Surfing the Internet-of-Things: Lightweight Access and Control of Wireless Sensor Networks Using Industrial Low Power Protocols

Zhengguo Sheng[1,2], Chunsheng Zhu[2,*] and Victor C. M. Leung[2]

[1]Department of Engineering and Design, University of Sussex, UK
[2]Department of Electrical and Computer Engineering, The University of British Columbia, Canada

Abstract

Internet-of-Things (IoT) is emerging to play an important role in the continued advancement of information and communication technologies. To accelerate industrial application developments, the use of web services for networking applications is seen as important in IoT communications. In this paper, we present a RESTful web service architecture for energy-constrained wireless sensor networks (WSNs) to enable remote data collection from sensor devices in WSN nodes. Specifically, we consider both IPv6 protocol support in WSN nodes as well as an integrated gateway solution to allow any Internet clients to access these nodes. We describe the implementation of a prototype system, which demonstrates the proposed RESTful approach to collect sensing data from a WSN. A performance evaluation is presented to illustrate the simplicity and efficiency of our proposed scheme.

Keywords: Internet-of-things, IPv6, RESTful, wireless sensor netorks

Received on 02 November 2014 ; accepted on 03 November 2014; published on 09 December 2014

doi: 10.4108/ inis.1.1.e2

1. Introduction

In recent years, the Internet-of-things (IoT) has emerged as an important research focus of both industry and academia. The concept of IoT can be traced back to the pioneering work done by Kevin Ashton in 1999 [1] on using radio frequency identification (RFID) tags in supply chain management. Soon after, this term became popular and is well known as a new type of communication system in which the Internet is extended to the physical world via wireless sensor networks (WSNs) [2].

With the rapid development of IoT technologies in the past few years, a wide range of intelligent and tiny sensing devices have been massively deployed in a variety of vertical applications, and several major standardization alliances or forums have emerged based on the interests of technology developments and commercial markets. Generally, sensing devices are constrained by limitations in energy resources (battery power), processing and storage capability, radio communication range and reliability, etc., and yet their deployment must satisfy the real-time nature of applications under little or no direct human interactions. Over the past decades, the research community has invested substantial efforts to develop networking systems called WSNs that meet the challenges stated above. With large-scaled deployments of WSNs and their interconnection into the global IoT, a new ecosystem supporting ubiquitous deployment of smart applications has been formed.

Technically speaking, current IoT solutions can be categorized as non-IP based or IP based solutions. Most off-the-shelf solutions belong to the former, especially those from some well-known standard alliances, such as ZigBee [3], Z-Wave [4], INSTEON [5] and WAVE2M [6]. However, most of these non-IP solutions are isolated within their own verticals, which hinder the

[★]This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), the NSERC DIVA Strategic Research Network, the ICICS/TELUS People & Planet Friendly Home Initiative at The University of British Columbia, TELUS and other industry partners.
[*]Corresponding author. Email: cszhu@ece.ubc.ca

global IoT development due to incompatibility across heterogeneous communication systems.

Motivated by the fact that the Transmission Control Protocol (TCP)/Internet Protocol (IP) suite is the de-facto standard for computer communications in today's networked world, IP based solutions could be the future for networks that form the IoT [7]. In order to tackle the technical challenges, such as extensive protocol overheads against memory and computational limitations of sensor devices, the Internet Engineering Task Force (IETF) has taken the lead to develop and standardize communication protocols for resource constrained devices, including Routing Protocol for Low Power and Lossy Networks (RPL) [8], and Constrained Application Protocol (CoAP) [9]. Besides, the IP Smart Object Alliance (IPSO) [10] also actively promotes the use of IP version 6 (IPv6) embedded devices for machine-to-machine (M2M) applications. Although it is still in its early stage to be commercialized, there are already a substantial number of IP-based WSN solutions supported by growing availability of products and systems.

To promote organic-growth of IoT systems, open technologies are preferred for integration into IoT, and IPv6-based solutions are promising. In order to well-maintain sensor devices as well as facilitate the efficient development of IoT applications, e.g., to monitor the performance of sensor devices and send commands to sensor nodes, trusted-entities in an IoT system should be provided with a reliable and efficient way to remotely monitor and control WSNs without consuming significant resources. We take an approach based on the Representational State Transfer (REST) paradigm [11] whereby a lightweight web server can be embedded in resource constrained sensor devices. In essence, not only can the proposed method integrate IoT devices into the network, but also connect them to the "web".

The following summarizes our contributions and key results:

- We present an implementation of the full IPv6 protocol stack on WSN nodes to enable wireless connectivity among sensor devices. Specifically, the 6LowPAN/IPv6/RPL/UDP/CoAP protocol stack has been deployed on WSN nodes employing the IEEE 802.15.4 radio platform.

- We integrate IEEE 802.15.4 connectivity into an open-platform gateway and implement the Hypertext Transfer Protocol (HTTP)-CoAP proxy using OpenWrt, an open-source operating system based on Linux in the gateway to realize remote access from any IP terminal to IPv6 sensor devices.

- We propose two alternative access methods to enable REST based applications with sensor devices. In the direct access method, the user can directly visit any sensor devices by sending CoAP request, whereas for the proxy access method, the user can use the normal HTTP requests to access sensor devices, but the gateway needs to help convert the HTTP requests to CoAP requests and vise versa.

The remainder of this paper is organized as follows. A survey of related works is provided in Section 2. The implementation of the RESTful protocol stack in WSNs is introduced and analyzed in Section 3. The prototype implementation of the remote access schemes is presented in Section 4 and performance evaluation results are shown in Section 5. Finally, concluding remarks are given in Section 6.

