IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DATA MINING FOR INTERNET OF THINGS

Received January 30, 2020, accepted February 11, 2020, date of publication February 21, 2020, date of current version March 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2975708

An Implementation of High Efficient Smart Street Light Management System for Smart City

YU-SHENG YANG[1], SHIH-HSIUNG LEE[2], GUAN-SHENG CHEN[3], CHU-SING YANG[3], (Member, IEEE), YUEH-MIN HUANG[1], (Senior Member, IEEE), AND TING-WEI HOU[1], (Member, IEEE)

[1]Department of Engineering Science, National Cheng Kung University, Tainan 701, Taiwan
[2]Department of Intelligent Commerce, National Kaohsiung University of Science and Technology, Kaohsiung 824, Taiwan
[3]Institute of Computer and Communication Engineering, National Cheng Kung University, Tainan 701, Taiwan

Corresponding author: Shih-Hsiung Lee (beargoer@gmail.com)

ABSTRACT Street light are among the most common infrastructure in cities. Street lights and sensors can be combined to generate an interface of data collection. The analysis of massive data serves as an integral element of a smart city. This paper proposes a highly efficient system for the configuration, deployment, and management of smart street lights. The features of fast deployment and high scalability of the container-based system management result in virtual deployment. Additionally, for database design, NoSQL and in-memory databases are integrated to realize flexible data management. In terms of data transmission, this paper designs an asymmetric key and an SSH encrypted tunnel. Moreover, when all the services are connected, it conducts legitimacy validation via a token. Therefore, this system can help meet the demands for data throughput, low-latency, configuration, and realization of a smart city. It boasts high efficiency and security. Besides, it offers a flexible data storage and management service to facilitate the massive data processing of a smart city. With respect to experiments, this paper designs a street lighting simulation system with edge computing devices (consisting of a micro-controller, a sensor, and an IP camera) and a street lighting function. The system collects real-time sensed environmental data, enables live streaming of images, and offers an API for historical data query. This paper utilizes container-based virtualization to deploy all edge computing devices on the server and validates the feasibility of simultaneous operation of multiple container-based services on edge computing devices. This system has high commercial value.

INDEX TERMS Street light management system, containerization, live streaming, SSH tunnel.

I. INTRODUCTION

Currently, over 50% of the world's population resides in cities [1]–[3]. On average, one person connects over six smart devices to the Internet [4]. In other words, billions of devices and systems are deployed in the urban infrastructure, from end-user equipment to urban infrastructure systems, such as smart street lighting, road management, pedestrian management, noise/air quality monitoring, waste management, and smart medical systems, forming a huge network ecosystem where all things are connected. Consequently, tremendous amounts of management data are generated for the security of urban residents, management of efficiency, productivity, and quality of life, and infotainment applications and services.

A smart city employs new computing technologies and communication resources to integrate, manage, and analyze tremendous amounts of data and achieve mutual benefits among urban applications, including: smart economy, smart governance, smart citizens, smart mobile, smart environment, and smart life [5]. In line with the definition of a smart city [6], elements of a smart city will become increasingly diversified along with the advances of science and technology. From the perspective of data management, a smart city integrates data analysis and processing with data security and privacy measures, and encourages application innovation to raise the overall quality of life of its citizens. A unified data management framework is vital for a smart city and its applications [7]. For example, street lighting and air quality monitoring are integrated. Air quality sensors are deployed on street lights to continuously and intensively monitor the air quality of urban

The associate editor coordinating the review of this manuscript and approving it for publication was Mu-Yen Chen.

areas. Data on atmospheric composition and air quality are collected everywhere at any time. Such data are analyzed and reported to local environmental protection departments for effective inspection and tracking. Thus, the overall quality of urban life can be improved. Characteristics of application development and examples of smart cities in the world are summarized in [8]. Souri *et al.* [74] proposed a systematic analysis of communication strategies for the Internet of Things. The purpose is to enable the device to connect and respond more quickly and flexibly. In analysis, there are four main directions: device-to-device, device-to-cloud, device-to-gateway and device-to-application. And analyze the existing papers of the Internet of Things and divide the technology into five categories including monitoring-based communications, routing-based communications, health-based communication, intrusion-based communication and resource-based communication. For resource management, Arani *et al.* [77] put forward a complete discussion of resource management on fog computing. Due to the limitations, heterogeneity and dynamics of resources on IoT devices, and the unpredictability of the fog environment, resource management is one of the challenging topics. Reference [77] divides the resource management methods into six categories including application placement, resource scheduling, task offloading, load balancing, resource allocation, and resource provisioning. This provides us with a good evaluation index and test method. In addition, Petritoli *et al.* [75] proposed a case comparison of the energy performance of smart lighting in smart cities. According to the Smart Street pilot system in Rome and the adaptive configuration of traffic flow to achieve the best energy efficiency. Mohandas *et al.* [76] further put forward the use of artificial neural network algorithms to effectively adjust the lighting system. Reference [76] used the lighting sensor, motion sensor, PIR sensor, artificial neural network and fuzzy logic controller into a five-level configuration scheme, ultimately reducing unnecessary utilization by 34% and reducing power consumption 13.5%. Therefore, the intelligent adjustment of power consumption is one of the important issues on the street light system. Our work focuses on container-based virtualization technology Docker to provide a powerful and highly scalable solution to deploy cloud and edge services. Our design scheme protects the communication between the edge and the cloud, including token verification and SSH-based encryption (with public key authentication). The system proposed in this paper is modular, scalable, easy to deploy, and security-oriented. In addition, the market forecast report of the Northeast Group reveals that more and more street lighting projects are connected [8]. Communication component suppliers and three of the four major street light suppliers in the world are no longer just communication module and light manufacturers. Therefore, smart street lights are regarded as a part of the concept of a smart city.

Generally, a street lighting management system is composed of three main parts: smart street lights, network infrastructure and management, and the control system [9]. This paper focuses on the following three parts: real-time lighting control is realized by manual setting or scheduling; real-time information, including the status of the lighting system, sensed environmental information, streaming of real-time images of the IP camera, is provided; and an API for historical data query is provided (including video storage). The system architecture proposed by this paper is mainly made up of a management platform, edge service orchestrators, and a web-based user interface. It is highly scalable, security- and privacy-oriented, and user-friendly. Along with the rapid development of smart street lights, the management system requires ever more computing, networking, and storage resources to simultaneously handle the many requests issued by street lights. Therefore, it is necessary to build a highly scalable system featuring easy migration, rapid deployment, and high resource utilization. This paper took advantage of container-based virtual deployment technology to realize rapid deployment and high scalability. Besides, with respect to database design, it utilized NoSQL and an in-memory database to achieve flexible data management. In accordance with the latest OWASP IoT Top 10 [10], more attention has been paid to insecure ecosystem interface and insecure data transmission (i.e., without encryption or access control). A security lapse not only results in wrong applications and services, but also unexpectedly becomes an intermediary for cyberattacks. For instance, in the 2016 Dyn cyberattack [11], a series of distributed denial-of-service (DDoS) attacks took advantage of substantial IoT devices infected with the Mirai malware. [12]. In line with the General Data Protection Regulation (GDPR), privacy management must be integrated into the solution for achieving the development of smart cities in Europe [13]. Therefore, when establishing a management system, one needs to consider how to protect and validate data transmission. This paper designed an asymmetric key and an SSH encrypted tunnel for data transmission. Moreover, it conducted legitimacy validation via a token when all services were connected. The edge processing function may be deployed on edge nodes with heterogeneous functions [14]. Therefore, this paper designed a program that facilitates the user's deployment of edge nodes, which was a great challenge. Furthermore, many aspects need to be considered in designing a practical remote observation and control system, such as a user-friendly interface, real-time feedback on status, and multi-user accessibility [15]. Another challenge lies in the diversity and number of streaming video devices. Hence, an effective edge node deployment mechanism and a user-friendly interface are essential for achieving an effective management system.

The rest of this paper is organized as follows. Section 2 discusses related works and introduces background concepts and technologies. Section 3 describes the entire system architecture and provides the implementation details of the cloud management platform, the edge node services, and the web-based user interface. Section 4 presents the results of the experiment and its verification. Finally, we conclude this paper.

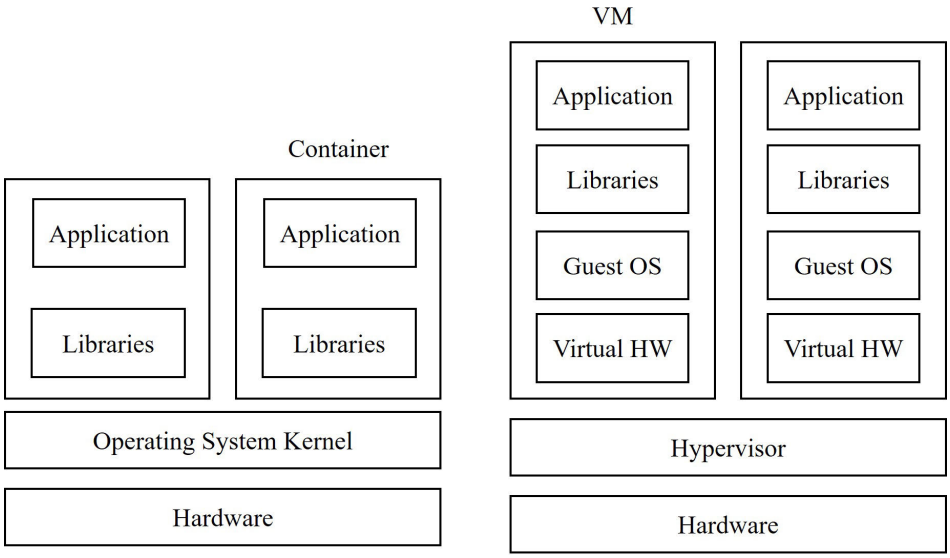


FIGURE 1. Architecture comparison of virtual machines and containers.

II. RELATED WORKS

A. SMART STREET LIGHT

Smart street light is one of the most significant applications in a smart city. Automatic switch of street lights for this pervasive infrastructure, but also take into account the connected urban digital platform. Álvarez *et al.* [16] proposed a taxonomy of use cases based on sensing technologies, databases, and actuation purposes. However, the identification of new creative cases requires the combination of multiple data streams. The first challenge was to enable street lights to collect data via different types of sensors and provide such data to a public digital platform. Most efforts spent on the building of a smart street lighting management system in recent years centered on the smart switch of LED street lights. Reference [17] integrated street lighting control with the current SmartGrid products of Portugal. In the technical abstraction layer, the lighting communication table was used to reduce the changes in circuit via a Web-based central management system. Daely *et al.* [18] put forward an architecture consisting of ZigBee-based wireless communication, a turnable correlated color temperature (CCT)-based LED array, and a central web server. This server can receive weather information and real-time sensed data from each lamp pole. In addition, this system switches the CCT to 5,000K or 3,000K based on the weather conditions from the weather API. In case of a traffic accident caused by low visibility due to fog, the system will recommend the use of a weak CCT light to reduce the possibility of such situation. Jia *et al.* [19] proposed a management platform based on a fog computing server with improved real-time response. Street lights periodically send their status to the server through NB-IoT network technology. However, the system could not guarantee 100% reliability. Its simulation results demonstrated that the average periodic maintenance time and the abnormal state of the server was approximately 20 minutes. Escrivá *et al.* [20] put forward a prototype architecture based on IoT devices and cloud computing. The modules included: a dedicated wireless lighting controller, one IoT sensor gateway set installed on LED lights, a smart collector, and a central management system. Through Apache Kafka, Apache Flink, and MongoDB, the smart collector analyzed and stored the data collected from the sensors. The smart central management system provides a Web interface that uses the real-time REST Web service and the IoT gateway to control and monitor street lights, including the status and location of lights. The difference between this paper and the existing researches lies in the fact that the former attaches more importance to scalable cloud and edge deployment, security against cyberattacks, and user-friendly Web-based interface.

