED, and DDNS methods. However, all of these methods take a variable amount of time to convert the IP, which has a significant impact on the test results. Therefore, in this study, two smart intercoms were configured in different locations and given fixed IPs. If there is a difference, one of the intercoms is affected by the network environment, and the test work has to be redone.
- Each resident has a different time of use for opening and closing the door. This study uses a system simulation instead of residents and sets the door opening and closing times to 2 s to facilitate comparison of response times for different system architectures.

5.2. Testing Criteria

The stress test tool is used to simulate residents operating the intercom. The stress test tool generates the results of each test case execution during the testing process, and the relevant generated data relationships are described as follows:

$$n = x + y,$$

where n denotes the number of tests, x denotes the number of successful transactions, and y denotes the number of failed transactions.

$$SR = x/n \times 100\%,$$

where S denotes the success rate.

$$AT = \sum_{i=1}^{x} STi,$$

where AT denotes the average successful response time in seconds. STi denotes the ith time in seconds for successful response when $1 \leq i \leq x$.

$$SD = \sqrt{\frac{1}{x-1}\sum_{i=1}^{x}(STi - AT)^2},$$

where SD denotes the standard deviation in seconds. STi denotes the ith time in seconds for successful response when $1 \leq i \leq x$.

$$E = \sum_{i=1}^{x} STi + \sum_{j=1}^{y} FTj + \sum_{k=1}^{x} Z,$$

where E denotes the elapsed times in seconds.
where STi denotes the ith time in seconds for successful response when $1 \leq i \leq x$. FTj denotes the jth time in seconds for failure response when $1 \leq j \leq y$. Z denotes the device operation time.

$$D = \sum_{i=1}^{x} SDi + \sum_{j=1}^{y} FDi,$$

where D denotes the size of the transferred data in bytes, SDi denotes the ith time in bytes for successful response when $1 \leq i \leq x$, and FDj denotes the jth time in bytes for failure response when $1 \leq j \leq y$.

$$TR = n/E,$$

where TR denotes the transaction rate in trans/second.

$$TP = D/E,$$

where T denotes the throughput in bytes/second.

$$LT = \max_{1 \leq \rangle \leq x} S\mathcal{T}\rangle,$$

where LT denotes the longest transaction response time in seconds.

$$ST = \min_{1 \leq \rangle \leq x} S\mathcal{T}\rangle,$$

where ST denotes the shortest transaction response time in seconds.

As the monolithic smart intercom system is installed on an embedded system and can serve only one user at a time, to compare the monolithic system with the two different load-balancing microservice systems, this study simulates the activation of the buzzer 50, 100, 150, and 200 consecutive times using a stress testing tool, whose device rings for 2 s at the end of each time and then proceeds to the next test.

Next, the stress testing tool was used to simulate 50, 100, 150, and 200 users using the same test script of the microservice system simultaneously to compare the native load balancer and the service mesh load balancer. Since the activation buzzer is an exclusive device that runs much longer than the microservice execution times, the actual activation buzzer was not activated in the experiments, and only the time of microservice execution was recorded.

5.3. Testing Results

As the monolithic smart intercom system is installed on an embedded system, it can serve only one user at a time. This test uses a stress test tool to simulate 50, 100, 150, and 200 consecutive activations of the buzzer by one user. The stress test tool records the results of each activation. To understand whether the two load-balancing microservice systems are superior to the monolithic system, relevant experiments were conducted, and the success rate was calculated after calculating the number of successful transactions, which are shown in Table 2.

Table 2. List of stress test success rates for three different architectures.

Architecture Type	50 Times		100 Times		150 Times		200 Times	
	Success Times (x)	**Success Rate (S)**	**Success Times (x)**	**Success Rate (S)**	**Success Times (x)**	**Success Rate (S)**	**Success Times (x)**	**Success Rate (S)**
Service mesh load balancer	50	100%	100	100%	150	100%	200	100%
Native load balancer	50	100%	100	100%	150	100%	200	100%
Monolithic	21	42%	21	21%	21	14%	21	10.5%

The monolithic smart intercom system experienced an exception after 21 activations of the buzzer, while both load-balancing microservice systems were able to provide stable

activation of the CoT device with a 100% success rate. The main reason for this is that both load-balancing microservice systems are deployed in the cloud, which is much richer in resources than a monolithic system deployed on an embedded system. As a result, the stability and performance of both load-balancing microservice systems are much better than those of the monolithic system.

The activation of the buzzer was omitted from the simulation tests due to the time-consuming operation of accessing the buzzer. Therefore, Z was set to 0 in the test criteria. The stress test tool was then used to simulate 50, 100, 150, and 200 users simultaneously using the same test script on both load balancers. After obtaining the test results of the stress test tool, the respective calculations according to the test criteria are shown in Table 3.

Table 3. List of test results for the simultaneous use of different load balancer microservices.

