ArduTalk: An Arduino Network Application Development Platform based on IoTtalk

Yun-Wei Lin, Yi-Bing Lin, *Fellow*, *IEEE*, and Ming-Ta Yang

***Abstract*—Several tools provide popular solutions for creating innovative Internet of Things (IoT) applications on single Arduino board. However, interactions among multiple Arduino boards need significant effort to be established. This paper proposes Arduino-IoTtalk (ArduTalk), a graphical user interface (GUI)-based platform aims to develop IoT network applications for interaction among multiple Arduino boards. ArduTalk utilizes IoTtalk, an IoT device management platform for quickly establishing connections and meaningful interactions between IoT devices without concerning the lower layer communication protocols. In this paper, the IoTtalk GUI has been significantly enhanced for ArduTalk. By integrating Arduino with the enhanced IoTtalk, ArduTalk allows a user to arbitrarily link and re-link sensors to actuators without or with little programming effort, and quickly generate Arduino applications for different purposes. We conduct measurements to investigate the time complexity for data delivery among multiple Arduino boards, and propose a damping mechanism to address the fairness issue caused by the discrepancies between local and remote delays among the Arduino boards.**

***Index Terms*—Arduino; Internet of Things; machine to machine communication; IoTtalk.**

I. Introduction

Internet of Things (IoT) interconnects sensors, actuators and heterogeneous computing devices within the existing Internet infrastructure and the development environments [1][2][3]. How to quickly create IoT applications is an important issue. To address this issue, several tools have been used to create innovative IoT applications on single Arduino board [4]. Details of these tools will be given in Section V. However, interactions among multiple Arduino boards need significant effort to be established. This paper proposes Arduino-IoTtalk (ArduTalk), a network application platform that allows a user to arbitrarily link and re-link sensors to actuators to create Arduino applications for different purposes. In the device domain, Arduino Yun is a popular solution for IoT device deployment. As a microcontroller board based on ATmega32u4 and Atheros AR9331, Arduino Yun provides convenient communication capabilities between the sensors/actuators and the Internet. ATmega32u4 (Fig. 1 (a)) controls electronic components and logic circuits through its input/output pins. Specifically, ATmega32u4 provides input pins to read analog signals from connected sensors (Fig. 1 (b)) and output pins to write digital signals to connected actuators (Fig. 1 (c)). Each pin can assign an independent task without interfering with other pins. Atheros AR9331 (Fig. 1 (d)) offers Wi-Fi capability (Fig. 1 (e)) to communicate with the ArduTalk server (to be elaborated next). The embedded Linux OS is running on Atheros AR9331, which allows the user to develop applications with various programming languages such as C, Java, and Python. Atheros AR9331 and ATmega32u4 communicate with each other through the Bridge software module (Fig. 1 (f)).

On the network side, ArduTalk utilizes IoTtalk [5][6], an application-layer IoT device management platform. 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, oneM2M, and so on). IoTtalk manages IoT devices based on a concept called "device feature". A device feature (DF) is a specific input or output "capability" of an IoT device. For example, a wearable ring with the temperature sensor has the input device feature (IDF) called "Temperature". A pair of wearable glasses with the optical head-mounted display has the output device feature (ODF) called "Display". An IoT device is connected to the IoTtalk server in the network (i.e., Internet) by using wired or wireless communications, and one or more network applications are automatically created/re-used at the IoTtalk server for the IoT device. 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. With this view, the IoT devices interact with each other through their features.

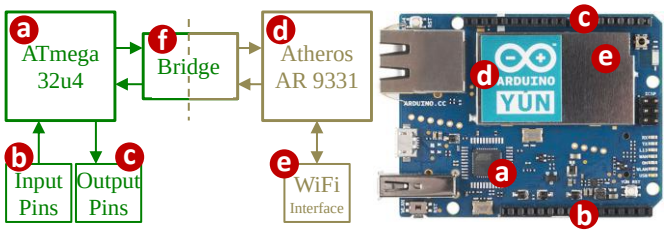

Fig. 1. The Arduino Yun architecture

We have enhanced the IoTtalk's device feature concept to perfectly match Arduino's I/O structure. By mapping the pins on an Arduino board to DFs in IoTtalk, the user can quickly develop an Arduino application. Specifically, every analog input pin that reads the sensed data from a connected sensor is

Yun-Wei Lin, Yi-Bing Lin and Ming-Ta Yang are with the Department of Computer Science, National Chiao Tung University, Taiwan (e-mails: jyneda@gmail.com, liny@cs.nctu.edu.tw, mingta@itri.org.tw). This work was supported in part by Ministry of Science and Technology (MOST) 104-2221-E-009-133-MY2-, 105-2221-E-009-119-, 105-2218-E-009-029-, 105-2622-8-009-010-, Academia Sinica AS-105-TP-A07.

mapped to an IDF. Similarly, every digital output pin that writes a command to a connected actuator is mapped to an ODF.

By integrating Arduino and IoTtalk, ArduTalk is a graphical user interface (GUI)-based development tool that allows a user to conveniently build Arduino programs using drag-and-drop operations with blocks (icons) and lines. In ArduTalk, a "physical" Arduino board is mapped to a "software" device model called "Arduino" in the server. The server provides a friendly GUI for the user to graphically create network applications, and to quickly link and re-link IDFs to ODFs.

By integrating Arduino and IoTtalk, ArduTalk is a graphical user interface (GUI)-based development tool that allows a user to conveniently build Arduino programs using drag-and-drop operations with blocks (icons) and lines. In ArduTalk, a "physical" Arduino board is mapped to a "software" device model called "Arduino" in the server. The server provides a friendly GUI for the user to graphically create network applications, and to quickly link and re-link IDFs to ODFs.

In ArduTalk, the enhanced IoTtalk server can be installed in a local environment or in the cloud as a virtual machine (VM). Fig. 2 illustrates the ArduTalk functional blocks, which consists of the ArduTalk server (Fig. 2 (a)) and the Arduino Yun (Fig. 2 (b)). The IoTtalk engine (Fig. 2 (c)) provides HTTP based RESTful application programming interfaces (APIs) for the Device Application (DA; see Fig. 2 (d)) to deliver/retrieve the IDF/ODF information. The ArduTalk GUI in Fig. 2 (e) provides a friendly web-based user interface to quickly establish connections and meaningful interactions among the IoT devices. Through the GUI, a user instructs the IoTtalk engine to execute desired tasks to create or set up device features, functions, and connection configurations. For more details of the IoTtalk server, the reader is referred to [5][6]. ArduTalk is particularly useful for developing applications to connect multiple Arduino Yun boards, which is seldom found in the existing Arduino solutions.

