An Intelligent Elevator Development and Management System

Lan-Da Van, *Senior Member, IEEE*, Yi-Bing Lin, *Fellow, IEEE*, Tsung-Han Wu, and Yu-Chi Lin

***Abstract*—This paper proposes ElevatorTalk, an elevator development and management system based on the Internet of Things (IoT) approach called IoTtalk. This system modularizes the software into elevator components so that we can develop flexible and scalable car scheduling algorithms. ElevatorTalk consists of three subsystems: cars, scheduler, and the elevator car operating (ECO) panel. The first two subsystems are used to develop the elevator systems, and the third subsystem is used to receive requests issued by the passengers. These three subsystems work in parallel, and communicate with each other through sending and receiving messages. ElevatorTalk can connect to a real elevator system to serve as the elevator management center. It can also emulate the existing elevator systems with different car scheduling algorithms. We propose an intelligent aggressive car scheduling with initial car distribution (ACSICD) algorithm in ElevatorTalk. Our paper indicates that ACSICD has better waiting/travel/journey time performance and/or accuracy than the previous proposed algorithms. We also show that in our approach, the car scheduling decision can be quickly made with 0.2010 ms, and therefore good performance in the time complexity is achieved.**

***Index Terms*—Car scheduling, elevator car operating (ECO) Panel, elevator system, emulator, intelligent, internet of things (IoT), sensor, waiting/travel/journey time.**

I. Introduction

As POPULATION increases, high-rise buildings have mushroomed around the world. Since Werner von Siemens [1] presented the world's first electric elevator in 1880, this invention has played an indispensable role in tall buildings to carry the passengers and goods. From the viewpoint of a passenger in a tall building, waiting for the elevator car during peak hours is frustrating experience. Therefore, it is essential that elevator scheduling optimizes the waiting/travel/journey times of the passengers. The elevator group control systems including single-car elevator [8], multicar elevator [3], [4], [6], [7], [9], [10], [12]–[20], [23], [24], and double-decker elevator [20] have been extensively explored in this field [2]–[24]. In [10], a simulator was developed to study the multicar elevator that consists of more than one car (cabin) in one shaft. Many researchers have applied neural network [2], [5], [16], fuzzy [4], [17], genetic methods [3], [6], [9], [13], [18], and reinforcement learning [15], [16], [21] to optimize elevator car scheduling. A good survey of elevator group control can be found in [20]. Some works have proposed solutions to resolve the elevator car scheduling problems [2]–[10], [12]–[20], [23]. Most recent results have been summarized and compared in [18], [20] and [21]. In this paper, we propose the ElevatorTalk system and compare our approach with the previous studies. The originality/contributions of this paper are described as follows.

1) The ElevatorTalk system is proposed to modularize the software into elevator components such that the flexible and scalable car scheduling algorithms can be easily developed.
2) The ElevatorTalk can connect to a real elevator system to serve as the elevator management center and can emulate the existing elevator systems with different car scheduling algorithms.
3) An intelligent aggressive car scheduling with initial car distribution (ACSICD) algorithm is proposed to reduce the waiting/travel/journey time compared with other algorithms.
4) The effects of the variable measured door open delay and fixed door open delay are discussed for accuracy.

The paper is organized as follows. Section II describes the ElevatorTalk architecture. Section III develops a parallel ACSICD implemented in the ElevatorTalk. Section IV compares the ACSICD with the previously proposed approaches. Section V summarizes our findings.

II. ElevatorTalk Architecture

Fig. 1(a) shows a four-car elevator model we built. A raspberry PI board [Fig. 1(b)] [25] is used to control the motors and doors of the elevator cars. This control board is connected to the car server [Fig. 1(c)]. The cars in the elevator system are driven by the cars subsystem implemented in the car server. The cars subsystem is instructed by the scheduler subsystem implemented in the scheduler server [Fig. 1(d)]. The scheduler subsystem reads the requests issued from the elevator car operating (ECO) panels [Fig. 1(e)]. An ECO panel can be a traditional panel mounted on the wall or a smartphone with the ECO panel app. The cars, the scheduler, and the ECO panel subsystems are implemented as cyber IoT devices in an application-layer IoT device management platform called IoTtalk [26]–[29] adopting the skills/resources [30]–[35]. Therefore, the car server, the

Manuscript received January 8, 2019; revised May 24, 2019; accepted May 25, 2019. Date of publication June 21, 2019; date of current version June 3, 2020. This work was supported in part by the Ministry of Science and Technology under Grant MOST 107-2218-E-009-020 and Grant MOST 106-2221-E-009-028-MY3 and in part by National Chiao Tung University PAIR Lab. *(Corresponding author: Lan-Da Van.)*

The authors are with the Department of Computer Science, National Chiao Tung University, Hsinchu 300, Taiwan, Republic of China (R.O.C.) (e-mail: ldvan@cs.nctu.edu.tw; liny@cs.nctu.edu.tw; stevensuperboy@gmail.com; geniusdavid869@gmail.com).

Digital Object Identifier 10.1109/JSYST.2019.2919967

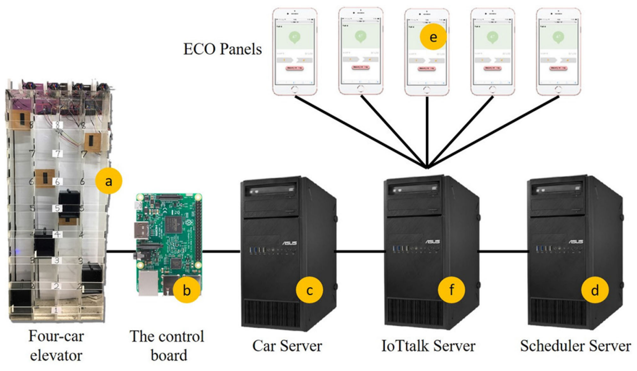


Fig. 1. ElevatorTalk hardware architecture.

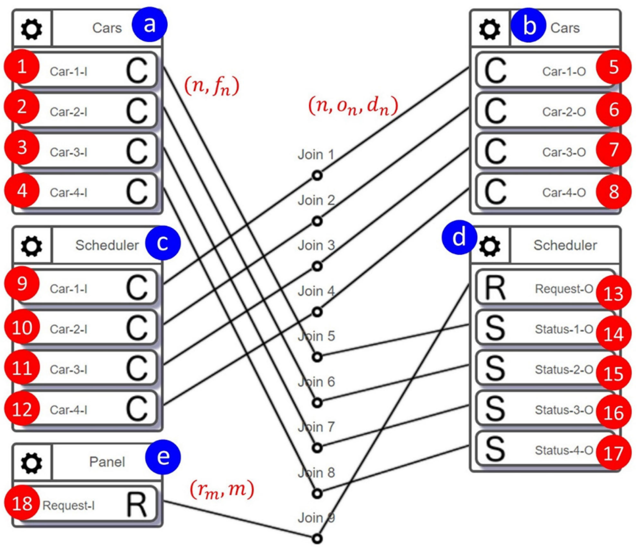


Fig. 2. Configuring ElevatorTalk through the IoTtalk GUI ($N = 4$).

scheduler server, and the ECO panel are connected to the IoTtalk server [Fig. 1(f)]. IoTtalk allows the user to quickly establish connections and meaningful interactions between IoT devices without concerning the lower layer IoT platforms/protocols (such as AllJoyn [36], oneM2M [37], and so on). IoTtalk manages IoT devices based on a concept called "device feature" (DF). A DF is a specific input or output "capability" of an IoT device. For example, an air quality monitoring device has an input DF (IDF) called the PM2.5 sensor that produces the particulate matter measuring 2.5 μg/m^3. A pair of wearable glasses with the optical head-mounted display has the output DF (ODF) called "Display." An IoT device is a collection of DFs. In IoTtalk, every IoT device can be partitioned into one input and one output devices. The input device is a subset of the IoT device consisting of the IDFs of that device. Similarly, the output device consists of the ODFs of that device. An IoT device is connected to the IoTtalk server through wired or wireless communications technologies, and network applications are automatically created/reused at the IoTtalk server for the IoT devices. When the IDFs of the IoT device produce new values, they are sent to the server, and the corresponding network application is executed to take actions, which may produce results to be sent to the ODFs of the same or other IoT devices.

In ElevatorTalk, the cars, the scheduler, and the ECO panel subsystems are cyber devices. In a typical elevator system, there is a physical ECO panel for each floor and a panel inside every car. In the elevator system, all physical panels are connected to the IoTtalk server through the ECO panel subsystem. The ECO panel subsystem represents both the floor panels and the car panels pressed by the passengers to specify the source (the start floors) and the destinations (the target floors). The scheduler subsystem accepts and handles the requests from the ECO panel subsystem, and schedules the movement of the cars. Following the instructions of the scheduler subsystem, the cars subsystem is in charge of the car operations, such as moving up or down the cars, and opening or closing doors. The scheduler server, the car server, the IoTtalk server, and the ECO panel subsystem can be installed in a local server, a private or a public cloud. The physical ECO panels and the cars/controllers are installed in the building.

IoTtalk provides a friendly graphical user interface (GUI) to specify the ElevatorTalk devices and their connection configurations. In this web-based GUI window, an input device is represented by an icon placed at the left-hand side of the window [Fig. 2(a), (c), (e)], which consists of smaller icons that represent IDFs [Fig. 2 (1)–(4), (9)–(12), and (18)]. The IDFs are appended with "-I." Similarly, an output device is represented by an icon placed at the right-hand side of the window [Fig. 2(b) and (d)], which includes ODF icons [Fig. 2 (5)–(8) and (13)–(17)]. The ODFs are appended with "-O." By connecting the IDFs to the ODFs in the GUI (joins 1–9), the devices interact with each other without any programing effort. The cars subsystem controlling N cars ($N = 4$ in this example) consists of both input and output devices. The input device has N IDFs. The IDF car-n-I [Fig. 2 (1)–(4)] provides the motion status of the nth car, where $1 \le n \le N$. The output device has N ODFs, where the ODF car-n-O [Fig. 2 (5)–(8)] accepts the requests for moving the nth car. Similarly, the scheduler subsystem consists of both input and output devices. The input device has N IDFs, where car-n-I [Fig. 2 (9)–(12)] instructs the nth car to move. The output device has $N + 1$ ODFs. The ODF request-O [Fig. 2 (13)] receives the passenger requests from the ECO panel, and the status ODF status-n-O [Fig. 2 (14)–(17)] receives the status information of the nth car. The panel subsystem has one IDF called request$-$I [Fig. 2 (18)] to send out the passenger requests.