B. VIRTUALIZATION TECHNOLOGY-CONTAINERS AND VIRTUAL MACHINES

Virtualization simplifies the replication and scaling of applications [21]. There are two main virtualization technologies: hardware-level virtualization and operating system-level virtualization. The container, using operating system-level virtualization, executes processes in the operating system. It uses namespace [21] to handle the resource isolation of each process (e.g. process ID, network interface, and mount point) and manage CGroups [23]. A virtual machine (VM) is independent of the operating system; it simulates VM management processes (e.g. VMware ESXi [24] and KVM [25]) on virtual hardware (e.g. CPU, memory, and network devices). Containers and virtual machines are compared in terms of architecture in Figure 1.

Docker [26] introduced an advanced API. It used a lib container-based [27] solution to create a container with Linux kernel virtualization. Its first version was released as an open source in March 2013. Since then, many applications have been shifted from VMs to containers. According to

TABLE 1. Comparison of docker and virtual machines. [73].

Feature	Docker	Virtual Machines
Start time	<50 ms	30 - 45 seconds
Stop time	<50 ms	5 - 10 seconds
System Overhead	No overhead	Overhead due to hypervisor
Storage space	Tens of MBs in size	Tens of GBs in size
Scalability	Highly scalable	Not easily scalable
CPU load when idle	Normal	~1.5% more than docker
Isolation	Less isolation due to software virtualization	More isolation due to hardware virtualization
Network round trip latency	~75 µs	~60 µs
I/O throughput (Read and Write)	100000 I/Os per second	50000 I/Os per second

Docker's blog [28] in March 2018, 3.5 million Dockerized applications and 37 billion containers have been downloaded. The main differences between Docker and VM are shown in Table 1. Previous studies [21], [29], [30] explored the relationship between containers and VMs. Sharma *et al.* [21] compared the performance of intensive workload between KVM and LXC [31] (another OS-level virtualization relying on the Linux kernel). Their results displayed that the performance overhead of KVM is negligible in CPU, memory, and network-intensive workload, but high in I/O-intensive applications because I/O is based on the hypervisor. Additionally, an interesting architecture different from a VM was considered; it enables the nesting of a container on a VM for the isolation of applications. Similar to a VM, it restricts soft container resources to slightly improve performance. Zhang *et al.* [29] compared the convenience of deployment, initiation efficiency, and the performance of Spark jobs (e.g. execution of Kmeans, logistic regression, page ranking, and SQL Join) between Docker container clusters and VM clusters. Their measurements showed that containers are more convenient in deployment and initiation. Containers have better scalability among different and multiple workflows and higher resource utilization in one workflow in the big data environment. Maheshwari *et al.* [30] deemed that the time, CPU, network, and disk I/O resource utilization of the Docker container required for start and stop are lower than those of Oracle VirtualBox [32]. However, they held that compared with containers, VMs enable greater isolation and provide better support for real-time migration.

Recent studies [14], [33]–[38] reveal the growing trend of Docker services in the IoT environment; they probed into the balance between flexibility and performance overhead. Bellavista and Zanni [33] containerized the IoT framework of open source codes and deployed the containers as fog gateways on the resource-constrained node (i.e., Raspberry Pi (RPi)). The application runtime and execution of multiple containers on a resource-constrained device indicated good scalability. The overhead derived from container-based virtualization in the experiments of [34] and [37] could be ignored, reflecting advantages in resource consumption, service activation time, and energy efficiency. Meanwhile, container-based virtualization features prominent manageability and scalability. Reference [36] applied the concept of container migration to the edge cloud architecture in the checkpoints stored and managed by Docker, and handled real-time internal and external container migration. The energy consumption at the edge of network was minimized via the cooperative game theory and Docker. Badiger *et al.* [38] proposed VIoLET, a virtual environment for defining and launching large-scale IoT deployment in cloud VMs. Containers were deployed in cloud VMs to simulate the computing resources matching the performance of native edge, fog, and cloud devices. Cao *et al.* [14] realized collaborative video processing. Various video processing functions were run on different edge nodes on a container-based edge computing platform. Their study implied that when the network link capacity is sufficient, collaborative processing is superior to baseline in terms of the total time for video processing. References [33] and [14], [37], [38] utilized Docker Swarm [39] which is a lightweight solution compared to Kubernetes [40] natively integrated with Docker; it can simplify the deployment and management of multiple containers and run across multiple physical or virtual hosts. In regard to security, [41] and [42] indicated that Docker provides an extremely secure container-based application development platform. They suggested running it for non-privileged users (i.e., non-root users) and initiating other enhanced solutions in the Linux kernel, such as AppArmor and SELinux. Therefore, this paper takes advantage of container-based virtual deployment technology to realize fast deployment and high scalability.

III. THE PROPOSED SYSTEM ARCHITECTURE

A. SYSTEM ARCHITECTURE

The proposed system architecture contains three main parts: the cloud server, edge orchestrators, and user applications. The whole system architecture is shown in Figure 2. Six modules were designed for the cloud management platform running on the cloud server: account management module, device management module, secure transmission module, edge service handling module, historical data query module, and video streaming module. For the edge, multiple devices are installed on the light pole, including one single-board computer, one IP camera, and one Arduino where multiple sensors are embedded. Three modules run on the single-board computer at the edge: secure transmission module, edge service handling module (communicating

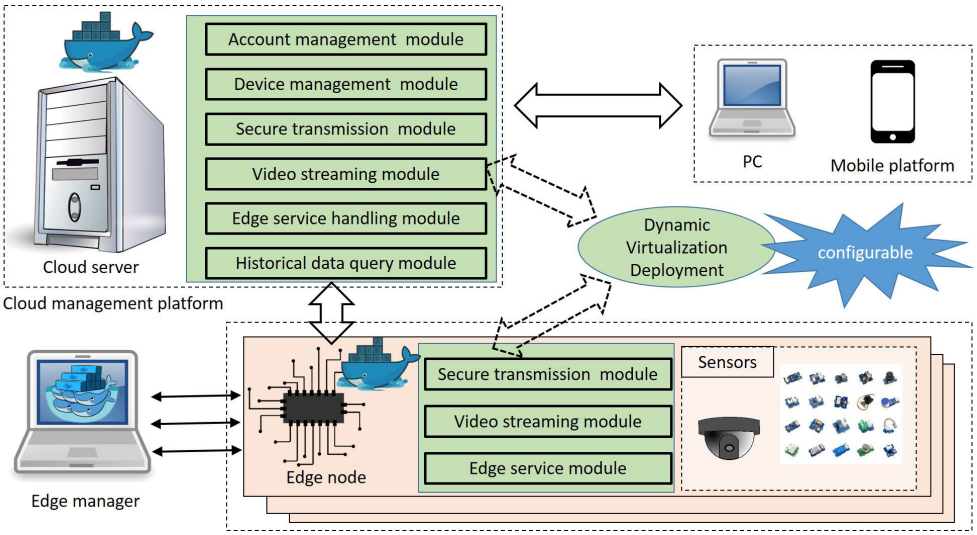


FIGURE 2. System architecture.

TABLE 2. Docker images in the cloud server.

	Base Image	Dockerfile Content	Image Tag	Size
Nginx	alpine:3.9	• Install dependent packages, logrotate • Download source: nginx-1.16.0 • Compile and install nginx with modules: ssl, http2, auth_request • Setup timezone	nginx:star	23 MB
Nginx-RTMP	alpine:3.9	• Install dependent packages, logrotate • Download source: nginx-1.16.0, nginx-rtmp-1.2.1 • Download source: ffmpeg-4.1.3 • Compile and install nginx with nginx-rtmp-module • Compile and install ffmpeg • Setup timezone	nginx-rtmp:star	112 MB
Web (Django)	python:3.7.3-alpine3.9	• Install dependent packages, logrotate, supervisor • Install python packages listed in requirements.txt • Modify "email uri" in django-allauth • Setup timezone	web:star	280 MB
MongoDB	mvertes/alpine-mongo:4.0.6-1	(Only specify the Base Image)	mongo:star	123 MB
Redis	redis:5.0.5-alpine3.9	(Only specify the Base Image)	redis:star	50.9 MB
SSHD	alpine:3.9	• Install OpenSSH, inotify-tools, logrotate, supervisor	sshd:star	57 MB
Certbot	docker:18.09.6 (base on alpine:3.9)	• Install dependent packages, certbot • Set ENTRYPOINT to the certificate check script	certbot:star	256 MB

with Arduino), and video streaming module (pulling/pushing video streams from the IP camera). The necessity of the edge manager depends on the swarm management of multiple street lights.

B. CLOUD MANAGEMENT PLATFORM

1) CLOUD SERVER ARCHITECTURE

The Docker CE engine is installed on the cloud server [43]. This engine creates and controls containerized Docker applications. The cloud management platform consists of seven Docker containers, as shown in Figure 3. Each container is connected to a software bridge network for communication among containers. Containers not connected to a bridge network are isolated from the rest of the containers.

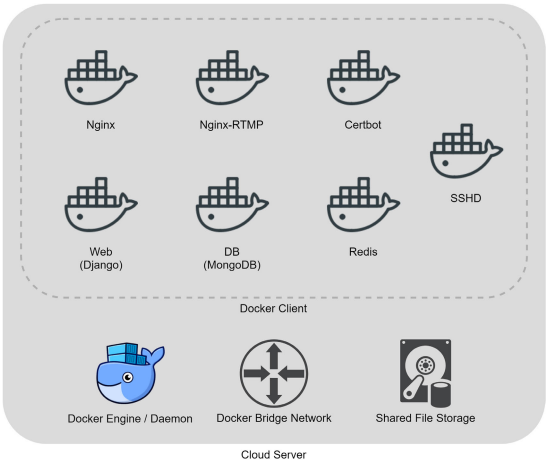


FIGURE 3. Cloud server architecture.

2) DETAILS OF CLOUD CONTAINERS

Table 2 lists the information of each image, including base image, Docker file content, image tag, and image size. Docker Compose [44] is a wonderful tool for the fast deployment of containers via simple commands. It is used to create images and containers, mount all volumes, wire up the network, and expose ports to docker-compose.yml to define the format of

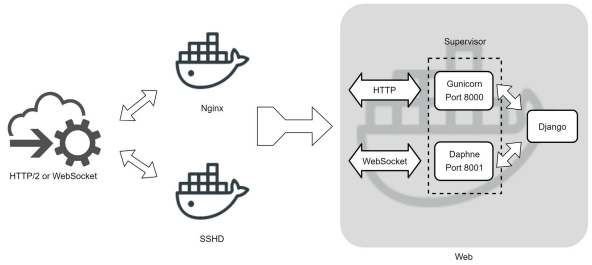


FIGURE 4. Web-based requests workflow.

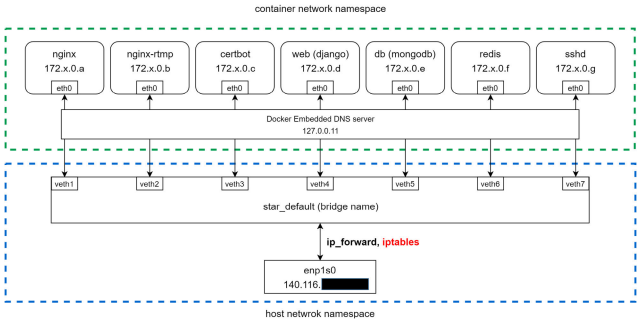


FIGURE 5. Cloud server network topology.