This paper is organized as follows. Section II proposes the ArduTalk platform. Section III describes how to develop Arduino applications using ArduTalk. Section IV investigates the communication delays of ArduTalk. Section V reviews the related works and the conclusions are given in Section VI.

II. The ArduTalk Device Application

The Device Application (DA) shown in Fig. 2 (d) is responsible for connecting IoT devices to the ArduTalk server, which is installed in an Arduino board (e.g., Arduino Yun [4]). The DA consists of two software components. The *Device Application to the Network* (DAN; see Fig. 2 (f)) is in charge of the interaction with the ArduTalk server for registration and data exchange through Wi-Fi. The DAN communicates with ArduTalk using the functions provided by an API. The *Device Application to IoT device* (DAI; see Fig. 2 (h)) is in charge of the interaction with the *IoT Device Application* (IDA; see Fig. 2 (g)) through the Bridge (Fig. 2 (i)).

The IDA is an Arduino program executed at ATmega32u4 to fetch values from the input pins (i.e., from the plugged sensors) or send commands to output pins (i.e., to the connected actuators). Through the Bridge and the DA, sensor values can be transmitted from ATmega32u4 to the ArduTalk server and the commands can be issued from the ArduTalk server to give instructions to ATmega32u4.

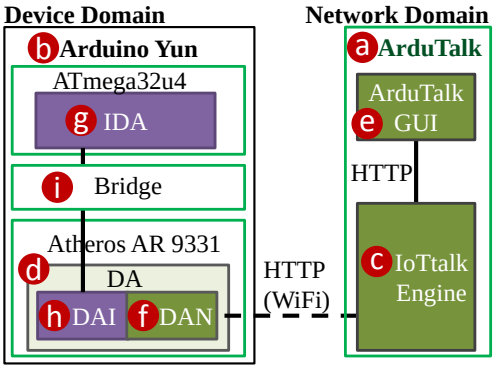

Fig. 2. The ArduTalk functional block diagram

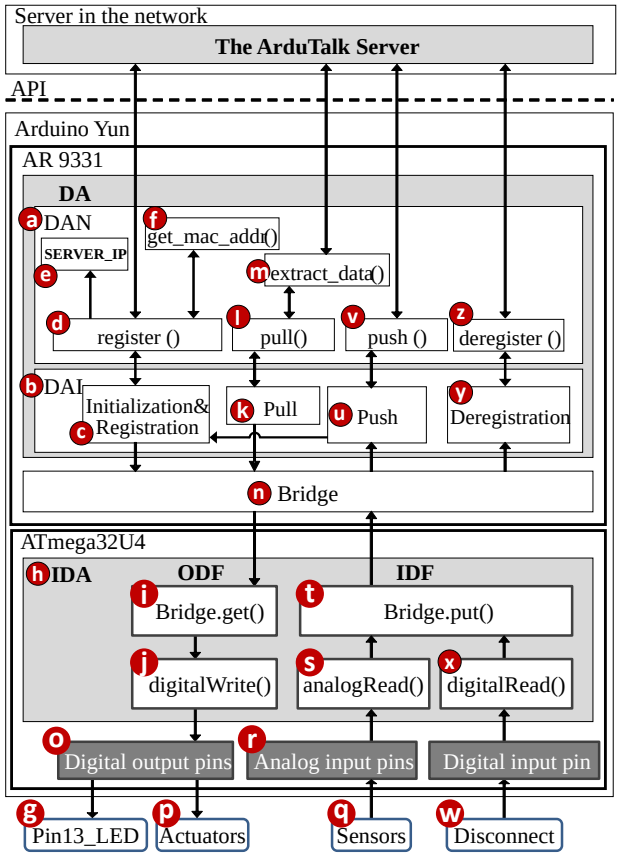

Fig. 3. The functional block diagram of the Arduino DA and IDA

Through the DAN (Fig. 3 (a)) and the DAI (Fig. 3 (b)), the DA implements four processes including Initialization & Registration, Pull, Push, and Deregistration. The details are elaborated as follows. Before an IoT device connects to ArduTalk, it must register to the ArduTalk server. In the Initialization & Registration process (Fig. 3 (c)), the function register() (Fig. 3 (d)) reads the static variable SERVER_IP (Fig. 3 (e)) to obtain the IP address of the ArduTalk server. SERVER_IP is pre-defined when the DA is installed in the Arduino Yun board. Then the DAN uses get_mac_addr() (Fig. 3 (f)) to fetch the Wi-Fi mac address that serves as the unique ID for this IoT device.

After the DA has successfully registered the Arduino Yun board to ArduTalk, the built-in LED on the Arduino Yun (controlled by digital input pin D13; see Fig. 3 (g)) is turned on to announce the successful registration. Specifically, the DA sends a notification to the IDA (Fig. 3 (h)) through the Bridge. The notification is obtained by Bridge.get() (Fig. 3 (i)), and then digitalWrite() (Fig. 3 (j)) turns on the built-in red LED to inform the user that this Arduino Yun board can start to transmit/receive the IDF/ODF data.

The Pull process (Fig. 3 (k)) periodically invokes pull() (Fig. 3 (l)) in the DAN to retrieve the data from the ArduTalk server. The function extract_data() (Fig. 3 (m)) is a JSON parser to extract the data from a JSON packet received from the ArduTalk server. Then the Pull process delivers the data to the IDA through the Bridge (Fig. 3 (n)). Specifically, Bridge.get() (Fig. 3 (i)) pulls the data from the DAI, and then digitalWrite() (Fig. 3 (j)) sends the received data to the output pin (Fig. 3 (o)) to control the actuator (Fig. 3 (p)).

The IDA is also in charge of transmitting IDF data from the sensors (Fig. 3 (q)) to the Bridge through analog input pins (Fig. 3 (r)). This task is achieved by executing analogRead() (Fig. 3 (s)) and Bridge.put() (Fig. 3 (t)).

The Push (Fig. 3 (u)) process decodes the received data from the Bridge (Fig. 3 (n)). Then the data are sent to ArduTalk by push() in the DAN (Fig. 3 (v)). Besides, the Push process can detect whether a sensor is connected to an analog input pin. If so, the push process will send the used analog input pin number to the Initialization process (Fig. 3 (c)) to update the list of IDFs. This list is used to create the IDF icons in the GUI.

There is no hardware mechanism to disconnect Arduino Yun from the ArduTalk server (except for unplugging the power line). We install a "disconnect" button (Fig. 3 (w)) that connects to digital input pin D12 for this purpose. When the button is pressed, pin D12 is triggered to instruct digitalRead() (Fig. 3 (x)) to invoke the Deregistration process (Fig. 3 (y)) to deregister the IoT device from the ArduTalk server. Specifically, Function deregister() in the DAN (Fig. 3 (z)) is called to disconnect the device from ArduTalk.

