On BeDIS Gateways and Zigbee Networks

Overview on functionalities, preliminary design and resources

Revision date	Contents	Contact
June 18, 2017	▪ Overview o Structure of smart gateway o Preliminaries on setup and initialization and self test ▪ Links to resources: tutorials, specifications, sample code on: (They are no longer so relevant, December 2017) o MQTT, MQTT-SN o Related topics, e.g., REST architecture and APIs	janeliu@iis.sinica.edu.tw
Dec 28, 2017	▪ Overview updated o NSI (Network setup and initialization) o BHM (Beacon Health Monitor) o Hardware configuration ▪ Pseudo code description of software structure of gateway and NSI	janeliu@iis.sinica.edu.tw

Overview

This note first presents a high-level description of smart gateways and zigbee networks of location beacons (Lbeacons) in BeDIS (Building- environment Data and Information System. It then describes the Network Setup and Initialization (NSI) and Beacon Health Monitor (BHM) components of gateways. NSI should be among the first components to be designed and implemented. The note presents a pseudo-code description of the software structure of the gateway and the NSI component.

Structure of BeDIS

Fig. 1 shows the overall structure of a BeDIS: Specifically, the figure depicts the structure of a gateway and its relationship with Lbeacons, BeDIS server and sources of emergency alert messages. The pdf file titled "BeDI enabled location specificity and IPS" contains a more detailed and complete description of the system and its components. You can find the file at OpenISDM homepage under documents-> Publications-> Journal/conference papers.

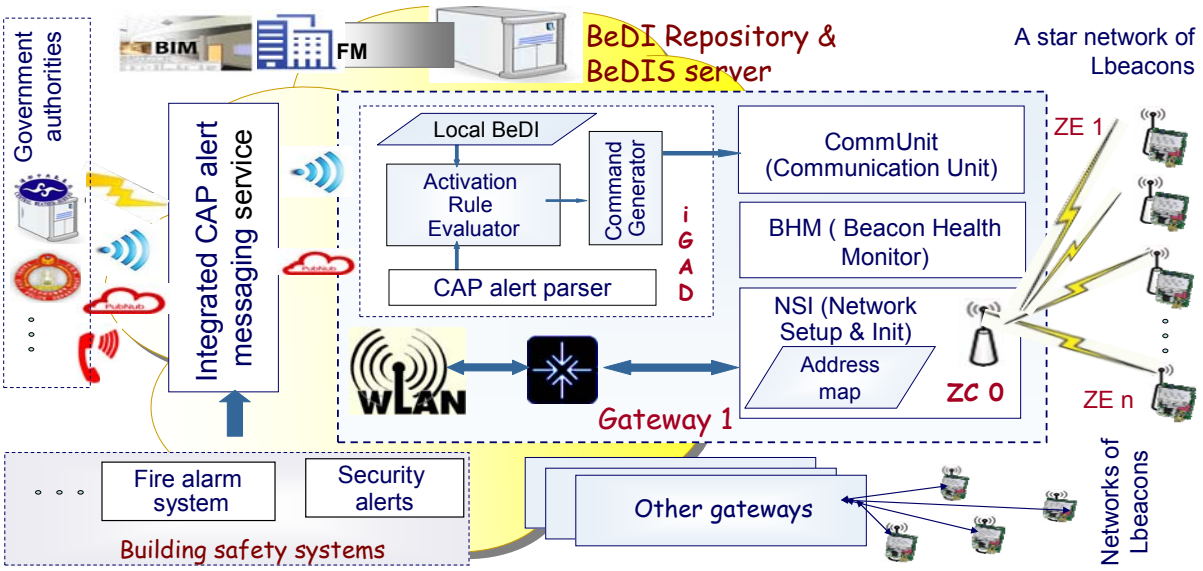

Fig. 1 Structure of smart gateway in a BeDIS

In addition to the gateway component linking the local Zigbee network to the local area network of the building and from there to the BeDIS server, the large rectangle in upper right half part of Fig. 1 encloses the major components of a typical smart gateway. They are described below. In the description, the term local Lbeacons refer to the beacons connected by the Zigbee star network served by the gateway.