Proxy Type	50 Users		100 Users		150 Users		200 Users	
	Istio	**Kube**	**Istio**	**Kube**	**Istio**	**Kube**	**Istio**	**Kube**
x	50	50	100	100	150	150	200	200
y	0	0	0	0	0	0	0	0
SR	100%	100%	100%	100%	100%	100%	100%	100%
AT	0.2854	0.196	0.4027	0.3348	0.5752	0.4681	0.7763	0.6890
SD	0.0671	0.0202	0.1223	0.0856	0.2055	0.1773	0.2850	0.2756
E	14.2700	9.8000	40.27	33.48	86.2800	70.2100	155.2500	137.7900
D	450	450	900	900	1350	1350	1800	1800
TR	3.5039	5.1020	2.4832	2.9869	1.7385	2.1364	1.2882	1.4515
TP	31.5347	45.9184	22.3491	26.8817	15.6467	19.2280	11.5942	13.0634
LT	0.38	0.24	0.58	0.44	0.88	0.69	1.19	1.04
ST	0.16	0.17	0.1	0.12	0.1	0.11	0.21	0.17

The stress test tool was used to simulate 50 users using the same test script in two load-balancing microservice systems. Based on the test results recorded by the stress test tool, Figure 10 was created according to the test criteria. It was observed that the native load balancer was better than the service mesh load balancer after the 10th user activated the buzzer.

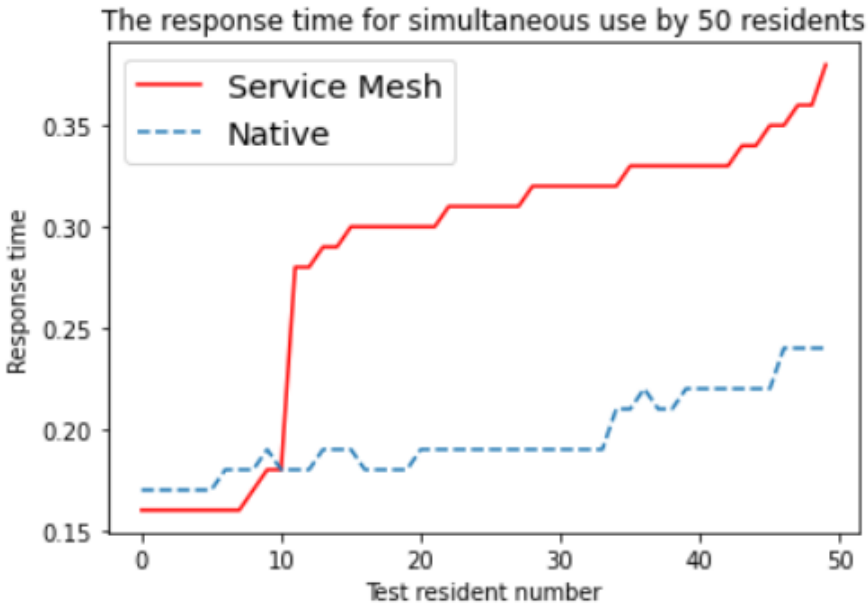


Figure 10. The response time for simultaneous use by 50 residents.

The stress test tool was used to simulate 100 users using the same test script in two load-balancing microservice systems. Based on the test results recorded by the stress

test tool, Figure 11 was created according to the test criteria. The native load balancer was found to outperform the service mesh load balancer.

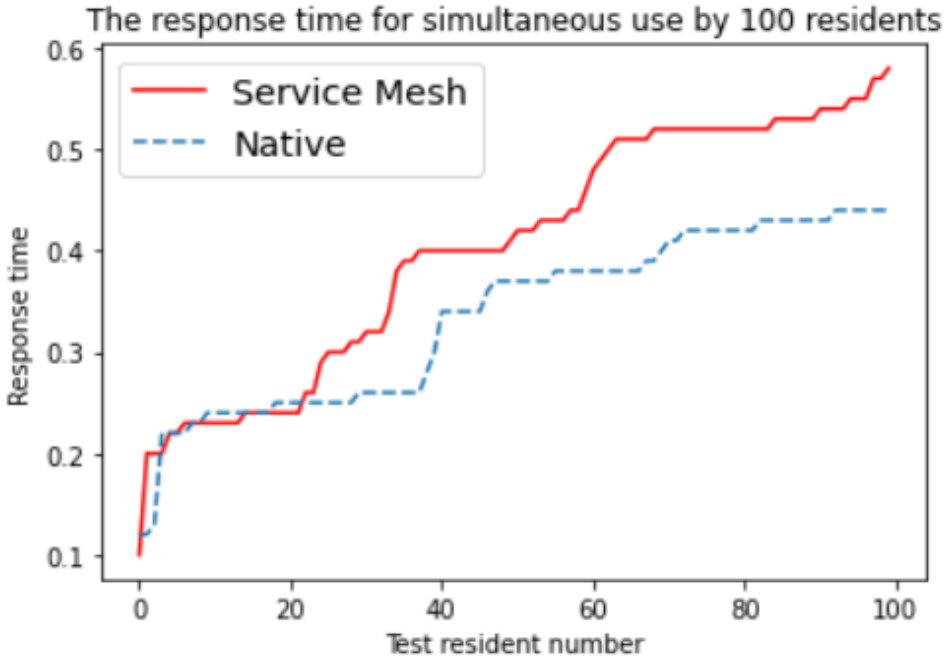


Figure 11. The response time for simultaneous use by 100 residents.