For Arduino boards without Atheros AR9331, ArduTalk still works by plugging a Wi-Fi shield into the Arduino board. Both the DA and the IDA are burned in, for example, the ATmega32u4 to exchange data with the ArduTalk server.

III. Application Development with ArduTalk

This section describes how to develop network applications by using the ArduTalk GUI. Fig. 4 illustrates the ArduTalk GUI, which consists of the menu bar, the Graphical Layout Window, and the Management Window. The menu bar (Fig. 4 (a)) has seven items. The "Project" item is a pull-down menu (Fig. 4 (b)) for the user to select a specific project. The "Model" item (Fig. 4 (c)) is another pull-down menu for choosing device models (e.g., the Arduino-based devices) to be manipulated in the Graphical Layout Window. Details of other menu bar items are out of the scope of this paper and can be found in [5][6].

The Graphical Layout Window (Fig. 4 (d)) illustrates IoT devices and their connections by using icons and line segments. The Management Window (Fig. 4 (e)) allows the user to configure the device features, the connections and the functions corresponding to the IoT devices in the Graphical Layout Window.

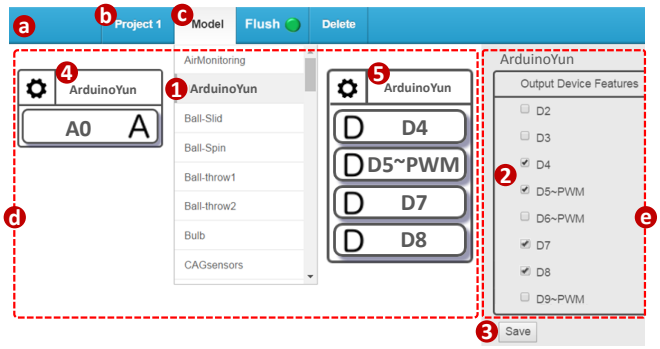

Fig. 4. Creating Arduino Yun icons in the ArduTalk GUI

Through the ArduTalk GUI, the user builds an Arduino program with network functions as follows. She/he first selects the "ArduinoYun" device model from the "Model" pulldown menu (Fig. 4 (1)). Then the Management Window shows the ODFs of the ArduinoYun device model, which follows the physical Arduino board's output pin layout. The IDFs representing the analog input pins are labelled from A0 to A5, and the IDF icons are automatically generated by the GUI without the involvement of the user. Assume that only pin A0 of the Arduino1 is connected to a sensor. As we mentioned, the used input pins are detected by the Push process in the DAI, and the corresponding IDFs are sent to the GUI to create the IDF icons. Therefore, the A0 icon is automatically shown in Fig. 4 (4). The ODFs representing the digital output pins are labelled as D2−D11 (Fig. 4 (2)), where D5 and D6 pins support the Pulse Width Modulation (PWM) signals to control, e.g., servomechanisms. D0, D1, D11, D12, and D13 are not included in ArduTalk's Arduino Yun model. We note that D0 and D1 are used by the Bridge, D13 is used by the Arduino LED, and D12 is used as an input pin for the "disconnect" button. The user can select the required ODFs and save the selected configuration through the Save button (Fig. 4 (3)). Then the icons for the ArduinoYun device model are automatically created in the Graphical Layout Window. The icons for IDFs are placed at the left-hand side of the Graphical Layout Window (Fig. 4 (4)), and the icons for ODFs are placed at the right-hand side of the Graphical Layout Window (Fig. 4 (5)).

When the user clicks the input device icon (Fig. 4 (4)) in the Graphical Layout window, the icon automatically binds to a real Arduino Yun called "Arduino1" (Fig. 5 (1)). After this binding, all signals of Arduino1's input pin A0 are automatically sent to the IDF. Similarly, when the user clicks the output device icon (Fig. 4 (5)) to bind to the same Arduino Yun (Fig. 5 (2)), the ODFs automatically send the data to Arduino1's output pins D4, D5, D7 and D8.

From the above description, the ArduTalk GUI automatically generates the IDF icons. For the ODFs, the user needs to select the ODFs because the output pins of the Arduino board do not send any signals to be read by the DAI, and

therefore the DAI cannot detect which output pins are used. To resolve this issue, we physically connect the unused output pins to D11 through a resistance of 100 ohm, then the DAI can detect through D11 that some IDFs have been incorrectly connected to some unused output pins through the ArduTalk GUI, and may falsely send data to these output pins. If so, the DAI will inform the GUI to show a warning message to the user so that the user can check the connections again to correct the mistakes. This resource discovery feature significantly reduces the possibility of false pin icon creation and connection. To our knowledge, no other IoT platforms can detect the used resources dynamically and reflect in their GUIs in real time like ArduTalk.

Now we show how multiple Arduino Yuns are created and connected through ArduTalk. Following the process of the device icon creation described in Fig. 4, we create another input Arduino Yun device icon (Fig. 5 (3)) and bind it to a real Arduino Yun called "Arduino2". We use Arduino1 and Arduino2 to develop a security light application (Project 1). In this application, when someone passes through the front door of this secured house, a spotlight is turned on to detect the movement and the curtain of the window next to the front door moves down (if it is up) to protect the privacy of the house.

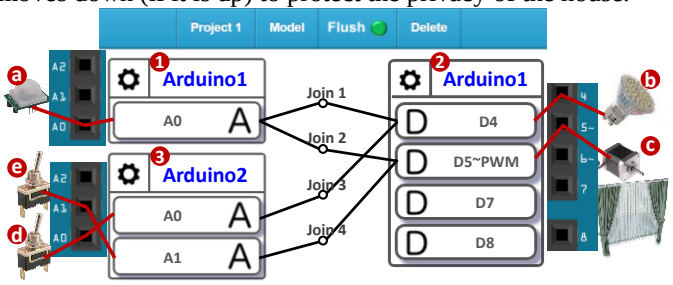

Fig. 5. Project 1: security light application

This application uses an outdoor PIR (Passive Infrared) motion sensor (Fig. 5 (a)) connected to input pin A0 of Arduino1. The outdoor spotlight (Fig. 5 (b)) connects to output pin D4 of Arduino1, and the motor of the curtain (Fig. 5 (c)) that receives PWM signals is connected to output pin D5. Two switches (Fig. 5 (d) and (e)) are connected to input pins A0 and A1 of Arduino2, respectively. Then we drag a line to connect A0 and D4 of Arduino1 (Join 1), and another line to connect A0 and D5 (Join 2). We also drag a line to connect A0 of Arduino2 and D4 of Arduino1 (Join 3), and another line to connect A1 of Arduino2 and D5 of Arduino1 (Join 4). Joins 3 and 4 allow the house owner to remotely control the spotlight and the curtain.