***iGaD* (*intelligent guard against disaster*)** as shown in the upper left part of the box: It is composed of a CAP alert parser, an activation rule evaluator and a command generator. They are, respectively, for extracting from each CAP alert parameters specifying the type, severity, affected areas and time, etc. of the emergency/disaster warned by the alert; for choosing the

action(s) to take in response to the alert based on the alert parameters and local building/environment data; and for generating the command(s) to be issued to Lbeacons based on the decision of the action rule evaluator: The inclusion of an iGaD enables the gateway to make location specific decisions on the action(s) to take by local Lbeacons in response to each CAP alert. A gateway may not contain an iGaD. In that case the command(s) to local beacons come from the BeDIS server. -- [***NB. December 2017: The alpha version does not contain this component.***]

Communication unit in the upper right corner: In the alpha version, the sole function of this component is to support the communication of NSI and BHM with location beacons and the BeDIS server. (In a later version that contains iGaD, this components also receives commands from iGaD in the gateway and the BeDIS server and broadcasts the commands to local LBeacons.) Messages exchanged for these purposes are short, easily fit in the 1016 bits payload allowed by Zigbee packets and the 548 bytes allowed for user data by UDP packets.

There will be need for file transfers between the server and gateways and gateways to Lbeacons. These activities are not likely to have demanding response time requirements and can take place in the background. Data transferred to update local files of Lbeacons consists of a number of relative short character strings. We will investigate later whether sequenced message transmission supported by TCP is required for transfer of such files.

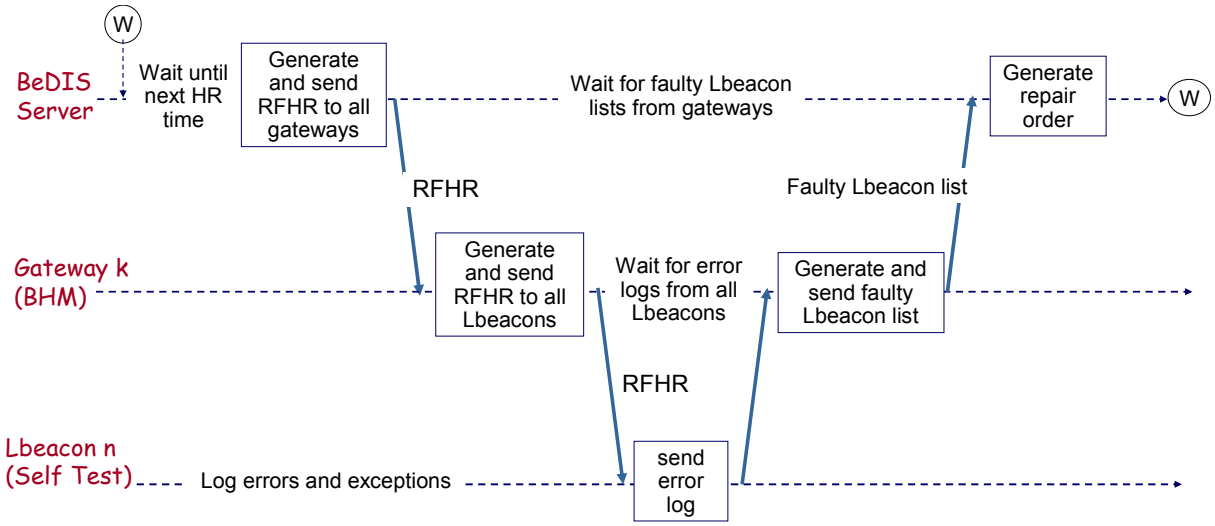

Fig. 2 Collaborative work by BeDIS server, gateways and Lbeacons for health monitoring

Beacon Health Monitor (BHM): One of the requirements of an IPS for a large building is that the health of the system can be assessed during runtime on a regular basis. The health monitor and self test module in each gateway is for this purpose. The interactions among the server, gateway and Lbeacons during each test are as shown in Fig. 2 and as explained below:

- The health of every Lbeacon is monitored by itself: In the alpha version, error log maintained by the beacon during its execution is used as its health report. The paper mentioned above describes more reliable self tests which later versions may adopt.