2. Related Works

Recent technology trends in the Web Services (WS) are primarily focused on two different architectures, namely Big WS (or WS-*) and RESTful WS. Cesare *et al.* in [12] compare these two architectures and argue that the RESTful WS can create a loosely coupled system that is better suited for simple and flexible integration scenarios, whereas WS-* can provide more advanced quality-of-service support for enterprise-class applications.

Many recent works are dedicated to the development of REST-style IoT systems to enable easy access from application servers to wireless sensor devices, since the REST-style device would not require any additional application programming interface (API) or descriptions of resources/functions. REST is a general architectural design style for developing lightweight WS to access resources over the Internet using standard protocols. It provides a design concept that all the objects in the Internet are abstracted as resources. Each resource corresponds to a unique identity. Through a general interface, all the operations on a resource do not change its identity as they are stateless. REST-style can make applications as sharable, reusable and loose coupling services. The uniform operation and interaction mechanisms on resources can help developers or decision makers to quickly react to market changes.

Weijun *et al.* in [13] propose an adaptation layer to integrate RESTful WS infrastructures, which can enable connectivity of embedded devices with mobile Internet applications. Vlad in [14] proposes a resource discovery mechanism based on RESTful principles, which enables a plug and play experience in web of things. Dominique *et al.* in [15] and [16] also propose a RESTful mechanism to integrate wireless energy monitors with application servers to build mashup applications. However, most of the embedded devices considered in the above works are not IP based, which means that a multiprotocol translation gateway is needed. As discussed in [2], network protocol translation can bring more complexity than just a packet format conversion, which usually involves semantics translations between different mechanisms and logics for routing, quality of service, security, etc.

There are some recent papers focusing on the implementation of IPv6 protocol stacks on various hardware platforms. Thomas *et al.* in [17] demonstrate an intelligent container testbed in which CoAP is implemented on the embedded operating system TinyOS [18]. Moreover, a couple of other implementations of CoAP are also available on the Contiki platform [19]-[21]. However, most of these implementations are only for the purpose of connectivity evaluations on different operation platforms and usually assume that a virtual gateway, which is usually a IEEE 802.15.4 USB dongle connected to a personal computer (PC), is mounted as a root node to collect upstream packets from leaf nodes.

Different to the above works, our contribution in this paper is that we consider both IPv6 protocol implementation on sensor devices as well as an integrated gateway solution to allow any normal Internet device (e.g., PC and smart phone) to access an IPv6 sensor device. Specifically, we integrate real-world things into the existing web by turning real objects into RESTful resources that can be retrieved directly using HTTP.

3. A RESTful Protocol Stack for WSN

We employ the IPv6 based protocol stack for WSNs. Some protocols within this stack, which have been developed for resource constrained networks, are introduced as follows.

3.1. 6LoWPAN

From the very beginning, IPv6 has been selected by IETF as the only choice to support wireless communications in IoT. Its key features such as universality, extensibility and stability, etc., have been designed to overcome many known problems in the existing version of IP, i.e., IP version 4, and therefore IPv6 is expected to be widely adopted for the future Internet. To develop a standard that enables IP connectivity in resource constrained WSNs, the 6LoWPAN working group [22] was established to work on protocol optimization of IPv6 over networks built on top of IEEE 802.15.4 [23]. Specifically, the 6LoWPAN protocol considers how to integrate IPv6 with the medium access control (MAC) and physical (PHY) layers of IEEE 802.15.4.

In fact, there are two key challenges to run IPv6 over the IEEE 802.15.4 network. On the one hand,

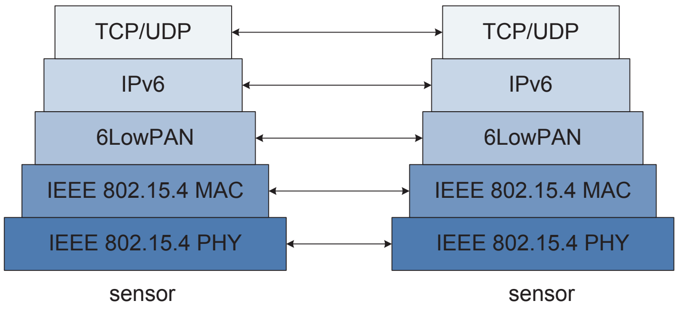


Figure 1. The position of 6LoWPAN in the IPv6 protocol stack

the maximum frame size supported by IEEE 802.15.4 is only 127 Bytes. Considering that significant header overheads are occupied by layered protocols (e.g., MAC layer header, IPv6 header, security header and transmission layer), the payload size available for the application layer is very limited. On the other hand, since the minimum value of maximum transmission unit (MTU) specified by IPv6 is 1280 Bytes (RFC 2460), if MTU supported by the under layer (i.e., IEEE 802.15.4) is smaller than this value, the data link layer must fragment and reassemble data packets. In order to address these issues, 6LoWPAN incorporates an adaptation layer right above the data link layer to fragment large IPv6 packets into small pieces required by the under layer and reassemble them at the receiving end. Moreover, 6LoWPAN specifies stateless compression methods for IP header in order to reduce the overhead of IPv6. The position of 6LoWPAN in the IPv6 protocol stack is shown in Figure 1.