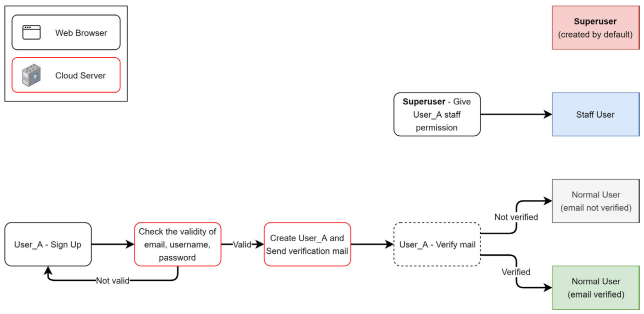


FIGURE 6. User registration workflow.

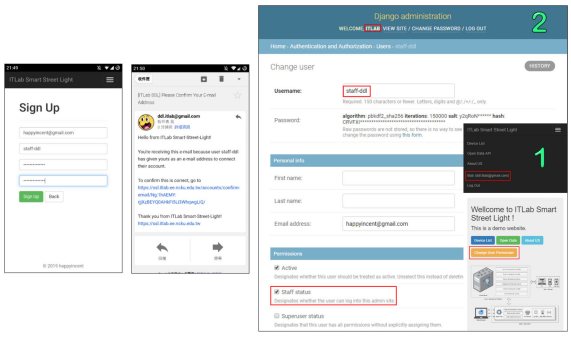


FIGURE 7. User Interface of user registration.

YAML [45]. In this paper, we adopted Version 3 of the file format of Compose to write the bridge network for each container. We specified to "always" restart each container and restart Docker daemon unlimited times to ensure that these containers always ran regardless of the reason of exit. All seven containers were created based on minimal image and Alpine Linux [46], a security-oriented lightweight version of Linux, as shown in Table 2. The total size of the seven images

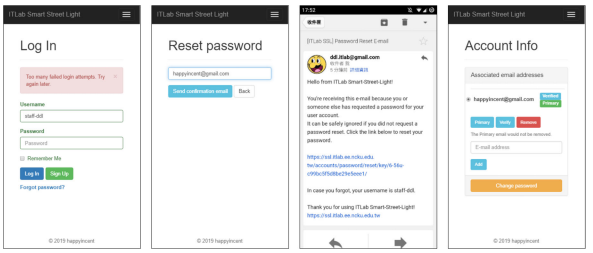


FIGURE 8. User Interface of user login and user modification.

TABLE 3. User information collection.

Key	Value Description	Type
_id	Primary key for each document in MongoDB	ObjectId
id	Primary key for the User collection	Int32
username	Login Username	String
password	Login Password (Salted Hash)	String
email	Email address	String
date_joined	A datetime designating when the account was created	Date
last_login	A datetime of the user's last login	Date
is_superuser	Designates that this user has all permissions	Boolean
is_staff	Designates whether this user can access the admin site	Boolean
is_active	Designates whether this user should be considered active	Boolean

TABLE 4. User permission.

	Get data/ streaming	Config LED (timetable)	Setup device configuration	Edit user permission
superuser	✓	✓	✓	✓
staff	✓	✓	✓(creator)	
email verified	✓			
email not verified				

TABLE 5. User information in redis.

Key	Value Description	TTL	Type
(session_key)	Session information includes: • _auth_user_id • _auth_user_backend • _auth_user_hash • _session_expiry	1 Day	Python pickle
(username)	Token for Open Data API	600 s	Python string

TABLE 6. Device configuration collection.

Key	Value Description	Type
_id	Primary key for each document in MongoDB	ObjectId
user_id	Device owner's User id	Int32
id	Primary key for the Device collection	String
longitude	Longitude of the device location	Double
latitude	Latitude of the device location	Double
token	An unique one-time token of the device	UUID
ssh_pub	SSH public key of the device	String
led_schedule	Weekly LED schedule in a JSON array	Array
pir_schedule	Weekly PIR schedule in a JSON array	Array

is less than 1 GB. As shown in Figure 4, for all Web-based requests, we used Django [47] to realize the web server. The requests fall into two categories: HTTP/2 [48] and WebSocket [49]. Two standard interfaces were used for the collocation of Django applications and to allocate two types of incoming requests: WSGI [50] and [51], respectively. In the web

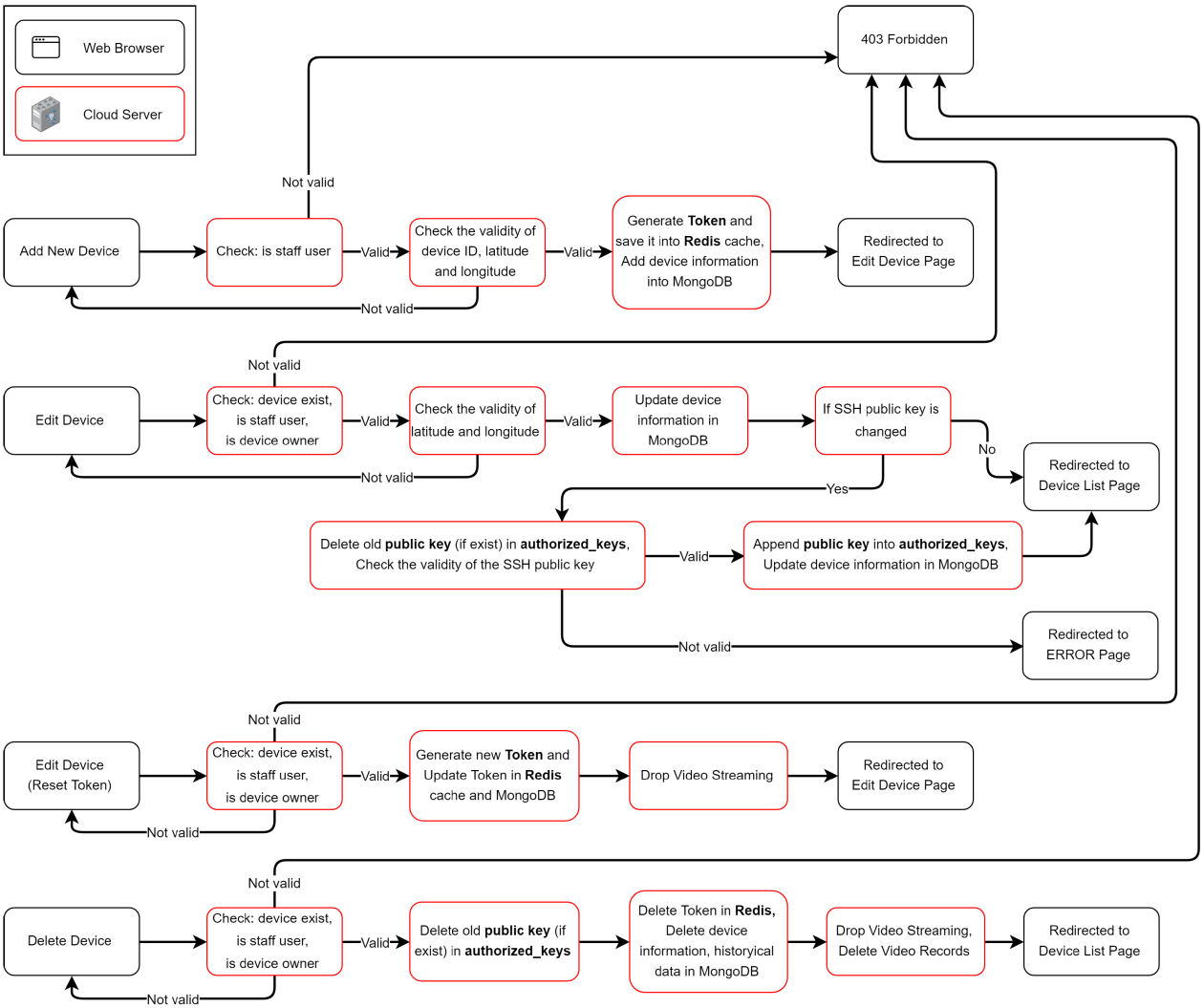


FIGURE 9. Device management workflow.

TABLE 7. Device information in redis.

Key	Value Description	TTL	Type
(Device ID)	Token for Edge services (same as the token in Table 6)	-1 (Infinite)	Python string
(Device ID)_channel	WebSocket channel name of the device	-1 (Infinite)	Python string
(Device ID)_info	Current information of the device	30 s	Python pickle

container, we used Supervisor [52] as a process control tool to monitor and automatically start the following processes:

- Gunicorn [53]: WSGI server. We used the asynchronous work procedure based on gevent [54]. Asynchronous IO was used to handle I/O binding requests with higher synchronization.
- Daphne [55]: A WebSocket protocol server for ASGI.

In addition, among the following containers: Nginx, Nginx-RTMP, Web, and SSHD, we adopted crond built in Alpine Linux for time synchronization and logrotating [56] jobs. It is used to manage log files (e.g. Nginx access log or Supervisord log).

3) CLOUD SERVER NETWORK TOPOLOGY

The network topology of the cloud server is shown in Figure 5. Each container has its own namespace. A namespace is regarded as an isolated network stack in the kernel with its own network interface and routing and firewall rules. There is also a Linux bridge (realized by the virtual switch inside the Linux kernel). It has multiple virtual Ethernet interfaces linked to the interfaces in each container; therefore, containers communicate with each other through the ARP protocol of the Layer 2 bridge. Moreover, containers on the network bridges defined by the same user can resolve the name of the other container into an IP address through the embedded DNS server of the Docker engine and DNS lookup.

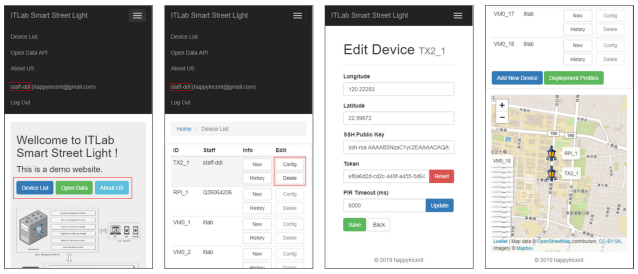


FIGURE 10. Device setting for device owner.

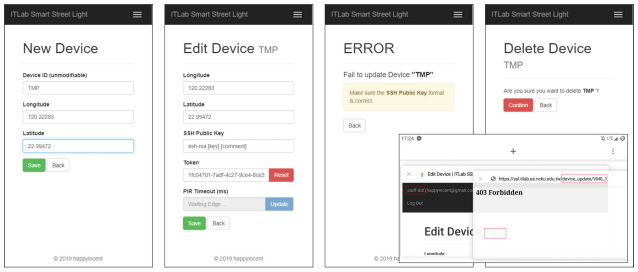


FIGURE 11. User Interface of device management with with wrong input or wrong permission.

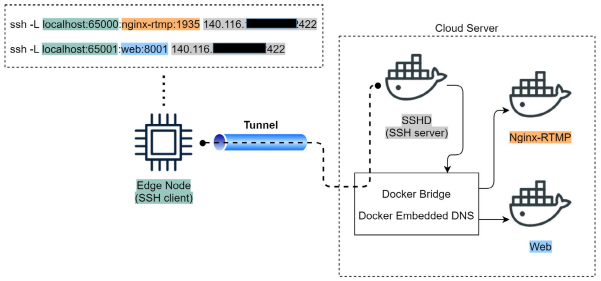


FIGURE 12. SSH tunnel between Edge and Cloud.

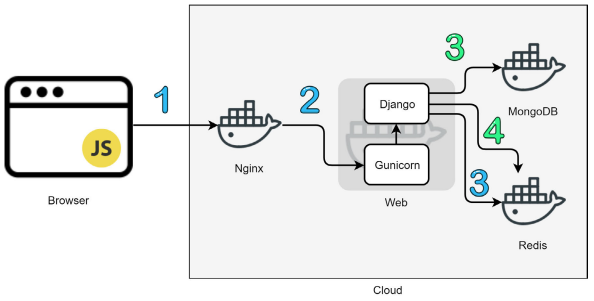


FIGURE 13. Browser authentication workflow of HTTP/2 request.