- Once or a few times a day, the BeDIS tests the health of the system based on health reports of all Lbeacons. Each time, the BeDIS server initiates a test by broadcasting a request for health report (RFHR) command to the beacon health monitor (BHM) of every gateway.
- Upon receiving the request, the BHM broadcasts its own request-for-health-report (RFHR) command to Lbeacons on the local star network.
- In response to the RFHR from the gateway, every Lbeacon sends its error log.
- After receiving reports from all location beacons on the local star network, the BHM generates and sends to the BeDIS server a global health report, listing the ID and location of every defective or failed beacon thus identified. (The absence of a report from the Lbeacon within the timeout interval signals to the gateway that the beacon warrants attention.)
- The server uses the report from every gateway as input in the generation or update of a repair order containing beacons to be repaired.

Network setup and initialization (NSI) module shown in the lower right part of the big box: in Fig. 1. As shown, NSI contains the coordinator node (ZC) of a Zigbee star network connecting all Lbeacons within a relative small area in the building. The module execute only during initialization and when individual beacons powers up.

More on Network Setup and Initialization

Setup Zigbee dongle with Raspberry Pi Zero W and Raspbian Each gateway is set up to be a full function device (FFD) and the coordinator node of a star network. Each Lbeacon is set up to function as a reduced function device: Lbeacons are end-devices, each in a Zigbee star network.

[Note: Set up instructions to be added later.]

NSI Operations Without loss of generality, we assume that the initialization of BeDIS has two phases. The initialization of all components except NSI are completed during phase 1. At the end of phase 1, the space for the address mapping table has been allocated, and the coordinator within NSI has been initialized and can start to process requests.

The goal of phase 2 initialization is to set up the communication paths between Lbeacons, the gateway and the BeDIS server. Specifically, during phase 2 initialization, each Lbeacon sends to a coordinator within its line of sight and range a join request containing its 3D coordinates. In response, the coordinator assigns a 16-bit network address to the beacon and sends the address to the beacon. The NSI module then updates an entry in the address table for the Lbeacon. The entry its Zigbee network address, its internal IDs and coordinates. The module then connects to the BeDIS server via WiFi and gets from the server the 2D barcode and location description of the Lbeacon and maintains the information in the address map. After initialization, the coordinator can address Lbeacons in the star network individually and identify the senders of packets from them. The former capability enables the gateway to support occasional transfers of data, including location specific commands and response instructions. The latter enables the gateway to play the key role in health monitoring as described above.

BeDIS Gataway Startup and Initialization

Fig. 4 shows a pseudo-code description of the start up and initialization code of gateways. Global constants and variables include the ones shown in Fig 3.

```
// Constants
// Maximum number of nodes (Lbeacons) per star network
MAX_NUMBER_NODES = 32;
// Maximum number of characters in location description
MAX_LENGTH_LOC_DESCRIPTION = 64;
// Length and struct of coordinates
COORDINATE_LENGTH = ??;
typedef struct coordinates {
   char X_coordinates[COORDINATE_LENGTH];
   char Y_coordinates[COORDINATE_LENGTH];
   char Z_coordinates[COORDINATE_LENGTH];
} coordinates;
// A global flag which is initially false and is set by main thread to
// true to tell other threads to shutdown, i.e., clean up and return.
  bool system_is_shutting_down;
 // A global flag that is initially false and is set by the main thread
 // to true when initialization completes.  Afterward, the flag is used
 // by other threads to inform the main thread the need to shutdown.
  bool ready_to_work;
 // A global flag set to true by a thread when its initialization failed
  bool initialization_failed;
```

```
/* Initialization of gateway components involve network activates
   that may take time. The main thread should wait until their
   initialization is sufficiently complete. These flags enable the
   modules to inform the main thread when this happens.  */
  bool NSI_initialization_complete;
  bool BHM_initialization_complete;
  bool CommUnit_initialization_complete;
/* A struct linking network address assigned to a Lbeacon to its
   UID, coordinates, and location description. Each beacon also
   has a 2D barcode on its enclosure. The barcode was assigned
   by BDE tool and is used during installation and maintenance. */
typedef  struct address_map {
      network_address; // 16 bit network address
      beacon_id;          // internal UID of the Lbeacon;
      coordinates beacon_coordinates;
      char loc_description [MAX_LENGTH_LOC_DESCRIPTION];
      beacon_barcode;
} address_map;
// an array of address maps
Address_map beacon_address [MAXIMUM_NUMBER_NODES];
// NSI is the only writer of beacon_address; it has many readers.
Beacon_address_lock;
```