The security light application in Fig. 5 can be easily modified to create a door opening application. In this application, when a visitor arrives at the front door, the spotlight is turned on, and a camera facing the visitor is automatically turned on. Then the house owner can remotely open the door and turn off both the spotlight and the camera. We first save Project 1 to Project 2 (Fig. 6), and switch the ODF of Join 2 from D5 to D7 of Arduino1, where D7 is connected to a camera (Fig. 6 (c)). Join 3 is modified to connect A0 of Arduino2 to both D4 and D7 of Arduino1. Join 4 in Fig. 5 is removed, and the new Join 4 is added in Project 2 to connect A0 of Arduino2 to D8 of Arduino1, where D8 is connected to the door lock (Fig. 6 (e)).

When the home owner turns on the switch (Fig. 6 (d)), the binary value 1 is sent to D8 through Join 4 to open the door. However, the value 1 passing through Join 3 must be flipped to 0 to turn off the spotlight and the camera. A function is implemented to flip the binary values. By clicking the Join 3 circle, a window (Fig. 7) is popped up for the user to write the function to flip the binary values. In Fig. 7, the input argument *args of the Python function run() (Line 1) is the binary value of the switch (where 0 is "off" and 1 is "on") connected to A0 of Arduino2. Line 2 performs the XOR operation on the input value and 1, and the result is sent back to the circle of Join 3.

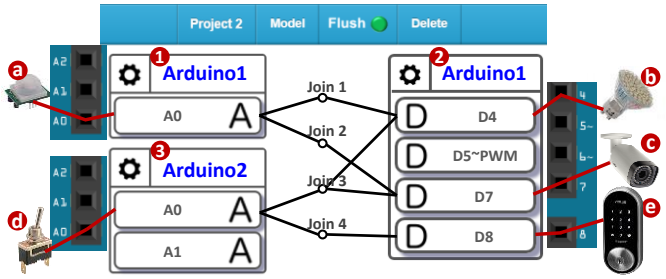

Fig. 6. Project 2: door opening application

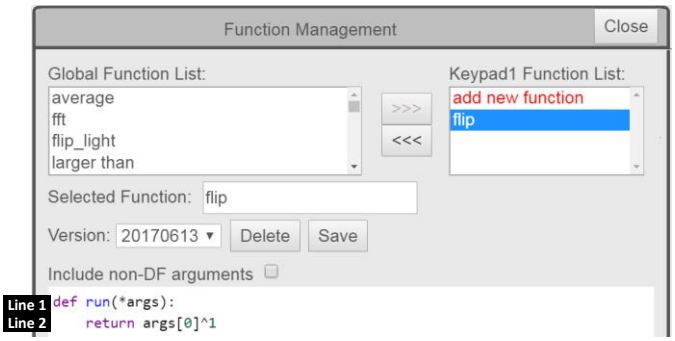

Fig. 7. The flip function

Then Join 3 sends the result to D4 and D7 of Arduino1.

Through the drag-and-drop GUI operations with no or little programming effort, the user can quickly modify one Arduino network application to generate another application.

IV. Performance Study

We conduct experiments to measure the delays of data delivered from one Arduino board to another as illustrated in Fig. 8. In terms of the locations for the ArduTalk server, there are two scenarios, local (L) and remote (R). For scenario i (= L or R), we measure the communication delay t_i from ArduinoI$_i$ to ArduinoO. The delay t_i includes the upstream delay $d_{i,U}$, downstream delay $d_{i,D}$, and the execution time of the ArduTalk server.

Scenario 1 (i=L): The ArduTalk server is installed in a local Wi-Fi AP. The CPU utilization in the local server is about 40% for each core under a dual-core processor (Intel i3-4005U) and the memory utilization is 689MB of 1.99 GB. Both the Arduino boards (Fig. 8 (1) and (2)) and the server ArduTalk$_L$ (Fig. 8 (3)) are placed in the same location. Each of $d_{L,U}$ and $d_{L,D}$ is a one-hop Wi-Fi wireless transmission delay.

Scenario 2 (i=R): The server ArduTalk$_R$ (Fig. 8 (5)) is installed in a VM in a commercial cloud at Chunghwa Telecom (CHT),

and its location is remote from ArduinoI$_R$ and ArduinoO. The CPU utilization in the CHT Cloud is about 50% for each core under a quad-core processor VM and the memory utilization is 795 MB of 1.95 GB. Each of $d_{R,U}$ and $d_{R,D}$ includes the delays for a one-hop Wi-Fi wireless transmission and the public Internet connection to the CHT cloud.

In our experiments, each of ArduinoI$_i$, ArduTalk$_i$, and ArduinoO is installed a timer module to synchronize with the Network Time Protocol (NTP). For each scenario, we conduct 1200 measurements in the CHT commercial network; that is, the remote data delivery path experiences the CHT commercial background traffic. Define the local deay t_L as the one-way data delivery delay in the local path (1)->(3)->(2) in Fig. 8, and the remote delay t_R as one-way data delivery delay in the remote path (4)->(5)->(2). These delays include the processing times at the ArduTalk servers (the execution of the application logic). Fig. 9 shows the histograms of t_i for i=L and R. The figure shows that the expected values for t_i are E[t_L]=31.68ms and E[t_R]=59.14ms. The variances are V[t_L]=0.0022E[t_L]^2 and V[t_R]=0.0026E[t_R]^2. These results indicate that the time complexity of ArduTalk at a VM in the cloud is larger than that of a local ArduTalk server. These results indicate that the time complexity for accessing the remote ArduTalk server is larger than that of a local one. In both scenarios, on the average the actuator can respond within 60 ms when it is triggered by a sensor through either a local or a remote ArduTalk server.