III. ElevatorTalk Procedures and the Intelligent Algorithm

This section describes the procedures developed by the ElevatorTalk subsystems. Then, we propose an intelligent ACSICD for the elevator system and show how the algorithm is implemented in ElevatorTalk. Specifically, Section III-A implements procedure *Panel* for the ECO panel subsystem [Fig. 2(e)]. Sections III-B and III-D elaborate on four procedures *ReqArrival, SchCar*(n), *UpTaskCar*(n), and *DownTaskCar*(n) implemented in the scheduler subsystem [Fig. 2(c) and (d)]. Section III-E describes Procedures *Car*(n) for the cars subsystem [Fig. 2(a) and (b)].

In the current implementation, procedures *Car*(n) are implemented as threads in the car server, which are run in parallel.

TABLE I
SYMBOLS USED IN THE ELEVATORTALK PROCEDURES

Symbol	Description
C_B	The set of busy cars.
d_n	The current moving direction of the n-th car; $d_n \in \{Up, Down, Idle\}$.
D_m	The moving direction of r_m; $D_m \in \{Up, Down\}$.
f_n	The current floor of the n-th car.
$f_s(m)$	The start floor of r_m.
$f_{t,r}(m,k)$	The k-th target floor in r_m, where $1 \le k \le k_m$.
$f_{t,c}(n,j)$	The j-th target floor in the n-th car's stop list $S_{t,c}(n)$, where $1 \le j \le j_n$.
F	The number of floors in the building.
j_n	The number of target floors in $S_{t,c}(n)$.
k_m	The number of target floors in r_m.
L_u	The set Uplist of all unhandled $f_s(m)$, where D_m is Up.
L_d	The set Downlist of all unhandled $f_s(m)$, where D_m is $Down$.
M	The number of requests issued during the observation period.
N	The number of cars in the elevator system.
o_n	A flag that indicates if the n-th car should open door or not. $o_n \in \{0,1\}$.
r_m	The m-th request from an ECO panel, where $1 \le m \le M$; $r_m = (f_s(m), D_m, S_{t,r}(m))$.
$S_{t,c}(n)$	The set of the target floors to be stopped by the n-th car in the current moving direction; $S_{t,c}(n) = \{f_{t,c}(n,1), f_{t,c}(n,2), \ldots, f_{t,c}(n,j_n)\}$.
$S_{t,r}(m)$	The set of target floors in r_m; $S_{t,r}(m) = \{f_{t,r}(m,1), f_{t,r}(m,2), \ldots, f_{t,r}(m,k_m)\}$.
t_d	The door open delay.
t_f	The delay of car moving up/down one floor.
t_m	The arrival time of r_m; by default, $t_0 = 0$.
τ_m	The inter-arrival time between request r_{m-1} and r_m, where r_0 is a null request, and $\tau_m = t_m - t_{m-1}$.
t_s	The delay between when a request r_m arrives and when a car opens the door to handle the request.
$T_{m,n}$	The waiting time of r_m if it is served by the n-th car.
x_n	The number of known stops on the moving path of the n-th car from f_n to $f_s(m)$. Let $L_{u,n}$ be the floors in L_u, where the n-th car stops to pick the passengers. Similarly, let $L_{d,n}$ be the floors in L_d, where the n-th car stops to pick the passengers. If $d_n = Up$ then $x_n = \{x \mid x \in S_{t,c}(n) \cup L_{u,n}$ and $f_n \le x \le f_s(m)\}$. If $d_n = Down$ then $x_n = \{x \mid x \in S_{t,c}(n) \cup L_{d,n}$ and $f_s(m) \le x \le f_n\}$.

Similarly, the procedures for the scheduler subsystem are implemented as independent threads. To describe the ElevatorTalk procedures, the symbols defined in Table I are used in this paper.

A. Procedure Panel

The ECO panel subsystem implements procedure *Panel* as a thread run on the IoTtalk server. Through the IoTtalk server, the ECO panel subsystem connects to the physical ECO panels. When a passenger presses a button of an ECO panel, the instruction is captured by procedure *Panel*. Let r_m be the *m*th request issued from a panel, which has the format $r_m = (f_s(m), D_m, S_{t,r}(m))$ where

1) $f_s(m)$ is the floor from which r_m is issued,
2) $D_m \in \{\text{Up}, \text{Down}\}$ is the moving direction of r_m,
3) $S_{t,r}(m)$ is the set of target floors in r_m.

Both $f_s(m)$ and D_m are known before the passenger enters the car, and $S_{t,r}(m)$ is determined after the passenger entered the car. When r_m arrives, procedure *Panel* sends r_m to the scheduler subsystem through join 9 in Fig. 2.

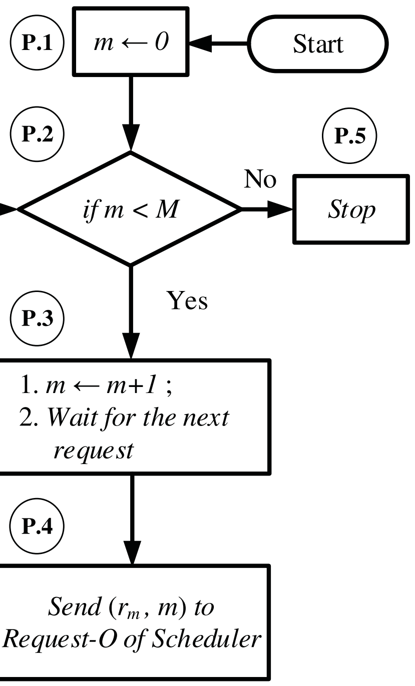

Start
P.1
m ← 0
P.2
if m < M
No
P.5
Stop
Yes
P.3
1. m ← m+1 ;
2. Wait for the next request
P.4
Send (rm, m) to Request-O of Scheduler

Fig. 3. Flowchart for procedure *Panel*.

In real world, an elevator system operates for a period of time and then is shut down for maintenance. Without loss of generality, we assume that procedure *Panel* runs to serve *M* requests, and then stops for system maintenance. The flowchart is illustrated in Fig. 3. At step P.1, there is no request, and m is set to 0 initially. Step P.2 checks if *M* requests have been served. If so, the service is stopped for maintenance at step P.5. Otherwise, wait for the next request at step P.3. At step P.4, request r_m is sent to the scheduler subsystem for service. We have also implemented a panel emulator that can generate a sequence of *M* requests to test the elevator system. The flowchart is the same as that in Fig. 3 except for step P.3. In the emulator, step P.3.2 is replaced by the following three substeps.

Step P.3.2: creates the interarrival time τ_m by a random number generator RNG() following the Poisson process (note that an arbitrary interarrival time distribution as well as trace-driven emulation can be implemented).

Step P.3.3: invokes function $\text{sleep}(\tau_m)$ to wait for a time period τ_m.

Step P.3.4: generates the next request r_m.

B. Procedure ReqArrival

Procedure *ReqArrival* is implemented as the top priority thread in the scheduler subsystem, which dispatches all tasks for scheduling. Let D_m be the moving direction for r_m. Several data structures manipulated at the scheduler subsystem are initialized by procedure *ReqArrival* (step R.1, Fig. 4), where uplist L_u is the set of all unhandled $f_s(m)$ with $D_m = \text{Up}$, downlist L_d is the set of all unhandled $f_s(m)$ with $D_m = \text{Down}$, and C_B is the set of busy cars. Since these data structures are accessed by all threads in the scheduler subsystem, they must

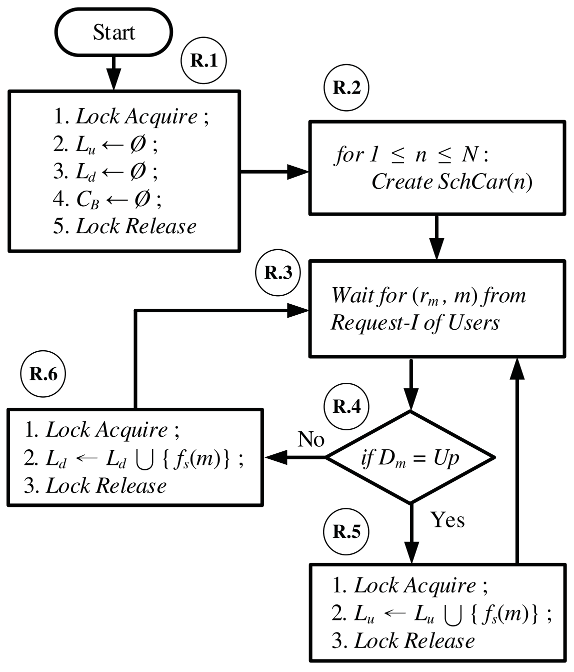

Start
R.1
1. Lock Acquire ;
2. $L_u \leftarrow \emptyset$;
3. $L_d \leftarrow \emptyset$;
4. $C_B \leftarrow \emptyset$;
5. Lock Release
R.2
for $1 \leq n \leq N$: Create SchCar(n)
R.3
Wait for (r_m, m) from Request-I of Users
R.4
if D_m = Up
No
Yes
R.5
1. Lock Acquire ;
2. $L_u \leftarrow L_u \bigcup \{f_s(m)\}$;
3. Lock Release
R.6
1. Lock Acquire ;
2. $L_d \leftarrow L_d \bigcup \{f_s(m)\}$;
3. Lock Release

Fig. 4. Flowchart for procedure *ReqArrival*.

be locked to avoid multiple updates at the same time. At step R.2, for $1 \leq n \leq N$, the scheduler subsystem creates the thread *SchCar*(*n*) to handle the request assignment for the *n*th car. Step R.3 waits for the request (r_m, *m*) issued from the ECO panel subsystem. If the moving direction of r_m is *Up* at step R.4, then the request is included in L_u (see step R.5). Otherwise, the request is included in L_d (see step R.6). Then the flow goes to step R.3 to wait for the arrival of the next request. When the elevator system is turned on, *ReqArrival* repeatedly executes the loop (see steps R.3–R.6) until the system is shut down.