The stress test tool was used to simulate 150 users using the same test script in two load-balancing microservice systems. Based on the test results recorded by the stress test tool, Figure 12 was created according to the test criteria. The native load balancer outperformed the service mesh load balancer.

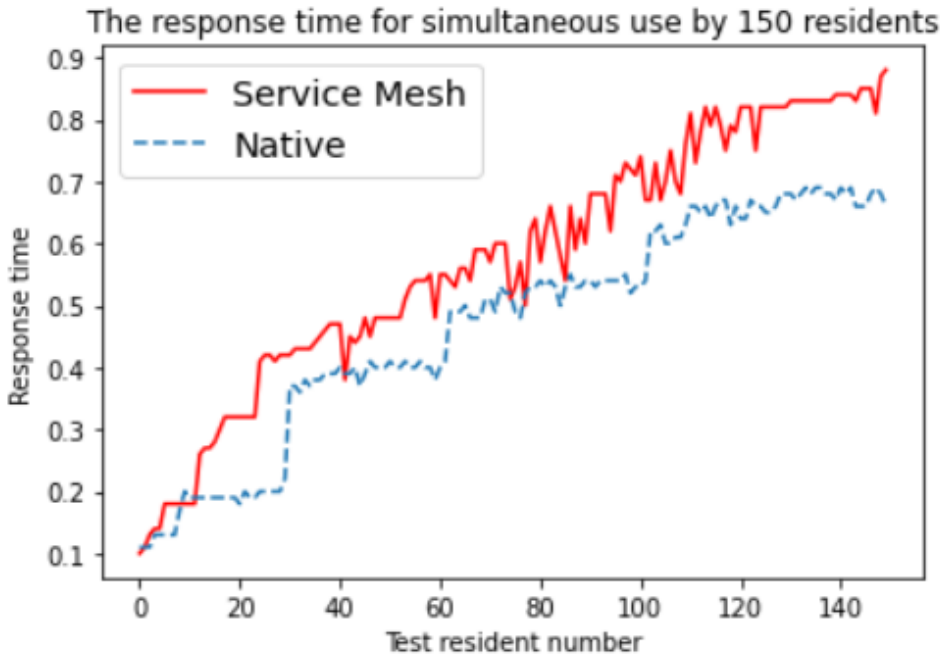


Figure 12. The response time for simultaneous use by 150 residents.

The stress test tool was used to simulate 200 users using the same test script in two load-balancing microservice systems. Based on the test results recorded by the stress

test tool, Figure 13 was created according to the test criteria. The native load balancer outperformed the service mesh load balancer.

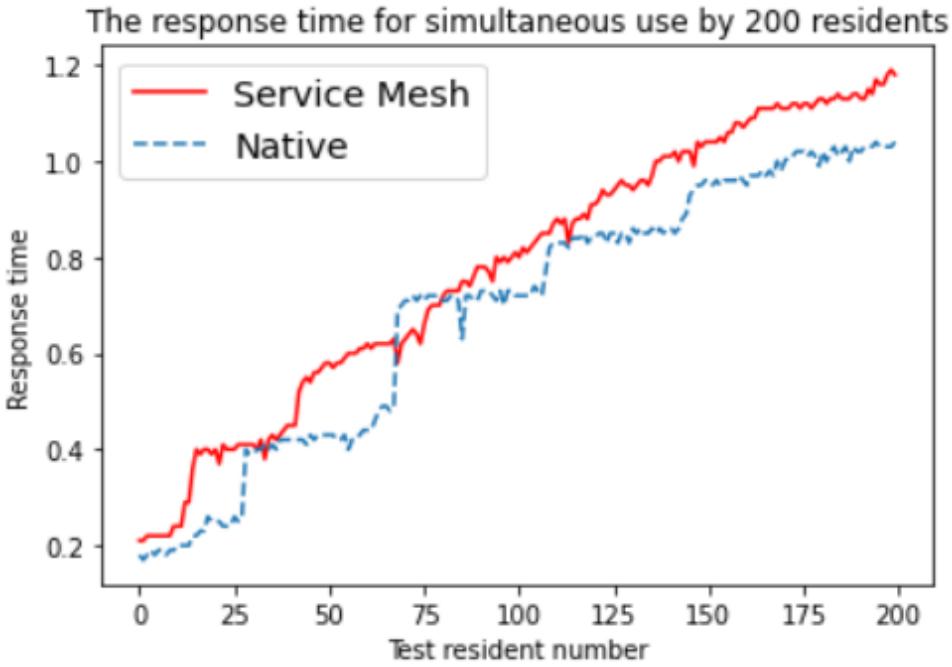


Figure 13. The response time for simultaneous use by 200 residents.

The data presented in Table 3 and the analysis of the four stress tests show that the kube proxy is better than the istio proxy in the microservice architecture, which means that the type of processing tested in this simulation is suitable for execution under a native load balancer with a kube proxy architecture. The more users there are executing simultaneously, the longer the average response time and the larger the standard deviation. In other words, the greater the number of users executing simultaneously is, the worse the execution efficiency of the two microservice architectures.