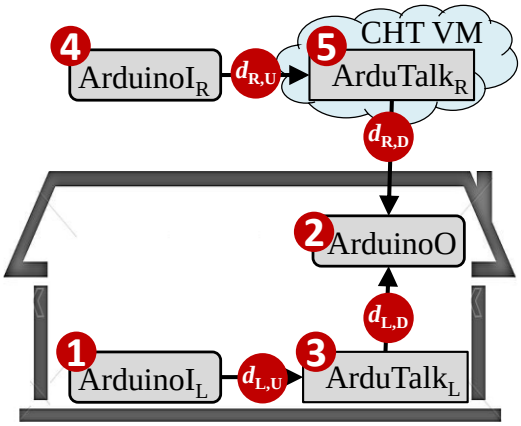

Fig. 8. Two scenarios for the ArduTalk server placement

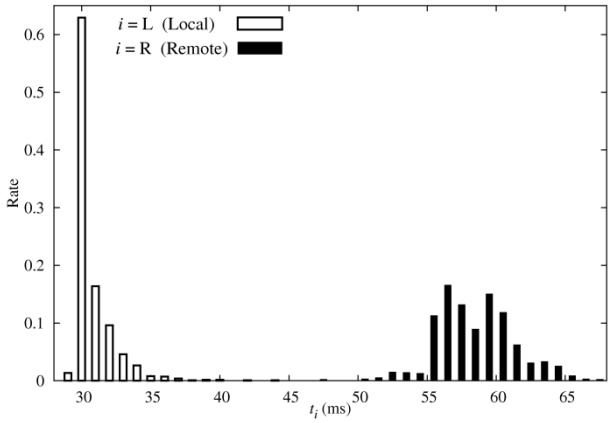

Fig. 9. The histograms of t_i for i=L and R

Fig. 9 shows that data transmission from the remote and the local paths incur different delays. Therefore, it is possible that even if ArduinoI$_R$ sends data to ArduinoO earlier than ArduinoI$_L$, ArduinoO is controlled by ArduinoI$_L$ because ArduinoI$_L$'s data arrive at ArduinoO earlier. This phenomenon causes the "fairness" issue, which is critical for some Arduino applications such as remote camera control. The access right of the remote camera control application is similar to that of distributed push-to-talk over cellular [12], where all participants can control the camera remotely depending on when they make the requests. Due to the discrepancies of the request data delivery delays, some participants may never be granted the control right. Although E[t_R] observed in Fig. 9 is small, as long as $t_R > t_L$, the fairness issue exists. For example, we have deployed an ArduTalk remote demo room (see Fig. 10) that experiences the fairness issue. Several manufacturers have donated more than 100 sensors and actuators to the demo room [13] and many of them are connected to ArduTalk. These IoT devices are installed in the cell-shape display racks of the demo room, and the data are collected every day for machine-learning analysis. A user from the USA can remotely execute an ArduTalk application connected to the sensors/actuators in the displaying cells of the demo room. A camera can be remotely controlled to point to the cell of the involved IoT devices. Once the user is granted the control of the camera, he/she also gains the control of other IoT devices in the room. Clearly, if a local user (ArduinoI$_L$) and a remote user (ArduinoI$_R$) request to gain the camera control (ArduinoO) at the same time through the paths like Join 2 and Join 3 in Fig. 6, the local user has better opportunity to gain the control.

To provide fair access to ArduinoO, we may add a damping delay at the local path. This delay depends on the $t_R - t_L$ distribution that can be derived as follows.

Fig. 10. ArduTalk demo room

From the t_i measurements, we can approximate t_i as a Gamma density function f_G with the shape parameter α_i and scale parameter $1/\beta_i$ (this approximation is validated by the Kolmogorov-Smirnov test)

$$f_G(t_i, \alpha_i, \beta_i) = \frac{\beta_i{}^{\alpha_i} t_i{}^{\alpha_i - 1} e^{-\beta_i t_i}}{\Gamma(\alpha_i)} \quad (1)$$

where for i = L, R, the mean E[t_i] and the variance V[t_i] are

$$\mathrm{E}[t_i] = \frac{\alpha_i}{\beta_i} \quad \text{and} \quad \mathrm{V}[t_i] = \frac{\alpha_i}{\beta_i{}^2} \quad (2)$$

When the ArduTalk server is installed in an environment other than the CHT cloud, the one-way delay measurements can also be approximated by the Gamma distribution or a mixture of Gamma distributions [14]. Let $t_d = t_R - t_L$, then

$$f_d(t_d | t_R > t_L) \Pr[t_R > t_L] = \int_{t_L=0}^{\infty} f_G(t_d + t_L, \alpha_R, \beta_R) f_G(t_L, \alpha_L, \beta_L) dt_L \quad (3)$$

Substitute (1) into (3) to yield

$$f_d(t_d|t_R > t_L)\Pr[t_R > t_L] = \int_{t_L=0}^{\infty}\left[\frac{{\beta_R}^{\alpha_R}(t_d+t_L)^{\alpha_R-1}e^{-\beta_R(t_d+t_L)}}{\Gamma(\alpha_R)}\right]\left[\frac{{\beta_L}^{\alpha_L}{t_L}^{\alpha_L-1}e^{-\beta_L t_L}}{\Gamma(\alpha_L)}\right]dt_L$$

For $\alpha_R = 1$,

$$f_d(t_d|t_R > t_L)\Pr[t_R > t_L] = \left[\frac{\beta_R{\beta_L}^{\alpha_L}}{(\beta_R+\beta_L)^{\alpha_L}}\right]e^{-\beta_R t_d}$$

For $\alpha_R \neq 1$,

$$f_d(t_d|t_R > t_L)\Pr[t_R > t_L] = \left[\frac{{\beta_R}^{\alpha_R}{\beta_L}^{\alpha_L}e^{-\beta_R t_d}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right] \times \int_{t_L=0}^{\infty}\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n}{t_L}^{\alpha_L+n-1}{t_d}^{\alpha_R-n-1}e^{-(\beta_R+\beta_L)t_L}dt_L$$

$$= \left[\frac{{\beta_L}^{\alpha_L}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right]\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n} \times \left[\frac{\Gamma(\alpha_R-n)\Gamma(\alpha_L+n)}{{\beta_R}^{-n}(\beta_R+\beta_L)^{\alpha_L+n}}\right]f_G(t_d, \alpha_R-n, \beta_R) \quad (4)$$

From (3), we have

$$\Pr[t_R > t_L] = \int_{t_d=0}^{\infty} f_d(t_d|t_R > t_L)\Pr[t_R > t_L]\,dt_d \quad (5)$$

Substitute (4) into (5) to yield

$$\Pr[t_R > t_L] = \left[\frac{{\beta_L}^{\alpha_L}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right]\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n}\left[\frac{\Gamma(\alpha_R-n)\Gamma(\alpha_L+n)}{{\beta_R}^{-n}(\beta_R+\beta_L)^{\alpha_L+n}}\right] \quad (6)$$