C. Procedure SchCar(n)

Procedure *SchCar*(*n*) is responsible for scheduling of the *n*th car. Fig. 5 illustrates the flowchart with two local data structures. The first data structure $d_n \in \{\text{Up}, \text{Down}, \text{Idle}\}$ is the current moving direction and the second data structure $S_{t,c}(n) = \{f_{t,c}\,(n, 1),\ f_{t,c}\,(n, 2), \ldots,\ f_{t,c}\,(n, j_n)\}$ is the set of the target floors to be stopped by the *n*th car in the current moving direction. In $S_{t,c}(n)$, $f_{t,c}(n, j)$ is the *j*th target floor in $S_{t,c}(n)$, where $1 \leq j \leq j_n$, and $j_n = |S_{t,c}(n)|$ is the number of target floors. At step S.1, d_n is set to idle, and $S_{t,c}(n)$ is empty. The *SchCar*(*n*) thread of the scheduler subsystem and the *Car*(*n*) thread of the cars subsystem must be synchronized. Specifically, steps S.1.3 waits for the current status (*n*, f_n) of the *n*th car from Car(*n*), and then step S.1.4 sends the acknowledgement (*n*, *0*, *Idle*) to Car(*n*) to make sure that the connection between these two threads is established. After initialization, *SchCar*(*n*) enters a loop. Step S.2 checks if the *n*th car is moving. If the car is idle, then step S.7 checks if the *n*th car should serve incoming requests based on the total number of unhandled requests. Before step S.7 is

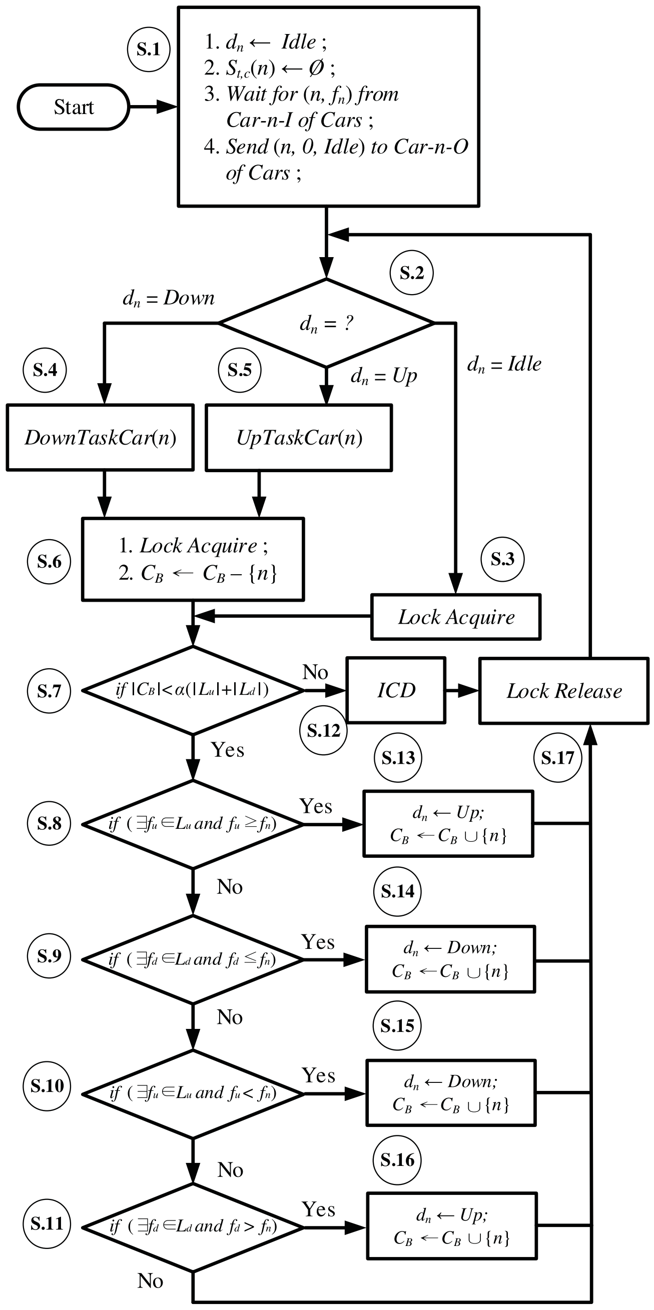

S.1
1. $d_n \leftarrow$ Idle ;
2. $S_{t,c}(n) \leftarrow \emptyset$;
3. Wait for (n, f_n) from Car-n-I of Cars ;
4. Send (n, 0, Idle) to Car-n-O of Cars ;
Start
S.2
d_n = ?
d_n = Down
d_n = Up
d_n = Idle
S.4
DownTaskCar(n)
S.5
UpTaskCar(n)
S.6
1. Lock Acquire ;
2. $C_B \leftarrow C_B - \{n\}$
S.3
Lock Acquire
S.7
if $|C_B| < \alpha(|L_u|+|L_d|)$
No
Yes
ICD
S.12
Lock Release
S.13
S.17
S.8
if ($\exists f_u \in L_u$ and $f_u \geq f_n$)
Yes
$d_n \leftarrow$ Up; $C_B \leftarrow C_B \cup \{n\}$
No
S.14
S.9
if ($\exists f_d \in L_d$ and $f_d \leq f_n$)
Yes
$d_n \leftarrow$ Down; $C_B \leftarrow C_B \cup \{n\}$
No
S.15
S.10
if ($\exists f_u \in L_u$ and $f_u < f_n$)
Yes
$d_n \leftarrow$ Down; $C_B \leftarrow C_B \cup \{n\}$
No
S.16
S.11
if ($\exists f_d \in L_d$ and $f_d > f_n$)
Yes
$d_n \leftarrow$ Up; $C_B \leftarrow C_B \cup \{n\}$
No

Fig. 5. Flowchart for procedure *SchCar*(*n*).

executed, step S.3 locks C_B, L_u, and L_d to ensure that other threads will not modify these global data structures at the same time. Step S.17 releases the lock and loops back to step S.2. At step S.7, the positive value $\alpha \leq 1$ is the aggressive factor that specifies how aggressive the elevator wants to serve the passengers. A large α value means that the system aggressively serves the requests with more cars. Specifically, if the number $|C_B|$ of busy cars is larger than or equal to $\alpha(|L_u| + |L_d|)$ then the *n*th car will not handle any request and then the function of initial car distribution (ICD) in step S.12 is executed. According to observations of the traffic patterns in an elevator system, a simple statistics method on big data [38] is used to determine

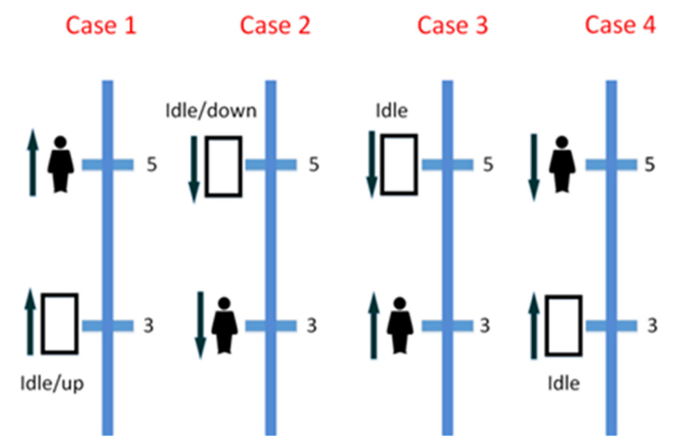


Fig. 6. Four cases for the relationship between a car and a request at floors 3 and 5.

ICD function to control the *n*th car to park at the floor that has the largest number of requests in certain periods. Specifically, ACSICD collects the traffic patterns in an elevator system. We used the statistics method to generate the histogram of the collected big data. Then from the histogram we find the floor with the highest probability for ICD. For example, in an office building, an idle car parks at the ground floor in the morning (when people are coming to their office) and parks at the highest floor in the evening (when people are leaving the building). After ICD, the control flow loops back to step S.2. Otherwise, the car takes care of an unhandled request in one of four cases illustrated in Fig. 6.

Case 1: (steps S.8 and S.13). For a passenger at a floor above the *n*th car, if he/she wants to go upward and the car is idle or moving up, then the car moves up to handle the request.

Case 2: (steps S.9 and S.14). If there is a passenger at a floor below the *n*th car, and he/she wants to go downward where the car is idle or moving down, then the car moves down to handle the request.

Case 3: (steps S.10 and S.15). If there is a passenger at a floor below the *n*th car who wants to go upward and the car is idle, then the car moves down to handle the request.

Case 4: (steps S.11 and S.16). If there is a passenger at a floor above the *n*th car, and he/she wants to go downward and the car is idle, then the car moves up to handle the request.

