Journal of Universal Computer Science, vol. 24, no. 9 (2018), 1170-1191
submitted: 30/12/17, accepted: 30/6/18, appeared: 28/9/18 © J.UCS

An Experimental System for MQTT/CoAP-based IoT Applications in IPv6 over Bluetooth Low Energy

Chi-Yi Lin
(Tamkang University, Taipei, Taiwan
chiyilin@mail.tku.edu.tw)

Kai-Hung Liao
(Tamkang University, Taipei, Taiwan
605420156@s05.tku.edu.tw)

Chia-Hsuan Chang
(Tamkang University, Taipei, Taiwan
604410059@s04.tku.edu.tw)

Abstract: With the rapid development of Internet of Things (IoT), it is an inevitable trend that all things will get connected to the Internet to form various intelligent services such as Industry 4.0, smart home, smart medical care, etc. To make such intelligent IoT services practicable, it is vital to have a low-power link-layer technology that can accommodate a diversity of upper-layer networking protocols. Currently, there are many popular low-power wireless networking technologies for IoT such as ZigBee and Bluetooth Low Energy (BLE). Because of the ubiquity of BLE-enabled smartphones nowadays, BLE has gained much attention in the IoT industry recently. In this research, we aim at implementing an IPv6 over BLE experimental system using Raspberry Pi 3 and nRF51-DK development boards, and then run the Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN) protocol and the Constrained Application Protocol (CoAP) over the protocol stack of IPv6/BLE. Specifically, in our experimental system every BLE node is IPv6-addressable and accessible through the MQTT/CoAP protocols from anywhere over the Internet. Moreover, to ease user accesses from ordinary web browsers, we build two gateways as the web servers for end users, which receive real-time sensor data via CoAP or MQTT-SN protocols and then push the data to end users' browsers. The gateways are also designed to routinely request sensor data and then forward the data to cloud database platforms, which serve as the data sources for historical sensor data. Preliminary results showed that our system is capable of achieving the designed goals and is user-friendly. Compared with the non-IP based BLE sensor networks, our implementation can be integrated into a variety of existing and widely used IP-based applications easily.

Keywords: Internet of Things, Bluetooth Low Energy, 6LoWPAN, MQTT, CoAP
Categories: H.3.4, H.4.3, H.5.2

1 Introduction

In the field of *Internet of Things* (IoT), wireless sensor networks with the characteristic of low power consumption are becoming more and more important. Bluetooth Low Energy (BLE) [Bluetooth SIG, 10c] is one of the latest short-range wireless communication technologies, which was merged to the 2010 version of Bluetooth standard and integrated with the Bluetooth core specification version 4.0

and simplify the protocol stack. The overall objective is to achieve low-cost, low-power consumption, long sleeping mode application technology that is suitable for IoT sensor networks. On the other hand, with the expansion of the IoT and the field of intelligent applications, vast amounts and varying types of sensor devices are used in the IoT field, meaning that there will be lots of different communication technologies and devices coexisting in the IoT environment. If we would like to integrate different types of sensor networks to form a larger IoT network, we face the problem of interoperability issue. Fortunately, IPv6 is regarded as the most promising solution, as long as every sensor device is capable of running IPv6.

The Internet Engineering Task Force (IETF) formulated the transmission of IPv6 packets over Bluetooth Low Energy using IPv6 over Low-power Wireless Personal Area Network (6LoWPAN) technology [Kushalnagar, 07], which is referred to as *IPv6 over BLE* [Nieminen, 15]. It allows sensor nodes to achieve end-to-end IPv6 communication, even between different link-layer technologies, overcoming the limitation of communication only within sensor networks of the same type. This clearly promotes the interoperability with different devices and sensor networks. 6LoWPAN was formulated by the IEFT working group to provide IPv6 routing solutions for low power and lossy networks (LLNs), and it is an adaptation layer between the link layer and the network layer of the OSI model. Due to the fact that the size of IPv6 packets is larger than BLE packets, therefore, 6LoWPAN provides header compression and fragmentation of datagrams. This allows transmission of IPv6 packets within low power wireless personal area networks, thereby creating IPv6-connected wireless sensor networks.

The devices within IoT environments are usually resource-limited. Therefore, the IETF formulated the *Constrained Application Protocol* (CoAP) [Shelby, 14] which is a specialized transfer protocol and lightweight M2M communication technology, applied in constrained networks and with constrained devices. CoAP uses UDP transmission combined with the reliable CoAP messaging mechanism thus providing advantages such as low-overhead, lightweight and reliable transmission. *Message Queuing Telemetry Transport* (MQTT) [Oasis, 14] is another popular lightweight application-layer messaging protocol, which has become an OASIS standard since 2014. MQTT runs on top of TCP/IP, and supports publish/subscribe messaging model. It has been used to build Facebook Messenger app for assured and faster message delivery. Other real-world applications of MQTT include Amazon IoT, Microsoft Azure IoT Hub, to name a few. However, the connection-oriented feature of TCP can still be a burden for some devices in the sensor networks. Therefore, IBM proposed MQTT for Sensor Networks (MQTT-SN) [Stanford-Clark, 13] which runs on top of UDP. In our experimental setup, our objective is to create a relatively lightweight smart home environment, building an IPv6 over BLE sensor network with sensor nodes running CoAP and MQTT-SN protocols. To ease the collection of environmental data, we also build two gateways: one serves as an HTTP server and a CoAP client; the other serves as an HTTP server and a MQTT subscriber. The collected environmental data are then stored in cloud-based databases, which can be used to create statistical charts for end users.

The rest of this paper is organized as follows: In Section 2 we show the background technologies, which mainly clarify the IPv6 over BLE architecture, the CoAP protocol, and the MQTT/MQTT-SN protocol. In Section 3 we introduce the

organization of our experimental system, from the perspective of the network architecture and the application architecture, respectively. Section 4 describes the details of the implementation, while the preliminary results are shown Section 5. Finally, in Section 6 we conclude our work and give some future directions.

2 Background and Related Work

2.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is a wireless personal area network technology designed by Bluetooth Special Interest Group. BLE was merged into main Bluetooth standard in 2010 with the adoption of the Bluetooth Core Specification Version 4.0. Compared with the classic Bluetooth, BLE is intended to provide considerably low power consumption by working at extremely low duty cycle, reducing the number of channels, allocating the advertising channels at frequency bands that do not overlap with 2.4GHz Wi-Fi channels, and with flexible packet size, etc. BLE operates in the spectrum range from 2.400 ~2.4835GHz ISM band and has 40 2-MHz channels, while the classic Bluetooth uses 79 1-MHz channels. The transmission rate of BLE is about 1 Mbit/s and it uses frequency hopping transmission to reduce interference. BLE's advertising channels are channel 37, 38, 39 and scanning devices will regularly listen to these channels to obtain advertising information for establishing any possible connections. All the remaining channels are used to transmit data.

Bluetooth 4.0 brings two new core protocols: *Attribute Protocol* (ATT) [Bluetooth SIG, 10a] and *Generic Attribute Profile* (GATT) [Bluetooth SIG, 10b]. GATT is built on top of the ATT, which uses GATT data to define the way that two BLE devices send and receive messages. There are two roles in GATT: *server* and *client*. The GATT server stores the data transported over ATT and accepts ATT requests/commands/confirmations from the GATT client. On receiving requests from the GATT clients, the GATT server responds by sending response messages. When configured, the GATT server can also send asynchronous indications/notifications to the GATT client once specified events occur on the GATT server.