5.4. System Implementation

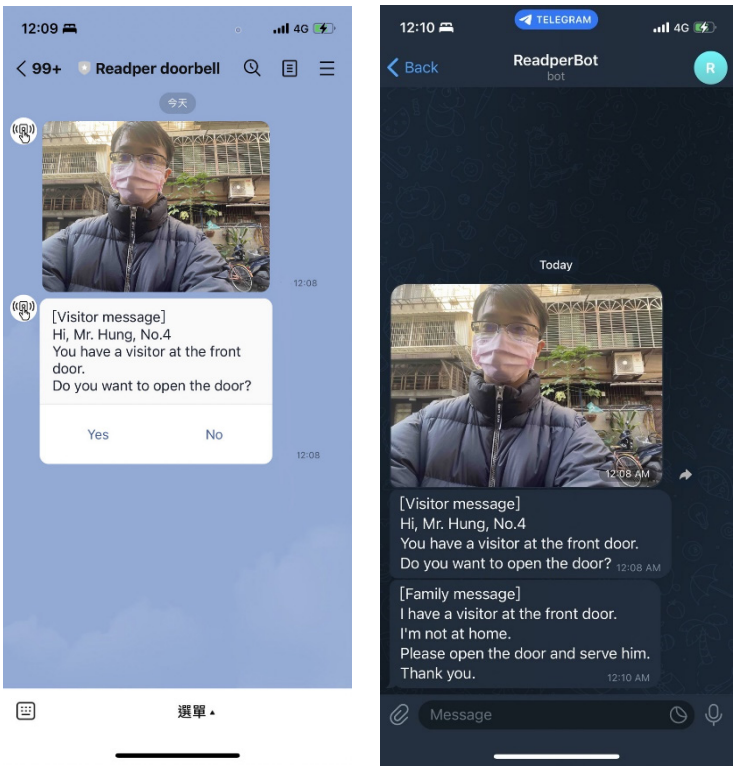
The smart intercom microservice system remotely controls the CoT device using the appropriate designated API and then transmits the execution results to the resident by instant messaging software.

The instant messaging microservice not only transmits system messages, but can also be used to implement platform broadcasts, building broadcasts, unit broadcasts, and family broadcasts to exchange messages between different instant messaging software to enable residents to communicate through the system. The implementation and results of the system are described below.

When a visitor presses a floor button on the intercom and takes a photo, the system looks up the family member to whom the button corresponds through the system. Members will receive the visitor's photo and message, even if they are using different instant messaging software, as shown in Figure 14a,b.

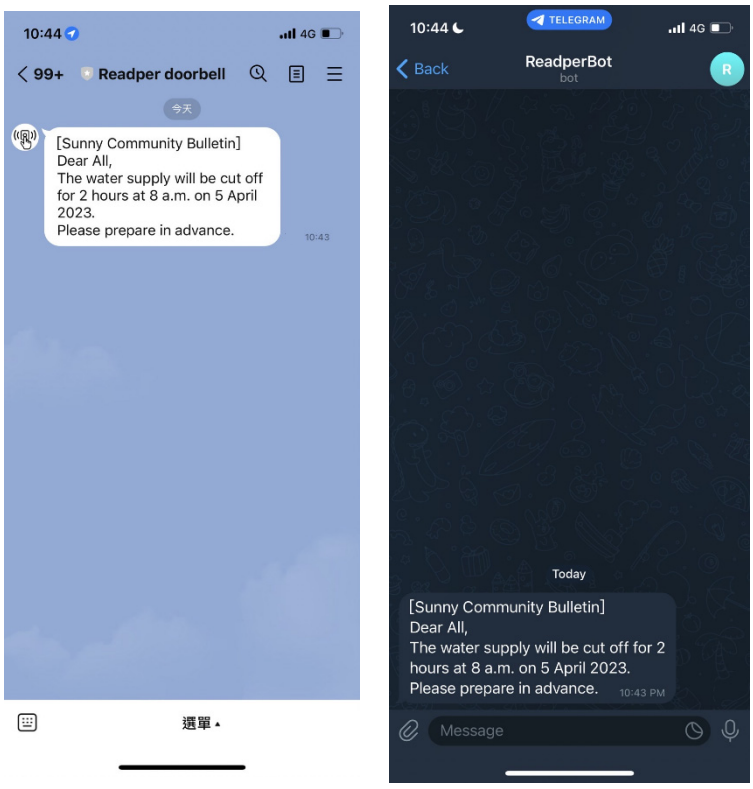
Residents who receive a photo or message will usually ignore it if they do not know the visitor. However, if the resident receiving the message is away and wishes to pass on the message to a family member to assist in receiving the visitor, the recipient can use the system to send a message of assistance to the family member using a different instant messaging software, as shown in Figure 14b.

Suppose the community needs to notify all residents of an emergency. In this case, a community announcement can be sent through the system, and residents can also receive the message using different instant messaging software, as in Figure 15a,b.



(**a**) (**b**)

Figure 14. (**a**) The resident receives messages by Line. (**b**) The resident receives messages by Telegram.



(**a**) (**b**)

Figure 15. (**a**) Residents receive community broadcasts using Line. (**b**) Residents receive community broadcasts using Telegram.