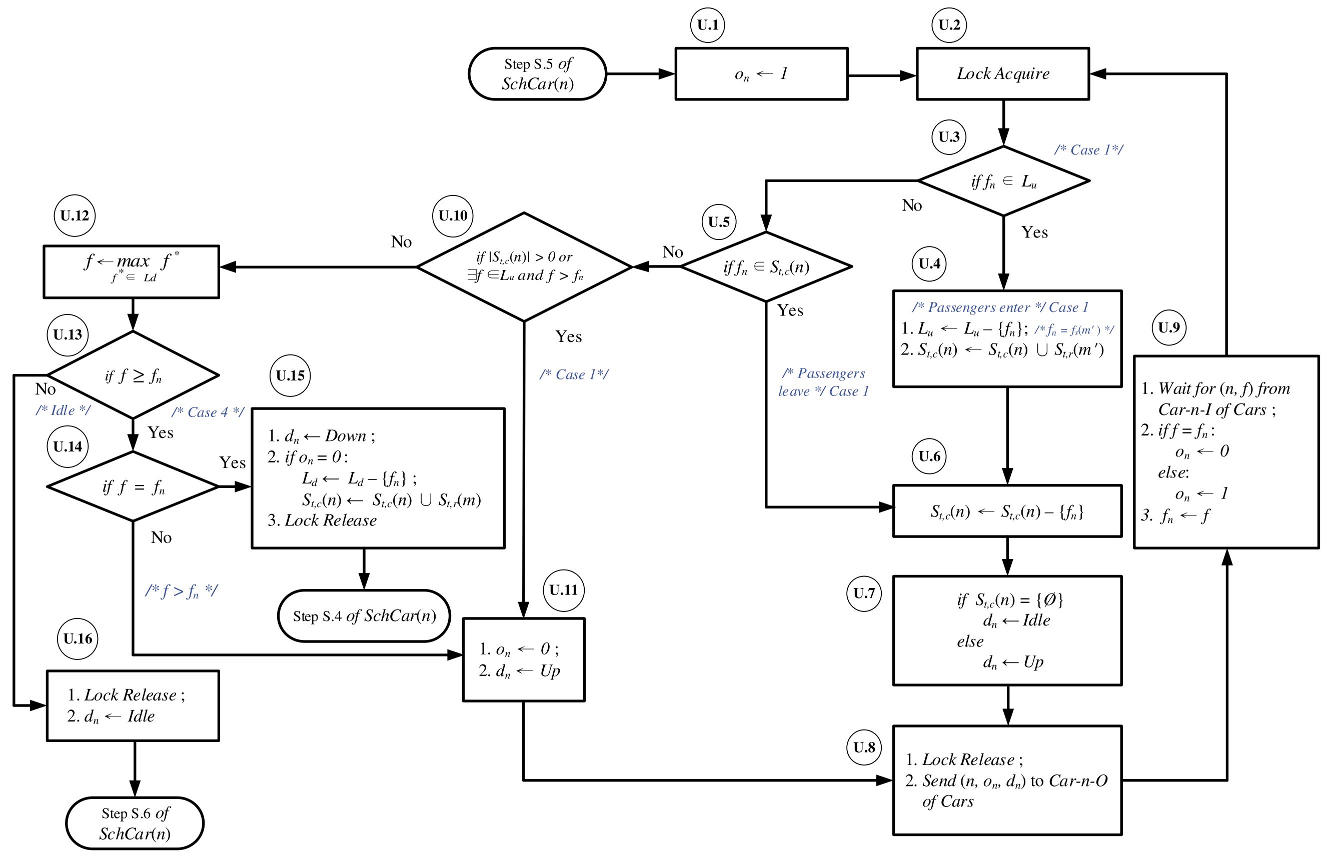
If d_n is *Down* at step S.2, the flow proceeds to step S.4 that invokes *DownTaskCar*(*n*) to handle all moving-down requests (to be elaborated in Section III-D). After all requests for the *n*th car have been served, step S.6 acquires the lock (just like step S.3) and proceeds to step S.7 to see if there are new tasks for the *n*th car. Similar to step S.4, if d_n is *Up* at step S.2, then step S.5 is executed to handle the moving-up requests.

D. Procedure UpTaskCar(n)

Procedure *UpTaskCar*(*n*) illustrated in Fig. 7 handles the moving-up requests. A flag $o_n \in \{0, 1\}$ indicates if the *n*th car should open the door or not when it arrives at the target floor ($o_n = 1$ for door open and $o_n = 0$ for no action). Initially, step U.1 assumes that the target floor should be opened in the coming action. Step U.3 checks if the current floor f_n of the *n*th car is in the L_u list. If so, it is case 1 in Fig. 6. As we pointed out in Section III-C, $S_{t,c}(n)$ is the set of the target floors to be stopped by the *n*th car in the current moving direction and $j_n = |S_{t,c}(n)|$. Then $S_{t,c}(n) = \{f_{t,c}(n, j)|1 \le j \le j_n\}$ where $f_{t,c}(n, j)$ is the *j*th target floor for the *n*th car. Let $S_{t,r}(m)$ be the set of target floors in r_m and $k_m = |S_{t,r}(m)|$. Then $S_{t,r}(m) = \{f_{t,r}(m, k)|1 \le k \le k_m\}$ where $f_{t,r}(m, k)$ is the *k*th target floor in r_m. Suppose that f_n is the start floor $f_s(m^*)$ of the m^*th request. Step U.4 starts serving the m^*th request by removing f_n from L_u and adding $S_{t,r}(m^*)$ to $S_{t,c}(n)$. Note that step U.4.2 actually occurs after the car door has been opened in a traditional elevator system. In a future elevator system, a passenger can specify the target floor through his/her smartphone before the car door opens. In this paper, we assume a traditional elevator system, and $S_{t,c}(n)$ is updated after the passenger has entered the car. Step U.6 removes f_n (i.e., $f_s(m^*)$) from $S_{t,c}(n)$. Step U.7 checks if $S_{t,c}(n)$ is empty after the m^*th request is served. If so, the car is idle. Otherwise, the car keeps moving up. Step U.8 instructs the *n*th car to take action based on flags o_n and d_n (to be elaborated). The message (n, o_n, d_n) is sent from car-*n*-I of the scheduler subsystem to car-*n*-O of the cars subsystem in Fig. 2. Step U.9 waits for the message (n, f) to be sent from car-*n*-I of the cars subsystem to status-*n*-O of the scheduler subsystem. This message indicates that the *n*th car has moved to floor f. If f is f_n, then the o_n flag is set to 0 to indicate that the door has already been opened at floor f and there is no need to open the door before the car visits this floor again. Then the current floor f_n is set to f. The procedure loops back to step U.2 to handle the next request in L_u.

If the current floor f_n of the *n*th car is not found in the L_u list at step U.3, then step U.5 checks if f_n is in the $S_{t,c}(n)$ list. If so, some passengers will leave the *n*th car at floor f_n, and steps U.6–U.9 are executed to serve this request. If f_n is not found in the $S_{t,c}(n)$ list at step U.5, it means that no passenger wants to enter or leave floor f_n. Step U.10 checks if there is a target floor that can be assigned to the *n*th car (i.e., $|S_{t,c}(n)| > 0$) or a passenger, at a floor above the *n*th car needs to move up (i.e., $\exists f \in L_u$ such that $f > f_n$). If so, steps U.11 and U.8 are executed to instruct the *n*th car to move up (d_n is set to *Up*) without opening the door at the current floor f_n; i.e., o_n is set to *0*. If the *n*th car does not need to move up to serve any request at step U.10 (i.e., the *n*th car is idle), then step U.13 checks if there is any passenger at a floor above or at f_n and needs to move down. If so (case 4 in Fig. 6 occurs) and if the passenger resides at f_n, then step U.15 serves this request by executing *DownTaskCar*(*n*) [i.e., Step S.4 of *SchCar*(*n*)]. If the passenger does not reside at f_n, then step U.11 sets the o_n flag to 0 (no need to open the door) and sets the d_n flag to *Up* (moving up the car to pick up the passenger). If step U.13 finds no passenger to move down at a floor above or at f_n, then step U.16 sets the *n*th car idle to exit *UpTaskCar*(*n*) and go back to step S.6 of *SchCar*(*n*).

The flowchart for *DownTaskCar*(*n*) is similar to that for *UpTaskCar*(*n*), which handles cases 2 and 3 in Fig. 6, and the details are omitted.



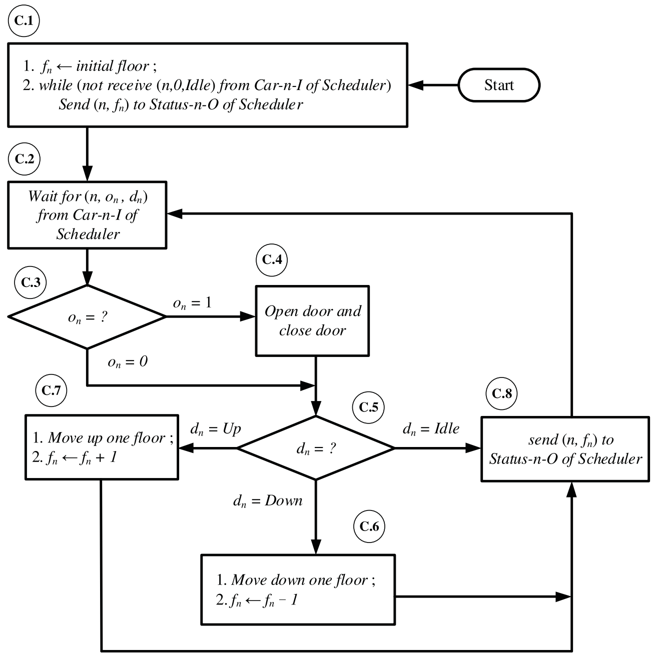
Step S.5 of SchCar(n)
U.1
$o_n \leftarrow 1$
U.2
Lock Acquire
U.3
/* Case 1*/
if $f_n \in L_u$
No
Yes
U.4
/* Passengers enter */ Case 1
1. $L_u \leftarrow L_u - \{f_n\}$; /* $f_n = f_r(m')$ */
2. $S_{t,c}(n) \leftarrow S_{t,c}(n) \cup S_{t,r}(m')$
U.5
if $f_n \in S_{t,c}(n)$
No
Yes
/* Passengers leave */ Case 1
U.6
$S_{t,c}(n) \leftarrow S_{t,c}(n) - \{f_n\}$
U.7
if $S_{t,c}(n) = \{\emptyset\}$ $d_n \leftarrow$ Idle else $d_n \leftarrow$ Up
U.8
1. Lock Release ; 2. Send (n, o_n, d_n) to Car-n-O of Cars
U.9
1. Wait for (n, f) from Car-n-I of Cars ; 2. if $f = f_n$: $o_n \leftarrow 0$ else: $o_n \leftarrow 1$ 3. $f_n \leftarrow f$
U.10
if $|S_{t,c}(n)| > 0$ or $\exists f \in L_u$ and $f > f_n$
No
Yes
/* Case 1*/
U.11
1. $o_n \leftarrow 0$; 2. $d_n \leftarrow$ Up
U.12
$f \leftarrow \max_{f^* \in L_d} f^*$
U.13
if $f \geq f_n$
No
/* Idle */
/* Case 4 */
Yes
U.14
if $f = f_n$
Yes
No
/* $f > f_n$ */
U.15
1. $d_n \leftarrow$ Down ; 2. if $o_n = 0$: $L_d \leftarrow L_d - \{f_n\}$; $S_{t,c}(n) \leftarrow S_{t,c}(n) \cup S_{t,r}(m)$ 3. Lock Release
Step S.4 of SchCar(n)
U.16
1. Lock Release ; 2. $d_n \leftarrow$ Idle
Step S.6 of SchCar(n)

Fig. 7. Flowchart for procedure *UpTaskCar*(n).