Docker daemon attaches rules to the iptables [57] of the host in order to filter a data package in Layer 3 and enable specific connection of containers, such as traffic forwarding and host port mapping.

4) ACCOUNT MANAGEMENT MODULE

This module takes charge of user registration, modification of user information, and user login. Table 3 shows that this module is stored in MongoDB [58] (a JSON-like document oriented NoSQL database). This container records both user information and the permission of each user account.

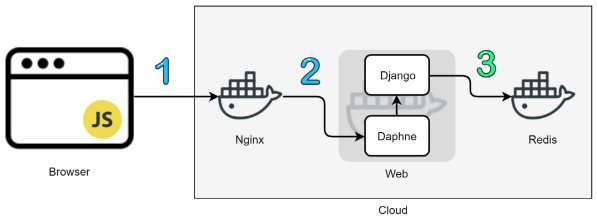


FIGURE 14. Browser authentication workflow of webSocket requests.

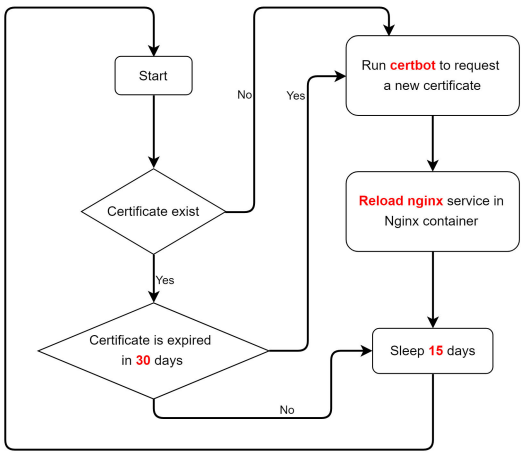


FIGURE 15. Certificate renewal workflow.

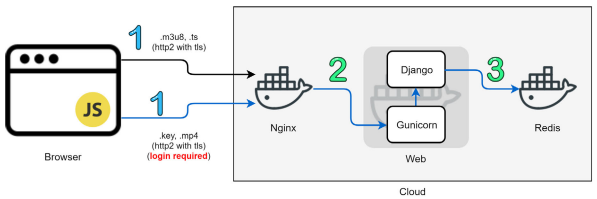


FIGURE 16. Browser workflow of video file requests.

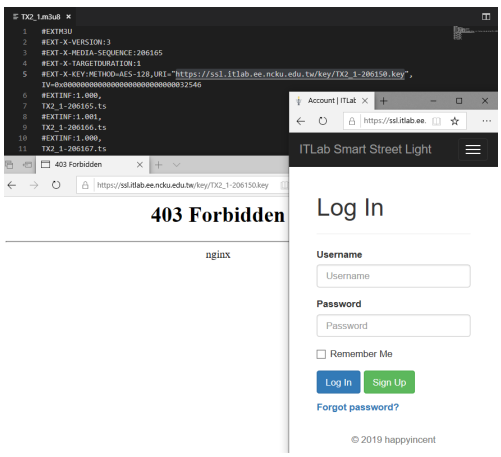


FIGURE 17. HLS playlist and authentication of HLS encryption key.

The permission of each type of user in the management platform is listed in Table 4. The simple user registration process and the user interface for user registration are introduced in Figures 6 and 7. As shown in Figure 8, the user login process includes login failure (the maximum failed

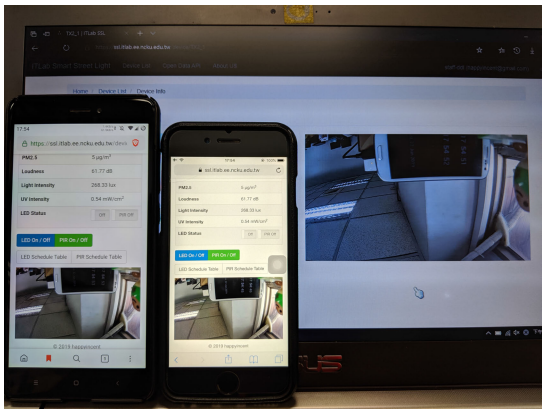


FIGURE 18. User interface of live streaming in cross platform cliens.

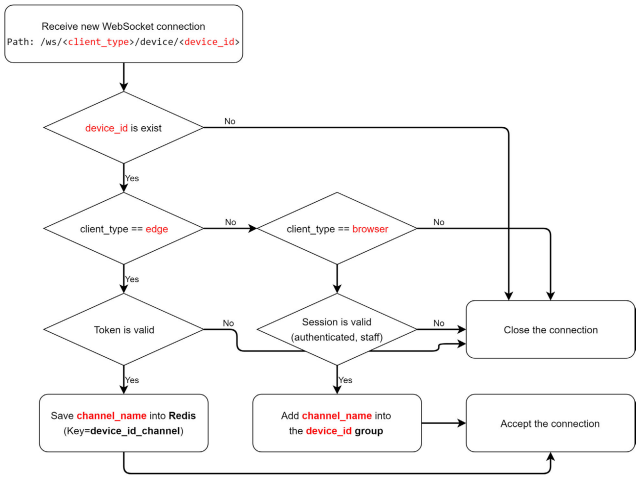


FIGURE 19. WebSocket connection workflow.

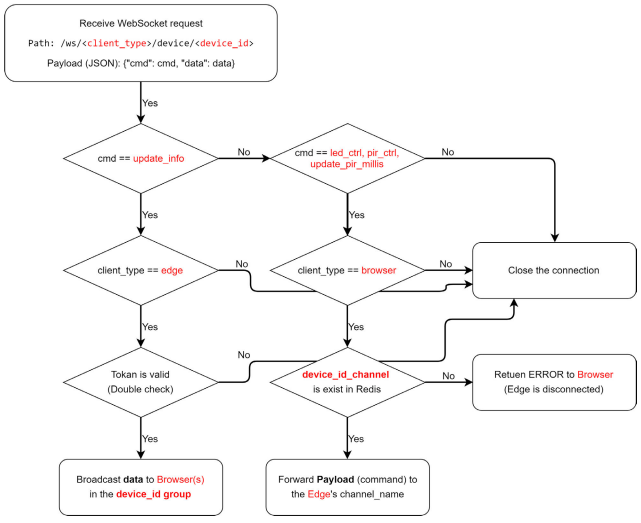


FIGURE 20. WebSocket requests handling workflow.

login attempts every five minutes is set to five times), password resetting, and account information. We stored two key-value pairs in the database container in Redis (an open source, in-memory data structure store used as a database or cache) in the management platform, as shown in Table 5, to

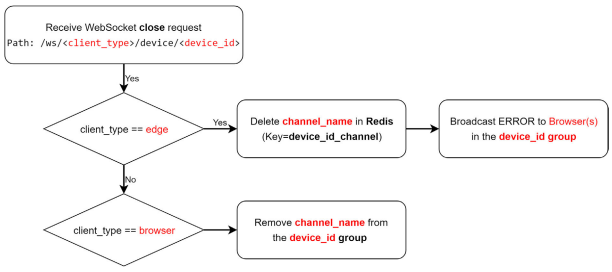


FIGURE 21. WebSocket Disconnection workflow.

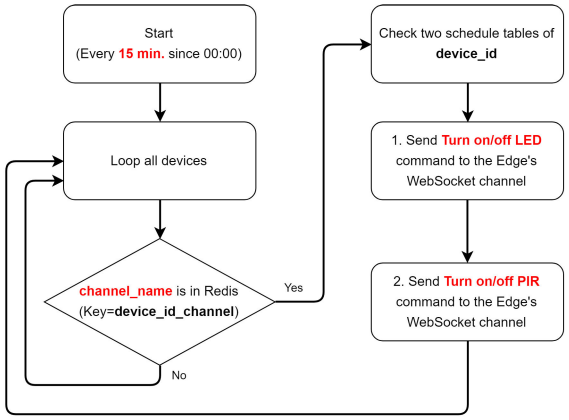


FIGURE 22. Street light ON/OFF scheduling workflow.

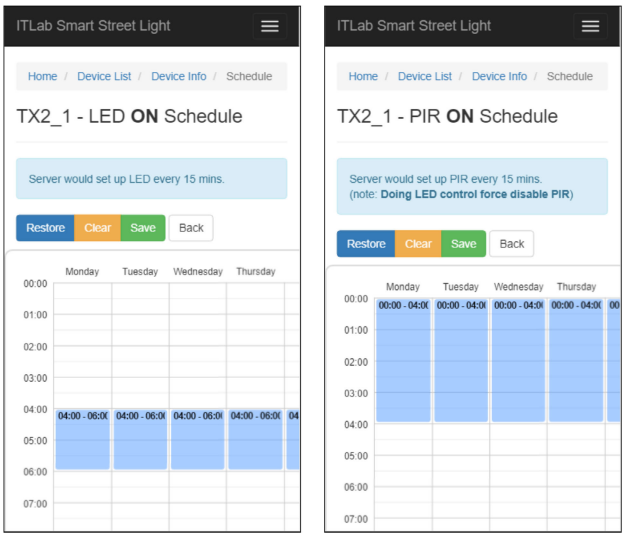


FIGURE 23. Uerser interface of edge service timetable.

accelerate each request that needs to be transmitted after the permission check.

5) DEVICE MANAGEMENT MODULE

This module manages the configuration, public keys, and device tokens. Table 6 describes the configuration and location of each device, secure transmission parameters, and lighting control schedules. Table 7 lists the device configuration for real-time data transmission and authentication stored in the Redis database container. The CRUD (i.e., Create, Read, Update, and Delete) of device configuration

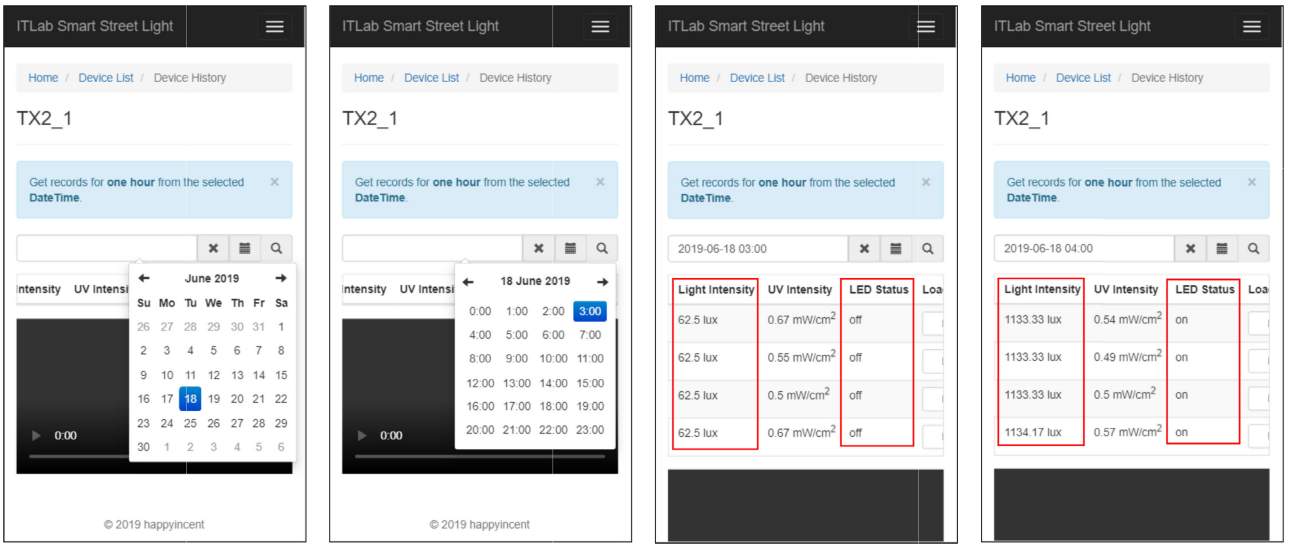


FIGURE 24. User interface of historical data query.