2.2 6LoWPAN

With an astronomical address space, IPv6 is by far the most suitable identifier for IoT devices. However, the sizes of IPv6 packets are too large for IoT devices that rely on low-power wireless area network technologies. If we would like to send IPv6 packets over low-power wireless area networks, IPv6 packets need to be fragmented and compressed [Hui, 11].

This idea was first realized in RFC 4944 [Montenegro, 07], proposed by an IETF working group called *IPv6 over Low-power Wireless Personal Area Network* (abbreviated as 6LoWPAN). RFC 4944 specifies how to transmit IPv6 packets over IEEE 802.15.4 (ZigBee) networks. In 2015, IETF published RFC 7668 – IPv6 over Bluetooth Low Energy [Nieminen, 15], which adds an IPv6 protocol stack on top of the Bluetooth LE L2CAP layer.

According to RFC 7668 there are two roles in a BLE subnet: *6LoWPAN Border Router* (6LBR) and *6LoWPAN Node* (6LN). As shown in Figure 1, the 6LBR is

located at the edge of a BLE network. It plays the role of a gateway between the BLE network and the Internet. Within the BLE network, the 6LBR is responsible for distributing IPv6 prefix to the 6LNs, which can auto-configure their IPv6 address given the IPv6 prefix.

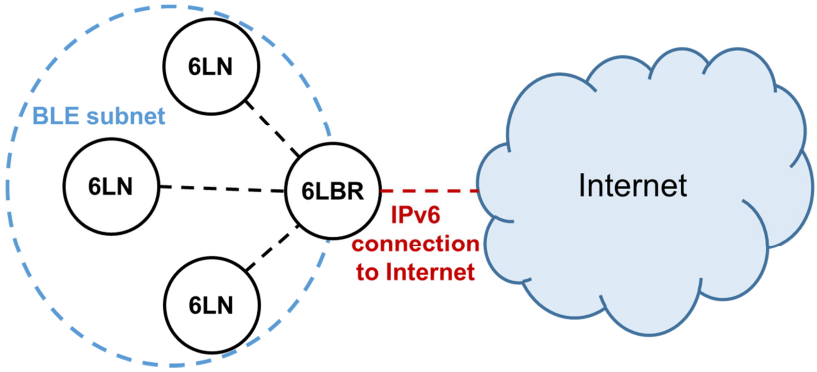

Figure 1: A BLE subnet connected to the Internet

2.3 CoAP

Constrained Application Protocol (CoAP) is a web transfer protocol based on the widely used REST model. Compared with HTTP, CoAP runs on top of UDP and has smaller header size. The low-overhead features of CoAP makes it suitable for use in the network devices with constrained resources. On top of UDP, CoAP defines the *Messaging* layer and the *Request/Response* layer. At the Messaging layer, four types of messages are exchanged over UDP between endpoints: *Confirmable* (CON), *Non-confirmable* (NON), *Acknowledgement* (ACK), and *Reset* (RST). When an endpoint receives a Confirmable message, it must acknowledge the CON message with an ACK message, which achieves reliable transmission in CoAP. In some cases that acknowledgements are not necessary, the messages can be transmitted less reliably by marking them Non-confirmable. At the Request/Response layer, CoAP uses client/server architecture and the available server resources are identified by URIs. CoAP clients access the server resources using methods such as GET, PUT, POST, and DELETE. Note that depending on specific needs of different application scenarios, the requests and responses can be either Confirmable or Non-confirmable.

2.4 MQTT and MQTT-SN

MQTT is a publish/subscribe-based lightweight messaging protocol for use on top of TCP/IP. With the publish/subscribe model, the publishers and subscribers do not need to know the existence of one another. Instead, all the data published by the publisher will be collected at the message broker, and then it is the responsibility of the message broker to push the data to the subscribers. In 2013, IBM proposed the MQTT-SN, where the term SN stands for *Sensor Networks*. As its name suggests, MQTT-SN is designed to be used in wireless communication environment with low bandwidth, high link failures, and short message length. It is also optimized for implementation

on low-cost and resource-constrained devices. Although MQTT-SN was originally developed to run on top of ZigBee, the specification emphasizes that it is agnostic of the underlying network services. Therefore, in resource-constrained devices, MQTT-SN can run on top of UDP rather than TCP.

Figure 2 shows the publish/subscribe model of MQTT and MQTT-SN. The MQTT broker is primarily responsible for receiving all messages, filtering them, deciding who has access to the data, and transmitting messages to legal subscribers. Subscribers play a passive role in the system. More specifically, subscribers do not need to query whether there are new data or not; they simply wait for new data to be pushed to them. In this way, both the publishers and the subscribers can have a low duty cycle, which means their power consumption can be greatly reduced.

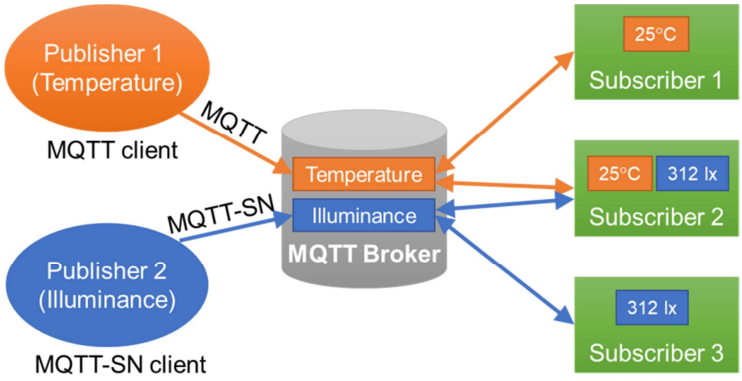

Figure 2: Publish/subscribe model of MQTT and MQTT-SN

2.5 Related Work

In 2013 Wang *et al.* [Wang, 13] implemented a prototype system to send IPv6 packets over BLE based on *BlueZ* [BlueZ, 17], the official Linux Bluetooth protocol stack. Their main contribution is the proposed context exchange mechanism between the sensor nodes and the router, which enhances the efficiency of the IPv6 header compression in RFC 6282 [Hui, 11]. Their experimental results showed that with the new context exchange mechanism, the number of frames to be transmitted can be reduced, and hence the transmission latency and the power consumption can also be reduced. In 2016 Kamma *et al*. [Kamma, 16] built a 6LoWPAN network over ZigBee, in which the 6LBR was built by using the Beagle Bone Black (BBB) development board. In the paper the authors have shown the needed software packages and the detailed procedures to set up the 6LBR. The functionality of the 6LBR was verified by using tools *tcpdump* and *Wireshark*. In 2016 Yoon *et al.* [Yoon, 16] designed and implemented an IPv6 over BLE platform to realize a patient-centric healthcare service. The unique feature of their system is the proposed *Advertising Data Transmission Protocol* which can send emergency data from the patients to a backend server through a non-paired gateway (such as a nearby person's smartphone) during critical medical situations. In their implementation, both 6LBR and 6LN were implemented by Raspberry Pi with CC2540 BLE USB dongle. Raspberry Pi carried out the

functionalities of 6LoWPAN, TCP/UDP, and CoAP, while CC2540 BLE USB dongle provided the BLE protocol stack. The authors didn't show how they configured Raspberry Pi to enable 6LoWPAN and CoAP in the paper; only the round-trip time and the throughput using UDP packets were measured. In 2017 Ahmed *et al.* [Ahmed, 17] developed a system that gathers the temperature data in a 6LoWPAN network over IEEE 802.15.4 and sends the data to a MQTT broker in the Internet. The 6LBR and 6LNs are implemented by running Thingsquare 6LoWPAN software stack and Contiki OS on Texas Instruments development boards (Tiva C Micro Controller Unit and CC1120 single-chip radio transceiver). Their system demonstrated a possible solution of controlling remote sensors and actuators from the cloud.