E. Procedure Car(n)

For $1 \leq n \leq N$, procedure *Car*(n) is run on a thread of the cars subsystem. Therefore, there are N threads executed in parallel. The flow chart of procedure *Car*(n) is shown in Fig. 8. When the nth car is turned on at step C.1, *Car*(n) reports the floor f_n where it resides, and then tries to conduct handshake with *SchCar*(n) to establish the communication path. Specifically, the cars subsystem sends (n, f_n) from car-n-I to status-n-O of the scheduler subsystem through joins 5–8 in Fig. 2 (see also steps S.1.3 and S.1.4 in Fig. 5), and waits for the response $(n, 0, \text{ Idle})$ from car-n-I of the scheduler subsystem. After handshaking, step C.2 waits for the next message $(n, o_n, \ d_n)$ from car-n-I of the scheduler subsystem (through joins 1–4 in Fig. 2).

If the o_n flag is 1 at step C.3, then the nth car opens the door, waits for the passengers to pass through the car door, and closes the door at step C.4. (In the emulation, this step executes the $\text{sleep}(t_d)$ function to emulate the delay of door opening/closing). If d_n is not idle, then the car moves down one floor at step C.6 or moves up one floor at step C.7, and updates the current floor f_n. If d_n is idle, then the car does nothing. In the emulation, both steps C.6 and C.7 execute the $\text{sleep}(t_f)$ function to emulate the delay of car movement. The flow proceeds to step C.8 to send (n, f_n) from car-n-I to status-n-O of the scheduler subsystem through joins 5–8 in Fig. 2. Then the procedure loops back to step C.2.



C.1
1. $f_n \leftarrow$ initial floor ; 2. while (not receive (n,0,Idle) from Car-n-I of Scheduler) Send (n, f_n) to Status-n-O of Scheduler
Start
C.2
Wait for (n, o_n, d_n) from Car-n-I of Scheduler
C.3
$o_n = ?$
$o_n = 1$
$o_n = 0$
C.4
Open door and close door
C.5
$d_n = ?$
d_n = Up
d_n = Idle
d_n = Down
C.7
1. Move up one floor ; 2. $f_n \leftarrow f_n + 1$
C.8
send (n, f_n) to Status-n-O of Scheduler
C.6
1. Move down one floor ; 2. $f_n \leftarrow f_n - 1$

Fig. 8. Flowchart of procedure *Car*(n).

TABLE II
COLLECTED ELEVATOR DATA IN 1 WEEK

Date	Number of door openings	E[t_d] (seconds)
2017/8/27 (Sun)	468	8.15
2017/8/28 (Mon)	1147	8.18
2017/8/29 (Tue)	1162	7.69
2017/8/30 (Wed)	1018	8.27
2017/8/31 (Thr)	1149	7.64
2017/9/1 (Fri)	992	7.4
2017/9/2 (Sat)	334	7.72

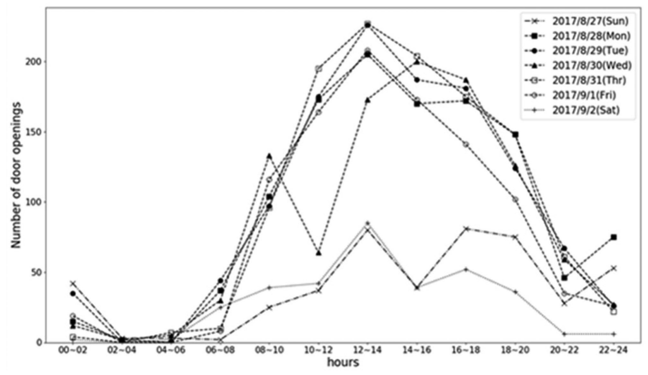


Fig. 9. Number of door openings in one week.

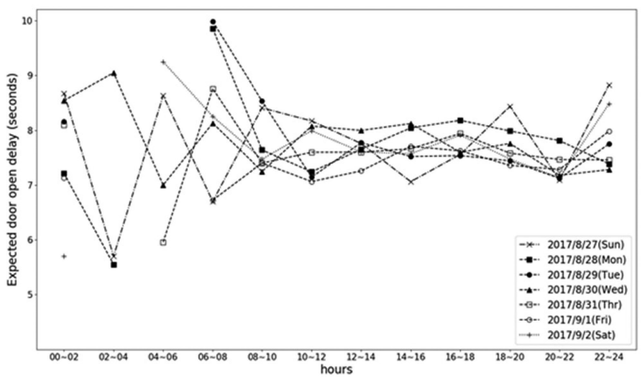


Fig. 10. Expected door open delay $E[t_d]$ in one week.

TABLE III
PERFORMANCE FOR VARIOUS SCHEDULING ALGORITHMS [18] WITH FIXED t_d

Algorithm	Waiting time (s)	Travel time (s)	Journey time (s)
Duplex	143	N/A	N/A
GA1	143	N/A	N/A
GA3	99	N/A	N/A
GA4	80	205	285
ACSICD	99	163	262

IV. PERFORMANCE EVALUATION

Performance analysis of ACSICD in Section III is evaluated and compared with the previous proposed algorithms in this section. In Section IV-A, real data with time delays and the number of door openings have been recorded since 2017. We particularly analyze the collected data during the week of August 27, 2017. In Section IV-B, the simulated passenger requests of the scenario in [18] are treated as the input of ACSICD to produce the emulation results. Emulation with fixed and variable door open delays is used to evaluate the performance of ACSICD as well as the previous approaches. In Section IV-C, the stability margin and reliability of ElevatorTalk are addressed. In Section IV-D, the limitation and possible usage variation are briefly discussed.

A. Measurements of Door Open Delays

In [22], a sensor system is set up to monitor a six-floor elevator in the Department of Computer Science Building in National Chiao Tung University (NCTU). The sensor system has collected the data including the car status (idle/moving up/moving down/door open/door close) and timestamps of door opening and closing since 2017. In this paper, the data collected during the week of August 27 in 2017 are used to investigate the door open delays t_d and the delays t_f of car moving up or down one floor. Table II lists the numbers of door openings and the expected value $E[t_d]$ in the week. The table indicates that the number of door openings in a week day is about three times of that in a weekend, and $E[t_d]$ ranges from 7.4 to 8.27 s. Fig. 9 illustrates the one-week histograms of the numbers of door openings occurring in 2-hour slots in a day. From the curves, we observe one major peak at the 12–14 slot and two small peaks at the 8–10 and the 16–18 slots. Fig. 10 plots $E[t_d]$ for every 2-h time slot in the week. The curves indicate that t_d may vary significantly and using fixed t_d to estimate the behavior of the elevator may not be practical (to be elaborated in Section IV-B). For car movement, we found that the delay t_f is roughly fixed to 2 s.

B. Comparing ACSICD With the Previously Proposed Algorithms

In Appendices A and B, we have described several previously proposed algorithms. Emulation of the proposed ACSICD in Section III is conducted with the same scenario as the simulation experiment performed in Appendix A [18]. In our emulation experiments, the actual delays are emulated through the sleep() function. We measure the waiting time (when a passenger arrives at the start floor and when a car door opens), the travel time (when the passenger enters the car and when the car arrives at the target floor), and the journey time (the waiting time + the travel time).

Based on the six-request scenario with fixed t_d in [18], Table III lists the measured times for various algorithms. Among them, the duplex algorithm [23] is a conventional approach that exercises full collective control of the cars. If the elevator system has completely served all incoming requests, then before the next request arrives, the idle cars are moved evenly among the floors. When the new requests arrive, the scheduler dispatches the cars to serve the nearest requests with the same moving direction. Details of other three genetic algorithms GA1, GA3, and GA4 are given in Appendix A.

The measured times for the duplex, the GA1, the GA3, and the GA4 algorithms are obtained directly from the results reported in [9] and [18]. From Table III, it is clear that our solution has

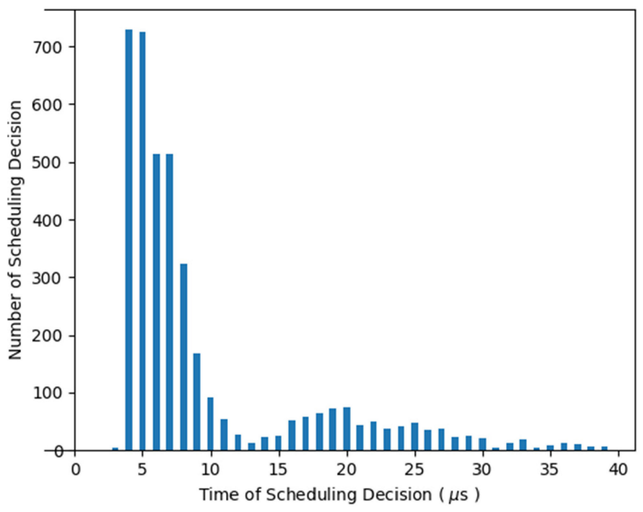


Fig. 11. Distribution of scheduling decision time delay t_s.