TABLE 8. Historical information collection.

Key	Value Description	Type
_id	Primary key for each document in MongoDB	ObjectId
id	Primary key for the Historical Info collection	Int32
device_id	Device ID	String
temperature	Temperature (C)	Double
humidity	Humidity (%)	Double
pmat25	PM2.5 (g/m^3)	Double
looudness	Looudness (dB)	Double
light_intensity	Light intensity (lux)	Double
uv_intensity	UV intensity (mW/cm^2)	Double
led_status	LED On/Off (True or Flase)	Boolean
timestamp	A datetime of the Information	Date

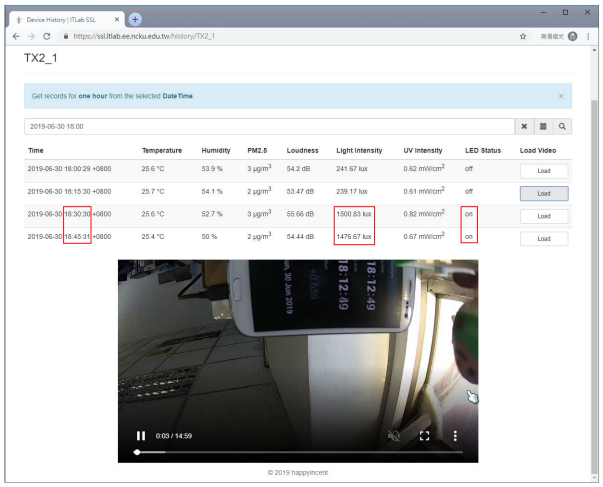


FIGURE 25. User interface of detailed historical data query.

management is described in Figure 9. Figure 10 shows that only the device owner can modify the device configuration. Figure 11 shows incorrect input, incorrect permissions, and incorrect pages requesting resetting.

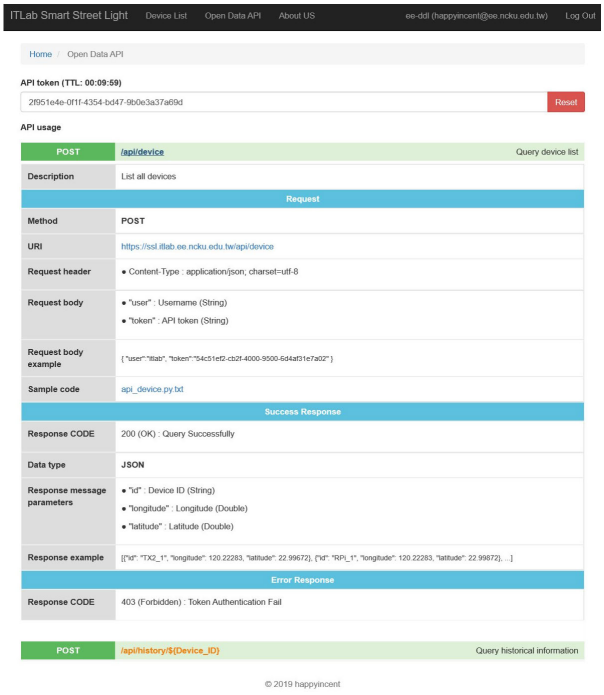


FIGURE 26. User Interface of open data API.

6) SECURE TRANSMISSION MODULE

This paper applied two communication protocols to the edge node: RTMP and WebSocket. For the communication between Edge and Cloud, we set up an encrypted SSH tunnel to forward local ports to remote ports and monitor the Docker bridge network inside the cloud server. As shown in Figure 12, the edge node binds ports 65000 and 65001 on localhost to the remote addresses of nginxrtmp: 1935 and web: 8001, respectively.

For Web browser to Cloud, this paper applied two communication protocols to the browser: HTTP/2 [48] and

TABLE 9. Docker images in the edge node.

	Base Image	Dockerfile Content	Image Tag	Size (amd64)	Size (arm64)	Size (armv7)
Autossh	alpine:3.9	• Install autossh • Create folder (store SSH keys) for volume mounting	itlabstar/autossh	11.5 MB	11.2 MB	8.71 MB
FFmpeg	alpine:3.9	• Install ffmpeg	itlabstar/ffmpeg	57 MB	47.9 MB	35.6 MB
Edge	python:3.7.3 -alpine3.9	• Install python packages: websocket-client, pyserial • Add edge.py into the image	itlabstar/Edge	93.7 MB	100 MB	78 MB

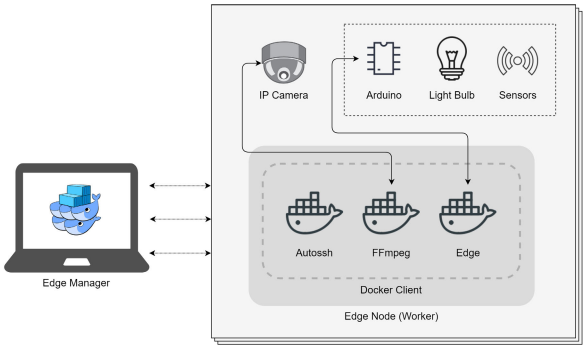


FIGURE 27. Edge nodes architecture.

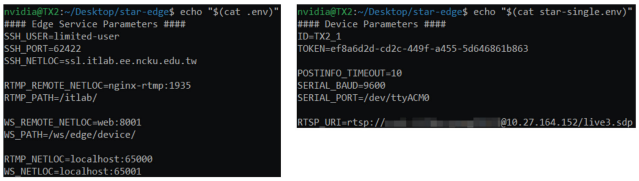


FIGURE 28. Environment variables in the edge node.

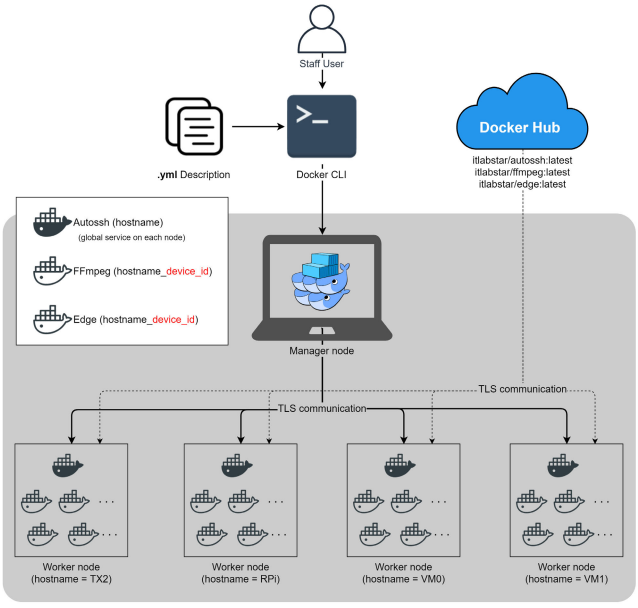


FIGURE 29. Cluster nodes architecture.

WebSocket. We adopted HTTPS [69] with a 4096-bit RSA certificate to encrypt the communication between the browser and the cloud. Figure 13 demonstrates the authentication workflow requested by HTTP/2. The user login workflow before the management platform is entered is as follows:

FIGURE 30. User interface of deployment profile.

TABLE 10. Cloud server environment.

	HP Z600
CPU	(Dual) Intel Xeon E5520 @ 2.27GHz
RAM	24 GB DDR3 1333MHz
Storage	• 291 GB (LSI RAID 0) • 2 TB
Ethernet	1 Gb/s
OS	Xubuntu 16.04.6 LTS
Linux kernel	4.15.0-54-generic x86_64
Software package	• Docker 18.09.7 • Docker Compose 1.24.0

- 1. The browser connects with the Nginx container via TLS encryption and HTTP/2.
- 2. The Nginx container uses the HTTP/2 proxy and forwards the request to the Web container.
- 3. The Web container checks whether the account and the password stored in MongoDB are correct.
- 4. The Web container adds session information to the Redis cache and returns the session key to the browser.

The authentication workflow requested for WebSocket of the browser is shown in Figure 14:

- 1. The browser sends the WebSocket request with a session key to the Nginx container via TLS encryption.
- 2. The Nginx container uses the HTTP/2 proxy and forwards the request to the Web container.
- 3. The Web container uses the session key stored in the Redis cache to check the session information.

After successful authentication, the session key stored in the Redis cache will be used for session check.

7) HTTP CERTIFICATE RENEWAL

HTTPS is the key to communication security between the browser and the cloud. HTTPS requires a digital certificate that allows the browser to verify the identity of the web server. Let's Encrypt [60] is a free, automatic, and open certificate authority, sponsored by the Electronic Frontier Foundation (EFF), Mozilla, and other organizations. We used Certbot [61] to obtain the HTTPS certificate and

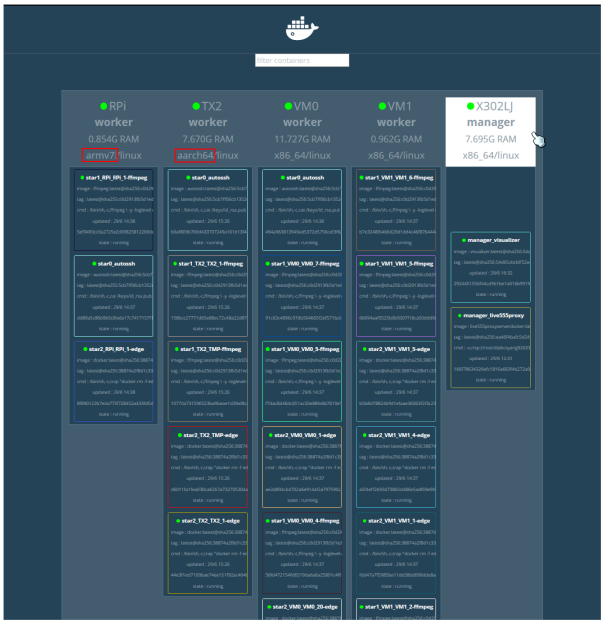


FIGURE 31. Swarm visualizer.

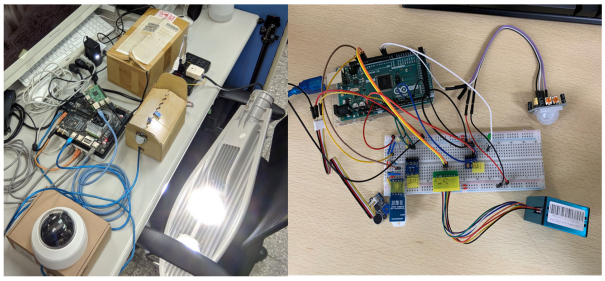


FIGURE 32. Our simulation devices.

```
itlab@Z600:~/Desktop/star-eval$ ./backup.sh
Stop all Docker containers ...

real    0m34.724s
user    0m1.165s
sys     0m0.119s

OK

Backup star project ...

real    0m15.012s
user    0m14.671s
sys     0m1.510s

OK

Backup Docker images ...

real    0m29.452s
user    0m53.618s
sys     0m2.806s

OK

List tar files ...
213M    star.tar.gz
74M     certbot:star.tar.gz
50M     mongo:star.tar.gz
38M     nginx-rtmp:star.tar.gz
7.3M    nginx:star.tar.gz
17M     redis:star.tar.gz
18M     sshd:star.tar.gz
83M     web:star.tar.gz
itlab@Z600:~/Desktop/star-eval$
```

```
itlab@Z600:~/Desktop/star-eval$ ./restore.sh
Remove star project ...
OK

Remove Docker images ...
OK

Restore star project ...

real    0m4.318s
user    0m4.302s
sys     0m1.842s

OK

Restore Docker images ...

real    0m11.014s
user    0m10.863s
sys     0m1.755s

OK

Start all Docker containers ...

real    0m20.819s
user    0m1.326s
sys     0m0.099s

OK
itlab@Z600:~/Desktop/star-eval$
```

FIGURE 33. Back up and restore the Cloud server.

ensure the validity of the certificate. Certbot is the full-featured and scalable client side of CA. Let's Encrypt CA was used. The domain name of the cloud server was configured and provided by NOC of NCKUEE [62]. Figure 15 shows the workflow of the script that checks whether the HTTPS certificate expires, when the Certbot container is started.