From (2), we have $\alpha_L \approx 455$, $\beta_L \approx 14.36$, $\alpha_R \approx 385$, $\beta_R \approx 6.51$, and

$$\Pr[t_R > t_L] \approx 0.999$$

which implies

$$f_d(t_d) \approx f_d(t_d|t_R > t_L)\Pr[t_R > t_L] \quad (7)$$

From (1), the Laplace transform ${f_d}^*(s)$ of $f_d(t_d)$ is expressed as

$${f_d}^*(s) = \int_{s=0}^{\infty} f_d(t_d)\, e^{-st_d}dt_d \quad (8)$$

The Laplace transform ${f_d}^*(s)$ can be approximated by using (5), (7) and (8) as

$${f_d}^*(s) = \int_{s=0}^{\infty}\left[\frac{{\beta_L}^{\alpha_L}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right]\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n}\left[\frac{\Gamma(\alpha_R-n)\Gamma(\alpha_L+n)}{{\beta_R}^{-n}(\beta_R+\beta_L)^{\alpha_L+n}}\right] \times f_G(t_d, \alpha_R-n, \beta_R)e^{-st_d}dt_d$$

$$= \left[\frac{{\beta_L}^{\alpha_L}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right]\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n}\left[\frac{\Gamma(\alpha_R-n)\Gamma(\alpha_L+n)}{{\beta_R}^{-n}(\beta_R+\beta_L)^{\alpha_L+n}}\right]\left[\frac{{\beta_R}^{\alpha_R-n}}{(s+\beta_R)^{\alpha_R-n}}\right] \quad (9)$$

The expected value $\mathrm{E}[t_d]$ of t_d can be obtained from its Laplace transform with the differential operation:

$$E[t_d] = -\frac{d{f_d}^*(s)}{ds}\Big|_{s=0} \quad (10)$$

Applying (10) on (9), the expected value $\mathrm{E}[t_d]$ of t_d can be obtained as follows:

$$\mathrm{E}[t_d] = \left[\frac{{\beta_L}^{\alpha_L}}{\Gamma(\alpha_R)\,\Gamma(\alpha_L)}\right]\sum_{n=0}^{\alpha_R-1}\binom{\alpha_R-1}{n}\left[\frac{\Gamma(\alpha_R-n)\Gamma(\alpha_L+n)}{{\beta_R}^{-n}(\beta_R+\beta_L)^{\alpha_L+n}}\right]\left(\frac{\alpha_R-n}{\beta_R}\right) \quad (11)$$

Substitute $\alpha_L \approx 455$, $\beta_L \approx 14.36$, $\alpha_R \approx 385$, $\beta_R \approx 6.51$ into (11) to yield

$$\mathrm{E}[t_d] \approx 27.45999733 \quad (12)$$

We also observe that

$$\mathrm{E}[t_R] - \mathrm{E}[t_L] = 59.14 - 31.68 = 27.46 \quad (13)$$

From (12) and (13) we conclude that for the example of the ArduTalk demo room,

$$\mathrm{E}[t_d] \approx \mathrm{E}[t_R] - \mathrm{E}[t_L] \quad (14)$$

We add the damping delay 27 ms to the local path, and the measurements indicates that

$$\Pr[t_R > t_L + 27] = 0.52$$

This probability is close to 0.5. As compared with the case when the damping mechanism is not used (where the probability is 0.999), we have effectively addressed the fairness issue. For local/remote delays other than what we observed in the demo room, same approach can be used to select the damping delay. Specifically, through the NTP, we measure t_L and t_R in the leaning process to derive the damping delay $\mathrm{E}[t_d]$. Then we remove the NTP mechanism and simply add the damping delay into the local delay; that is, ArduinoO always waits for $\mathrm{E}[t_d]$ before the ArduinoI$_\mathrm{L}$'s data is processed. Alternatively, we can execute an adaptive damping mechanism with the following steps.

Step 0. Set the initial $\mathrm{E}[t_d]$ value to 0.

Step 1. ArduinoO receives the next message at τ_O. If the message is sent from ArduinoI$_\mathrm{R}$, then go to Step 2. Otherwise, go to Step 3.

Step 2. The message received by ArduinoO is ($\mathrm{data_R}$,τ_R), where τ_R is the time when ArduinoI$_\mathrm{R}$ sent the message. ArduinoO processes $\mathrm{data_R}$ and computes $t_R = \tau_\mathrm{O} - \tau_\mathrm{R}$. Go to Step 6.

Step 3. The message received by ArduinoO is ($\mathrm{data_L}$,τ_L), where τ_L is the time when ArduinoI$_\mathrm{L}$ sent the message. ArduinoO computes $t_L = \tau_\mathrm{O} - \tau_\mathrm{L}$. If ArduinoO receives the next message ($\mathrm{data_R}$,τ_R) from ArduinoI$_\mathrm{R}$ before $\tau_\mathrm{O} + \mathrm{E}[t_d]$, then go to Step 4. Otherwise, go to Step 5.

Step 4. If $\tau_\mathrm{R} < \tau_\mathrm{L}$, then ArduinoO processes $\mathrm{data_R}$ before $\mathrm{data_L}$, else it processes $\mathrm{data_L}$ before $\mathrm{data_R}$. In both cases, ArduinoO computes t_R. Go to Step 6.

Step 5. ArduinoO processes $\mathrm{data_L}$ at $\tau_\mathrm{O} + \mathrm{E}[t_d]$ and computes $t_L = \tau_\mathrm{O} - \tau_\mathrm{L}$. Go to Step 6.

Step 6. Collect t_R and/or t_L, and may update $\mathrm{E}[t_d]$ using (11). Go to Step 1.

Without loss of generality, we assume that ArduinoI$_\mathrm{L}$ does not send subsequent messages within the interval $\mathrm{E}[t_d]$. If not, Step 3 is modified with steps that are more tedious. To reduce the overhead of the damping mechanism, ArduinoO may not compute $\mathrm{E}[t_d]$ every time Step 6 is executed. On the other hand, to speed up the execution of Step 6, we may use Equation (14) to approximate $\mathrm{E}[t_d]$.

The fairness issue is also found in other IoTtalk applications and network games. It is particularly meaningful that we raise this fairness issue in ArduTalk because previous Arduino solutions are aiming for single Arduino board applications, and this issue does not exist in these solutions. When ArduTalk connects multiple Arduino boards, this issue is amplified and therefore analyzed in this section.

V. Comparison and Integration with Other Solutions

This section first compares ArduTalk with other Arduino-related tools. Then we discuss how to integrate ArduTalk with other IoT platforms. In Table 1, all tools considered are Scratch-based except for ArduTalk. All of them

target for Arduino microcontroller board (MCU) except Webduino that targets for the ESP8266 MCU [7].