TABLE IV
PERFORMANCE FOR VARIOUS SCHEDULING ALGORITHMS

Algorithm	**Max waiting time (s)**	**Average waiting time (s)**	**Average travel time (s)**	**Average journey time (s)**	**Average moves per day**
ACSICD (α = 1)	218	24.07	36.09	60.16	6504
ACSICD (α = 2)	90	18.85	35.69	54.54	7325
LQF [21]	253	19.76	38.91	58.67	N/A
RLS [21]	218	17.42	40.65	58.07	N/A

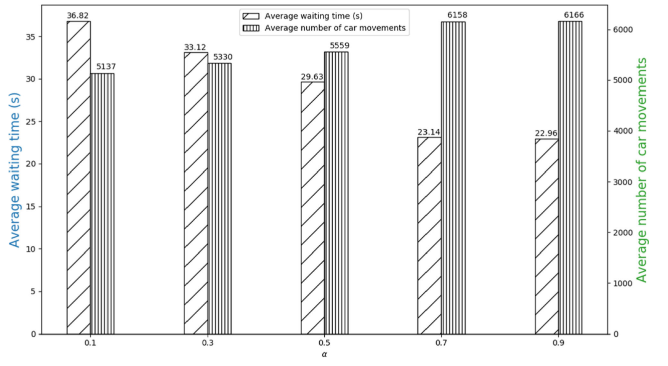


Fig. 12. Impact of α $(N = 4)$.

improved the waiting times of the duplex and the GA1 algorithms. Both GA3 and ACSICD have the same waiting time performance. GA4 has best waiting time performance, but has much longer travel time than ACSICD. Therefore, in terms of the journey time, ACSICD is better than GA4.

The above results are obtained based on fixed t_d, which may not reflect the real world. If we consider the t_d delays measured in Section IV-A, then the expected total waiting time for ACSICD is 106.38 s in a weekday and 103.88 s in a weekend. Clearly, using the measured t_d from a real elevator system, the results are quite different. Therefore, simply assuming that t_d is fixed to 7 s as did in [18] is not appropriate. We use the measured data in Section IV-A to conduct ACSICD to select the cars, and check if the selected cars are the same as GA3's estimation using fixed t_d. Our paper indicates that up to 49.2% of the prediction made by GA3 with fixed t_d is different from prediction made with the measured t_d. Therefore it is not appropriate to conduct estimation using fixed t_d. Among the previous car scheduling approaches, GA3/GA4 has demonstrated very good performance. Through ElevatorTalk, we pointed out the direction to enhance the accuracy of GA3/GA4.

Note that GA1–GA4 have to handle a batch of requests at a time, and the next batch of requests is ignored when the current batch is handled. On the other hand, ACSICD proposed in this paper is not restricted by the inflexibility of "batch handling."

The time complexity of a car scheduling algorithm is defined as the delay t_s between when a request m is made and when a car opens the door to handle the request. The schedule time performance is not reported in [9] and [18]. However, it is well known that the execution of the genetic algorithm is time consuming [11]. On the other hand, ACSICD can schedule a request in real time. We have conducted 4397 measurements, and Fig. 11 shows the t_s histogram where $E[t_s] = 0.201$ ms. The measurements indicate that ElevatorTalk can schedule a request in real time.

We further compare the performance of ACSICD with RLS, a machine learning scheduler implemented in reinforcement learning in [21]. The implementation details are described in Appendix B. The traffic patterns for 100 days (from 7:00 AM to 9:00 PM every day) [21] are used to show the performance including the average waiting time, the average travel time, the average journey time, and the average number of car movements of the proposed approaches listed in Table IV. The table indicates that, ACSICD $(\alpha = 2)$ has the best performance in terms of average travel time and average journey time as compared with the existing methods in [21]. ACSICD $(\alpha = 2)$ also has smallest maximum waiting time, which means that this algorithm is more stable (has small variance).

Table IV also shows that when α is large, ACSICD dispatches more cars at the same time (i.e., the algorithm is more aggressive). In particular, when $\alpha = 2$, ACSICD has the best expected journey time performance of the compared algorithms. On the other hand, when α is small $(\alpha = 1)$, the required number of movement of the cars is less, which implies less energy consumption. The car movement number is an important output measure for energy consumption, but is not reported in the previous studies.

We use the same request traffic patterns in Table IV to investigate the impact of α on ACSICD with four cars (i.e., $N = 4$). Fig. 12 plots the total waiting time and the total numbers of car movement per day for different α values. The figure indicates the intuitive results that as α increases, the waiting time decreases and the total number of car movement increases. Specifically, by increasing α from 0.1 to 0.9, the waiting time is decreased by 38% at the cost of increasing the number of movement by 20%. Therefore, the operator can trade off the user experience (waiting) and energy consumption (car movement) by choosing an appropriate α value.

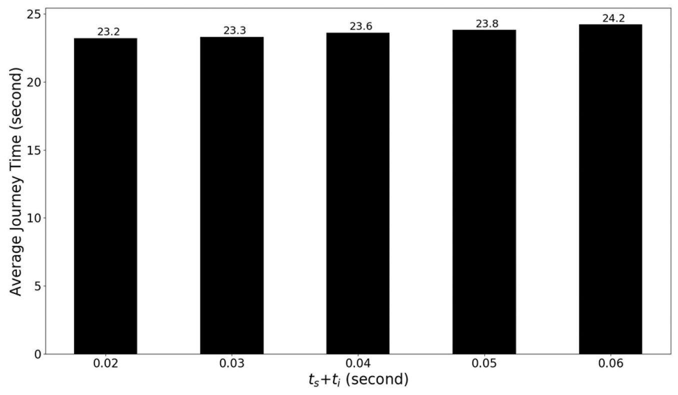


Fig. 13. Effect of $t_s + t_i$ on the average journey time.

C. Stability Margin and Reliability

The stability margin for communication is determined by the processing/communication delays and the movement/door operation delays of cars. The delays for ElevatorTalk decision is $E[t_s] = 0.201$ ms. We also measured the delays t_i for communications, where $i = L$ represents the local scenario; i.e., the servers [Fig. 1(c), (d), and (f)] are connected together in one hop. For $i = R$, it is the remote scenario where the IoTtalk server is located in a cloud. The expected values are $E[t_L] = 31.68$ ms and $E[t_R] = 59.14$ ms. The variances are $V[t_L] = 0.0022E[t_L]^2$ and $V[t_R] = 0.0026E[t_R]^2$. On the other hand, the car movement takes 2 s for one floor and a door operation takes even much longer time. We have conducted experiments to see how different $t_s + t_i$ values affect the journey times in Fig. 13. The figure indicates that for the local scenario, $t_s + t_i \approx 0.03$ s, and the journey time is affected by 0.43%. In the remote scenario (where the IoTtalk server is located in the cloud), $t_s + t_i \approx 0.06$ s, and the journey time is affected by 4.3%. The experiments indicate that the car scheduling results are not affected in the local scenario, and is not significantly affected in the remote scenario.

In terms of reliability, the two-way communication protocols for the current ElevatorTalk version include https to ensure security and message queuing telemetry transport (MQTT) to ensure reliability. At the application level, IoTtalk is a commercial-grade mechanism, which has been intensively tested by various IoT applications in the past 5 years. These applications are not for demo. Instead, they are sustainably operated for commercial scenarios. For example, in AgriTalk [39], IoTtalk has been used in the large-scale farms for smart control since 2016. Therefore, reliability of IoTtalk has been strictly tested.

D. Limitation and Usage Variation

We are integrating ElevatorTalk with the elevator system in the Student Dorm 3 of NCTU. This elevator system was constructed by Yungtay Engineering Co., Ltd [40], the largest elevator company in Taiwan. All features described in this paper are being integrated in Yungtay's elevator system. Some features that collect sensor data for car conditions (such as motor vibration) will be integrated with ElevatorTalk in the future. Due to government regulations, the features about failures and emergency must follow the original Yungtay design, and will not be integrated with ElevatorTalk. Examples include emergency power, emergency exit, and so on. In the future, we will consider ElevatorTalk version of failure and emergency scenarios to impact government regulations. For variations of usage, we are planning to integrate ElevatorTalk with other IoT applications through the IoTtalk server. For example, in a hotel, a robot delivers a meal order from the kitchen to a guest room in different floors. By installing the ECO panel software in the robot, the routing algorithm of the robot application instructs the elevator car to carry the robot to the target floor through ElevatorTalk.

V. Conclusion

This paper proposed ElevatorTalk, an IoT-based elevator scheduling system that allows the designer to develop a parallel car scheduling algorithm for an elevator system with multiple cars. We showed how to modularize the software for elevator components so that flexible and scalable car scheduling algorithms can be easily developed. ElevatorTalk consists of three subsystems: the cars, the ECO panel, and the scheduler subsystems. To create new car scheduling algorithms, the developer only needs to modify the scheduler subsystem. The first two subsystems are standard, and can be reused for all elevator systems. These three subsystems work in parallel, and communicate with each other through sending and receiving messages. ElevatorTalk can connect to a real elevator system to serve as the elevator management center. We proposed ACSICD in ElevatorTalk and compare it with the previous proposed algorithms. Our paper indicated that the ACSICD has better performance than the previous solutions. Through this comparison, we pointed out where the previous approaches can be improved. We also show that the car scheduling decision can be quickly made in our approach within 0.201 ms, and good performance in the time complexity is achieved.

IoTtalk has been tested with various applications in several years and is considered reliable. At present, the reliable communication protocol MQTT is used in the current ElevatorTalk version. The study in [41] pointed out that both MQTT and advanced message queuing protocol (AMQP) based architectures ensure reliability for IoT applications. On the other hand, the study in [42] used data distribution service (DDS) for reliable IoT implementation. Besides MQTT, we will consider AMQP and DDS in next ElevatorTalk version. In the future, we will also conduct theoretical proof for ElevatorTalk reliability using bigraph, a universal computational model defined by Milner [43] for modeling interacting systems that evolve in time and space.