3 System Architecture

In this section, we will introduce the network architecture and the application architecture in our experimental system.

3.1 Network Architecture

Figure 3 shows the network architecture in our experimental system. There are two BLE subnets, which consist of five *Raspberry Pi 3* development boards and one *Nordic nRF51-DK* development board. Two Raspberry Pi 3 act as the 6LBRs for the two BLE subnets, respectively. All other boards act as the 6LNs, which are connected to the environmental sensor devices such as temperature sensors, humidity sensors, and ambient light sensors. The reason we choose Raspberry Pi 3 and Nordic nRF51-DK as the BLE nodes is their built-in support of Bluetooth 4.0 and the 6LoWPAN module. Raspbian, the official Linux-based operating system for Raspberry Pi, contains a number of useful modules which help us build the desired functionalities. The modules we used will be detailed in Section 4. The BLE networks are connected to the Internet via a router. We also set up a web server and a database server, to display and save the environmental data from the sensor nodes, respectively. Furthermore, we deploy a cloud database as a backup storage for environmental data, which can also be accessed by the clients.

Here we describe the IPv6 address assignments in our experiment. By default, an IPv6 address consists of a 64-bit network prefix and a 64-bit interface identifier. We have been allocated the network prefixes of 2001:288:d003:a000::/64 and 2001:288:d003:a010::/64 to be used inside our two BLE subnets, respectively. Specifically, in each BLE subnet, the 6LBR broadcasts the prefix to the network hosts so that they can generate their own addresses by using *stateless address auto-configuration* specified in RFC 4862 [Thomson, 07]. As for the Ethernet interfaces on the 6LBRs which serve as the WAN side, they are on the IPv6 subnet with the network prefix of 2001:288:d003:1127::/64. Each facing two different IPv6 subnets, the 6LBRs play the role of a gateway between the BLE subnet and the Internet.

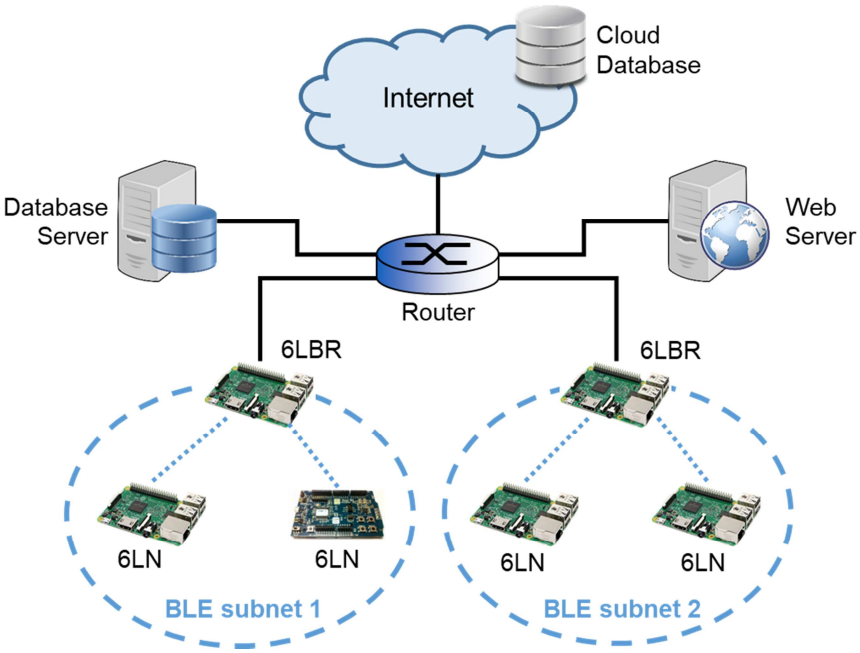

Figure 3: Network architecture in our experimental system

Now let's describe the procedure of how the 6LNs connect to the 6LBR and finish their IP configurations. This is basically a three-step process explained as follows:

Step 1: Establish link-layer connection.
According to RFC 7668, before any IP-layer communications can take place over BLE, 6LNs and 6LBRs have to discover each other and establish a suitable link-layer connection. Therefore, when a 6LN is powered on, the first step is to establish a link-layer connection to the 6LBR. The discovery and connection procedures are defined in BLE *Generic Access Profile* (GAP) [Bluetooth SIG, 10]. At the link layer, the 6LN plays the *peripheral* role which advertises itself to be scanned by the 6LBR, which plays the *central* role. When the 6LN is detected by the 6LBR, the 6LBR can initiate a link-layer connection establishment procedure with the 6LN. Once the link-layer connection has been established successfully, they can move to the next step of creating an L2CAP channel which is responsible for protocol multiplexing capability, segmentation, and reassembly operation for data exchanged between the two ends.

Step 2: Establish L2CAP channel.
According to the *Internet Protocol Support Profile* (IPSP) [Bluetooth SIG, 14] standardized by the Bluetooth SIG, the L2CAP channel type shall be an *LE Credit Based Connection*, and the L2CAP channel establishment shall also be initialized by the 6LBR. Note that the purpose of the IPSP is to allow devices

to discover and communicate to other devices that support IPSP. There are two roles defined by IPSP – *Node* role and *Router* role. The Router role is used for devices that can route IPv6 packets, which is exactly the 6LBR's responsibility. The 6LNs, however, only originate or consume IPv6 packets, play the Node role. In this use case of supporting IP service over BLE, the 6LNs implement the *Generic Attribute Profile* (GATT) server role and expose *IP Support Service* (IPSS) to serve the GATT clients (i.e., 6LBRs).

Step 3: Configure IPv6 address.

On top of the L2CAP channel sits the *6LoWPAN for BLE* layer to provide functionalities of stateless IPv6 address auto configuration, neighbour discovery, and header compression. Basically, the neighbour discovery process follows RFC 6775 [Shelby, 12]. The 6LN sends *Router Solicitation* (RS) messages to the 6LBR, and the 6LBR responds with *Router Advertisement* (RA) messages containing the network prefix. With stateless address auto configuration, the 6LN is able to generate a global IPv6 address. The complete process of how a 6LN connects to a 6LBR and finishes its IP configuration is shown in Figure 4.

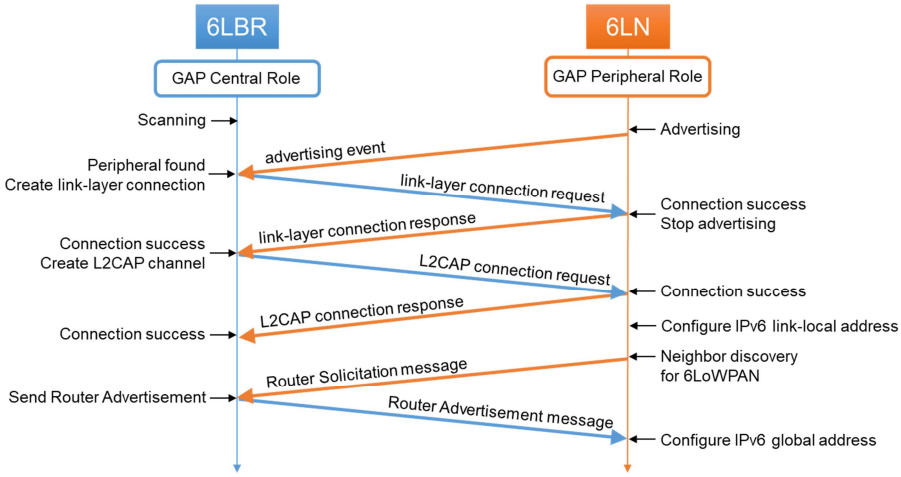

Figure 4: Network configuration procedure in IPv6 over BLE network

3.2 Application Architecture

As soon as all the BLE devices have completed their IP configuration, they are ready to execute IP-based applications. Common application-layer protocols based on IP such as HTTP, SSH, and FTP can be applied to our system. However, considering the fact that IoT nodes are mostly resource-constrained, we choose MQTT-SN and CoAP as the light-weight application-layer protocols to be used in the BLE networks. The application architecture in our experimental system is shown in Figure 5. We can see that the two BLE subnets run different application protocols. In BLE subnet 1, the two 6LNs play the role of the CoAP servers, while $6LBR_1$ plays the role of a CoAP client as well as a web server. Specifically, we make $6LBR_1$ a gateway device that accepts HTTP