Note that the fundamental purpose of header compression methods is to remove the redundant information from the header by using compression encoding schemes. Although the IPv6 header takes 40 Bytes, most of information bits can be compressed in the link layer. The compression methods for each field of IPv6 header are as follows:

1. Version (4 bits): The value is 6. It can be omitted in the IPv6 network.

2. Traffic Class (8 bits): It can be compressed by compression encoding methods.

3. Flow label (20 bits): It can be compressed by compression encoding methods.

4. Payload Length (16 bits): It can be omitted because the length of IP header can be obtained through the Payload Length field in the MAC header.

5. Next Header (8 bits): It can be compressed by compression encoding methods if the next header is assumed to be one of UDP, ICMP, TCP or extended header.

6. Hop Limit (8 bits): This is the only field that cannot be compressed.

7. Source Address (128 bits): It can be compressed by omitting the prefix or Interface Identifier (IID).

8. Destination Address (128 bits): It can be compressed by omitting the prefix or IID.

In order to implement the stateless compression on IPv6 header, the 6LoWPAN working group has specified two compression algorithms: LOWPAN_HC1 (RFC4944) [24] and LOWPAN_IPHC (RFC6282) [25]. HC1 algorithm is applicable to networks using link-local addresses. The prefix of a node's IPv6 address is fixed as FE80::/10 and IDD can be obtained via the MAC address. Since this algorithm cannot efficiently compress global/routable addresses or broadcast addresses, it cannot be used to connect a 6LoWPAN with the Internet. LOWPAN_IPHC, however, is proposed to improve the efficiency of compressing routable addresses.

Both LOWPAN_HC1 and LOWPAN_IPHC define an 8-bit dispatch field after the MAC header. Its possible values as shown in Table I determine the specific format of the type-specific header and algorithm. For example, if the first 8 bits is 01000010, the following filed is the header corresponding to the LOWPAN_HC1 algorithm; if the first 3 bits is 011, the following field is the header corresponding to the LOWPAN_IPHC algorithm.

Table 1. 6LoWPAN dispatch field

Type	Header type
00 xxxxxx	NALP - Not a LoWPAN frame
01 000001	IPv6 - Uncompressed IPv6 Addresses
01 000010	LOWPAN_HC1 - LOWPAN_HC1 compressed IPv6
...	Reserved
01 010000	LOWPAN_BC0 - LOWPAN_BC0 broadcast
...	Reserved
01 XXXXXX	IPv6 header compressed by LOWPAN_IPHC
01 000000	ESC-There are others subsequent header
10 xxxxxx	MESH - Mesh Header
11 000xxx	FRAG1- Fragmentation Header (first)
11 100xxx	FRAGN - Fragmentation Header (subsequent)

The dispatch field is immediately followed by the type-specific header, which consists of some indicating bits. The indicating bits indicate specific compression schemes for IPv6. Readers can refer to RFC4944 for more details.

In addition to stateless IPv6 header compression, 6LoWPAN also includes other relevant standards including schemes supporting mesh routing, simplified IPv6 neighbour discovery protocol, use cases and routing requirements. In summary, the 6LoWPAN working group provides the fundamental of IETF on IoT communications.

3.2. RPL

IETF Routing over Lossy and Low-power Networks working group (RoLL) was established in February 2008. It focuses on routing protocol design and is committed to standardize the IPv6 routing protocol for lossy and low power networks (LLN). Its tasks start with the routing requirements of various application scenarios. So far, the routing requirements of four application scenarios have been standardized, i.e., Home Automation (RFC5826), Industrial Control (RFC5673), Urban Environment (RFC5548) and Building Automation (RFC 5867).

In order to develop suitable standards for LLN, RoLL first provides an overview of existing routing protocols for wireless sensor networks. The literature [26] analyzes the characteristics and shortcomings of the relevant standards and then discusses the quantitative metrics for constructing routing in the routing protocol. RFC6551 [27] introduces two kinds of quantitative metric: node metrics including node state, node energy and hop count, and link metrics including throughput, latency, link reliability, expected transmission count (ETC) and link colour object. In order to assist dynamic routing, nodes can incorporate objective functions to determine the rule for path selection based on the quantitative metrics.

Based on the results of routing requirements and quantitative static link metrics, RoLL has developed a routing protocol for LLN (RPL) as specified in RFC6550 [28]. RPL supports three kinds of traffic flows including point-to-point (between devices inside the LLN), point-to-multipoint (from a central control point to a subset of devices inside the LLN) and multipoint-to-point (from devices inside the LLN towards a central control point). RPL is a distance-vector routing protocol, in which nodes construct a Directed Acyclic Graph (DAG) by exchanging distance vectors. Through broadcasting routing constraints, the DAG root node (i.e., central control point) filters out the nodes that do not meet the constraints and select the optimum paths according to the metrics.

3.3. CoAP

CoAP, as specified by the IETF Constrained RESTful Environments working group (CoRE) [9], is a specialized web transfer protocol for resource constrained nodes and networks. CoAP conforms to the REST

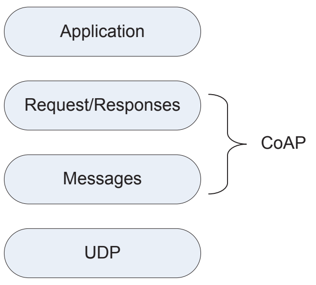


Figure 2. CoAP protocol stack

style. It abstracts all the objects in the network as resources. Each resource corresponds to a unique Universal Resource Identifier (URI), based on which the resources can be operated upon in a stateless manner using commands including GET, PUT, POST, DELETE and so on.