6. Discussion

The system was implemented to migrate from a smart intercom monolithic system to a microservice architecture and integrated with the security mechanism of the instant messaging software platform to make the system more secure. As the monolithic system is installed on an embedded system, a single user will experience a system crash after 21 consecutive uses, which will cause the monolithic system to stop providing services. Two different load-balancing microservice architectures were able to provide stable services after 200 consecutive uses. Not only is it known that microservice architectures using two different load balancers are more stable and scalable when accessing CoT devices, but they are also better than monolithic architectures.

Next, we tested the buzzer or the door lock under two different load-balancing microservice architectures to see the actual response times. Neither of the two load-balancing operations of the microservice architectures activate a buzzer or door lock opening operation. This study uses a stress testing tool to simulate 50, 100, 150, and 200 users simultaneously, requesting specific functions from two microservice architectures with different load balancers to see which one is more suitable for accessing the CoT device.

The two different load balancers of the microservice systems use a native load balancer with a kube proxy and a service mesh load balancer with an istio proxy. According to the average response time and standard deviation of the two architectures based on the data logged by the stress test tool, the kube proxy slightly outperformed the istio poxy for microservice architectures based on CoT. The study identified numerous issues that need to be discussed:

1. This study confirmed that L4 load balancers process requests faster than L7 balancers [48]. A native load balancer with a kube proxy is a network load balancer that can handle only L4 protocols. A service mesh load balancer with an istio proxy is an application load balancer that can handle L7 protocols.
2. The average response time length for both the native load balancer and the service mesh load balancer microservice architectures varied with the number of concurrent residents, as indicated by the simulation using the stress testing tool. In other words, as the number of concurrent residents increases, the average response time increases due to the increase in system load.
3. Although the native load balancer was better than the service mesh load balancer, further analysis of the four stress tests showed that the response times of the two load balancers were similar for the first 12 users. After the 12th user, the native load balancer outperformed the service mesh load balancer. Even so, the native load balancer had sudden increases in response time after approximately the 30th, 65th, and 70th completed users, as shown in Figures 16 and 17.
4. As the instant messaging software does not provide an API to integrate a third-party live calling functionality in the system, the instant messaging software mentioned in this study does not yet include the live calling functionality, which is a potential drawback of the system compared to other intercom systems. However, the system allows residents and visitors to still hear each other's voices by exchanging voice files.
5. Although strict security mechanisms and procedures have been designed to capture the connection settings of CoT devices, and CoT device control instructions are transmitted over an encrypted channel, the security mechanism for controlling CoT devices is still undergoing continuous improvement.

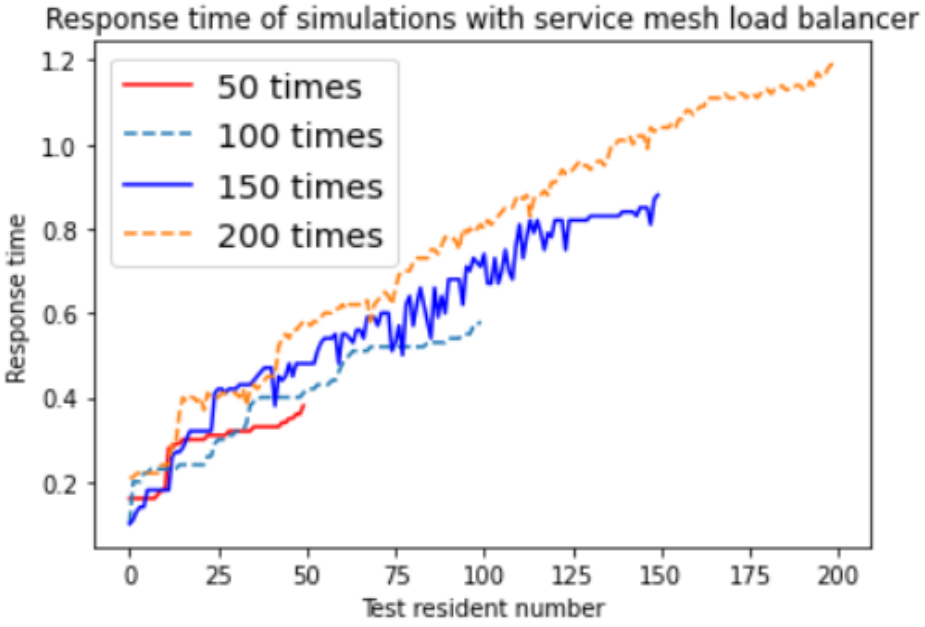


Figure 16. Response time of simulations with service mesh load balancer.