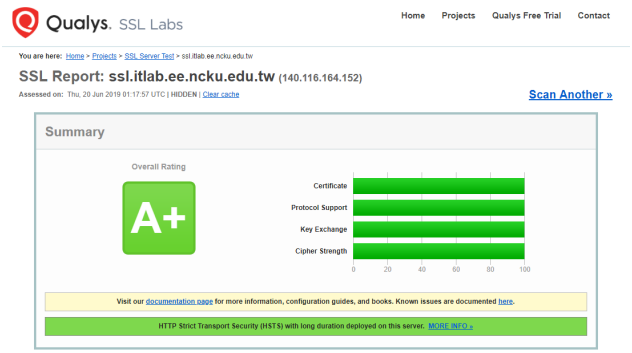


FIGURE 34. SSL Report by SSL Labs [72].

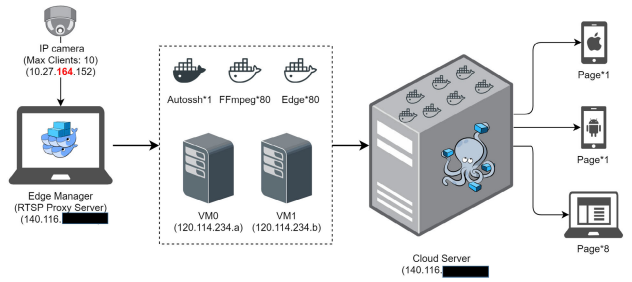


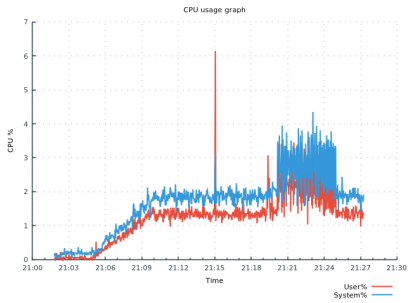
FIGURE 35. Experiment architecture of cloud nodes.

8) VIDEO STREAMING MODULE

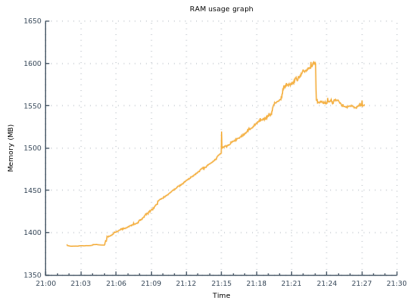
In order to provide real-time video streams, this module uses the nginx rtmp module [63] to receive the RTMP [64] stream issued by the edge node and utilizes FFmpeg [65] to transcode it into HLS [66]. We regard Nginx and Nginx-RTMP as two separate containers to make it easy to add more computing resources in the future to raise the scalability of the system. The Nginx-RTMP container has the following functions:

- It monitors Port 1935 (internal) to receive the RTMP stream with token inspection.
- It generates the HLS file.
  - HLS playlist (.m3u8)
  - Encrypted media segment (.ts)
  - AES encryption key (.key)
- It records the RTMP stream and converts it to MP4 via FFmpeg. information. The maximum length of historical videos is 15 minutes.
- It monitors Port 80 (internal) to receive the control messages requested by the Web container in order to disconnect the RTMP client side later.
  - The token of the device has been reset.
  - The device has been removed.

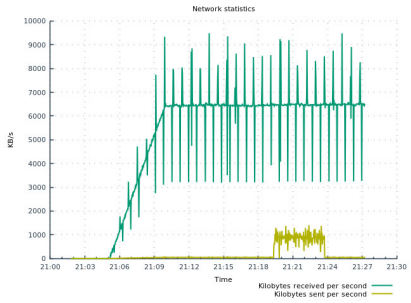
The Nginx container safely provides all video files to the browser through HTTPS encryption. This container and the Nginx-RTMP container share volumes with each other. As the HLS media segment is encrypted, only a request by HLS encryption key for historical videos requires session check for authentication by Nginx. The workflow of requesting a video file is shown in Figure 16. The HLS playlist and request of an unauthenticated HLS encryption key are



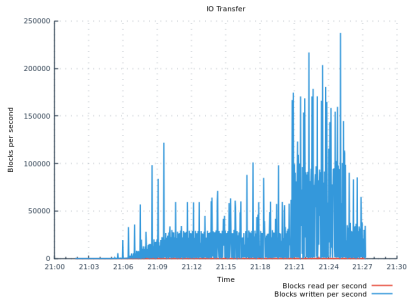
(a) CPU usage



(b) RAM usage



(c) Network throughput



(d) Disk I/O

FIGURE 36. Performance of Cloud Server.

shown in Figure 17. Figure 18 demonstrates that the real-time video streams on our management platform are running across platforms, including Android, iOS, and Windows.

9) EDGE SERVICE HANDLING MODULE

This module provides edge services, including:

- It receives information from the edge and broadcasts it to the browser (real-time).

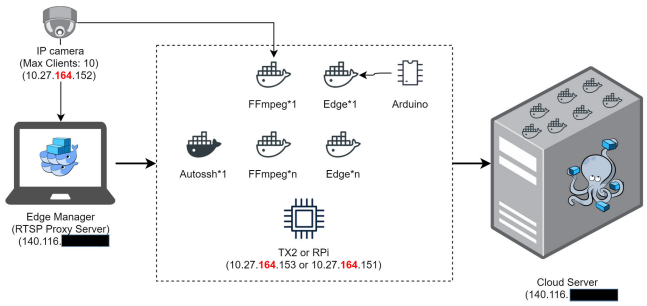


FIGURE 37. Experiment architecture of edge nodes.

- It receives commands sent from the browser to the specific edge (real-time).
- It controls LED lights based on two schedules: (1) Manual control, (2) Automatic control in accordance with the value displayed on the Passive Infrared Sensor (PIR).

Figure 19 shows the workflow of this module when WebSocket is connected. Figure 20 displays the workflow when multiple commands are used to process the WebSocket request, including updating Edge information, turning on/off LED lights or the PIR, and configuring the PIR timeout value. All data frames in WebSocket are encoded as JSON (a data structure standard in Web applications). Figure 21 shows the workflow of this module during the disconnection of WebSocket.

The built-in crond service is used in the cloud server. According to the two schedules stored in MongoDB, each device will have its own cron job. In this way, the user can control street lights more easily. Figure 22 describes the process of a cron job. Figure 23 shows the user interface of schedules based on jQuery Schedule [67] and Ajax [68].

10) HISTORICAL DATA QUERY MODULE

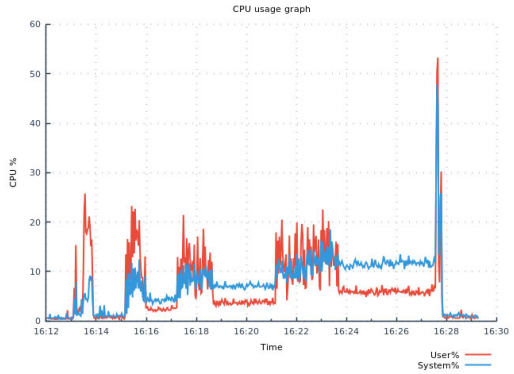
This module enables the query of historical data in different methods, including:

- Query via a Web-based user interface (session check).
- Query via the Open Data API (token check).

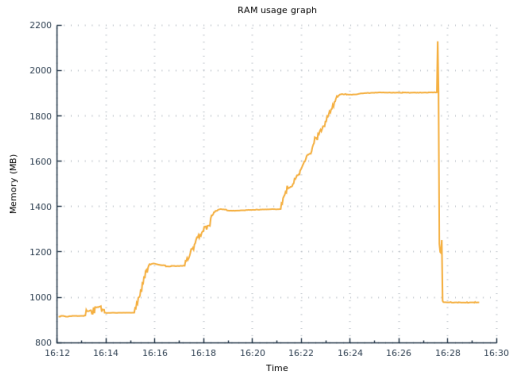
Table 8 describes each historical file stored in the MongoDB container and records the values of all sensors and the Led status of the edge nodes. Besides the above keys in the historical information set, the two keys of devices in the current information are stored in the Redis database. For the query request via a logged-in session in the Web user interface, the server will simultaneously return the historical information and video of the device, as shown in Figures 24 and 25. In contrast, For the query request via a user token and the Open Data API, the server will only return the historical information of the device, as shown in Figure 26.

C. EDGE SERVICE ORCHESTRATORS

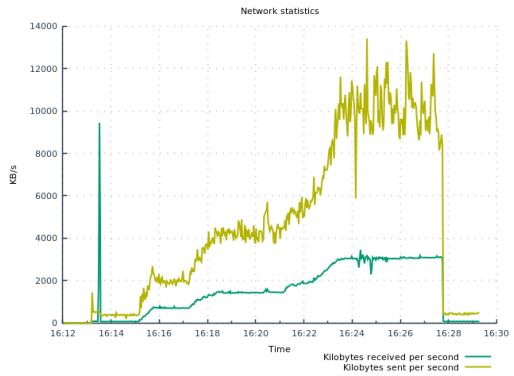
We deployed each edge service on the edge node as a Docker container, as shown in Figure 27. The image information of each edge container is shown in Table 9, including base image, Docker file content, image tag, and size of images on



(a) CPU usage

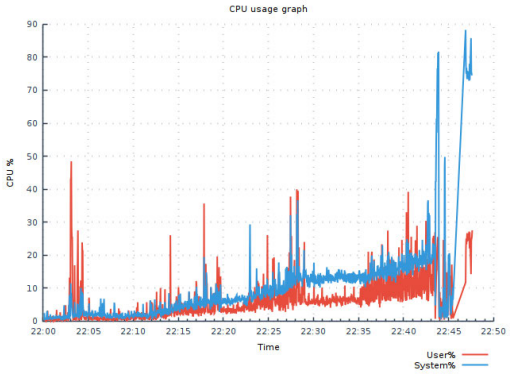


(b) RAM usage

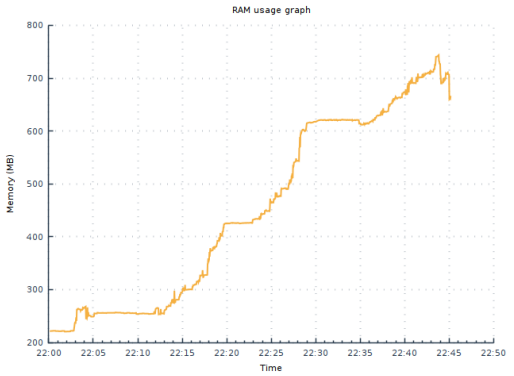


(c) Network throughput

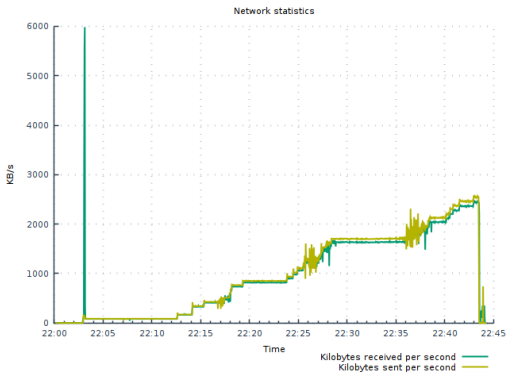
FIGURE 38. Performance of Node Server(TX2).