Appendix A
The Genetic Algorithm-Based Scheduling

Several genetic algorithms for car scheduling are proposed in [18], and the experiments with six requests in a batch were conducted to measure the waiting, the travel, and the journey times. In the experiments, the door open delay t_d and the car movement delay t_f are fixed to be 7 and 2 s, respectively. From the measurements in Section IV-A, it is reasonable to assume a fixed t_f of 2 s. On the other hand, it is misleading to assume that t_d is fixed. However, to make a fair comparison with the

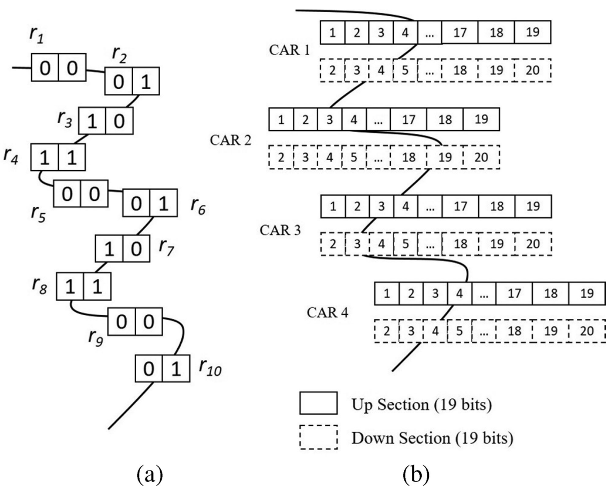


Fig. 14. Chromosome representations in (a) GA1 [9] and (b) GA3 [18].

algorithms proposed in [18], we follow the fixed t_d assumption. We relax this restriction by using the real data measured in Section IV-A, and show that the fixed t_d assumption may be misleading.

GA1 [9] and GA3 [18] are genetic algorithm-based approaches, where GA3 is an enhancement of GA1. As in a traditional elevator system, both GA1 and GA3 as well as ACSICD assume that the target floors are determined after the passengers have entered the car. In genetic algorithm-based car scheduling, every request allocation assignment is represented by a chromosome encoded as a binary string. For example, Fig. 14(a) illustrates the chromosome of GA1. This chromosome is composed of 20 genes, where every gene is represented by one bit. For $1 \leq i \leq 20$, the *i*th and the $(i+1)$ th bit (genes) are grouped to represent the elevator car that serves the $\frac{i+1}{2}$th request. Therefore, GA1 is designed for a four-car system (every car is represented by two bits), and can handle at most ten requests at a time. Denote a GA1 chromosome as a 20-bit string g_1, where the car that serves the *m*th request is represented by the $(2m-1)$th and $(2m)$th bits, which is denoted as $g_1(m)$. When the requests arrive, the GA1 algorithm is executed with the following steps.

Step GA1.1: Randomly generate six chromosomes with ten requests.

Step GA1.2: Compute the fitness value of each chromosome according to the fitness function $F(g_1)$ for every chromosome g_1, which is the inverse of the total waiting time for serving the ten requests based on the car scheduling encoded in g_1, that is,

$$\frac{1}{F\left(g_1\right)} = \left(\frac{1}{10}\right) \sum_{m=1}^{10} T_{m,\, g_1(m)}$$

where $T_{m,\, g_1(m)}$ is the waiting time for car $g_1(m)$ to serve the request r_m.; i.e., the delay between when the passenger makes the request and when the car arrives at the floor to pick the passenger. $T_{m,\, g_1(m)}$ for GA1 is similar to (A.1)–(A.6) for GA3 to be elaborated later. Clearly, the larger the $F(g_1)$ value, the better the waiting time performance.

Step GA1.3: Rank the six chromosomes according to their fitness values.

Step GA1.4: Replace the chromosome that has lowest fitness value with the one that has maximum fitness value.

Step GA1.5: Execute crossover process on chromosomes in pairs with a given probability.

Step GA1.6: Execute mutation process to randomly change the bit value of one gene in every chromosome.

Step GA1.7: Go to step GA1.2 if the newly produced generation of chromosomes does not meet the converge condition. Otherwise, go to step GA1.8.

Step GA1.8: Choose the chromosome with the best fitness value. Instruct the cars to serve the requests based on this chromosome.

Step GA1.9: Go to step GA1.1. to handle next ten requests.

GA3 extends the GA1 chromosome to 152 bits [Fig. 14(b)] to deal with more requests (at most $2 \times (F-1)$) within four cars where $F = 20$. Since it is a four-car system, the 152-bit chromosome g_3 is divided into four parts, one for each car. For $1 \leq n \leq 4$, the *n*th part ranges from bit $38n-37$ to bit $38n$, which indicates the floors to be stopped by the *n*th car. Because the elevator system has 20 floors, the *n*th part is further partitioned into two 19-bit subparts. The first subpart (from bit $38n-37$ to bit $38n-19$) represents floors 1–19 and the value of a bit indicates if the car will stop at this floor when it moves up. On the other hand, the second subpart (from bit $38n-18$ to bit $38n$) represents floors 2–20 and the value of a bit indicates if the car will stop at this floor when it moves down. The GA3 algorithm executes the following steps.

Step GA3.1: Generate 50 chromosomes randomly.

Step GA3.2: Compute the fitness value of each chromosome according to the fitness function (g_3), which is the inverse of the total waiting for serving the *M* requests based on the car scheduling encoded in g_3. This function is similar to (g_1), and the details are omitted.

Step GA3.3: Randomly select a pair of chromosomes (the parents).

Step GA3.4: The selected parent chromosomes produce a pair of offspring chromosomes (children) by crossover and mutation.

Step GA3.5: Replace the parent chromosomes with offspring chromosomes.

Step GA3.6: Go to step GA3.2 if the number of generations is less than 100 (the converge criterion).

Step GA3.7: Choose the chromosome with the best fitness value. Instruct the cars to serve the requests based on this chromosome.

Step GA3.8: Go to step GA3.1 to handle the next batch of requests.

To attain more precise waiting time estimation, GA3 uses an argument x_n that denotes the number of known stops on the moving path of the *n*th car from f_n to $f_s(m)$. Except for chromosome encoding and the definition of waiting time equations, GA1 and GA3 use similar processes to determine request assignment. Specifically, both GA1 and GA3 assume that the target floors of the requests are unknown during scheduling. We elaborate more on GA3 about the heuristics used in this algorithm.

If the *n*th car is assigned to serve r_m, the waiting time $T_{m,n}$ is defined as the sum of delays for moving the *n*th car from f_n to the start floor $f_s(m)$ of r_m and the door open delays of the known stops within the movement path. In GA3, $T_{m,n}$ is estimated by one of the following equations [18]. For $d_n = \text{Up}$,

$$T_{m,n} = \begin{cases} [f_s(m) - f_n]t_f + x_n t_d;\ f_s(m) \geq f_n \text{ and } D_m = Up & \text{(A.1)} \\ [2F - f_n - f_s(m)]t_f + x_n t_d;\ D_m = \text{Down} & \text{(A.2)} \\ [2F - f_n + f_s(m) - 2]t_f + x_n t_d;\ f_s(m) < f_n \\ \qquad \text{and } D_m = \text{Up} & \text{(A.3)} \end{cases}$$

Symmetrical to (A.1)–(A.3), for $d_n = \text{Down}$, we have

$$T_{m,n} = \begin{cases} [f_n - f_s(m)]t_f + x_n t_d; f_s(m) \leq f_n \text{ and } D_m = \text{Down} & \text{(A.4)} \\ [f_n + f_s(m) - 2]t_f + x_n t_d;\ D_m = Up & \text{(A.5)} \\ [2F + f_n - f_s(m) - 2]t_f + x_n t_d; f_s(m) > f_n \\ \qquad \text{and } D_m = \text{Down} & \text{(A.6)} \end{cases}$$

Note that (A.1) estimates the waiting time for case 1 in Fig. 6, (A.2) and (A.6) for case 4, (A.3) and (A.5) for case 3, and (A.4) for case 2. Through (A.1)–(A.6), GA3 calculates the estimated waiting time of r_m according to the chromosome at each iteration. Also note that computation for $T_{m,n}$ of GA1 is similar to that for GA3 except that the factor x_n is not considered. GA4 is the same as GA3, except that it is used in a nontraditional elevator system where the target floors are determined before the passengers have entered the car.

Appendix B
Reinforcement Learning-Based Scheduling

Several scheduling algorithms are proposed in [21], including dynamic load balancing, collective, longest queue first (LQF), and AI methods based on reinforcement learning. Among the three non-AI algorithms, LQF has the best performance, which always serves the request with the longest waiting time. The reinforcement learning scheduling (RLS) algorithm is implemented by extending LQF (Factor "a" to be elaborated later). Whenever the RLS assigns a request to a car, a reward called priority value is calculated. The assignment of the *m*th request to the *n*th car is called an action. The history of the actions, the corresponding priority values p and the current states of cars, and requests are stored as records in a replay memory. After sufficient number of records have been collected, RLS learns to decide the best action in the current state. The priority value p is determined as [21]

$$p = \frac{\left(1 + \frac{a}{14}\right) \times c \times \left(1 + \frac{d}{5}\right) \times e}{(2 + b) \times f \times g} \qquad \text{(B.1)}$$

In (B.1), the factors a to g are defined in Table V.

ACSICD captures factors a, b, f and g listed in Table V in different ways. Factor a is basically the same as LQF where the request with the longest waiting time is served first. This factor is captured by ACSICD that serves requests in the FIFO order in L_u and L_d. According to ACSICD simulation for 100 days,

TABLE V
FACTORS FOR RLS [21]

Factor	Description
a	Current waiting time of the request at this floor.
b	Percentage of requests that start from the same floor according to previous days in the same hour; by default $b = -1$; otherwise, b is the percentage value if previous data is available.
c	The value c is increased if the waiting time of the request is longer than the average waiting time according to previous days in the same hour; by default $c = 1$; otherwise, if the waiting time of the request exceeds the average waiting according to previous days in the same hour, $c = 2$.
d	The number of people waiting at the same floor.
e	If there is only one person in the request and the car is the closest one to the request, $e = 1$; otherwise, $e = 0$.
f	If the target floor of the other car is the same as the request, $f = 12$; otherwise, $f = 1$.
g	If the request already assigned to the other car, $g = 8$; otherwise, $g = 1$.

there are at most three target floors where the car has to stop in the same direction (i.e., there are at most three requests in the queue). Hence, this factor does not have significant impact on performance. Table IV reports that ACSICD outperforms LQF.

Factor b is influenced by the percentage of the number of requests at every floor in each hour according to previous-day data. A request has higher priority to be assigned to a car by RLS if it is at the floor that had higher number of requests in previous days. This factor is captured by the ICD of ACSICD. From the learning and big data analysis similar to [38] on the request traffic of 100 days, when a car is idle and no request comes, the car is parked at the ground floor during 7:00AM–8:00AM. During 5:00PM–21:00PM, one idle car is parked at the middle floor and the other idle car is parked at the highest floor in the period.

