|  |
| --- |
|  |
| **SECE: SENSE EVERYTHING CONTROL EVERYTHING** |
| **SENSOR AND ACTUATOR PROTOTYPE** |
|  |
| **Hyunwoo Nam** |
| **5/10/2012** |

|  |
| --- |
| **This technical report describes overall progresses and experiment results for implementing sensor and actuator prototypes in SECE project.** |

Table of Contents

1. Introduction 5
2. Arduino microcontroller6
   1. Arduino specification 7
   2. Arduino IDE8
   3. Connect Arduino to the Internet8
   4. Connect sensors and actuators to Arduino9
3. Communication with SECE server10
   1. Getting IP address11
   2. SECE server discovery from Arduino11
   3. Websocket12
   4. Http and JSON between Arduino and SECE server12
4. Sensor and actuator prototype 13
   1. Temperature sensor (tmp36) 13
   2. Zigbee protocol14
   3. IR transmitter and detector16
   4. X10 protocol18
   5. Event notification system19
   6. Communication and sensor layers in Arduino20
   7. Timer and reset23
   8. Find SECE service using Bonjour
5. Concluding remarks25
6. References26

List of Figures

1. The big picture of SECE5
2. Microcontroller between SEVER server and sensors6
3. Arduino7
4. Arduino IDE8
5. Arduino Ethernet shield9
6. Phidgets, Zigbee and X10 modules9
7. Wiring sensors to Arduino10
8. Connection establishments between Arduino and SECE server11
9. WebSocket12
10. Messages between Arduino and SECE server13
11. Get temperature data using TMP3614
12. Get temperature data over Zigbee protocol15
13. Arduino Xbee shield and dongle16
14. IR transmitter and detector wired on Arduino16
15. Characteristics of PNA460217
16. Implement IR detector and transmitter17
17. Control a lamp using X10 protocol18
18. X10 wired and wireless transmitters18
19. X10 PLC transmitter wired on Arduino19
20. Event notification system using Arduino19
21. FSM of a light detector event notification system20
22. Flow chart of communication and sensor layers in Arduino21
23. Arduino timer: without Timer (left) and with Timer (right)22
24. FSM of timer function to get temperature data from Arduino23
25. Find SECE services via Bonjour24
26. Obtain sensor information using HTTP and JSON25
27. Find SECE services using mDNS on Android smartphones 26

List of Tables

1. Arduino board model7
2. **INTRODUCTION**

SECE (Sense Everything, Control Everything) allows even non-technical users to create various services that combine communication, presence, social networks, calendaring, location and devices in the physical world. This report addresses how to control physical devices such as lights, TVs, and air conditioners over the Internet in our SECE project. We introduce a gate way based sensor and actuator prototype for controlling the networked devices. A gate way plays a key role in the communication between actuators and SECE server and granting interoperability among the actuators which use different communication technologies.

Figure 1 The big picture of SECE

This report is organized as follows. Section2 introduces an Arduino microcontroller as a selected gate way and the overall communication system between Arduino and SECE server is mentioned in Section3, respectively. The details of implementing different sensor and actuator prototypes and experiment results are addressed in Section 4.

1. **ARDUINO MICROCONTROLLER**

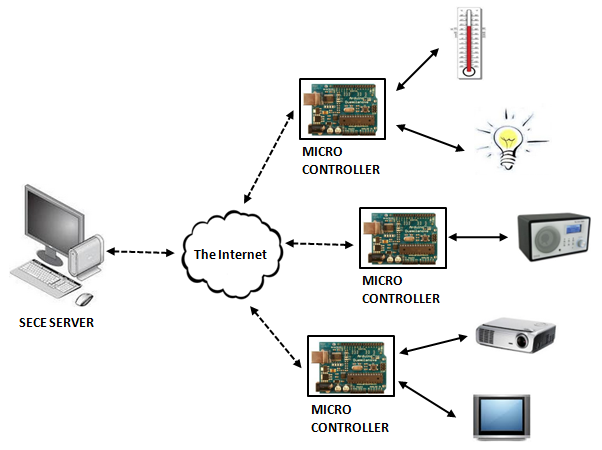
In the Internet of Things (IoT), we assume that all general devices can be connected to the Internet. However, it is hard to get access general appliances such as lights, projectors, and video players over the Internet without installing any relay device. Also, it is costly to replace the general appliances with IP capable ones. Our simple solution is to put an Ethernet available microcontroller between those general sensors and SECE server. As shown in Figure 2, microcontrollers are responsible for communicating with SECE server directly over the Internet on behalf of the non-IP capable sensors and actuators. 