(a) CPU usage



(b) RAM usage



(c) Network throughput

FIGURE 39. Performance of Node Server(RPi).

different platforms. We adopted Docker Compose to deploy three edge services in a single node, according to the compose file. The file defines the configuration for building images, creating containers, mounting volumes, wiring up network, as well as the configuration of environment variables. It is defined in .env and star-single.env, as shown in Figure 28.

D. DEPLOYMENT OF CLUSTER NODES

We provided a tool to deploy edge services on many edge nodes, according to Docker Swarm, Docker Compose, Buildx [70], and DockerHub. Figure 29 demonstrates a swarm cluster system structure consisting of one manager node and four worker (edge) nodes. Figure 30 shows the deployment profile of each edge node designed by this paper. Furthermore, the user can utilize the Docker swarm visualizer [71] to monitor and visualize the container status in the swarm. Figure 31 displays an example of a visualization service. This service is deployed on the swarm manager node.

IV. EXPERIMENT AND SIMULATION

A. ENVIRONMENT

Tables 10 and 11 display the specifications of the cloud server and devices on edge nodes used in this paper. Figure 32 shows the relevant devices that we used to simulate light

TABLE 11. Edge nodes environment.

	Nvidia Jetson TX2	Raspberry Pi 3 Model B	VM0 (ESXi 6.0)	VM1 (ESXi 5.5)
CPU	2 x 2 GHz + 4 x 2 GHz	4 x 1.2 GHz	8 x 3.70 GHz	8 x 3.40 GHz
GPU	Nvidia Pascal 256-core	Broadcom VideoCore	None	None
RAM	8GB	1 GB	12 GB	2 GB
Ethernet	1 Gb/s	100 Mb/s	1 Gb/s	1 Gb/s
OS	Ubuntu 16.04.6 LTS	Raspbian 9.9 (stretch)	Xubuntu 16.04.6 LTS	Ubuntu Server 18.04.2 LTS
Linux kernel	TX2 4.4.38-tegra **aarch64**	RPi 4.19.50-v7+ **armv7l**	4.15.0-52-generic **x86_64**	4.15.0-45-generic **x86_64**
Software package	Docker 18.09.7	Docker 18.09.0	Docker 18.09.6	Docker 18.09.6

TABLE 12. Secure transmission verification.

Connection	Transmission method	Encryption method	Authentication method
Cloud to Browser (hls key, vod mp4)	TCP / HTTP2	HTTPS	Session
Cloud to Browser (edge service)	TCP / WebSocket	HTTPS	Session
Edge to Cloud (edge service)	TCP / WebSocket	SSH tunnel	Token
Edge to Cloud (streaming)	TCP / RTMP	SSH tunnel	Token
IP Camera to Edge (streaming)	TCP / RTSP	None (LAN)	Fixed password

TABLE 13. CPU (%) and RAM (MB) usage of Cloud Server (average with 30 records).

	nginx	nginx-rtmp	certbot	web	mongodb	redis	sshd
0 edge, 0 client	0.00% 27.74MB	0.00% 4.60MB	0.00% 3.47MB	0.01% 231.31MB	0.03% 17.86MB	0.01% 2.79MB	0.00% 20.70MB
1 edge, 0 client	0.00% 27.74MB	0.03% 5.20MB	0.00% 3.47MB	0.01% 236.82MB	0.02% 18.04MB	0.01% 2.87MB	0.02% 22.58MB
1 edge, 1 client	0.00% 28.06MB	0.03% 5.36MB	0.00% 3.47MB	0.05% 251.67MB	0.02% 18.30MB	0.01% 2.95MB	0.02% 22.57MB
1 edge, 10 clients	0.02% 29.10MB	0.03% 5.56MB	0.00% 3.47MB	0.01% 261.81MB	0.03% 18.84MB	0.01% 3.14MB	0.02% 22.61MB
80 edges, 0 client	0.00% 28.34MB	1.43% 33.37MB	0.00% 3.47MB	0.17% 270.95MB	0.03% 19.57MB	0.04% 3.53MB	0.98% 27.77MB
80 edges, 10 clients	0.02% 29.64MB	1.34% 44.95MB	0.00% 3.47MB	0.15% 272.37MB	0.03% 20.04MB	0.04% 3.59MB	0.94% 28.30MB

street scenarios, including: the IP camera, the Arduino micro-controller, and sensors for PM2.5, ambient sound, temperature and humidity, intensity of illumination, ultraviolet, and motion sensing.

B. FUNCTION VERIFICATION

This paper proposes the following system design objectives:

- High scalability.
- Security and privacy orientation.
- User-friendly interface.

Indeed, this platform has high scalability. The cloud server consists of seven Docker containers that are able to be deployed on multiple physical machines at the swarm cluster with the built-in multi-host networking. In light of the results shown in Figure 33, the scalability of deployment is met. The backup and restoration of cloud Docker containers on a single host are explained. As expected, it took approximately 20 seconds and 34 seconds to start and stop the system, respectively. It only took nearly 500MB to back up cloud projects, including source codes, static files, database volumes, and seven Docker images. From the perspective of the edge, this paper proposed and described two (container-based) deployment methods in Section 3. The security and privacy of the platform are satisfactory. Three TCP ports were exposed in the cloud server and forwarded from the Internet to the internal Docker bridge network via iptables. The three TCP ports are 80 (for HTTPS certificate renewal), 443 (for HTTPS requests), and 62422 (for SSH tunnels). Table 12 describes the security method for each connection type. The configuration of the HTTPS certificate was verified by SSL Labs [72]. The test results are shown in Figure 34.

C. PERFORMANCE EVALUATION

1) PERFORMANCE OF THE CLOUD SERVER

The following experiments were carried out in this paper, as shown in Figure 35. Many edge services were deployed on two VMs (i.e., VM0 and VM1). Ten real clients were browsing the page of real-time device status. Many edge services were deployed on embedded devices. The input to FFmpeg and Edge containers running on edge nodes are described below:

- FFmpeg: The RTSP stream from the IP camera proxy (maximum connections = 10).
- Edge: Random value (simulation of the information collected by Arduino).

According to the experimental results shown in Figure 36, 80 edge nodes were deployed at 21:05. From 21:19 to 21:24, 10 client pages were browsed. Table 13 shows the average

TABLE 14. Service deployment Timetable (TX2).

	Autossh	FFmpeg	Edge
16:13	1	1	1
16:15	1	10	10
16:17	1	20	20
16:21	1	40	40
16:28	1	1	1

TABLE 15. Service deployment Timetable (RPi).

	Autossh	FFmpeg	Edge
22:03	1	1	1
22:12	1	10	10
22:24	1	20	20
22:36	1	40	40

use of CPU and RAM of each cloud container in the client side serving different numbers of edge nodes and browsers (measured based on docker statistics).

2) PERFORMANCE OF EDGE NODES

Experimental scenarios are shown in Figure 37. Figure 38 shows the edge services deployed on TX2 based on the time in Table 14. Figure 39 displays the deployment of edge services on RPi according to the time in Table 15.

The reason why we did not deploy more containers on RPi was that, after 20 FFmpeg and 20 Edge containers were deployed, RPi would crash randomly. As expected, in terms of CPU usage and time for container deployment, Nvidia Jetson TX2 is better than RPi.

V. CONCLUSIONS AND FUTURE STUDIES

This paper proposes a street lighting management system consisting of one Web-based cloud management platform, one set of edge devices (a single-board computer, a microcontroller, sensors, and an IP camera), and the real-time lighting control function. The system can provide street pole information to the user in real-time and three major functions, including the historical data query API, which has been verified and discussed in detail. We put forward a novel architecture differing from that of the existing street lighting management systems. The architecture integrates the container-based virtualization technology, Docker, to provide a strong and highly scalable solution to the deployment of the cloud and edge services. Furthermore, our design protects the communication between the edge and the cloud, including token authentication and SSH-based encryption (with public key authentication). In general, the proposed system is modular, scalable, easy to deploy, and security-oriented. Therefore, this system has a high commercial value.

It is suggested that future studies explore the development of some smart applications and integrate them into our system. For instance, the machine learning algorithm can be applied to the development of a smart lighting control mechanism based on the environment data provided by the sensors. Besides, the neural network can be integrated into edge devices through the IoT edge devices installed on light poles to execute low-latency AI applications, such as real-time target detection (e.g. pedestrian or vehicle detection). As a result, the application value of this system architecture will be raised.

REFERENCES

[1] *World's Population Increasingly Urban With More Than Half Living in Urban Areas*. New York, NY, USA, Dec. 2014.

[2] United Nations. (2018). *World Urbanization*. [Online]. Available: https://www.un.org/development/desa/publications/2018-revision-of-world-urbanization-prospects.html

[3] H. Ritchie. (2018). *How is an Urban Area Defined?* [Online]. Available: https://ourworldindata.org/how-urban-is-the-world

[4] D. Evans, "The Internet of Things: How the next evolution of the Internet Is Changing everything," CISCO, San Jose, CA, USA, White Paper 1, 2011.

[5] S. Pellicer, G. Santa, A. L. Bleda, R. Maestre, A. J. Jara, and A. G. Skarmeta, "A global perspective of smart cities: A survey," in *Proc. 7th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput.*, Jul. 2013, pp. 439–444.

[6] V. Fernandez-Anez, *Stakeholders Approach to Smart Cities: A Survey on Smart City Definitions*. Cham, Switzerland: Springer, 2016, pp. 157–167.

[7] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 112–121, Apr. 2014.

[8] *Global LED Smart Street Lighting: Market Forecast (2017- 2027)*, Northeast Group, 2017.

[9] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart cities: A survey on data management, security, and enabling technologies," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2456–2501, Aug. 2017.

[10] (2018). *OWASP Internet of Things Top 10*. [Online]. Available: https://www.owasp.org/images/1/1c/OWASP-IoT-Top-10-2018-final.pdf

[11] *2016 Dyn Cyberattack*. [Online]. Available: https://en.wikipedia.org/wiki/2016_Dyn_cyberattack

[12] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT: Mirai and other botnets," *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[13] SCC Europe Staff. (2018). *How the EU's GDPR Will be a Game Changer for Cities Using Open Data*. [Online]. Available: https://eu.smartcitiescouncil.com/article/how-eus-gdpr-will-be-game-changer-cities-using-open-data

[14] Y. Cao, Z. Xu, P. Qin, and T. Jiang, "Video processing on the edge for multimedia IoT systems," 2018, *arXiv:1805.04837*. [Online]. Available: http://arxiv.org/abs/1805.04837

[15] M.-H. Jia, Y.-Q. Chen, G.-Y. Zhang, P. Jiang, H. Zhang, and J. Wang, "A Web service framework for astronomical remote observation in antarctica by using satellite link," *Astron. Comput.*, vol. 24, pp. 17–24, Jul. 2018.

[16] R. Álvarez, F. Duarte, A. AlRadwan, M. Sit, and C. Ratti, "Re-imagining streetlight infrastructure as a digital urban platform," *J. Urban Technol.*, vol. 24, no. 2, pp. 51–64, Apr. 2017.

[17] F. Campos, A. Simões, I. Sousa, R. Almeida, and P. Daniel, "Smart ip–central management system for public lighting in portugal," *CIRED–Open Access Proc. J.*, vol. 2017, no. 1, 2017, pp. 1471–1481.

[18] P. T. Daely, H. T. Reda, G. B. Satrya, J. W. Kim, and S. Y. Shin, "Design of smart LED streetlight system for smart city with Web-based management system," *IEEE Sensors J.*, vol. 17, no. 18, pp. 6100–6110, Sep. 2017.

[19] G. Jia, G. Han, A. Li, and J. Du, "SSL: Smart street lamp based on fog computing for smarter cities," *IEEE Trans Ind. Informat.*, vol. 14, no. 11, pp. 4995–5004, Nov. 2018.