Table 1. Comparison of Solutions

Item	Solution	Ardublock	mBlock	Minibloq	S4A	Webduino	ArduTalk
1	Firmware burning	Multiple times	Multiple times	Multiple times	One time	One time	One time
2	On-board Logic	Scratch (Limited)	Scratch	Scratch	×	×	×
3	Off-board logic	×	×	×	Scratch	Scratch, JavaScript	GUI, Python
4	On-board I/O connection	Scratch	Scratch	Scratch	Scratch	Scratch	GUI
5	Off-board I/O connection cost	×	Scratch (High)	Scratch (High)	Scratch (Medium)	JavaScript (Medium)	GUI (Low)
6	Wireless communication	Bluetooth	Bluetooth	Bluetooth	Bluetooth, USB	Bluetooth, Wi-Fi	Bluetooth, Wi-Fi

Depending on where the application logic is executed, these tools can be categorized into two groups. In the on-board group including Ardublock [8], mBlock [9] and Minibloq [10], the application logic is executed in the microcontroller board (MCU). In the off-board group including Scratch for Arduino (S4A) [11], Webduino and ArduTalk, the application logic is executed outside the MCU (see items 2 and 3, Table 1).

In Ardublock, every graphical block contains a number of codes representing an actuator or a sensor (such as a LED, a motor, or a temperature sensor). The user develops an Arduino program like playing a jigsaw puzzle, and then uploads this program to an Arduino board. Ardublock does not allow the user to create any new block. In mBlock, a series of Arduino boards (such as Uno, Leonardo and Mega 2560) are supported. The user builds the Arduino program with the blocks like Ardublock. User-defined blocks can be created with standard Arduino programs. Minibloq is specifically tailored for robotic applications of Arduino. In these tools, every time the service logic is modified (e.g., to change the threshold of triggering an action), the program (including DA and IDA) need to be re-burned into the board through the Arduino IDE (item 1, Table 1).

In the off-board group, the DA and IDA firmware is initially burned into the Arduino board and will not be re-burned. Since the application logic is executed off-board, the program is not burned in the MCU (item 1, Table 1). A tool in the off-board group can implement complex service logic with features such as large-scale data processing and the machine learning algorithms. On the other hand, for a tool in the on-board group, the features of the service logic are limited due to the MCU's restricted computing power.

The S4A service logic is written in C, which needs to be compiled before execution. The Webduino allows using a web browser to create simple logic through Scratch, and insert JavaScript code in the Scratch-created program for complex service logic. The ArduTalk service logic can be automatically generated through the GUI. For specific features not found in ArduTalk, they are implemented by Python functions that can be reused as parts of the ArduTalk tool. Both the Webduino and the ArduTalk programs are executed without compilation (items 2 and 3, Table 1).

All solutions except for ArduTalk use Scratch for on-board IO connection. ArduTalk uses the enhanced IoTtalk GUI to automatically create on-board IO connection (item 4, Table 1). For multi-board I/O connection, only ArduTalk provide GUI setup where multiple boards can easily interact with each other. Other solutions need extra programming effort (item 5, Table 1). That is, these tools are very efficient to develop applications for single Arduino board, but require extra tedious effort to build multiple Arduino board connection to interact with each other via the Internet.

All tools use built-in Bluetooth to communicate with the outside world typically through the Bluetooth gateways. Both Webduino and ArduTalk provide plug-and-play Wi-Fi communications. In S4A, the off-board service logic communicates with the on-board IDA through the USB link (item 6, Table 1).

Now we show how ArduTalk can be easily adopted (accommodated) by other IoT platforms and become a part of these platforms. There are two ways to integrate ArduTalk with other platforms. In the first alternative, ArduTalk is ported on top of the target platforms such as AllJoyn and oneM2M (Fig. 11). It works because AllJoyn and oneM2M provide APIs to allow the user to build network applications. Therefore, when we port ArduTalk on top of, for example, oneM2M, it is considered as a network application of oneM2M. The porting uses oneM2M API (API1 in Fig. 11) to implement four functions: connect, disconnect, push and pull. Then all physical oneM2M devices become IoTtalk devices and can interact with the Arduino boards through the ArduTalk GUI. In Fig. 11, the ArduTalk server is ported on both oneM2M (API1) and AllJoyn (API2). Actually, the first version of the IoTtalk Engine was successfully built on top of an early version of oneM2M called openMTC. In Fig. 11, through the ArduTalk server, the sensor of Arduino Board can control the light bulb of oneM2M, and the smart watch of oneM2M can control water spray of AllJoyn or the fan connected to the Arduino board. Therefore, an oneM2M user may implement his/her IoT device following the oneM2M protocols and still can enable the device to interact with the ArduTalk devices.

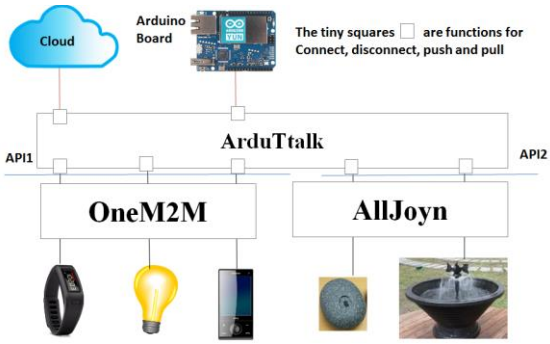

Fig. 11. Integrating ArduTalk with other platforms

In the second alternative, the ArduTalk GUI creates an icon for a platform to be integrated with ArduTalk. For example, ArduTalk has successfully connected to Chunghwa Telecom's

commercial IoT Gateway using this approach (Fig. 12). Both the ArduTalk server and the CHT IoT Gateway are installed in a Raspberry Pi 3. We have created an ArduTalk device called CHT-GW. The DAN of CHT-GW is the same as that in Fig. 2 (f). The DAI connects to the CHT IoT Gateway through a web socket. In this alternative, CHT provides a command table to CHT-GW. For example, "1" represents fan, and then under this command, the subcommand 0 represents "turn off" and so on. With this integration, the temperature sensor of the Arduino board can interact with CHT's home appliances through the command table.

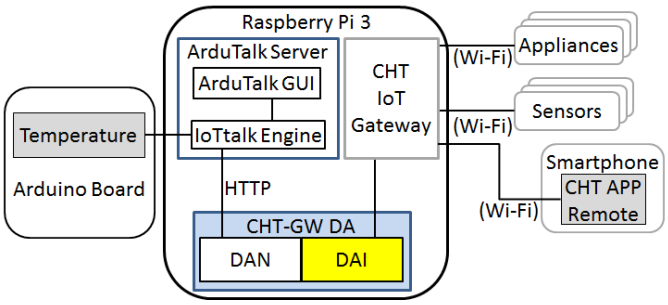

Fig. 12. Integrating ArduTalk with CHT IoT Gateway

Note that in both alternatives, ArduTalk does not modify other IoT platforms. Such integration only requires minimal porting efforts and ArduTalk does not need to know the low level details of other IoT platforms. Also note that ArduTalk can be integrated with other solutions in Table 1 to extend their ability for network application development. For example, following the second alternative, we use Webduino's JavaScript editor to write a Webduino DA for ArduTalk as what we have done for Arduino in Fig. 2.