requests from remote users with ordinary web browsers. If a remote user clicks a button to turn on a light in BLE subnet 1, $6LBR_1$ will issue a CoAP request to the 6LN that connects to the light. After turning on the light, the 6LN replies with a CoAP response that indicates the latest state of the light is ON. When the CoAP response is received at $6LBR_1$, the latest state is updated on the web browser via a HTTP response. The MongoDB server and the *xively* cloud platform [xively, 17] are used to store historical data for the sensors in BLE subnet 1.

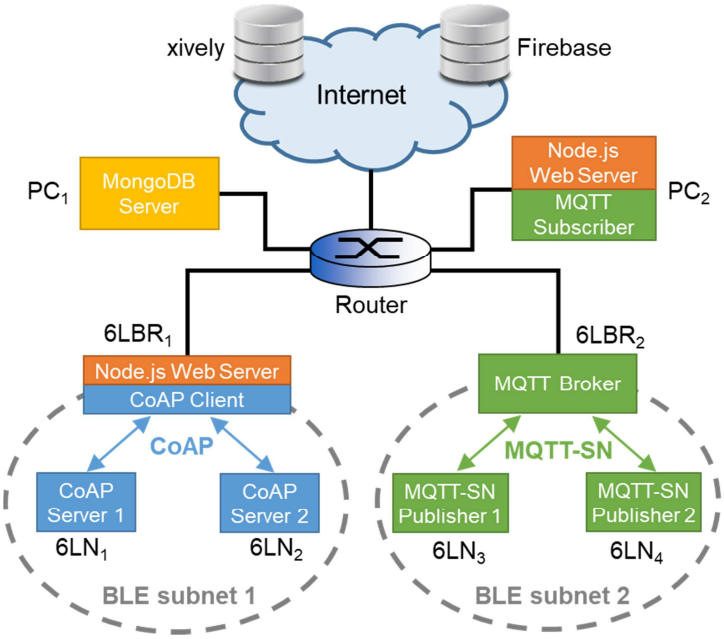

Figure 5: Application architecture in our experimental system

In BLE subnet 2, the two 6LNs play the role of the publishers, while the 6LBR plays the role of a broker. The message exchanges between the publishers and the broker are based on the MQTT-SN protocol. PC_2 plays the role of a subscriber as well as a web server, which communicates with the broker via the MQTT protocol. Specifically, the web server utilizes a MQTT client library to subscribe to the sensor data. The real-time sensor data are displayed at the front end and then pushed to the *Firebase* cloud database [Firebase, 17]. Note that the broker needs to support both MQTT and MQTT-SN protocols.

4 Implementation

In this section, we will first show the procedures of implementing IPv6 over BLE, and then describe how to enable CoAP and MQTT-SN in the 6LoWPAN networks. Note that we only show the configurations for Raspberry Pi 3 in the following subsections. For the nRF51-DK development board, its functionalities of 6LoWPAN over BLE and

CoAP were coded in C language with the support of several built-in libraries from the nRF51 IoT SDK [Nordic Semiconductor, 18]. For simplicity, we chose to skip the tedious C codes.

4.1 Implementing IPv6 over BLE

In Section 2 we mentioned that the 6LoWPAN technique was originally designed for IEEE 802.15.4. Until late 2015 the specification of 6LoWPAN over BLE was also proposed. In fact, as we mentioned in the related work, Wang *et al.* [Wang, 13] has successfully implemented the first prototype system to transmit IPv6 packets over BLE based on BlueZ. Therefore, we also use BlueZ to set up the BLE connection. First, we wake up the BLE interface of the 6LN and make it do advertising. Afterwards, the 6LBR can establish a link-layer connection to the 6LN. Figure 6 shows the configuration result at the 6LBR after the link-layer connection with the 6LN has been established, in which B8:27:EB:2C:43:A7 is the Bluetooth MAC address of the 6LN. From Figure 6 we can also see that at this stage, three network interfaces (eth0, lo, and wlan0) are up and running, but the Bluetooth network interface (bt0) does not exist.

```
root@gateway:~# hcitool con
Connections:
root@gateway:~# hcitool lecc B8:27:EB:2C:43:A7
Connection handle 64
root@gateway:~# hcitool con
Connections:
        < Unknown B8:27:EB:2C:43:A7 handle 64 state 1 lm MASTER
root@gateway:~# ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:87:e7:80
          inet addr:163.13.127.114  Bcast:163.13.255.255  Mask:255.255.0.0
          inet6 addr: 2001:288:d003:1127::114/64 Scope:Global
          inet6 addr: fe80::b070:7d0c:7ec6:ecb5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:109842 errors:0 dropped:2956 overruns:0 frame:0
          TX packets:4182 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10397495 (9.9 MiB)  TX bytes:607317 (593.0 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr b8:27:eb:d2:b2:d5
          inet6 addr: fe80::5895:eb47:dbb5:8db6/64 Scope:Link
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:6 errors:0 dropped:6 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:823 (823.0 B)  TX bytes:0 (0.0 B)
```

Figure 6: Network configuration at the 6LBR

To create the Bluetooth network interfaces for 6LBRs and 6LNs, we need to load the bluetooth_6lowpan module in the Raspbian OS and enable the 6LoWPAN functionality. The updated network configuration results of the 6LBR and the 6LN are shown in Figure 7, where we can see that the Bluetooth network interface bt0 has been created successfully with the IPv6 link-local address starting with 0xFE80.

```
root@gateway:/home/pi# echo "connect B8:27:EB:2C:43:A7 1" > /sys/kernel/debug/bluet
ooth/6lowpan_control
root@gateway:/home/pi# ifconfig
bt0       Link encap:UNSPEC  HWaddr B8-27-EB-FF-FE-2D-4D-2A-00-00-00-00-00-00-00-00

          inet6 addr: fe80::ba27:ebff:fe2d:4d2a/64 Scope:Link
          UP POINTOPOINT RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:9 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:298 (298.0 B)  TX bytes:492 (492.0 B)
```

```
pi@blue:~ $ ifconfig
bt0       Link encap:UNSPEC  HWaddr B8-27-EB-FF-FE-2C-43-A7-00-00-00-00-00-00-00-00

          inet6 addr: fe80::ba27:ebff:fe2c:43a7/64 Scope:Link
          UP POINTOPOINT RUNNING MULTICAST  MTU:1280  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:492 (492.0 B)  TX bytes:298 (298.0 B)
```