Factors f and g ensure that the RLS do not assign a request to multiple cars. ACSICD partially captures these factors through the aggressive factor α to prevent multiple cars moving toward a request simultaneously.

References

[1] [Online]. Available: https://www.siemens.com/history/en/news/1043_elevator.htm

[2] B. L. Whitehall, D. J. Sirag Jr., and B. A. Powell, "Elevator control neural network," U.S. Patent 5,672,853, Sep. 30, 1997.

[3] A. Fujino, T. Tobita, K. Segawa, K. Yoneda, and A. Togawa, "An elevator group control system with floor-attribute control method and system optimization using genetic algorithms," *IEEE Trans. Ind. Electron.*, vol. 44, no. 4, pp. 546–552, Aug. 1997.

[4] R. Gudwin, F. Gomide, and M. A. Netto, "A fuzzy elevator group controller with linear context adaptation," in *Proc. IEEE World Congr. Comput. Intell.*, May. 1998, Vol. 1, pp. 481–486.

[5] B. L. Whitehall, T. M. Christy, and B. A. Powell, "Method for continuous learning by a neural network used in an elevator dispatching system," U.S. Patent 5,923,004, Jul. 13, 1999.

[6] J. H. Kim and B. R. Moon, "Adaptive elevator group control with cameras," *IEEE Trans. Ind. Electron.*, vol. 48, no. 2, pp. 377–382, Apr. 2001.

[7] A. Rong, H. Hakonen, and R. Lahdelma, "Estimated time of arrival (ETA) based elevator group control algorithm with more accurate estimation," *TUCS Tech. Rep. 584*, 2003, Finland, http://danielparejaortiz.es/ascensores/documentos/TR584.pdf.

[8] S. Tanaka, Y. Uraguchi, and M. Araki, "Dynamic optimization of the operation of single-car elevator systems with destination hall call registration: Part I. Formulation and simulations," *Eur. J. Oper. Res.*, vol. 167, no. 2, pp. 550–573, Dec. 2005.

[9] W. Ghareib, *Optimal Elevator Group Control Using Genetic Algorithms*. Cairo, Egypt: Faculty of Engineering, Ain Shams Univ. 2005.
[10] T. Miyamoto and S. Yamaguchi, "MceSim: A multi-car elevator simulator," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. E91-A, no. 11, pp. 3207–3214, Nov. 2008.
[11] M. B. Nia and Y. Alipouri, "Speeding up the genetic algorithm convergence using sequential mutation and circular gene methods," in *Proc. IEEE 9th Int. Conf. Intell. Syst. Des. Appl.*, Pissa, Italy, Nov. 2009, pp. 31–36.
[12] J. Sun, Q. C. Zhao, and P. B. Luh, "Optimization of group elevator scheduling with advance information," *IEEE Trans. Autom. Sci. Eng.*, vol. 7, no. 2, pp. 352–363, Apr. 2010.
[13] Z. Hu, Y. Liu, Q. Su, and J. Huo, "A multi-objective genetic algorithm designed for energy saving of the elevator system with complete information," in *Proc. IEEE Int. Energy Conf.*, 2010, pp. 126–130.
[14] A. Valdivielso, and T. Miyamoto, "Multicar-elevator group control algorithm for interference prevention and optimal call allocation," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 41, no. 2, pp. 311–322, Mar. 2011.
[15] M. Ikuta, K. Takahashi, and M. Inaba, "Strategy selection by reinforcement learning for multi-car elevator systems," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2013, pp. 2479–2484.
[16] W. Liu, N. Liu, H. Sun, G. Xing, Y. Dong, and H. Chen, "Dispatching algorithm design for elevator group control system with Q-Learning based on a recurrent neural network," in *Proc. Chin. Control Decis. Conf.*, May 2013, pp. 3397–3402.
[17] J. Fernandez, P. Cortes, J. Muñuzuri, and J. Guadix, "Dynamic fuzzy logic elevator group control system with relative waiting time consideration," *IEEE Trans. Ind. Electron.*, vol. 61, no. 9, pp. 4912–4919, Sep. 2014.
[18] E. O. Tartan, H. Erdem, and A. Berkol, "Optimization of waiting and journey time in group elevator system using genetic algorithm," in *Proc. IEEE Int. Symp. Innov. Intell. Syst. Appl.*, 2014, pp. 361–367.
[19] H. Ishihara and S. Kato, "The effectiveness of dynamic zoning in multi-car elevator control," in *Proc. IEEE 3rd Global Conf. Consumer Electron.*, Oct. 2014, pp. 601–604.
[20] J. R. Fernandez and P. Cortes, "A survey of elevator group control systems for vertical transportation," *IEEE Control Syst. Mag.*, vol. 35, no. 4, pp. 38–55, Aug. 2015.
[21] A. Elias and L. Marcus, "Impact of machine learning on elevator control strategies: A comparison of time efficiency for machine learning elevator control strategies and static elevator control strategies in an office building," Degree Project in Computer Science, DD143X, KTH Royal Inst. Technol., Stockholm, Sweden, 2015.
[22] D.-H. Cheng, Low-cost and nonintrusive elevator operations detection service, M.S. thesis, Dept. Comput. Sci., National Chiao Tung Univ., 2017.
[23] *Passenger Lift Planning Guide*, OTIS Elevator Company, pp. 15–16, USA.
[24] [Online]. Available:https://multi.thyssenkrupp-elevator.com/en/
[25] [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[26] Y.-B. Lin *et al.*, "EasyConnect: A management system for IoT devices and its applications for interactive design and art," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 551–561, Dec. 2015.
[27] T. H. Wu, C. H. Chang, Y. W. Lin, L. D. Van, and Y.-B. Lin, "Intelligent plant care hydroponic box using IoTtalk," in *Proc. IEEE Int. Conf. Internet Things (iThings 2016)*, Chengdu, China, Dec. 2016, pp. 398–401.
[28] Y.-B. Lin, Y. W. Lin, C. M. Huang, C. Y. Chih, and P. Lin, "IoTtalk: A management platform for reconfigurable sensor devices," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1552–1562, Oct. 2017.
[29] Y. W. Lin, Y.-B. Lin, M. T. Yang, and J. H. Lin, "ArduTalk: An Arduino network application development platform based on IoTtalk," *IEEE Syst. J.*, vol. 13, no. 1, pp. 468–476, Mar. 2019.
[30] Representational state transfer (REST) 2016 [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer
[31] [Online]. Available:https://labs.mediatek.com/en/platform/overview
[32] [Online]. Available:https://openwrt.org/
[33] [Online]. Available:http://www.atmel.com/devices/atmega32u4.aspx
[34] [Online]. Available:https://www.python.org/
[35] [Online]. Available:http://scikit-learn.org/stable/modules/svm.html
[36] Open Connectivity Foundation. [Online]. Available: https://openconnectivity.org. Accessed: May 2018.
[37] oneM2M. Standards for M2M and the Internet of Things. [Online]. Available: http://www.onem2m.org/
[38] Y.-T. Chen, E. W. Sun, and Y.-B. Lin, "Coherent quality management for big data systems: A dynamic approach for stochastic time consistency," *Annals Oper. Res.*, vol. 277, no. 1, pp. 3–32, Jun. 2019.
[39] W. L. Chen *et al.*, "AgriTalk: IoT for precision soil farming of turmeric cultivation", *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5209–5223, Jun. 2019.
[40] [Online]. Available: http://www.yungtay.com.tw/
[41] T. Sultana and K. A. Wahid, "Choice of application layer protocols for next generation video surveillance using Internet of video Things," *IEEE Access*, vol. 7, pp. 41607–41624, 2019.
[42] A. A. Al-Roubaiey, T. R. Sheltami, A. S. H. Mahmoud, and K. Salah, "Reliable middleware for wireless sensor-actuator networks," *IEEE Access*, vol. 7, pp. 14099–14111, 2019.
[43] R. Milner, *The Space and Motion of Communicating Agents*. Cambridge, U.K.: Cambridge Univ. Press, 2009.

Lan-Da Van (S'98–M'02–SM16) received the Ph.D. degree from National Taiwan University (NTU), Taipei, Taiwan, in 2001.

In 2006, he joined the Faculty of the Department of Computer Science, National Chiao Tung University (NCTU), Hsinchu, Taiwan, R.O.C., where he is currently an Associate Professor. From 2015, he served the Deputy Director of NCTU M2M/IoT R&D Center. His research interests are in digital signal processing and learning computation algorithms, architectures, chips, systems and applications.

Dr. Van was the recipient of the Best Poster Award in the iNEER Conference for Engineering Education and Research (iCEER) in 2005. In 2014, he received the Best Paper Award in the IEEE International Conference on Internet of Things (iThings2014). He served as the Chairman of the IEEE NTU Student Branch in 2000. In 2001, he was the recipient of the IEEE Award for Outstanding Leadership and Service to the IEEE NTU Student Branch. He has served as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTERS (2014–2018), and has been serving as an Associate Editor for IEEE ACCESS (2018–present).

Yi-Bing Lin (M'96–SM'96–F'03) received the Ph.D. degree from the University of Washington, Washington, DC, USA, in 1990.

From 1990 to 1995 he was a Research Scientist with Bellcore. In 2010, he became a Lifetime Chair Professor of National Chiao Tung University (NCTU) Hsinchu, Taiwan, R.O.C., and in 2011, the Vice President of NCTU. During 2014–2016, he was the Deputy Minister, Ministry of Science and Technology, Taiwan. He has coauthored several books, *Wireless and Mobile Network Architecture* (Wiley, 2001), *Wireless and Mobile All-IP Networks* (Wiley, 2005), and *Charging for Mobile All-IP Telecommunications* (Wiley, 2008).

Dr. Lin is a Fellow of the AAAS, ACM, and IET.

Tsung-Han Wu received the B.S. and M.S. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2012 and 2014. He is currently working toward the Ph.D. degree with National Chiao Tung University, Hsinchu, Taiwan R.O.C.

His current research interests include Internet of Things, machine learning, and elevator scheduling.

Yu-Chi Lin received the B.S. and M.S. degrees in computer science from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2016 and 2019, respectively.

His current research interests include neural network and reinforcement learning.