VI. Conclusion

This paper proposed ArduTalk, a mechanism that utilizes enhanced IoTtalk to provide a GUI-based tool for Arduino network application development. We pre-install the DA and the IDA in the Arduino Yun board, which is one-time installation with the same overhead as the standard procedure to burn sketches into the Arduino board. The user can build Arduino network programs using drag-and-drop operations to draw lines between analog and digital pins in the ArduTalk GUI, which allows quick deployment of IoT network applications. Furthermore, the user can easily link and re-link sensors to actuators to create and modify Arduino applications for different purposes. We have conducted measurements to investigate the data delivery delays among multiple Arduino boards, and observed that the expected local and remote delays are within 60 ms in the ArduTalk demo room example. Through analytic analysis, we also derived the discrepancy distribution for the remote and the local delays to resolve the "fairness" issue.

As a final remark, ArduTalk also supports embedded systems such as MediaTek LinkIt Smart 7688 duo, ROHM IoT kit and ESP8266 ESP-12F with the same DA software.

Acknowledgement

The sensors and actuators in the ArduTalk demo room are provided by A. B.-F. Tseng of Connected Automation Global Inc., S. Chen of Qblinks, and J. Hwang of EverMore Technology.

References

[1] S. Verma, Y. Kawamoto, Z. Fadlullah, H. Nishiyama, N. Kato. A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues. IEEE Communications Surveys & Tutorials, 2017

[2] C.-S. Shih, C.-C. Chuang and H.-Y. Yeh. Federating Public and Private Intelligent Services for IoT Applications. In 13th International Wireless Communication and Mobile Computing Conference (IWCMC), Valencia, SPAIN, June 2017.

[3] T.-Y. Chan, Y. Ren, Y.-C. Tseng, and J.-C. Chen, "eHint: An Efficient Protocol for Uploading Small-Size IoT Data", IEEE WCNC, 2017.

[4] Arduino Yun (2017). [Online]. https://www.arduino.cc/en/Main/ArduinoBoardYun

[5] Y.-B. Lin, Y.-W. Lin, C.-C. Tai, C.-Y. Chih, T.-Y. Li, Y. -C. Wang, C.-C. Huang, and S.-C. Hsu, "EasyConnect: A Management System for IoT Devices and Its Applications for Interactive Design and Art", IEEE Internet of Things Journal, Vol. 6, No. 2, pp. 551-561, 2015.

[6] Y-B. Lin, Y.-W. Lin, C.-M. Huang, C.-Y. Chih, P. Lin. IoTtalk: A Management Platform for Reconfigurable Sensor Devices. Accepted and to appear in IEEE Internet of Things Journal, 2017.

[7] Webduino.io, https://webduino.io/en/ , 2015.

[8] ArduBlock (2017). [Online]. http://blog.ardublock.com/

[9] mBlock (2017). [Online]. http://www.mblock.cc/zh-download

[10] Minibloq (2017). [Online]. http://blog.minibloq.org/

[11] Scratch for Arduino (S4A) (2015). [Online]. http://s4a.cat/

[12] M.-H. Tsai and Y.-B. Lin. Talk Burst Control for Push-to-Talk over Cellular. IEEE Transactions on Wireless Communications, 7(7): 2612-2618, 2008.

[13] W. Huang. M-Echo Project. Alpha Networks Inc. (2017). [Online]. http://www.alphanetworks.com/

[14] Kelly, F.P. Reversibility and Stochastic Networks. John Wiley & Sons, 1979.

Yun-Wei Lin received the B.S. degree in computer and information science from Aletheia University, Taipei, Taiwan, in June 2003, and the M.S. and Ph.D. degrees in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in 2005 and 2011, respectively. He has been an Assistant Research Fellow at National Chiao Tung University since 2013. His current research interests include mobile ad hoc network, wireless sensor network, vehicular ad hoc networks, and IoT/M2M communications.

Yi-Bing Lin (M'96-SM'96-F'03) received his Bachelor's degree from National Cheng Kung University, Taiwan, in 1983, and his Ph.D. from University of Washington, USA, in 1990. From 1990 to 1995 he was a Research Scientist with Bellcore. He then joined National Chiao Tung University (NCTU) in Taiwan, where he remains. In 2010, Lin became a lifetime Chair Professor of NCTU, and in 2011, the Vice President of NCTU. During 2014 - 2016, Lin was Deputy Minister, Ministry of Science and Technology, Taiwan.

Since 2016, Lin has been appointed as Vice Chancellor, University System of Taiwan (for NCTU, NTHU, NCU, and NYM). Lin is an Adjunct Research Fellow, Institute of Information Science, Academia Sinica, Research Center for Information Technology Innovation, Academia Sinica, and a member of board of directors, Chunghwa Telecom. He serves on the editorial boards of IEEE Trans. on Vehicular Technology. He is General or Program Chair for prestigious conferences including ACM MobiCom 2002. He is Guest Editor for several journals including IEEE Transactions on Computers. Lin is the co-author of the books *Wireless and Mobile Network Architecture* (Wiley, 2001), *Wireless and Mobile All-IP Networks* (John Wiley, 2005), and *Charging for Mobile All-IP Telecommunications* (Wiley, 2008). Lin received numerous research awards including 2005 NSC Distinguished Researcher, 2006 Academic Award of Ministry of Education, 2008 Award for Outstanding contributions in Science and Technology, Executive Yuen, 2011 National Chair Award, and TWAS Prize in Engineering Sciences, 2011 (the Academy of Sciences for the Developing World). He is Chair of IEEE Taipei Section. Lin is AAAS Fellow, ACM Fellow, IEEE Fellow, and IET Fellow.

Ming-Ta Yang (M'17) received his Bachelor's degree in Electrical Engineering, minor in Information Management, from Yuan Ze University (YZU), Taiwan, in 2003, and his M.S. degree in Electrical Engineering from National Chung Cheng University (CCU), Taiwan, in 2006.

Yang is the Senior Research and Development Engineer of Industrial Technology Research Institute (ITRI), and serves as the Deputy Technical Manager. He is also working toward his Ph.D. degree at the Department of Computer Science, National Chiao Tung University (NCTU), Taiwan. His current research interests include Internet of Things (IoT), media streaming, and virtual reality.