Figure 2 Microcontroller between SECE server and actuators

To show the feasibility for SECE sensor and actuator prototype, an ***Arduino*** microcontroller [1]is selected among the other microcontrollers such as Beagleboard [2] and Netduino [3] for the following reasons.

* Cheap: $21 (basic version)
* Well developed programming environment (C/C++ language supported) and AVR-Studio
* Easy to start with
* Compatible with Phidgets, X10, Zigbee and other general gadgets
* Huge community
* Software development on Mac OS-X and Windows
  1. **ARDUINO SPECIFICATION**

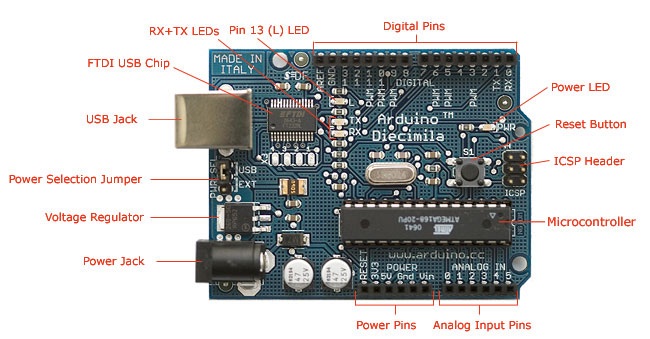


Figure 3 Arduino UNO

Arduino is an [open source](http://en.wikipedia.org/wiki/Open-source_hardware) [single board microcontroller](http://en.wikipedia.org/wiki/Single-board_microcontroller). It is able to sense the environment by receiving inputs from a variety of sensors and can affect its surroundings by controlling lights, motors, and other actuators. The microcontroller on the board is programmed using wiring-based language similar to C++ language in Arduino development environment. Arduino projects can be stand-alone or they can communicate with software running on a computer. shows the various versions of Arduino. For a gate way based sensor and actuator prototype in SECE project, Arduino UNO version is selected.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Arduino** | **Processor** | **Flash**  **(KB)** | **EEPROM (KB)** | **SRAM**  **(KB)** | **Digital pins** | **Analog pins** |
| Due | ATMEL SAM3U | 256 | 0 | 50 | 54 | 16 |
| Mega2560 | ATmega2560 | 256 | 4 | 8 | 54 | 16 |
| Mega | ATmega1280 | 128 | 4 | 8 | 54 | 16 |
| Leonardo | Atmega32u4 | 32 | 1 | 2 | 14 | 12 |
| Fio | ATmega328P | 32 | 1 | 2 | 14 | 8 |
| Uno | ATmega328P | 32 | 1 | 2 | 14 | 6 |
| Duemilanove | ATmega168/328P | 16/32 | 0.5/1 | 2-Jan | 14 | 6 |
| Diecimila | ATmega168 | 16 | 0.5 | 1 | 14 | 6 |
| LilyPad | ATmega168V or ATmega328V | 16 | 0.5 | 1 | 14 | 6 |
| Nano | ATmega168 or ATmega328 | 16/32 | 0.5/1 | 2-Jan | 14 | 8 |

Table 1 Arduino board model

* 1. **ARDUINO IDE**

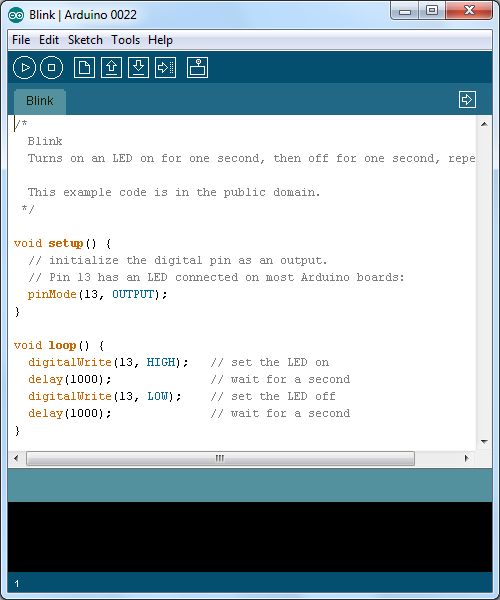


Figure 4 Arduino IDE

The Arduino IDE is a cross-platform application written in Java. This IDE is available on Windows or Mac-OS-X. Using the well-developed IDE, users do not need to create makefiles every time they compile their codes. The Arduino IDE supports a [C](http://en.wikipedia.org/wiki/C_(programming_language))/C++ library called "*Wiring*", which makes many common input/output operations much easier. With the C/C++ libraries, users only need to define two functions to make a runnable application.