Strictly speaking, CoAP is not a HTTP compression protocol. On the one hand, CoAP realizes a subset of HTTP functions and is optimized for constrained environments. On the other hand, it offers features such as built-in resource discovery, multicast support and asynchronous message exchanges.

Unlike HTTP, CoAP utilizes a datagram-oriented transport protocol underneath, such as UDP. In order to ensure reliable transmissions over UDP, CoAP introduces a two-layer structure as shown in Figure 2. The messaging sublayer is used to deal with asynchronous interactions using UDP. Specifically, there are 4 kinds of CoAP messages:

1. Confirmable (CON): ACK is needed.
2. Non-confirmable (NON): ACK is not needed.
3. Acknowledgment (ACK): To represent that a Confirmable message is received.
4. Reset (RST): To represent that a Confirmable message is received but can't be processed.

The Request/Response interaction sublayer is used to transmit resource operation requests and the request/response data. As a summary, CoAP has the following features:

- Constrained web protocol fulfilling M2M requirements.
- Asynchronous message exchanges.
- Low header overhead and parsing complexity.
- URI and Content-type support.
- Simple proxy and caching capabilities.
- Built-in resource discovery.

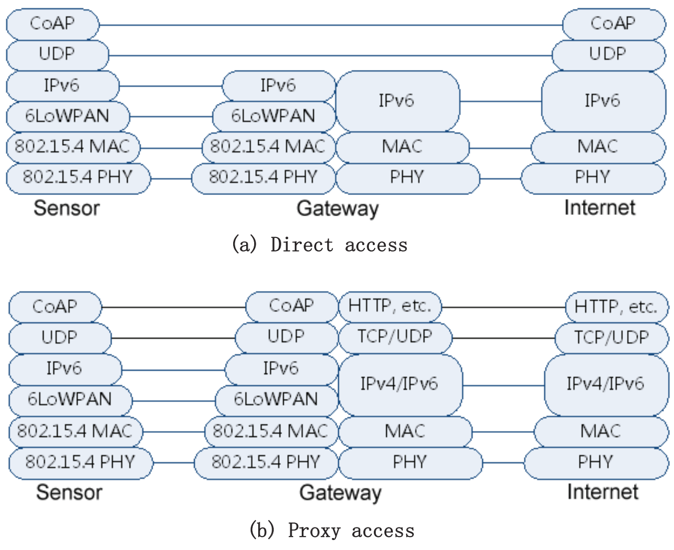


Figure 3. Direct access vs. Proxy access

- UDP binding with optional reliability supporting unicast and multicast requests.
- A stateless HTTP-CoAP mapping, allowing a proxy to provide access to CoAP resources via HTTP in a uniform way and vice versa.

3.4. HTTP–CoAP protocol implementation

Applying REST-style network structure in WSN can largely facilitate connection between WSN and the Internet. By applying CoAP protocol on wireless sensors devices, Internet services can access WSNs as resources directly or via gateway as a proxy. Basically, there are two methods to enable remote access from an Internet client to a sensor device.

Direct access. Direct access means that the an Internet user accesses a WSN through a gateway that only implements protocol conversions between the IPv6 network layer and 6LoWPAN, but does not process the upper layers protocols (e.g., CoAP). As an example shown in Figure 3 (a), a sensor node in WSN can be accessed through an IPv6 address and the gateway only needs to implement conversion between IPv6 and 6LoWPAN, which significantly reduces the processing overhead.

Proxy access. Proxy access means that an Internet user accesses a WSN through a proxy that can convert an incompatible data format from outside networks into a WSN compatible data format. For example, in our case, the proxy can have functions of protocol conversion from a HTTP request to a CoAP request, and vice versa, payload conversion and blockwise segmentation of large data packets (e.g., those representing an image), etc.

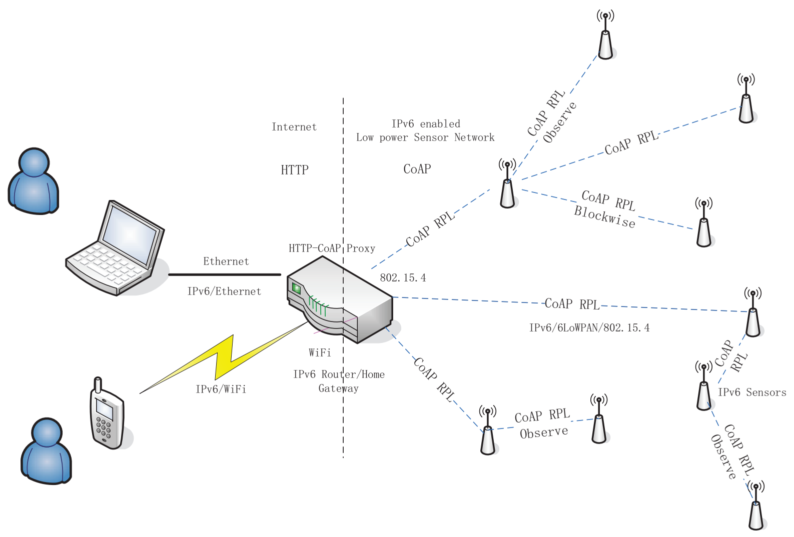


Figure 4. System Architecture

The advantage of this method is that current Internet services can easily access WSN resources without any changes, because of the existence of the proxy gateway. Moreover, since low power sensor nodes cannot support TCP efficiently, the proxy mechanism can buffer and process the requests to avoid TCP time out. However, the protocol conversion increases the complexity of the gateway and thereafter affects communication efficiency. Figure 3 (b) illustrates the protocol conversion between HTTP and CoAP via a gateway.