Fig. 3 Some of the global variables

```
/* Psuedo code description of gateway start up and initialization.
   Assumptions include the following:
   - The zigbee dongle on the gateway has been set up with
     Raspberry Pi Zero W as a full function device (FFD). It is to
     be a coordinator after power up. Similarly, each Lbeacon was
     set up to be RFD, an end-device. After initialization each
     Lbeacon sends a request-to-join packet the coordinator
     in the gateway within its line of sight.
   - WiFi has been configured when RPi zero W was set up
   -  During system-wide shutdown, each gateway is normally shut
     down by a command from the server: Upon receiving a
     shutdown command, the communication unit tells the main
     thread (and there by other threads) to shutdown by
     setting the ready_for_work flag to false. Shutting down
     individual gateway via ctl-C can be handled the same way
     as in Lbeacons.
   -  TBD: Add in Lbeacon code for communicating with gateway.
*/
// The following function is executed by the gateway's main thread
Int main (…) {
   // Define and initialize all important global variables, including
   system_is_shutting_down = false;
   ready_to_work = false;
   initialization_failed = false;
   NSI_initialization_complete = false;
   BHM_initialization_complete = false;
   CommUnit_initialization_complete = false;
   …….
   // Allocate and zero memory space for address map
   beacon_address = calloc (MAX_NUMBER_NODES *
                                 sizeof (address_map));
   if (beacon_address == NULL), return (error_code);
   …..
  // Create the main threads of NSI, BHM, and CommUnit, using
  // startThread() in Lbeacon.c, which should have been revised to
  // return an error code upon failure.
  if ( startThread(NSI_routine) != NO_ERROR) clean_up_exit;
  if (startThread(BHM_routine) != NO_ERROR) clean_up_exit;
  if (startThread(CommUnit_routine != NO_ERROR) clean_up_exit;
  // Do some initialization work and then go to wait for components
  // to complete their initialization
  while ( (NSI_initialization_complete == false)    ||
          (BHM_initialization_complete == false) ||
          CommUnit_initialization_complete = false) ) {
     sleep(A_SHORT_TIME);
     if (initialization_failed == true) {
         system_is_shutting_down = true;
        clean_up_exit ( ); // the function returns after all threads have
                           // exited and resources freed
     }
   }
   // Initialization completed;
   ready_to_work = true;
   while (ready_to_work == true) {
    // do a chunk of work or sleep for a short time;
   }
   // need to shutdown either due to failure or normal shutdown
   system_is_shutting_down = true
  // wait for all threads to exit and free all resources and return
  clean_up_exit ( );
}
```

```
/* Gateway components may be multi-threaded. Below are thread
   routines of their main thread.*/
// start routine of the main thread in NSI
int  NSI_routine (…) {
    /* coordinator initializes the zigbee network:
       - if (PAN ID == 0) scan nearby network and chooses a PAN ID;
       - channel scan to find a good operating channel;
       - ready to access join requests from Lbeacons;
    */
    // make sure WiFi has been correctly configured
    ….
    /* initialize beacon_address []
       - enter a 16-bit network address in each address_map struct
         in the array
    …..
    // start a thread to maintain beacon_address map. The thread
    // should also check system_is_shutting_down flag periodically
    // and returns when it finds the flag is true.
    if (startThead (address_map_manager) != NO_ERROR) {
        initialization_failed = true;
        NSIcleanupExit( );
    }
    // finish phase 2 initialization (in ways TBD)
    ….
    NSI_initialization_complete = true;
    // wait for other components to complete initialization
    while ( (system_is_shutting_down == false) &&
              (ready_to_work == false)) {
         sleep (A_SHORT_TIME);
     }
    // ready to work, check for system shutdown flag periodically
    while (system_is_shutting_down == false) {
        - do a chunk of work and/or sleep for a short time
        /* Upon fatal failure, set ready_to_work = true and
           then call NSIcleanupExit( ) */
    }
    NSIcleanupExit(); // wait for all threads to have exited then returns
}

// start routine of the main thread in BHM
int  BHM_routine (…) {
   ….
   // when initialization completes,
   BHM_initialization_complete = true
   while (system_is_shutting_down == false) {
        do a chunk of work and/or sleep for a short time
    }
    BHMcleanupExit();
}
// start routine of the main thread of CommUnit
int  CommUnit_routine (…) {
    ….
   when initialization completes,
            CommUnit_initialization_complete = true;
   while (system_is_shutting_down == false) {
        do a chunk of work and/or sleep for a short time
    }
    CommUnitcleanupExit();
}
```

Fig. 4 Pseudo-code description of gateway start up, initialization and shutdown

Resources for Beginners

This section lists links to overviews, tutorials, etc. on related topics. They can provide a beginner with good places to start.