Figure 7: Bluetooth network interface bt0 created at both 6LBR and 6LN

According to the IPv6 neighbor discovery protocol, IPv6 routers are responsible for sending *Router Advertisement* (RA) [Narten, 07] messages which carry the IPv6 prefix and the network configuration information to the IPv6 host. To do that, we use a Linux software named *Linux IPv6 router advertisement daemon* (a.k.a. *radvd*) [radvd, 17] in the 6LBR. Figure 8 shows the radvd configuration on the interface bt0, where the advertisement function is turned on and the network prefix is listed. Now the 6LBR is ready to advertise the IPv6 prefix.

```
 GNU nano 2.2.6                File: /etc/radvd.conf

interface bt0
{
    AdvSendAdvert on;
    prefix 2001:288:d003:a010::/64
    {
        AdvOnLink off;
        AdvAutonomous on;
        AdvRouterAddr on;
    };
};
```

Figure 8: radvd configuration at the 6LBR

The next step is to configure the IPv6 global addresses in the 6LNs. As shown in Fig. 8, since we would like to advertise the network prefix of 2001:288:d003:a010::/64 to the 6LNs, the 6LBR itself must be configured with an IPv6 address within the same subnet. Therefore, we assigned 2001:288:d003:a010::0000 for the 6LBR. Then, the 6LBR runs the radvd service so the 6LNs can receive the router advertisements. As stated in Section 3, on receiving router advertisements, 6LNs can configure their own global IPv6 addresses by using stateless address auto-configuration (RFC4862). In this way we can build the IPv6 over BLE architecture successfully. Then, we can test the network connectivity from the 6LNs to the Internet. Figure 9 shows the *ping* test result from the 6LN to the Google IPv6 server. Note that we specified the bt0 interface to send the ping packets. As the

figure shows, the ping test is successful, which means that the 6LN is able to reach the Internet via the 6LBR.

```
pi@blue:~ $ ping6 -I bt0 ipv6.google.com
PING ipv6.google.com(tm-in-x8a.1e100.net) from 2001:288:d003:a010:ba27:ebff:fe2c:43
a7 bt0: 56 data bytes
64 bytes from tm-in-x8a.1e100.net: icmp_seq=1 ttl=41 time=197 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=2 ttl=41 time=141 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=3 ttl=41 time=153 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=4 ttl=41 time=164 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=5 ttl=41 time=175 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=6 ttl=41 time=186 ms
64 bytes from tm-in-x8a.1e100.net: icmp_seq=7 ttl=41 time=197 ms
^C
--- ipv6.google.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6007ms
rtt min/avg/max/mdev = 141.537/173.753/197.818/20.277 ms
pi@blue:~ $
```

Figure 9: IPv6 ping test from the 6LN to the Internet

4.2 Implementing CoAP over 6LoWPAN and the Gateway

To run the CoAP protocol in the 6LoWPAN network, we use the *node-coap* package (Node.js CoAP library) [node-coap, 17] to code the CoAP server on the 6LNs, and the *aiocoap* package (Python CoAP library) [aiocoap, 17] to code the CoAP client on the 6LBR. Figure 10 shows a CoAP request/response transaction in BLE subnet 1 between the 6LBR (CoAP client) and the 6LN (CoAP server) connected to a HTU21D humidity sensor. In the request message, the CoAP client specifies the message type as *Confirmable* and uses the GET method to request data from the remote endpoint with IPv6 address 2001:288:…:FE42:DD51 and port number 5683. In the response message, we can see the 2.05 response code which acknowledges the Confirmable request and echoes the message ID of 17739, carries the humidity value 50.

```
pi@gateway:~/aiocoap $ sudo python3 coaptestn2.py
Request Information :

<bound method Message.get_request_uri of <aiocoap.Message at 0x76544
290: Type.CON GET (ID 17739, token b'\x00\x00\x95\x9b') remote <UDP6
EndpointAddress [2001:288:d003:a000:24a:ddff:fe42:dd51]:5683>, 3 opt
ion(s)>>

Response Information :

State of HTU21D: b'50' Response Code: 2.05 Content MessageID: 17739
```

Figure 10: A CoAP request/response transaction

We also implemented the Observe function of the CoAP protocol defined in RFC 7641 [Hartke, 15], so the CoAP client can register its interest in a certain resource by initiating an extended GET to the server. In other words, the CoAP client acts as an observer, and the CoAP server notifies the client whenever the state of a resource changes. Figure 11 shows the result that the 6LBR observes the humidity resource state at the 6LN, where the 6LN is programed to send a notification every 3 seconds.

```
pi@gateway:~/aiocoap/mytest_coap $ sudo python3 coaptestobserpiv1.py
Humidity: b'50.3     time: 2017-06-01 10:45:50' observe ID:  0  Token: b'\x00\x00\xe8\xe9'

Humidity: b'50.7     time: 2017-06-01 10:45:55' observe ID:  2  Token: b'\x00\x00\xe8\xe9'

Humidity: b'50.3     time: 2017-06-01 10:45:58' observe ID:  3  Token: b'\x00\x00\xe8\xe9'

Humidity: b'50.3     time: 2017-06-01 10:46:01' observe ID:  4  Token: b'\x00\x00\xe8\xe9'

Humidity: b'50.6     time: 2017-06-01 10:46:04' observe ID:  5  Token: b'\x00\x00\xe8\xe9'
```

Figure 11: The CoAP client observes the resource state at the CoAP server

In Section 3.2 we mentioned that the 6LBR in BLE subnet 1 plays the role of a gateway between the HTTP-based Internet and the CoAP-based BLE subnet. To provide real-time sensor data to the end users, the CoAP client on the 6LBR acts as an observer that continuously receives the updated resource states. As shown in Figure 12, whenever an updated state is received at the CoAP client, the HTTP server at the 6LBR pushes the data to the web browser via the WebSocket technology [Fette, 11]. Therefore, the web browser does not need to poll the web server for the latest sensor data.

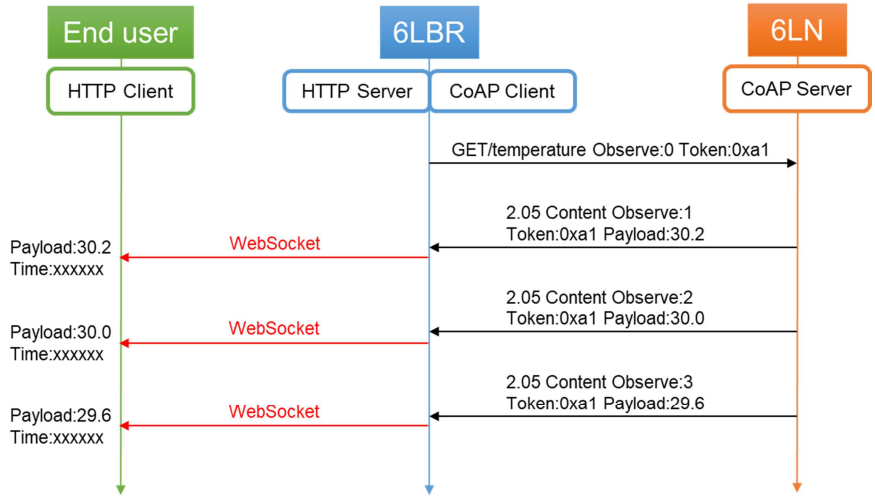

Figure 12: Sensor data is forwarded to the web browser by WebSocket

4.3 Implementing MQTT-SN over 6LoWPAN and the MQTT Broker

To build the MQTT/MQTT-SN environment over the 6LoWPAN network, we use the *Really Small Message Broker* (RSMB) [Eclipse, 14] from the open source *Eclipse Mosquitto* project [Eclipse, 17]. RSMB is an MQTT/MQTT-SN message broker produced freely by IBM and made available for personal use. After RSMB is installed on the 6LBR, we run the executable file called broker_mqtts to start the broker service. With default configuration, port 1883 and 1884 are turned on to listen to MQTT and