* setup(): run once at the start of a program that initializes all settings
* loop(): run repeatedly until the board powers off

In our SECE project, Arduino 0022 and 0023 IDEs are used to implement SECE applications.

* 1. **CONNECT ARDUINO TO THE INTERNET**

Figure 5 Arduino Ethernet shield

One of the most attractive things for using Arduino is that we can choose stackable shields depending on project types. Since we need the Internet connection from Arduino, an Ethernet shield is selected which is shown in Figure 5. The Internet standard protocols such as TCP/IP and DHCP stacks are already built in the Arduino IDE. Using the Ethernet library in the IDE, a few applications are developed to connect to SECE server and exchange data between the actuators wired to Arduino and SECE server.

* 1. **CONNECT SENSORS AND ACTUATORS TO ARDUINO**



Figure 6 Phidgets, Zigbee and X10 modules

There are many kinds of commercial sensors and actuators. For example, Phidgets [4] provide a variety of plug in and play sensors such as temperature, potentiometer and flex sensors. X10 [5] modules become popular in implementing home automation systems by controlling home appliances over power lines. For wireless communications, Zigbee [6] protocol based on 802.15.4 is widely used. Arduino is capable of connecting with these commercial sensors. Since each sensor uses its own protocol, Arduino should be flexibly wired and programmed depending on the different communication types.

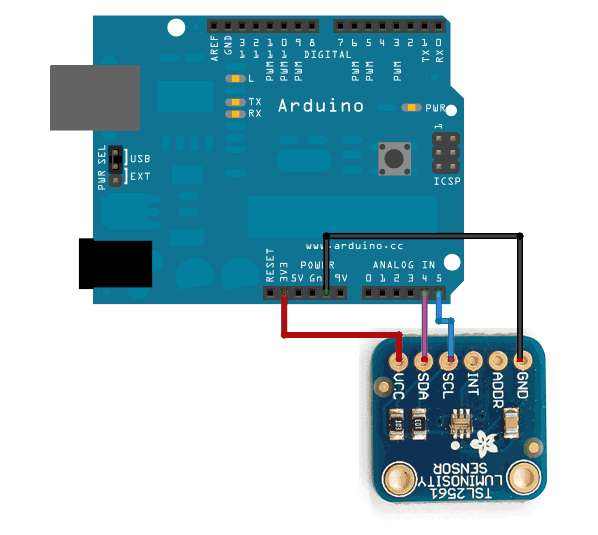


Figure 7 Wiring sensors to Arduino

To connect sensors to Arduino, first they must be physically wired to Arduino. Depending on their hardware types, Arduino is connected with them via analog or digital ports. Through those ports Arduino reads values from the sensors or sends signals to the actuators. Secondly, Arduino needs to be programmed to communicate with the sensors. For the well-known home automation systems such as Zigbee and X10, the libraries for Arduino are already available in the current Arduino IDEs. The details of each setup are introduced in section 4.

1. **COMMUNICATION WITH SECE SERVER**

Arduino is connected to SECE server over TCP/IP. Since Arduino Ethernet shield already supports a TCP/IP stack, we have focused on implementing software to connect it with SECE server. The overall flow of the connection establishment between Arduino and SECE server is shown in Figure 8.

Figure 8 Connection establishments between Arduino and SECE server

**3.1 GETTING IP ADDRESS**

When Arduino is turned on, it first gets an IP address from a local DHCP server. This function is called when Arduino initializes setups.

***void Init\_DHCP(){***

***Serial.println("Attempting to obtain a DHCP lease......");***

***EthernetDHCP.begin(mac\_address);***

***}***

Once Arduino obtains an IP address, it still needs to extend the allocated IP time period while it is running. An IP maintain function is continuously called in a loop*.* It keeps monitoring the connection status and requests extension if needed.