4. Prototype Implementation

In this section, we present our prototype to illustrate the implementation of the RESTful access methods to IPv6 wireless sensor devices, considered as the representative of future embedded devices in IoT. A RESTful gateway supports both IEEE 802.11 Wi-Fi and IEEE 802.15.4 interfaces for communications. The web resources in sensor devices are accessible through the RESTful APIs. The system architecture is shown in Figure 4, where a PC acts as a client to retrieve sensor resources via the RESTful gateway.

4.1. Sensor node

We deploy wireless sensor devices to monitor air temperature and humidity, detect movements and take photos. All these sensors are equipped with the same ATmega1284P MCU and AT86RF231 radio transceiver to support 250kbps data transmissions at 2.4GHz using the IEEE 802.15.4 protocol. To support IPv6 connectivity, all the sensor devices run the Contiki v2.6 operating system and incorporate 6LowPAN, IPv6 and RPL protocols on top of IEEE 802.15.4. The web service running on the sensor devices relies on the application protocol CoAP. A snapshot of the sensor platform is illustrated in Figure 5 and the detailed technical specifications are shown in Table II.

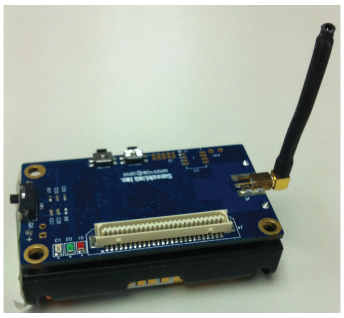


Figure 5. A snapshot of sensor platform

4.2. RESTfull Gateway

To ease access from Internet applications to sensor resources, especially for those Internet users without CoAP support, we integrate IEEE 802.15.4 connectivity into an open-platform gateway and port the HTTP-CoAP proxy implementation to the OpenWrt, the operation system of the gateway, to realize remote access from an ordinary IP terminal to an IPv6 sensor device. Figure 6 gives the hardware architecture of the RESTful gateway, which technical specifications are provided in Table III.

The HTTP-CoAP (HC) proxy provides translation and mapping between HTTP and CoAP protocol. CoAP can be directly mapped to HTTP, because CoAP actually implements a subset of HTTP functions. The mapping is performed only at the Request/Response interaction sublayer of the CoAP protocol and is invisible to the

Table 2. Technical specifications of sensor device

	Parameters	Note
CPU Performance		
Internal storage	128KB	
External storage	16KB	
EEPROM	4KB	
Serial communication	UART / USART	TTL Transmission Level
A/D converter	10-bit ADC	8 channels, 0-3V input
Other Interfaces	Digital I/O, I2C,SPI	
Maximum Current	18mA	Work mode
	2uA	Sleep mode
RF transceiver		
Frequency band	2400-2485MHz	ISM global free band
Data rate	250Kbps/ 1000Kbps/ 2000Kbps	
RF power	3.2 dBm	
Receiving sensitivity	-104 dBm	
Adjacent Channel Suppression	36 dBc	+5M Channel bandwidth
	34 dBc	-5M Channel bandwidth
Outdoor transmission	$\geq$ 300m	
Indoor transmission	$\geq$ 10m	
Maximum Current	12mA	Receiving Mode
	14mA	Tx -3dBm
Extended interface	51 pins	

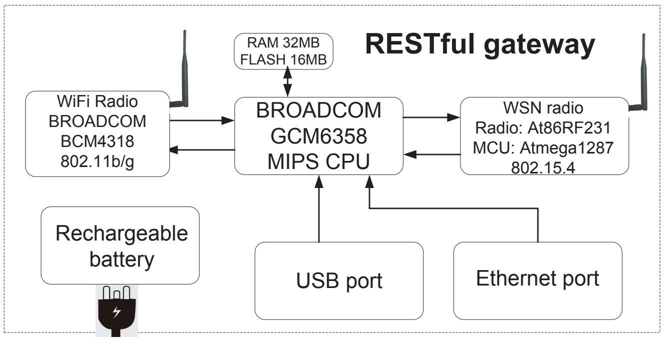


Figure 6. Hardware architecture of gateway

Table 3. Technical specifications of gateway

	Parameters	Note
CPU frequency	300MHz	
RAM	32MB	
Flash	16MB	
Serial communication	UART / USART	TTL Transmission Level
A/D converter	10-bit ADC	8 channels, 0-3V input
USB HOST	2	
RJ45	4	
WiFi	1	IEEE 802.11abg
OS	OpenWrt	v12.09-beta2
Protocol	IPv6, IPv4	

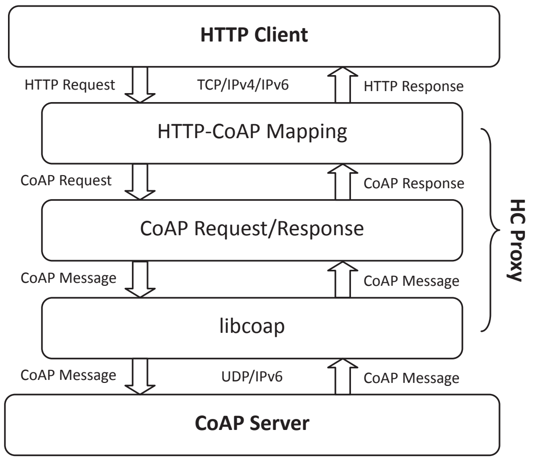


Figure 7. Interaction process of HC proxy

messaging sublayer. There are two kinds of mapping: CoAP-to-HTTP and HTTP-to-CoAP. In our case, we only realize HTTP-to-CoAP mapping, which is implemented by specifying CoAP-URI as the request address for transmitting HTTP request to the HTTP-CoAP proxy. Note that compared to CoAP-to-HTTP mapping, HTTP-to-CoAP mapping is more complex since it is necessary to determine whether to ignore the content or report an error by checking unsupported HTTP request methods, response codes, content-types and options.