[20] D. M. Llido Escriva, J. Torres-Sospedra, and R. Berlanga-Llavori, "Smart outdoor light desktop central management system," *IEEE Intell. Transp. Syst. Mag.*, vol. 10, no. 2, pp. 58–68, 2018.

[21] P. Sharma, L. Chaufournier, P. Shenoy, and Y. Tay, "Containers and virtual machines at scale: A comparative study," in *Proc. 17th Int. Middleware Conf.*, 2016, pp. 1–13.

[22] *Linux Namespaces*. [Online]. Available: http://man7.org/linux/man-pages/man7/namespaces.7.html

[23] *Linux Control Group v2*. [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v2.txt

[24] *VMware vSphere Hypervisor*. [Online]. Available: https://www.vmware.com/products/vsphere-hypervisor.html

[25] *KVM—Kernel-Based Virtual Machine*. [Online]. Available: https://www.vmware.com/products/vsphere-hypervisor.html

[26] Docker. [Online]. Available: https://www.docker.com/

[27] *Libcontainer—A Native Go Implementation for Creating Containers*. [Online]. Available: https://github.com/opencontainers/runc/tree/master/libcontainer
[28] D. Messina. (2018). *5 Years Later, Where Are You on Your Docker Journey?* [Online]. Available: https://blog.docker.com/2018/03/5-years-later-docker-journey/
[29] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, and W. Zhou, "A comparative study of containers and virtual machines in big data environment," 2018, *arXiv:1807.01842*. [Online]. Available: http://arxiv.org/abs/1807.01842
[30] S. Maheshwari, S. Deochake, R. De, and A. Grover, "Comparative study of virtual machines and containers for DevOps developers," 2018, *arXiv:1808.08192*. [Online]. Available: http://arxiv.org/abs/1808.08192
[31] *LXC-LinuxContainers*. [Online]. Available: https://linuxcontainers.org/
[32] *OracleVirtualBox*. [Online]. Available: https://www.virtualbox.org/
[33] P. Bellavista and A. Zanni, "Feasibility of fog computing deployment based on docker containerization over RaspberryPi," in *Proc. 18th Int. Conf. Distrib. Comput. Netw. (ICDCN)*, 2017, pp. 1–10.
[34] R. Morabito, "Virtualization on Internet of Things edge devices with container technologies: A performance evaluation," *IEEE Access*, vol. 5, pp. 8835–8850, 2017.
[35] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-Service at the edge: trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Commun.*, vol. 24, no. 3, pp. 48–56, Jun. 2017.
[36] L. Deshpande and K. Liu, "Edge computing embedded platform with container migration," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2017, pp. 1–6.
[37] R. Morabito, I. Farris, A. Iera, and T. Taleb, "Evaluating performance of containerized IoT services for clustered devices at the network edge," *IEEE Internet Things J.*, vol. 4, no. 4, pp. 1019–1030, Aug. 2017.
[38] S. Badiger, S. Baheti, and Y. Simmhan, "Violet: A large-scale virtual environment for Internet of Things," in *Proc. Eur. Conf. Parallel Process.* Springer, 2018, pp. 309–324.
[39] *Docker Swarm*. [Online]. Available: https://docs.docker.com/engine/swarm/
[40] *Kubernetes: Production–Grade Container Orchestration*. [Online]. Available: https://kubernetes.io/
[41] T. Bui, "Analysis of docker security," 2015, *arXiv:1501.02967*. [Online]. Available: http://arxiv.org/abs/1501.02967
[42] R. Yasrab, "Mitigating docker security issues," 2018, *arXiv:1804.05039*. [Online]. Available: http://arxiv.org/abs/1804.05039
[43] *Docker CE*. [Online]. Available: https://docs.docker.com/install/linux/docker-ce/ubuntu/
[44] *Docker Compose*. [Online]. Available: https://docs.docker.com/compose/
[45] *YAML—A Human Friendly Data Serialization Standard*. [Online]. Available: https://yaml.org/
[46] *Alpine Linux*. [Online]. Available: https://alpinelinux.org/
[47] *Django*. [Online]. Available: https://www.djangoproject.com/
[48] *RFC7540-HTTP/2*. [Online]. Available: https://tools.ietf.org/html/rfc7540
[49] *RFC6455-WebSocket*. [Online]. Available: https://tools.ietf.org/html/rfc6455
[50] *WSGI (Python Web Server Gateway Interface)*. [Online]. Available: https://www.python.org/dev/peps/pep-3333/
[51] *ASGI (Asynchronous Server Gateway Interface)*. [Online]. Available: https://asgi.readthedocs.io/en/latest/specs/main.html
[52] *Supervisor*. [Online]. Available: http://supervisord.org/
[53] *Gunicorn*. [Online]. Available: https://gunicorn.org/
[54] *Gevent*. [Online]. Available: http://www.gevent.org/
[55] *Daphne*. [Online]. Available: https://github.com/django/daphne
[56] *Logrotate-automatic rotation compression, removal log files*. [Online]. Available: https://github.com/logrotate/logrotate
[57] *iptables*. [Online]. Available: https://netfilter.org/projects/iptables/index.html
[58] *MongoDB*. [Online]. Available: https://www.mongodb.com/
[59] *Redis*. [Online]. Available: https://redis.io/
[60] *Let's Encrypt*. [Online]. Available: https://letsencrypt.org/
[61] *Certbot*. [Online]. Available: https://certbot.eff.org/
[62] *Network Operation Center of NCKUEE*. [Online]. Available: https://noc.ee.ncku.edu.tw/homepage/index.html
[63] *NGINX RTMP–Module—NGINX–Based Media Streaming Server*. [Online]. Available: https://githubcom/arut/nginx-rtmp-module
[64] *Real–Time Messaging Protocol (RTMP) Specification*. [Online]. Available: https://www.adobe.com/devnet/rtmp.html
[65] *FFmpeg-A Complete, Cross–Platform Solution to Record, Convert and Stream Audio and Video*. [Online]. Available: https://ffmpeg.org/
[66] *RFC8216—HTTP Live Streaming*. [Online]. Available: https://tools.ietf.org/html/rfc8216
[67] *jQuery Schedule—A Schedule Management With jQuery*. [Online]. Available: https://github.com/Yehzuna/jquery-schedule
[68] *Ajax-Asynchronous JavaScript + XML*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX
[69] *RFC2818-HTTPOverTLS*. [Online]. Available: https://tools.ietf.org/html/rfc2818
[70] *Buildx-Docker CLI Plugin for Extended Build Capabilities With BuildKit*. [Online]. Available: https://github.com/docker/buildx
[71] *Docker Swarm Visualizer—A Visualizer for Docker Swarm Mode*. [Online]. Available: https://github.com/dockersamples/docker-swarm-visualizer
[72] *SSL Server Test (Powered by Qualys SSL Labs)*. [Online]. Available: https://www.ssllabs.com/ssltest/
[73] R. Sairam, S. S. Bhunia, V. Thangavelu, and M. Gurusamy, "NETRA: Enhancing IoT security using NFV-based edge traffic analysis," *IEEE Sensors J.*, vol. 19, no. 12, pp. 4660–4671, Jun. 2019.
[74] A. Souri, A. Hussien, M. Hoseyninezhad, and M. Norouzi, "A systematic review of IoT communication strategies for an efficient smart environment," *Trans. Emerg. Telecommun. Technol.*, to be published, doi: 10.1002/ett.3736.
[75] E. Petritoli, F. Leccese, S. Pizzuti, and F. Pieroni, "Smart lighting as basic building block of smart city: An energy performance comparative case study," *Measurement*, vol. 136, pp. 466–477, Mar. 2019.
[76] P. Mohandas, J. S. A. Dhanaraj, and X.-Z. Gao, "Artificial neural network based smart and energy efficient street lighting system: A case study for residential area in Hosur," *Sustain. Cities Soc.*, vol. 48, Jul. 2019, Art. no. 101499.
[77] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: A comprehensive review," *J. Grid Comput.*, pp. 1–42, Sep. 2019.

YU-SHENG YANG is currently pursuing the Ph.D. degree with the Department of Engineering Science, National Cheng Kung University, Taiwan. Her research interests include the Internet of Things, ubiquitous learning, and network security.

SHIH-HSIUNG LEE received the B.Sc. degree from the Department of Applied Mathematics, National Chung Hsing University, in 2007, the M.Sc. degree from the Department of Computer Science and Information Engineering, National Cheng Kung University, in 2009, and the Ph.D. degree from the Institute of Computer and Communication Engineering, National Cheng Kung University, in 2018. He is currently an Assistant Professor with the Department of Intelligent Commerce, National Kaohsiung University of Science and Technology. His research interests include the Internet of Things, intelligent computing, signal processing, deep learning, machine learning, and computer vision.

GUAN-SHENG CHEN received the B.S. degree in computer science engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, and the M.S. degree in computer and communication engineering from National Cheng Kung University, Tainan, Taiwan. He is currently working with Chunghwa Telecom Company Ltd., Taipei, Taiwan. His research interests include web design, network security, and the Internet of Things.

CHU-SING YANG (Member, IEEE) received the B.Sc. degree in engineering science and the M.Sc. and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1976, 1984, and 1987, respectively. He joined the Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan, as an Associate Professor, in 1988, where he has been a Professor with the Department of Computer Science and Engineering, since 1993. He was the Chair of the Department of Computer Science and Engineering, National Sun Yat-sen University, from August 1995 to July 1999, where he was the Director of the Computer Center, from August 1998 to October 2002. He joined the Department of Electrical Engineering, National Cheng Kung University, as a Professor, in 2006. He participated in the design and deployment of Taiwan Advanced Research and Education Network. He was the Deputy Director of the National Center for High-performance Computing, Taiwan, from January 2007 to December 2008. He is currently a Professor of electrical engineering with the Institute of Computer and Communication Engineering, National Cheng Kung University. His research interests include future classroom/meeting room, intelligent computing, and network virtualization. He was the Program Chair of ICS 1996 and the Program Co-Chair of ICPP 2003, and MTPP 2010.

YUEH-MIN HUANG (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electrical engineering from The University of Arizona, in 1988 and 1991, respectively.

He is currently a Chair Professor with the Department of Engineering Science, National Cheng Kung University, Taiwan. He has received many funded research grants from the National Science Council, Ministry of Education, the Industrial Technology of Research Institute, and the Institute of Information Industry. He has supervised over 60 Ph.D. and 250 M.S. theses students. He has coauthored three books and published more than 250 refereed journal research articles. In e-learning area, he has published more than 100 SSCI-indexed journal articles and edited three special issues in SSCI-indexed journals. His research interests include e-learning, multimedia communications, and artificial intelligence.

Dr. Huang is a Fellow of the British Computer Society, in 2011. He has received many research awards, such as the Taiwan's National Outstanding Research Award, in 2011 and 2014, which were given to Taiwan's Top 100 Scholars. According to an article published in BJET, he was ranked 3 in the world, on terms of the number of Educational Technology articles published in the period of 2012–2017. He has been invited to give talks or served in the Program Committee at the National and International Conferences. He is the Funding Chair of the International Symposium of Emerging Technologies for Education (SETE) and the International Conference of Innovative Technologies and Learning (ICITL). He served as the Director for the Disciplines of Applied Science Education and the Innovative Engineering Education, Ministry of Science and Technology, Taiwan. He serves on the Editorial Board of several international journals in the areas of educational technology, computer communications, and web intelligence, including three SSCI-indexed e-learning.

TING-WEI HOU (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from National Cheng Kung University, Tainan, Taiwan, in 1983, 1985, and 1990, respectively. From 2007 to 2014, he was the Director of the Department of Medical Informatics, National Cheng Kung University Hospital. He is currently a Professor and the Chairman of the Department of Engineering Science, National Cheng Kung University. His major research interests include smart home computing systems, medical information systems, and smart hospital computing systems.