***void loop() {***

***EthernetDHCP.maintain();***

***}***

**3.2 SECE SERVER DISCOVERY FROM ARDUINO**

When Arduino is programmed, we encode the host name of SECE server in Arduino. Once Arduino gets an IP address, it resolves the SECE server’s IP address via DNS using the host name. Depending on further strategies, we may switch it to DNS SRV/NAPTR to connect to the remote SECE server.

**3.3 WEBSOCKET**

**Figure 9 WebSocket**

WebSocket provides bi-directional, [full-duplex](http://en.wikipedia.org/wiki/Full-duplex) communication channels over a single TCP [socket](http://en.wikipedia.org/wiki/Internet_socket). To upgrade to WebSocket, it needs to be designed both in [web browsers](http://en.wikipedia.org/wiki/Web_browser) and [web servers](http://en.wikipedia.org/wiki/Web_server), but it can be used by any client or server application. Nowadays, the other ports except the port number 80 are frequently blocked by administrators. WebSocket helps us overcome this restriction and available for multiplexing several WebSocket services over a single TCP port. WebSocket can be easily setup by handshaking between Arduino and SECE server via HTTP. It initiates from Arduino.

***GET /arduino HTTP/1.1***

***Host: ryngle.net***

***Upgrade: websocket***

***Connection: Upgrade***

***Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==***

***Sec-WebSocket-Origin: Open WebSocket***

***Sec-WebSocket-Version: 8***

SECE server replies by sending the response message below.

***HTTP/1.1 101 Switching Protocols***

***Upgrade: websocket***

***Connection: Upgrade***

***Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=***

Since it maintains a single TCP connection while communicating, it is scalable for the large number of concurrent users. It does not need to re-establish TCP connection every time it sends a message. Furthermore, since HTML5 also takes advantages of WebSocket [7], it would be also useful for the future web based system in SECE project.

**3.4 HTTP AND JSON BETWEEN ARDUINO AND SECE SERVER**

Figure 10 Messages between Arduino and SECE server

GET and POST HTTP methods are used for the communication between Arduino and SECE server. The contents of messages are created in JSON format. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for human beings to read and write. It is also simpler for machines to parse and generate messages than using XML. Currently, we use a simple JSON format to obtain states of actuators such as names and current values of the devices. For more details of the contents in JSON format, refer to the RFC4627 document [8].

1. **SENSOR AND ACTUATOR PROTOTYPE**

In this section, we introduce various kinds of experiment results to access different digital and analog sensors from Arduino using the current existing communication technologies. In addition, we address possible system scenarios to fulfill the various SECE services.

**4.1 TEMPERATURE SENSOR (TMP36)**

**Goal:**

* Enable to receive temperature data via Arduino

Figure 11 Get temperature data using TMP36

For the temperature sensor prototype using Arduino, the TMP36 temperature sensor is selected and wired on analog port in Arduino. This sensor has three pins; ground, signal and +5 volts. It generates 10 mVolts per degree centigrade on the signal pin. To convert this from the digital value to degrees, the math function in Arduino library was used. The output values were obtained from Arduino over a serial connection. The results were displayed on a terminal console on a desktop.

***void loop(){***

***float temperature = getVoltage(temperaturePin);***

***temperature = (temperature - .5) \* 100;***

***Serial.println(temperature);***

***}***

***float getVoltage(int pin){***

***return (analogRead(pin) \* .004882814);***

***}***

**4.2 ZIGBEE PROTOCOL**

**Goal:**

* Establish wireless connection from Arduino using Xbee
* Get temperature data via Zigbee protocol

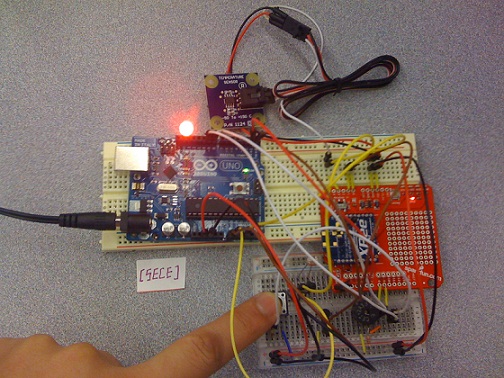
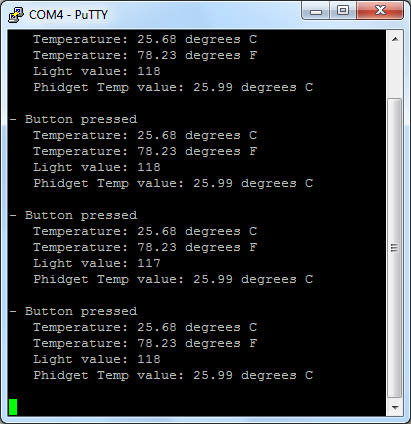
 