In our prototype gateway, the HC proxy is implemented based on libcoap [29] which is an open-source C-Implementation of CoAP and conforms to GPL v2 or higher licenses.

The interaction process of the HC proxy is shown in Figure 7. Specifically, for each of the HC proxy layers, we have the following implementations:

libcoap layer. libcoap implements the CoAP messaging sublayer based on UDP. It defines CoAP message structure and methods to operate CoAP messages.

CoAP Request/Response layer. The CoAP Request/Response layer implements the function of the Request/Response interaction sublayer in Figure 2 by encapsulating the data structure and methods relevant to CoAP Requests and Responses. It is responsible to transmit CoAP requests in the form of CoAP messages through the messaging sublayer and generate CoAP response based on received CoAP messages. Because CoAP messaging sublayer adopts unreliable UDP, certain issues need to be solved in order to implement a reliable transmission, including CoAP message acknowledgement, message retransmission for timeout, message process, asynchronous message process and segmented message process, etc. The following code header is provided to illustrate the implementation of CoAP Request/Response.

```
/* request.h */

typedef struct {
  unsigned char msgtype;
  method_t method;
  coap_list_t *optlist;
  str proxy;
  unsigned short proxy_port;
  str payload;
  int ready;
  char lport_str[NI_MAXSERV]
  coap_uri_t uri;
  int flags;
  coap_block_t block;
  unsigned int wait_seconds;     /*
      default timeout in seconds */
  coap_tick_t max_wait;          /*
      global timeout (changed by
      set_timeout()) */
  unsigned int obs_seconds;      /*
      default observe time */
  coap_tick_t obs_wait; /* timeout for
      current subscription */
} coap_request_t;

void coap_request_method(coap_request_t
    *request, char *arg);
void coap_request_uri(coap_request_t *
    request, char *arg);
int coap_request_proxy(coap_request_t *
    request, char *arg);
void coap_option_content_type(
    coap_request_t *request, char *arg,
    unsigned short key);
int coap_option_blocksize(coap_request_t
    *request, char *arg);
void coap_option_subscribe(
    coap_request_t *request, char *arg);
void coap_option_token(coap_request_t *
    request, char *arg);
int coap_send_request(coap_request_t *
    request, void *context);

void coap_init_request(coap_request_t *
    request);
void coap_register_request_handler(int
    (*handler) (coap_pdu_t *pdu, void *
    context));
void coap_register_request_data_handler(
    int (*handler) (const unsigned char
    *data, size_t len, void *context));
```

HTTP-CoAP mapping layer. This layer implements mappings from HTTP requests to CoAP requests and vice versa. When converting a HTTP request to a CoAP request, the HC proxy needs to convert the HTTP request method, URI, header/option and payload, respectively. If a proxy encounters an error, it has to generate the corresponding error response. The C function defined for handing the HTTP-CoAP mapping is also provides as follows.

```
int coap_response_map_code(int code);
char* coap_response_map_content_type(int
     content_type);

BOOL coap_proxy_handler(SOCKET
    localwebuser, char *szLineBuffer,
    int nLineBuffer)
```

5. Performance Evaluations

In this section, we provide evaluation results of the prototype system. Especially, we experimentally evaluate the performance over the prototype system at two layers: the routing layer where the round trip times (RTTs) and packet loss rates of multi-hop transmissions in the WSN are measured and the application layer where web resources of sensor devices are retrieved using RESTful methods.

5.1. System configuration

Our prototype system is composed of three different sensor devices, one HC proxy gateway and one PC for the tests. In order to ease the setup of WSN in a multi-hop fashion, we manually assign IPv6 addresses for the sensor devices as follows:

Camera sensor	2001:2::19
Humidity & temperature sensor	2001:2::14
Approach detecting sensor	2001:2::16

We deploy the prototype system in an open office area. The HC proxy gateway and sensor devices are connected wirelessly via IEEE 802.15.4 over channel 26. The PC client is connected to the gateway through the Wi-Fi link. The network topology is built with a maximum number of 2 hops, where the camera

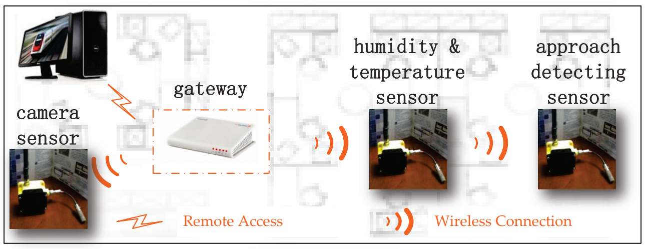
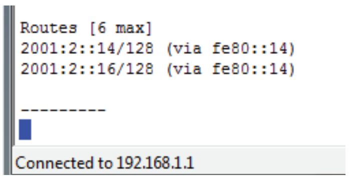