MQTT-SN messages, respectively. Figure 13 shows the configuration file for RSMB, in which the term MQTT-S actually refers to MQTT-SN. At the 6LNs, we use the MQTTSClient program in RSMB to perform the publisher's function.

```
 GNU nano 2.2.6                          File: broker.cfg

# will show you packets being sent and received
trace_output protocol

#predefined topics
predefined_topics_file /home/pi/rsmb/rsmb/src/pdtf

# setting the user management and topic access control
password_file /home/pi/rsmb/rsmb/src/passwd

#allow_anonymous false

# normal MQTT listener
listener 1883 INADDR_ANY
max_connections 10
ipv6 true

# MQTT-S listener
listener 1884 INADDR_ANY mqtts
max_connections 5
ipv6 true
```

Figure 13: Configuration file for RSMB

As described in Section 3.2, PC2 in our implementation is a subscriber who subscribes to the sensor data. In our design, the 6LNs send the sensor data every 30 seconds to the broker, including humidity, temperature, and illumination. At the same time, PC2 is also a web server which provides the sensor data to end users. The web server is based on Node.js, in which we installed *MQTT.js*, a client library for the MQTT protocol. Similar to what $6LBR_1$ does, when PC2 receives the subscribed sensor data, the data is pushed to the web browser by using *Socket.IO* JavaScript library. With these data, our web server can visualize them by showing line charts. Besides, since all BLE devices have their own global IP addresses, system administrators are able to send *ping* packets to confirm whether the devices are alive, and perform remote management tasks via SSH when necessary. Moreover, we run NTP in 6LNs so they can synchronize their local clocks with the time server. With this capability, our 6LNs can send environmental data with exact timestamps. We also deployed a cloud-based database using Google Firebase, which serves as the data store for the sensor data. If an end user would like to look up the historical data, the web server will send the query to Firebase, and then plot the returned data by Chart.js.

5 Preliminary Results

In this section, we would like to show some preliminary results from our implementation. First of all, we have mentioned that our 6LNs are with IPv6 global addresses, so they can be accessed directly from the Internet. For example, if an end user runs a CoAP client program, he/she would be able to access the 6LNs in BLE subnet 1. To test this use case, we installed *Copper* [Copper, 17] (CoAP user agent for Firefox) in the Firefox browser on one of the desktop computers in our campus

network and use it to access the 6LN connected with a humidity sensor and a LED light. Figure 14 is a snapshot of the Firefox browser by which we directly specified the URL of the CoAP server and used the Observe option to get the timestamped humidity data. In this figure we can also see a list of offered services on this CoAP server. From the GUI end users can press the button to send the desired commands such as GET, POST, PUT, and DELETE to manipulate the resource.

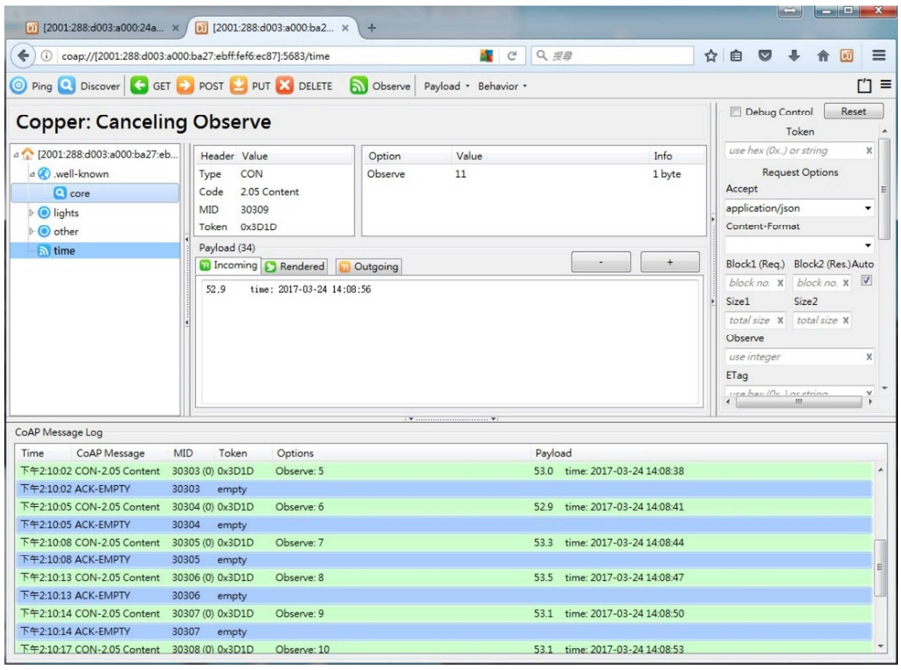

Figure 14: Firefox browser with Copper plug-in as a CoAP client

Apart from using the Firefox browser with Copper plug-in, end users from the Internet can also access the sensor data using ordinary web browsers. Figure 15 is a snapshot of the web interface connected to the web server at $6LBR_1$. From this web page end users can get the latest humidity data, turn on/off the LED light and the electrical relay, all by clicking a single button. On receiving the clicking events, the web server will send the commands to the 6LN using CoAP requests. For end users who would like to observe the humidity data, he/she can click on the *Humidity Observe* link to open the web page as shown in Figure 16, from which the user can start and stop the observation.

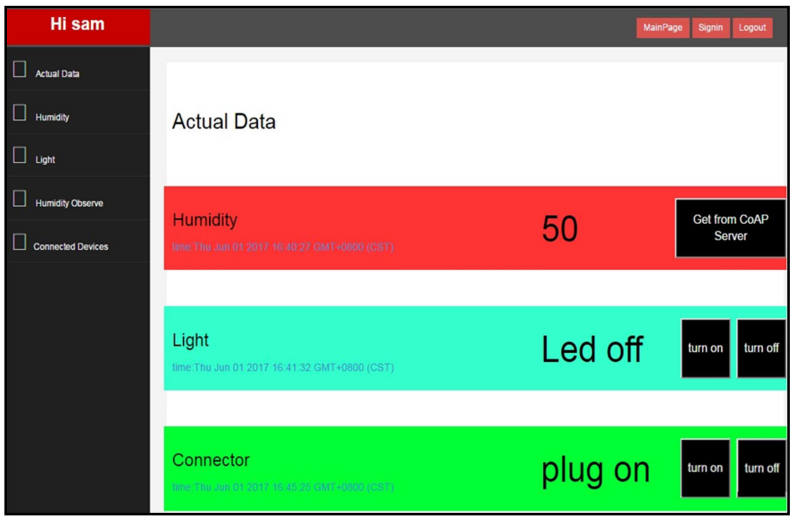

Figure 15: Web interface connected to the web server at $6LBR_1$

Figure 16: Web interface for observing humidity resource in BLE subnet 1

For viewing the historical humidity data in BLE subnet 1, we take advantage of the *xively* IoT cloud platform. When we add a new IoT device into the platform, the system will automatically generate a *Feed ID* and an *API key* for our gateway ($6LBR_1$) to upload the sensor data securely. The gateway is programed to use the built-in *crontab* to issue regular CoAP requests to the CoAP server. Whenever a new piece of humidity data is received at $6LBR_1$, the data is encapsulated with JSON format and

sent to the cloud platform. Figure 17 shows the web interface of the xively IoT cloud platform, where the historical humidity data is plotted as a line chart.