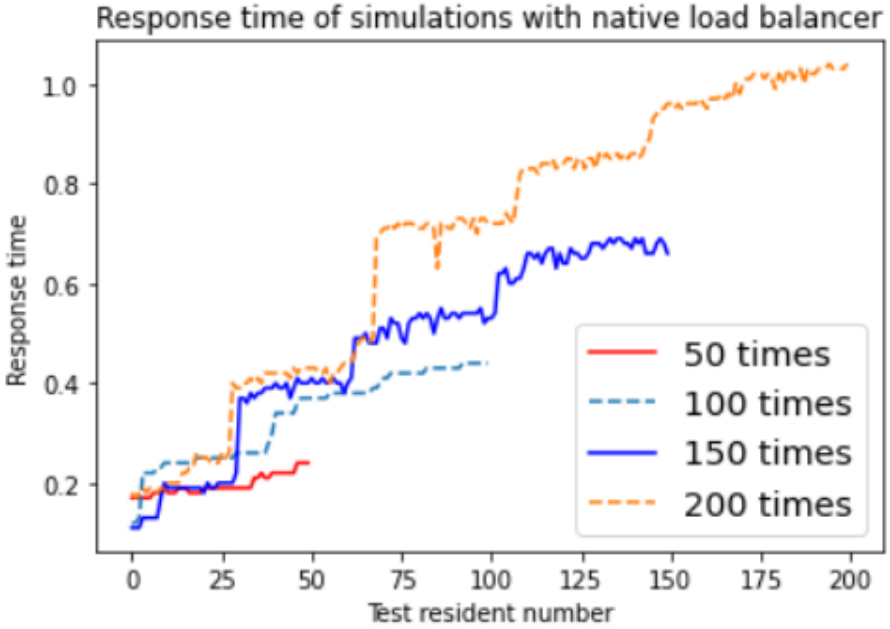


Figure 17. Response time of simulations with native load balancer.

7. Conclusions

This study completed the deployment of a microservice architecture and enhanced related functions based on a monolithic smart intercom system to serve more communities and residents and to enhance the system's security, stability, scalability, and convenience.

An analysis of the results of the implementation and testing proved that the microservices of the smart intercom system are indeed better than those of the monolithic system. The response time of the kube proxy was found to be slightly better than that of the istio proxy by comparing the two different microservice architectures. Furthermore, the system enables residents to use different instant messaging software, thereby increasing the convenience for residents.

The system enables residents to use different instant messaging software and still enables community broadcasts, platform broadcasts, unit broadcasts, and family broadcasts

to exchange messages between residents. This study also developed an OpenAPI for the smart intercom. The smart intercom can use the system's features and resources to provide services through the OpenAPI.

In the future, the system will need to continue to address the issue of instant messaging software not providing the ability for third parties to use live phone calls. Furthermore, although the connection settings for fetching CoT devices undergo a stringent security check process, the security mechanism for controlling CoT devices can still be continuously advanced.

Author Contributions: Conceptualization, H.-Y.H., C.-H.H. and Y.-Y.F.; methodology, H.-Y.H. and Y.-Y.F.; software, H.-Y.H., H.-Y.T. and B.-H.L.; validation, H.-Y.H. and Y.-Y.F.; writing—original draft preparation, H.-Y.H., H.-Y.T. and B.-H.L.; writing—review and editing, H.-Y.H. and Y.-Y.F.; supervision, Y.-Y.F.; funding acquisition, Y.-Y.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially sponsored by the National Science and Technology Council (Taiwan) under Grants MOST 110-2221-E-030-001 and MOST 111-2221-E-030-010.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Newman, S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
2. Huang, H.-Y.; Fanjiang, Y.-Y.; Hung, C.-H.; Lee, C.-A. Design and Implementation of a Smart Intercom System through Web Services on Web of Things. *Telecom* **2022**, *3*, 675–691. [CrossRef]
3. Kazanavičius, J.; Mažeika, D. Migrating legacy software to microservices architecture. In Proceedings of the 2019 Open Conference of Electrical Electronic and Information Sciences (eStream), Vilnius, Lithuania, 25 April 2019; pp. 1–5.
4. De Lauretis, L. From Monolithic Architecture to Microservices Architecture. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 27–30 October 2019; pp. 93–96. [CrossRef]
5. Raharjo, A.B.; Andyartha, P.K.; Wijaya, W.H.; Purwananto, Y.; Purwitasari, D.; Juniarta, N. Reliability Evaluation of Microservices and Monolithic Architectures. In Proceedings of the 2022 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM), Surabaya, Indonesia, 22–23 November 2022; pp. 1–7. [CrossRef]
6. Zhang, H.; Li, S.; Jia, Z.; Zhong, C.; Zhang, C. Microservice architecture in reality: An industrial inquiry. In Proceedings of the 2019 IEEE International Conference on Software Architecture (ICSA), Hamburg, Germany, 25–26 March 2019; pp. 51–60.
7. Hofman, D.; Leu, J.-S.; Troller, P. Evolution from a Door Bell into an IP Door Phone. In Proceedings of the 2019 4th International Conference on Intelligent Green Building and Smart Grid (IGBSG), Yichang, China, 6–9 September 2019; pp. 287–290.
8. Ejidokun, T.O.; Oke, O.J.; Omitola, I.M.; Oduneye, T. A Cost-Effective Two-Way Household Wireless Door Intercom System. *J. Commun.* **2021**, *16*, 379–385. [CrossRef]
9. Sivapriyan, R.; Rao, K.M.; Harijyothi, M. Literature Review of IoT based Home Automation System. In Proceedings of the 2020 Fourth International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 8–10 January 2020; pp. 101–105.
10. Ahtsham, M.; Yan, H.Y.; Ali, U. IoT Based Door Lock Surveillance System Using Cryptographic Algorithms. In Proceedings of the 2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC), Banff, AB, Canada, 9–11 May 2019; pp. 448–453.
11. Baikerikar, J.; Kavathekar, V.; Ghavate, N.; Sawant, R.; Madan, K. Smart Door Locking Mechanism. In Proceedings of the 2021 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE), Navi Mumbai, India, 15–16 January 2021; pp. 1–4.
12. Singh, H.K.; Verma, S.; Pal, S.; Pandey, K. A step toward Home Automation using IOT. In Proceedings of the 2019 Twelfth International Conference on Contemporary Computing (IC3), Noida, India, 8–10 August 2019; pp. 1–5.
13. Mustafa, B.; Iqbal, M.W.; Saeed, M.; Shafqat, A.R.; Sajjad, H.; Naqvi, M.R. IOT Based Low-Cost Smart Home Automation System. In Proceedings of the 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 11–13 June 2021; pp. 1–6.