Figure 8. Network topology of prototype system

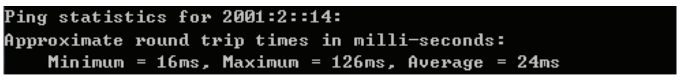


```
Routes [6 max]
2001:2::14/128 (via fe80::14)
2001:2::16/128 (via fe80::14)

---------
```

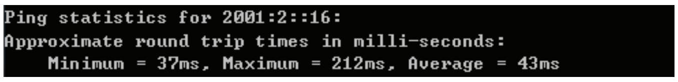
Connected to 192.168.1.1

(a) Routing table



```
Ping statistics for 2001:2::14:
Approximate round trip times in milli-seconds:
    Minimum = 16ms, Maximum = 126ms, Average = 24ms
```

(b) RRTs from one-hop sensor device



```
Ping statistics for 2001:2::16:
Approximate round trip times in milli-seconds:
    Minimum = 37ms, Maximum = 212ms, Average = 43ms
```

(c) RRTs from two-hop sensor device

Figure 9. Routing table and RTTs evaluations

sensor and humidity&temperature sensor are directly connected to the gateway over single hops, and the approach detecting sensor is the leaf node of the humidity&temperature sensor and it is two hops away from the gateway. Figure 8 provides the network topology of the prototype system.

5.2. RTTs and Packet loss evaluations of RPL routing

Wireless sensor networks should be capable of forming multi-hop transmissions among peer sensor devices. In this evaluation, the RTTs and packet loss rate in a single-hop and multi-hop scenarios using RPL routing are measured. After setting up of the system, we use the simple ping commands to evaluate the RTTs from the PC client to the humidity&temperature sensor and approach detecting sensor, respectively. The payload size for each transmission packet is 32 bytes and the RTTs results are averaged over 100 measurements. Figure 9 (a) shows the routing table via the secure shell client. As can be seen from Figure 9 (b), for one-hop transmissions, the average RTTs is 24ms. When the routing extends to two hops, the results as shown in Figure 9 (c) are degraded to 43ms average RTTs.

To further evaluate the performance of a large scale network, we set up another test to evaluate the packet loss rate in a multi-hop environment. The test is carried out in an open office area with strong Wi-Fi background noise and lowest possible WSN radio frequency output power to ensure a multi-hop fashion, which makes a sensor device can only communicate to each other within around 30 cm.

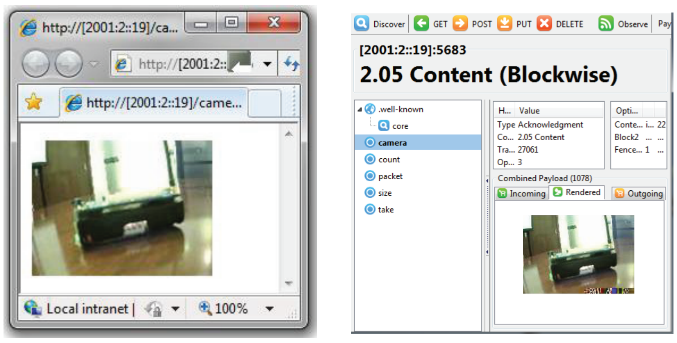
A maximum number of 6 hops can be obtained by optimizing the communication system. To retrieve the onboard resources via GET request (i.e., < /.well-known/core >) over the same number of measurements, Table 4 shows the packet loss rate in a multi-hop scenario. We can observe that the packet loss rate increases dramatically with an increasing number of hops, because of severe environmental interference and channel congestions, etc. Moreover, additional configurations to ensure a multi-hop transmission, such as one way communication, low output power and RPL settings, also contribute to the high loss.

Table 4. Packet loss rate in a multiple-hop network

	Hop 2	Hop 3	Hop 4	Hop 5	Hop 6
Received	2020	1704	1173	1112	944
Lost	161	474	1003	1068	1234
Packet loss rate	7.38%	21.76%	46.09%	48.9%	56.65%

5.3. RESTful method to retrieve sensor resources

To illustrate IoT applications, we initiate a trial to ’GET’ an image from the camera sensor device. Specifically, we use both proxy access and direct access methods to retrieve the sensor data via the gateway. Figure 10 (a) shows the proxy access result by sending a HTTP GET request along with the URI `http://[2001:2::19]/camera`. The HC proxy then converts the HTTP request to CoAP request and forwards the request to the camera sensor. As a comparison, Figure 10 (b) shows the direct access result by sending a CoAP request `coap://[2001:2::19]:5683/camera` directly from the CoAP browser [30] on the PC. Since the picture takes about 27 kBytes, which exceeds the payload size defined by the CoAP client, the CoAP protocol adopts the blockwise transfer by dividing the response into 64-Byte blocks in such a way that the web server can handle each block transfer separately, with no need for a connection setup or other server-side memory of previous block transfers. In summary, both methods show an acceptable performance.



(a) Proxy access using HTTP (b) Direct access using CoAP

Figure 10. HTTP vs. CoAP methods

6. Conclusion

We have implemented the 6LowPAN/IPv6/RPL/CoAP protocol stack on an IEEE 802.15.4 radio platform to enable wireless sensor communications. Furthermore, by integrating IEEE 802.15.4 connectivity and HTTP-CoAP proxy into an open-platform gateway, we have realized remote access from any IP node to IPv6 sensor devices. We have presented performance evaluations, which have shown that the IP based solution is promising to drive IoT development. In the future work, we plan to design a more robust and reliable IP solution for IoT. Especially, how to deploy large scale networks with decent performance is a critical issue and we need to continue to optimize both hardware and software implementations. Moreover, other issues, such as device management and control of sensor devices, can also be explored via RESTful methods.