Figure 12 Get Temperature data over 802.15.4 protocol

Xbee module takes 802.15.4 protocol (the basis for Zigbee) and wraps it into a simple way to use a serial command set. It allows very reliable and simple communications between microcontrollers, computers, systems and really anything with a serial port. Point to point and multi-point networks are supported. To connect Xbee module to Arduino, Xbee shield is selected which is shown in Figure 13. At the receiver part, Xbee dongle is connected to a desktop. This setup of implementing wireless communication based on 802.15.4 protocol in Arduino will be used for the indoor location tracking system as a part of SECE services.

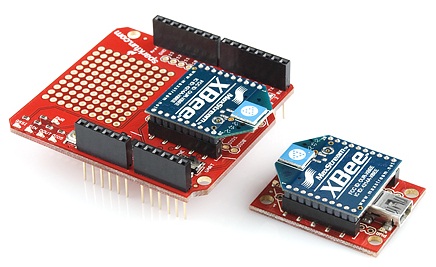


Figure 13 Arduino Xbee shield and dongle

**4.3 IR TRANSMITTER AND DETECTOR**

**Goal:**

* Decode and generate IR signals from Arduino

Figure 14 IR transmitter and detector wired on Arduino

As we see the characteristics of the IR detector (PNA4602) shown in Figure 15, this detector has the peak frequency detection at 38 KHz and the peak LED color at 940 nm. We programmed Arduino to generate IR signals using a high output LED in order to meet the specifications. Using this IR setup, it is expected to generate consumer IR signals for controlling the general IR electronics and decode the IR signals of the devices from Arduino.



Figure 15 Characteristics of PNA4602

Through this experiment, we figured out that the IR detector is able to decode the correct IR signal within 4~5 meters under the line-of-sight condition. The results can vary depending on the properties of IR LED/detector and environments.

Figure 16 Implement IR detector and transmitter

Most well-known electronic companies follow the general standards for consumer IR signals such as RC-5, RC-6 and NEC. The specific IR signals for the standards can be obtained online [9]. This IR setup will be used for SECE system to generate IR signals to control IR devices such as TVs and projectors. After SECE users provide the model numbers and brand names of their devices on our SECE web site, SECE system will automatically find the right IR signals from online. Then it sends the signals to the home gate way over the Internet and the gate way generates them to control the devices using the IR LED.

**4.4 X10 PROTOCOL**

**Goal:**

* Control electronics from Arduino over power lines using X10 protocol

Figure 17 Control a lamp using X10 protocol

X10 home automation protocol allows controlling general electronics via power lines. The electronics are directly controlled from X10 modules. For instance, the X10 module can provide power to them or cut it to turn off the devices. It can also control the amount of power to dim a light. It monitors the X10 signals from X10 transmitters on the same power line. Wired or wireless X10 transmitters can be connected to Arduino. The transmitters are shown in Figure 18.

Figure 18 X10 wired and wireless transmitters

In this experiment, X10 power line control (PLC) transmitter was wired on Arduino to control the lamp. When Arduino receives a command from SECE server over the Internet, it generates the right X10 signal and sends it over a power line. The X10 module receives it and then adjusts the amount of power to control the lamp. The following figure shows how to connect the X10 PLC transmitter to Arduino. 4 wires of telephone cable were used between the transmitter and Arduino.