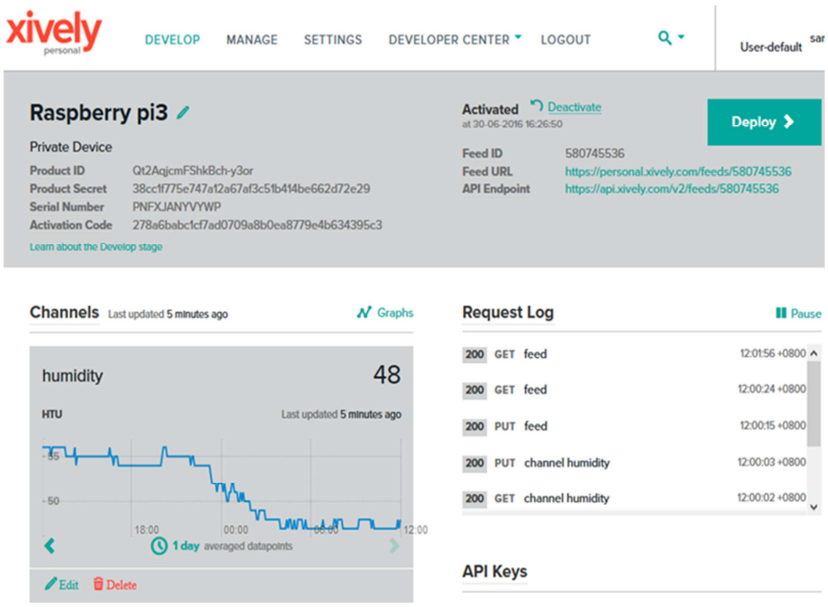

Figure 17: xively cloud platform showing historical humidity data in BLE subnet 1

For the sensor data in BLE subnet 2, since we use the publish/subscribe application architecture, there is no need for end users to make a direct connection into the 6LNs in BLE subnet 2. They simply use ordinary web browsers to connect to the web server at PC2. In BLE subnet 2, we have humidity, temperature, and illumination sensors. The web server acting as a subscriber subscribes to all the three topics, and shows real-time data on the web page as shown in Figure 18. Here the units for the brightness, humidity, and temperature readings are lx, %RH, and °C, respectively. Whenever a new piece of sensor data arrives at the web server, the data is then sent to the web browser by using Socket.IO, and finally the web page is updated dynamically by using jQuery. Through the web page, end users may also look up historical sensor data. Figure 19 shows the web interface that plots the line chart of the historical data. Users first select the period of time he/she would like, and then the query will be sent to the Firebase cloud database where the historical sensor data is stored. Upon receiving the sensor data at the web server, the last step is to plot the line chart by using Chart.js. The green, yellow, and red lines represent illumination, temperature, and humidity values, respectively.

Figure 18: Web interface showing real-time sensor data in BLE subnet 2

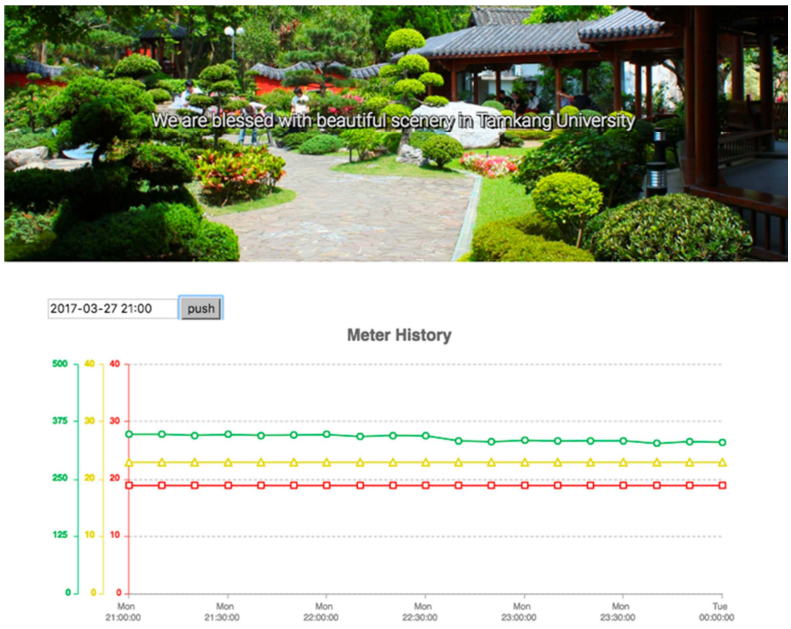

Figure 19: Web interface showing historical data in BLE subnet 2

Apart from the above web interfaces which show the functionalities of our experimental system, we also captured some BLE packets over the air and analysed the content to ensure the correctness of the operations. Specifically, we ran the *tcpdump* utility on both $6LBR_1$ and $6LBR_2$, and captured CoAP responses (Figure 20(a)) and MQTT-SN PUBLISH messages (Figure 20(b)) from the 6LNs in two BLE subnets, respectively. From both figures we can see that the two UDP/IPv6 packets are captured over the bt0 network interface. In Figure 20(a), the hexadecimal content on the row of `0x0030` include the CoAP header ("`0x6445 8f03 0000 2b1c ff`") and the payload (the last 2 bytes "`0x35`" and "`0x31`"). The first hexadecimal value of `6` is actually `0110` in binary, where the value `01` refers to the CoAP version of 1, and the value `10` means that this is an *acknowledgement* message. The hexadecimal value of the last 2 bytes are the ASCII code for characters "5" and "1", respectively. In fact, it refers to the humidity value of 51. In Figure 20(b), the hexadecimal content starting on the row of `0x0030` include the MQTT-SN header ("`0x1f0c 0400 0100 00`") and the payload (from "`0x39`" to the last byte "`0x31`"). The second byte of value `0x0c` means that this is a PUBLISH message, and the payload content is again encoded as ASCII code. For example, "`0x39`", "`0x31`", "`0x2e`", and "`0x37`" refers to characters "9", "1", ".", and "7", respectively. This is exactly the illumination value of 91.7 from the light sensor. The above descriptions are summarized in Table 1 for better readability.

```
pi@gateway:~ $ sudo tcpdump udp port 5683 -X -i bt0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on bt0, link-type LINUX_SLL (Linux cooked), capture size 262144
bytes
10:56:30.255096 IP6 2001:288:d003:a000:24a:ddff:fe42:dd51.5683 > 2001:288:
d003:a000::.32818: UDP, length 11
        0x0000:  6000 0000 0013 1140 2001 0288 d003 a000  `......@........
        0x0010:  024a ddff fe42 dd51 2001 0288 d003 a000  .J...B.Q........
        0x0020:  0000 0000 0000 0000 1633 8032 0013 39ce  .........3.2..9.
        0x0030:  6445 8f03 0000 2b1c ff35 31              dE....+..51
```

(a)

```
pi@gateway:~ $ sudo tcpdump -i bt0 -X dst host 2001:288:d003:a010::
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on bt0, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
01:44:11.828999 IP6 2001:288:d003:a010:ba27:ebff:fe2c:43a7.58111 > 2001:288:d003
:a010::.1884: UDP, length 31
        0x0000:  600a bbe6 0027 1140 2001 0288 d003 a010  `....'.@........
        0x0010:  ba27 ebff fe2c 43a7 2001 0288 d003 a010  .'...,C.........
        0x0020:  0000 0000 0000 0000 e2ff 075c 0027 93a2  ...........\.'..
        0x0030:  1f0c 0400 0100 0039 312e 372c 3230 3137  .......91.7,2017
        0x0040:  2d30 362d 3037 2030 313a 3434 3a31 31    -06-07.01:44:11
```