14. Thirrunavukkarasu, R.R.; Kumar, S.M.; Praveen, P.; Devi, T.M.; Pradeep, S.; Prabu, S.G. Customization In Home Automation Using IoT. In Proceedings of the 2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 19–20 March 2021; pp. 1062–1067.
15. Lee, Y.-S.; Fanjiang, Y.-Y.; Hung, C.-H.; Li, W.-D.; Zhang, T.-M. Design and Implement the Convenient Home Appliances Control with Instant Messaging Software. In Proceedings of the 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE), Kobe, Japan, 13–16 October 2020; pp. 604–605.
16. Ma, L.; Li, Z.; Zheng, M. A Research on IoT Based Smart Home. In Proceedings of the 2019 11th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Qiqihar, China, 28–29 April 2019; pp. 120–122.
17. Xu, R.; Jin, W.; Hong, Y.; Kim, D.-H. Intelligent Optimization Mechanism Based on an Objective Function for Efficient Home Appliances Control in an Embedded Edge Platform. *Electronics* **2021**, *10*, 1460. [CrossRef]
18. Chomklin, A.; Tanthavech, N.; Pakornmanokul, S. Prototype of Air Conditioners Control Systems via Line Chatbot using Raspberry Pi. In Proceedings of the 2021 6th International Conference on Business and Industrial Research (ICBIR), Bangkok, Thailand, 20–21 May 2021; pp. 1–6.
19. Yue, C.Z.; Ping, S. Voice activated smart home design and implementation. In Proceedings of the 2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST), Shenzhen, China, 14–16 April 2017; pp. 489–492.
20. Isyanto, H.; Arifin, A.S.; Suryanegara, M. Performance of Smart Personal Assistant Applications Based on Speech Recognition Technology using IoT-based Voice Commands. In Proceedings of the 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Republic of Korea, 21–23 October 2020; pp. 640–645.
21. Gondkar, S.S.; William, P.; Pardeshi, D.B. Design of a Novel IoT Framework for Home Automation using Google Assistant. In Proceedings of the 2022 6th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 25–27 May 2022; pp. 451–454.
22. Hamdan, O.; Shanableh, H.; Zaki, I.; Al-Ali, A.R.; Shanableh, T. IoT-Based Interactive Dual Mode Smart Home Automation. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics (ICCE), Las Vegas, NV, USA, 11–13 January 2019; pp. 1–2.
23. Huang, H.-Y.; Fanjiang, Y.-Y.; Hung, C.-H.; Lee, C.A. Design and Implement a Smart Intercom System with Remote Interactive Control. In Proceedings of the 2020 IEEE 9th Global Conference on Consumer Electronics (GCCE), Kobe, Japan, 13–16 October 2020; pp. 584–585.
24. Velepucha, V.; Flores, P. Monoliths to microservices—Migration Problems and Challenges: A SMS. In Proceedings of the 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST), Quito, Ecuador, 23–25 March 2021; pp. 135–142.
25. Kwon, D.; Ok, K.; Ji, Y. IBFRAME: IoT Data Processing Framework for Intelligent Building Management. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 5233–5238. [CrossRef]
26. Márquez, G.; Taramasco, C.; Astudillo, H.; Zalc, V.; Istrate, D. Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System. *IEEE Access* **2021**, *9*, 9411–9428. [CrossRef]
27. Kalubi, N.; Sajal, S. Cloud Computing: Arduino Cloud IoT Integration with REST API. In Proceedings of the 2022 IEEE International Conference on Electro Information Technology (eIT), Mankato, MN, USA, 19–21 May 2022; pp. 473–476. [CrossRef]
28. Larrinaga, F.; Ochoa, W.; Perez, A.; Cuenca, J.; Legaristi, J.; Illarramendi, M. Node-RED Workflow Manager for Edge Service Orchestration. In Proceedings of the NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 25–29 April 2022; pp. 1–6. [CrossRef]
29. Miyagoshi, K.; Teranishi, Y.; Kawakami, T.; Yoshihisa, T.; Shimojo, S. Proposal of a Logical Sensor Architecture using WoT-Based Edge Microservices. In Proceedings of the 2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC), Madrid, Spain, 13–17 July 2020; pp. 1223–1228. [CrossRef]
30. Aazam, M.; Khan, I.; Alsaffar, A.A.; Huh, E.-N. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of the 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST), Islamabad, Pakistan, 14–18 January 2014; pp. 414–419.
31. Eugster, P.; Kumar, S.; Savvides, S.; Stephen, J.J. Ensuring Confidentiality in the Cloud of Things. *IEEE Pervasive Comput.* **2019**, *18*, 10–18. [CrossRef]
32. Nguyen, D.C.; Pathirana, P.N.; Ding, M.; Seneviratne, A. Integration of Blockchain and Cloud of Things: Architecture, Applications and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 2521–2549. [CrossRef]
33. Aazam, M.; Islam, S.U.; Lone, S.T.; Abbas, A. Cloud of Things (CoT): Cloud-Fog-IoT Task Offloading for Sustainable Internet of Things. *IEEE Trans. Sustain. Comput.* **2022**, *7*, 87–98. [CrossRef]
34. Karn, R.R.; Das, R.; Pant, D.R.; Heikkonen, J.; Kanth, R. Automated Testing and Resilience of Microservice's Network-link using Istio Service Mesh. In Proceedings of the 2022 31st Conference of Open Innovations Association (FRUCT), Helsinki, Finland, 27–29 April 2022; pp. 79–88. [CrossRef]
35. Shitole, A.S. Dynamic Load Balancing of Microservices in Kubernetes Clusters Using Service Mesh. Master's Thesis, National College of Ireland, Dublin, Ireland, 2022.
36. Niu, Y.; Liu, F.; Li, Z. Load Balancing Across Microservices. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 198–206. [CrossRef]