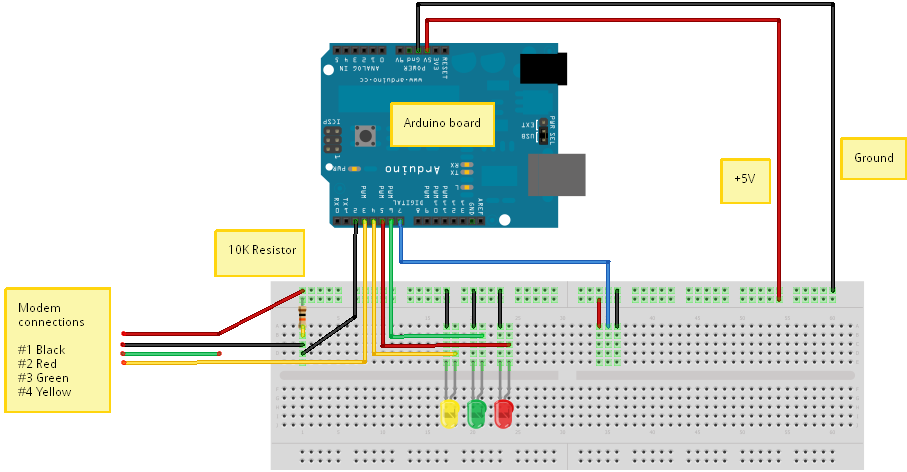


Figure 19 X10 PLC transmitter wired on Arduino

Note that it may not perform well when a power line has a noise. The X10 signals are likely to be interfered if many electronics are connected on the same power line.

**4.5 EVENT NOTIFICATION SYSTEM**

**Goal:**

* Implement an event notification system using Arduino

Figure 20 Event notification system using Arduino

An event notification system can be implemented in Arduino based on measurements from sensors and predefined If-conditions. Figure 20 shows the one of event notification systems using a light detector. Arduino is able to keep monitoring light values from the light detector. The light value expressed as integer number from the light detector changes depending on how much light is shining on its face. When it receives the data from the detector, it compares it with a predefined threshold. The threshold can be configured from SECE server over HTTP. If the measurement is over the threshold, it triggers the predefined actions. For instance, it can send a notification to SECE server or turn on the other switches. HTTP PUT method is used to send a notification message to SECE server. HTTP POST method from SECE server is used to activate or deactivate the notification function in Arduino. Finite state machine of the event notification system on Arduino side is shown in Figure 21.

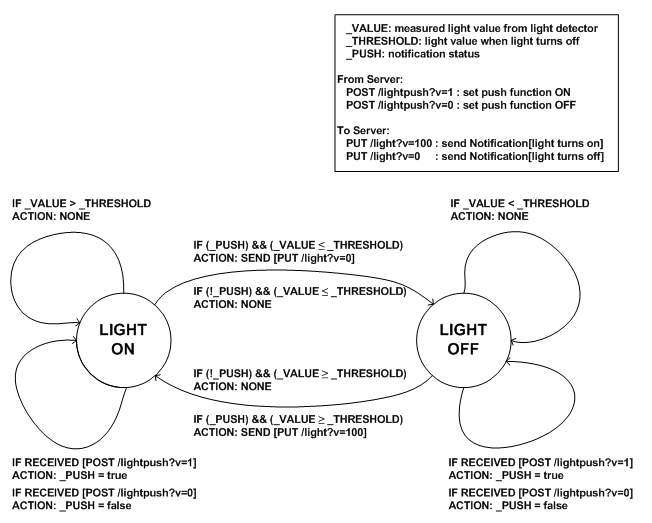


Figure 21 FSM of a light detector event notification system

**4.6 COMMUNICATION AND SENSOR LAYERS IN ARDUINO**

**Goal:**

* Define Arduino communication and sensor layers

**Figure 22 Flow chart of communication and sensor layers in Arduino**

Arduino consists of communication and sensor layers. The libraries in the communication layer are used to establish a reliable connection and communicate with SECE server. The sensor libraries are used to control actuators wired on Arduino and create output messages in JSON format. Also, it includes functions to monitor or configure sensors for SECE event driven system.  The libraries in the Arduino 0022 and 0023 IDEs are modified and new functions are designed for the various SECE services. Through these libraries, it is expected for SECE users simply to write sketches on Arduino in order to connect sensors with SECE server and manipulate them in order to create their own services. Figure 22 shows the flowchart of communication and sensor layers in Arduino.

**4.7 TIMER AND RESET**

**Goal:**

* Implement Arduino timer and reset functions

A timer function is used to send notifications to SECE server from Arduino periodically. For example, when SECE server is required to receive a temperature value every 30minutes from Arduino, this function is called to configure the Arduino. Once the timer is activated, Arduino reports SECE server with the measured temperature data by periods without any further request.