(b)

Figure 20: Captured CoAP request and MQTT-SN PUBLISH message at 6LBRs

Location	Content (HEX)	Interpretation
0x0030 in Figure 20(a)	6445 8f03 0000 2b1c ff35 31	1. The first hex value 6 is 0110 in binary, where 01 means CoAP version 1, and 10 means an *acknowledgement* message. 2. The last two bytes 35 and 31 are the ASCII codes for characters "5" and "1", respectively, which means the humidity value of 51 (%RH).
0x0030 & 0x0040 in Figure 20(b)	1f0c 0400 0100 0039 312e 372c 31	1. The second byte 0c means a *PUBLISH* message. 2. The four bytes 39, 31, 2e, and 37 are the ASCII codes for characters "9", "1", ".", and "7", respectively, which means the illumination value of 91.7 (lx).

Table 1: Interpretation of the captured messages shown in Figure 20

6 Conclusions and Future Work

To make IoT applications feasible and pervasive, one of the most important considerations is the low power consumption of IoT devices so they can last a long time without replacing batteries. Although there are a number of existing low-power wireless network technologies, in this research we chose BLE because it is universally supported by smartphones. Specifically, we implemented an IPv6 over BLE experimental system based on Raspberry Pi 3 and Nordic nRF51-DK development boards, by using the 6LoWPAN module and the IPv6 router advertisement daemon. With an IPv6 global address on every BLE node, they become directly accessible through the Internet. Next, we deploy two popular light-weight application-layer protocols, namely CoAP and MQTT-SN, on top of the IPv6/BLE protocol stack. With CoAP and MQTT-SN, sensor nodes are able to provide the environmental data either through the CoAP request/response interactions or the MQTT publish/subscribe model. In both application scenarios, we built a gateway that receives real-time sensor data via CoAP or MQTT protocols and pushes the data to end users with ordinary web browsers in the Internet. We also deployed xively IoT cloud platform and Google Firebase as the data store for storing historical sensor data. Finally, we used tcpdump to capture and analyse the BLE packets transmitted over the Bluetooth network interfaces to make sure that everything is working as expected.

In the near future, we would like integrate our two BLE subnets into one heterogeneous sensor network. In the heterogeneous network, sensor nodes can run CoAP or MQTT-SN freely, and the 6LBR must support both CoAP and MQTT-SN protocols simultaneously. We would also like to measure the power consumption on 6LNs and the packet delivery ratio in the 6LoWPAN network with respect to different application scenarios of different sensor data rates. Furthermore, we would like to study the feasibility of incorporating the newly published BLE mesh specification

[Bluetooth SIG, 17] into our system. With the capability of mesh networking, the deployment of BLE-based IoT systems can be more flexible.

References

[Ahmed, 17] Ahmed, S., Topalov, A., Shakev, N.: A robotized wireless sensor network based on MQTT cloud computing, In Proc. 2017 Int. Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), 2017, 1-6.

[aiocoap, 17] aiocoap, 2017, https://github.com/chrysn/aiocoap

[Bluetooth SIG, 10] Bluetooth Core Specification v4.0, Vol 3 Part C: Generic Access Profile, June 2010.

[Bluetooth SIG, 10a] Bluetooth Core Specification v4.0, Vol 3 Part F: Attribute Protocol (ATT), June 2010.

[Bluetooth SIG, 10b] Bluetooth Core Specification v4.0, Vol 3 Part G: Generic Attribute Profile (GATT), June 2010.

[Bluetooth SIG, 10c] Bluetooth Core Specification v4.0, Vol 6 Core System Package (Low Energy Controller volume), June 2010.

[Bluetooth SIG, 14] Bluetooth Internet Protocol Support Profile Specification Version 1.0.0, December 2014.

[Bluetooth SIG, 17] Bluetooth Mesh Profile v1.0, July 2017.

[BlueZ, 17] BlueZ – Official Linux Bluetooth protocol stack, 2017, http://www.bluez.org/

[Copper, 17] Copper (Cu) CoAP user-agent, 2017, https://github.com/mkovatsc/Copper

[Eclipse, 14] Really Small Message Broker, 2014, http://git.eclipse.org/c/mosquitto/org.eclipse.mosquitto.rsmb.git/

[Eclipse, 17] Eclipse Mosquitto, 2017
https://projects.eclipse.org/projects/technology.mosquitto

[Firebase, 17] Google Firebase, 2017, https://firebase.google.com/

[Fette, 11] RFC 6455: The WebSocket Protocol, December 2011, https://tools.ietf.org/html/rfc6455

[Hartke, 15] RFC 7641: Observing Resources in the Constrained Application Protocol (CoAP), September 2015, https://tools.ietf.org/html/rfc7641

[Hui, 11] RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks, September 2011, https://tools.ietf.org/html/rfc6282

[Kamma, 16] Kamma, P. K., Palla, C. R., Nelakuditi, U. R., Yarrabothu, R. S.: Design and implementation of 6LoWPAN border router, In Proc. 2016 13th Int. Conf. on Wireless and Optical Communications Networks (WOCN), 2016, 1-5.

[Kushalnagar, 07] RFC 4919: IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, August 2007, https://tools.ietf.org/html/rfc4919

[Montenegro, 07] RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks, September 2007, https://tools.ietf.org/html/rfc4944

[Narten, 07] RFC 4861: Neighbor Discovery for IP version 6 (IPv6), September 2007, https://tools.ietf.org/html/rfc4861

[Nieminen, 15] RFC 7668: IPv6 over Bluetooth Low Energy, October 2015, https://tools.ietf.org/html/rfc7668

[node-coap, 17] node-coap, 2017, https://github.com/mcollina/node-coap

[Nordic Semiconductor, 18] nRF51 IoT SDK, 2018, https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51-IoT-SDK

[Oasis, 14] MQTT version 3.1.1, 2014, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf

[radvd, 17] Linux IPv6 Router Advertisement Daemon (radvd), 2017, http://www.litech.org/radvd/

[Stanford-Clark, 13] MQTT for Sensor Networks (MQTT-SN) Protocol Specification v1.2, 2013, http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf

[Shelby, 12] RFC 6775: Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs), November 2012, https://tools.ietf.org/html/rfc6775

[Shelby, 14] RFC 7252: The Constrained Application Protocol (CoAP), June 2014, https://tools.ietf.org/html/rfc7252

[Thomson, 07] RFC 4862: IPv6 Stateless Address Autoconfiguration, September 2007, https://tools.ietf.org/html/rfc4862

[Wang, 13] Wang, H., Xi, M., Liu, J., Chen, C.: Transmitting IPv6 packets over Bluetooth low energy based on BlueZ, In Proc. 2013 15th Int. Conf. on Advanced Communications Technology (ICACT), 2013, 72-77.

[xively, 17] IoT Platform for Connected Devices, 2017, https://www.xively.com/

[Yoon, 16] Yoon, W., Kwon, K., Ha, M., Kim, D.: Transfer IPv6 packets over Bluetooth Low Energy with ensuring emergency data transmission, In Proc. 2016 6th Int. Conf. on Communications and Electronics (ICCE), 2016, 136-141.