References

[1] Ashton K. (2009) *That 'internet of things' thing*, RFID Journal.

[2] Vasseur J.-P. and Dunkels A. (2010) *Interconnecting smart objects with ip: The next internet*, Morgan Kaufmann.

[3] ZigBee Alliance (2007) *Zigbee home automation public application profile*, IEEE J. Select. Areas Commun..

[4] Z-Wave (2007) *Z-wave protocol overview*.

[5] Darbee P. (2005) *Insteon: The details*.

[6] Garcia-Hernando A. et al. (2008) *Problem solving for wireless sensor networks*, Springer.

[7] Sheng Z. and Yang S. and Yu Y. and Vasilakos A. and McCann J. and Leung K. (2013) *A survey on the IETF protocol suite for the internet of things: standards, challenges, and opportunities* in IEEE Wireless Communications Magazine, vol. 20, no. 6, pp. 91-98.

[8] IETF *Routing Over Low power and Lossy networks (roll)*, Available at: http://datatracker.ietf.org/wg/roll/charter.

[9] IETF *Constrained RESTful Environments (core)*, Available at: http://datatracker.ietf.org/wg/core/charter.

[10] IP Smart Object Alliance (IPSO) *Available at: http://www.ipso-alliance.org*.

[11] Fielding R. T. (2000) *Architectural styles and the design of network-based software architectures*, Doctoral dissertation, University of California, Irvine, USA.

[12] Pautasso C. and Zimmermann O. and Leymann F. (2008) *Restful web services vs. "big" web services: Making the right architectural decision*, in Proc. 17th International Conference on World Wide Web (WWW), pp. 805-814.

[13] Qin W. and Li Q. and Sun L. and Zhu H. and Liu Y. (2011) *Restthing: A restful web service infrastructure for mash-up physical and web resources* in Proc. IFIP 9th International Conference on Embedded and Ubiquitous Computing (EUC), pp. 197-204.

[14] Stirbu V. (2008) *Towards a restful plug and play experience in the web of things* in Proc. IEEE International Conference on Semantic Computing (ICSC), pp. 512-517.

[15] Guinard D. (2009) *Towards the web of things: Web mashups for embedded devices* in Proc. Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009) in conjunction with 18th International Conference on World Wide Web (WWW).

[16] Guinard D. and Trifa V. and Wilde E. (2010) *A resource oriented architecture for the web of things* in Proc. Internet of Things (IOT), pp. 1-8.

[17] Potsch T. and Kuladinithi K. and Becker M. and Trenkamp P. and Goerg C. (2012) *Performance evaluation of coap using rpl and lpl in tinyos* in Proc. 5th International Conference on New Technologies, Mobility and Security (NTMS), pp. 1-5.

[18] Iyengar S. S. and Parameshwaran N. and Phoha V. V. and Balakrishnan N. and Okoye C. D. (2011) *Fundamentals of sensor network programming: Applications and technology*, Wiley-IEEE Press.

[19] Dunkels A. and Gronvall B. and Voigt T. (2004) *Contiki - a lightweight and flexible operating system for tiny networked sensors* in Proc. 29th Annual IEEE International Conference on Local Computer Networks (LCN), pp. 455-462.

[20] Kovatsch M. and Duquennoy S. and Dunkels A. (2011) *A low-power coap for contiki* in Proc. IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 855-860.

[21] Schnwlder J. and Tsou T. and Sarikaya B. (2011) *Protocol profiles for constrained devices* Available at: www.iab.org/wp-content/IAB-uploads/.../Schoenwaelder.pdf.

[22] IETF *IPv6 over Low power WPAN (6lowpan)*, Available at: http://datatracker.ietf.org/wg/6lowpan/charter.

[23] IEEE Computer Society (2003) *Ieee std. 802.15.4-2003*.

[24] Montenegro G. and Kushalnagar N. and Hui J. and Culler D. *Transmission of ipv6 packets over ieee 802.15.4 networks*, RFC4944, available at: https://datatracker.ietf.org/doc/rfc4944/.

[25] Hui J. and Thubert P. *Compression format for ipv6 datagrams over ieee 802.15.4-based networks*, RFC6282, available at: http://datatracker.ietf.org/doc/rfc6282/.

[26] Levisi P. and Tavakoli A. and *Dawson-Haggerty S. Overview of existing routing protocols for low power and lossy networks*, Internet-Draft, available at: http://tools.ietf.org/html/draft-ietf-roll-protocols-survey-07.

[27] Vasseuri J. and Kim M. and Pister K. and Dejean N. and Barthel D. *Routing metrics used for path calculation in low-power and lossy networks*, RFC 6551, available at: http://datatracker.ietf.org/doc/rfc6551/.

[28] Winter T. and Thubert P. and Brandt A. and Hui J. and Kelsey R. and Levis P. and Pister K. and Struik R. and Vasseur J. and Alexander R. *Rpl: Ipv6 routing protocol for low-power and lossy networks*, RFC 6550, available at: http://datatracker.ietf.org/doc/rfc6550/.

[29] Bergmann O. *libcoap: C-implementation of coap*, Available at: http://libcoap.sourceforge.net.

[30] Copper (Cu) CoAP Browser *A firefox add-on to browse the internet of things*, Available at: https://github.com/mkovatsc/Copper.