**Figure 23 Arduino timer: without Timer (left) and with Timer (right)**

**Figure 24 FSM of timer function to get temperature data from Arduino**

This function can be also used to periodically notify SECE server of the current state of Arduino such as an IP address and attached sensor information.

A reset function initializes all setups on Arduino in software. It performs the same as does when the reset button on Arduino is pressed. If there are conflicts on the communication with SECE server, Arduino will be reset and try to connect with SECE server again. This function can be initiated from SECE server using a HTTP POST message. Also, Arduino itself calls this function when it finds exceptional errors while controlling actuators.

***void software\_Reset(){***

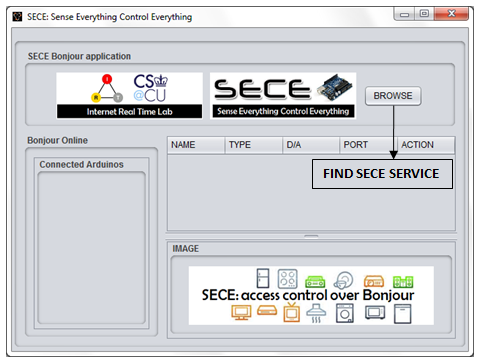
***asm volatile (" jmp 0");***

***}***

**4.8 FIND SECE SERVICE USING BONJOUR**

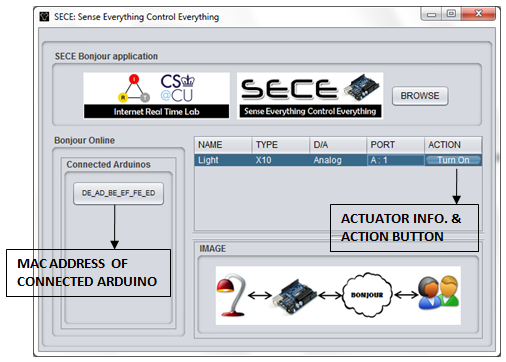
SECE Bonjour GUI application running on computers allows users to access SECE services over Bonjour in a local network. The following steps show how this application finds SECE service, resolves the address, obtains actuator information and controls the actuators.

* **Service registration**: SECE services are registered by local Arduinos, SECE.\_http.\_tcp.\_local. The Bonjour API for Arduino can be downloaded from online [10].
* **Browse and resolve**: A SECE Bonjour application browses SECE services using the service name, SECE. If SECE found, it resolves the addresses. The Arduinos in a local network reply the request with URL.



**Figure 25 Find SECE services via Bonjour**

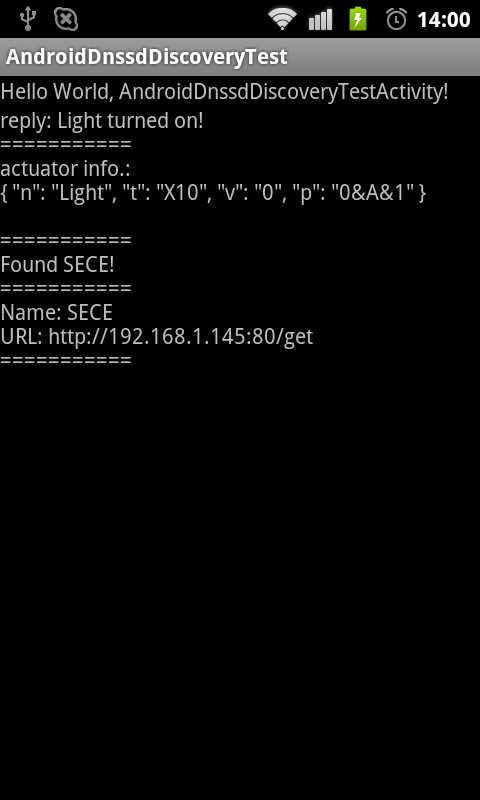
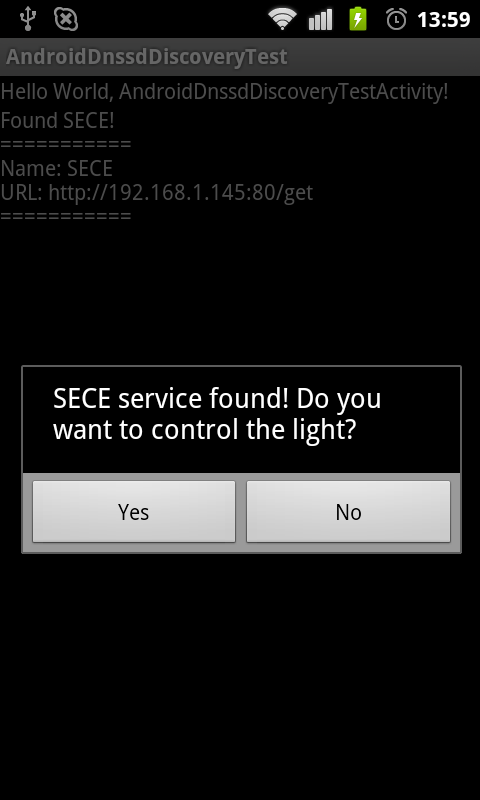
* **Get actuator information**: SECE Bonjour application obtains sensor information by visiting the resolved addresses. Currently, the information such as device name, communication type, current status and port number is stored in the Arduino itself, which is sent to the application in JSON format. For example, the following figure shows that this application finds one Arduino with MAC address (DE:AD:BE:EF:FE:ED) over Bonjour in a local network. One light actuator is connected to the Arduino via X10 protocol.