37. Yu, R.; Kilari, V.T.; Xue, G.; Yang, D. Load Balancing for Interdependent IoT Microservices. In Proceedings of the IEEE INFOCOM 2019—IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 298–306. [CrossRef]
38. Song, M.; Liu, Q.; Haihong, E. A MircoService Tracing System Based on Istio and Kubernetes. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019; pp. 613–616. [CrossRef]
39. He, X.; Deng, F. Research on Architecture of Internet of Things Platform Based on Service Mesh. In Proceedings of the 2020 12th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), Phuket, Thailand, 28–29 February 2020; pp. 755–759. [CrossRef]
40. Ferreira, L.C.; Borchardt, A.D.R.; Cardoso, G.D.S.; Lemes, D.A.M.; de Sousa, G.R.D.R.; Neto, F.B.; de Lima, E.R.; Fraidenraich, G.; Cardieri, P.; Meloni, L.G.P. Edge Computing and Microservices Middleware for Home Energy Management Systems. *IEEE Access* **2022**, *10*, 109663–109676. [CrossRef]
41. Koyama, T.; Kushida, T. Log message with JSON item count for root cause analysis in microservices. In Proceedings of the 2023 6th Conference on Cloud and Internet of Things (CIoT), Lisbon, Portugal, 20–22 March 2023; pp. 55–61. [CrossRef]
42. Huang, Y.-W.; Ma, S.-P.; Wang, S.K. MsdoBot: An Extensible Chabot Platform for Microservice Development and Operations. In Proceedings of the 2022 IEEE International Conference on e-Business Engineering (ICEBE), Bournemouth, UK, 14–16 October 2022; pp. 124–129. [CrossRef]
43. Kaur, G.; Thangaraju, B. Event Driven Microservices based Information Bot. In Proceedings of the 2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT), Bangalore, India, 8–10 July 2022; pp. 1–5. [CrossRef]
44. Muslih, M.; Supardi, D.; Multipi, E.; Nyaman, Y.M.; Rismawan, A. Developing Smart Workspace Based IOT with Artificial Intelligence Using Telegram Chatbot. In Proceedings of the 2018 International Conference on Computing, Engineering, and Design (ICCED), Bangkok, Thailand, 6–8 September 2018; pp. 230–234.
45. Ahmed, S.; Paul, D.; Masnun, R.; Shanto, M.U.A.; Farah, T. Smart Home Shield and Automation System Using Facebook Messenger Chatbot. In Proceedings of the 2020 IEEE Region 10 Symposium (TENSYMP), Dhaka, Bangladesh, 5–7 June 2020; pp. 1791–1794.
46. Gos, K.; Zabierowski, W. The Comparison of Microservice and Monolithic Architecture. In Proceedings of the 2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH), Lviv, Ukraine, 22–26 April 2020; pp. 150–153. [CrossRef]
47. Blinowski, G.; Ojdowska, A.; Przybyłek, A. Monolithic vs. Microservice Architecture: A Performance and Scalability Evaluation. *IEEE Access* **2022**, *10*, 20357–20374. [CrossRef]
48. Hosseini, S.M.; Jahangir, A.H.; Daraby, S. Session-persistent Load Balancing for Clustered Web Servers without Acting as a Reverse-proxy. In Proceedings of the 2021 17th International Conference on Network and Service Management (CNSM), Izmir, Turkey, 25–29 October 2021; pp. 360–364.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.