**Figure 26 Obtain sensor information using HTTP and JSON**

* **Control actuators**: Users can control the actuators by clicking the button and see the status from the application. When the action button is pressed, the application sends commands to the Arduino using http to control the actuator.

The SECE Bonjour GUI application is written in JAVA. This application uses a Bonjour API downloaded from the Apple developer site [11]. The Bonjour SECE client application can be converted to a mobile application running on Android smartphones. The following figure shows the Demo. To place Arduino and a smartphone connected in the same network, an access point is used in this experiment.



**Figure 27 Find SECE services using mDNS on Android smartphones**

1. **CONCLUDING REMARKS**

This technical report introduces how to access sensors and actuators over the Internet in SECE project. For a gate way based sensor and actuator prototype in SECE project, an Arduino microcontroller is selected for the following reasons. First of all, well-developed Arduino IDE provides C/C++ programming environment to compile codes on Arduino and the stackable Ethernet shield for Arduino allows actuators to communicate with SECE server over the Internet. Secondly, the DHCP and TCP stacks are already built in the Arduino IDE. Furthermore, it has been shown that most commercial home automation systems are compatible with Arduino. For instance, X10 and Phidgets modules are easily accessible from Arduino. To save memory usage in Arduino, a simple JSON format is used to exchange data messages between sensors and SECE server. Each sensor prototype is implemented depending on different sensor types such as a temperature, infrared LED and Zigbee protocol and system scenarios. In addition, based on the current existing Arduino libraries, we have implemented event notification, timer and reset, and Bonjour applications to fulfill various SECE services. We expect even non-technical users able to create their own sensor and actuator services simply by following our sensor and actuator prototypes.

1. **REFERENCES**

[1] Arduino official web site, *http://www.arduino.cc/*

[2] Beagleboard official web site, *http://beagleboard.org/*

[3] Netduino official web site, *http://netduino.com/*

[4] Phidgets official web site, *http://www.phidgets.com/index.php*

[5] Xbee radio, [*http://www.ladyada.net/make/xbee/ref.html*](http://www.ladyada.net/make/xbee/ref.html)

[6] Zigbee, [*http://en.wikipedia.org/wiki/ZigBee*](http://en.wikipedia.org/wiki/ZigBee)

[7] HTML5 Websocket, [*http://www.websocket.org/aboutwebsocket.html*](http://www.websocket.org/aboutwebsocket.html)

[8] JavaScript Object Notation (JSON), [*http://www.ietf.org/rfc/rfc4627.txt*](http://www.ietf.org/rfc/rfc4627.txt)

[9] IR library,[*http://lirc.sourceforge.net/remotes/*](http://lirc.sourceforge.net/remotes/)

[10] Bonjour library for Arduino, [*http://gkaindl.com/software/arduino-ethernet/bonjour*](http://gkaindl.com/software/arduino-ethernet/bonjour)

[11] Bonjour API,[*https://developer.apple.com/opensource/*](https://developer.apple.com/opensource/)