On Raspberry Pi Zero W and Raspbian:

- Getting started with RPi Zero W: https://learn.sparkfun.com/tutorials/getting-started-with-the-raspberry-pi-zero-wireless -- This is an excellent tutorial. It also contains references to together useful tutorials and links to downloads (e.g., to the latest version of Raspbian).
- Instruction for headless set up of Raspberry Pi Zero W (i.e., without GUI and keyboard): See either https://learn.adafruit.com/raspberry-pi-zero-creation/overview or at https://davidmaitland.me/2015/12/raspberry-pi-zero-headless-setup/

On ZigBee

- The current version uses XBee S2C Zigbee RF module. See XBee/XBee Pro S2C User Guide at https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf
- ZigBee/IEEE 802.15.4 Overview: Two sets of slides can give you a quick overview on Zigbee. The one at http://www.diit.unict.it/users/llobello/ss/IEEE802154.pdf and the other is used for a NCTU course at http://ocw.nctu.edu.tw/upload/classbfs1210030519111563.pdf
- "ZigBee/IEEE 802.15.4 Summary" by S. C.Ergen, http://home.iitj.ac.in/~ramana/zigbee.pdf This 2004 summary may be out of date at places, but is a better place to start then Wikipedia.
- Zigbee tutorial: https://iotpoint.wordpress.com/zigbee-tutorial/ - Very basic tutorial
- Code, hardware, and instructions to use the CC2520 with the Raspberry Pi, at https://github.com/lab11/raspberrypi-cc2520 : CC2520 is TI's Second generation 2.4 GHz ZigBee RF transceiver.
- Joining Zigbee networks, https://www.digi.com/resources/documentation/digidocs/90001399-13/references/r-create-join-zigbee-network.htm
- https://www.controlanything.com/Relay/Relay/ZB_ZIGBEE_COORDINATORS,
- https://www.dresden-elektronik.de/fileadmin/Downloads/Dokumente/Produkte/ZLL/RaspBee-BHB-en.pdf RaspBeeTM User Manual, December 10, 2017: This is the latest user manual for RaspBee – ZigBee addon board: "The RaspBee™ is a ZigBee addon board for the Raspberry Pi (RPi). By using the RaspBee the Raspberry Pi becomes a full functional wireless node which can be seamlessly integrated into ZigBee networks." – The hardware include Raspberry Pi 2 or 3.
- http://zboss.dsr-wireless.com/projects/zboss/wiki/Build_Compile_for_Linux_ARM, Zigbee Open Stack:

- https://www.linuxquestions.org/questions/linux-newbie-8/device-driver-for-zigbee-for-the-linux-kernel-2-6-20-or-2-6-24-a-666663/
- ZigBee Communication Using Raspberry Pi, at http://www.rhydolabz.com/wiki/?p=10868: This article describes an interface of Zigbee with Raspberry Pi2 for a proper wireless communication.

On MQTT-SN -- *We will not use MQTT-SN and MQTT within BeDIS*

MQTT-SN stands for MQ Telemetry Transport for Sensor Networks (a variation of the MQTT protocol) for embedded devices on non-TCP/IP networks. You can find discussions on MQTT versus MQTT-SN.

- The specification of MQTT-SN at http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf
- Short and sweet answers to why use MQTT-SN, not MQTT at https://stackoverflow.com/questions/27560549/when-mqtt-sn-should-be-used-how-is-it-different-from-mqtt
- Pros and cons of MQTT versus MQTT-SN at https://www.quora.com/What-are-the-pros-and-cons-of-MQTT-versus-MQTT-S-as-network-protocols-in-IoT-Internet-of-Things
- Tutorial: Prototyping a Sensor Node and IoT Gateway with Arduino and Raspberry Pi at https://thenewstack.io/tutorial-prototyping-a-sensor-node-and-iot-gateway-with-arduino-and-raspberry-pi-part-1/ -- Take a look at this tutorial first.
- http://mqtt.org/2013/12/mqtt-for-sensor-networks-mqtt-sn containing links to Eclipse Mosquitto
- MQTT-SN protocol client implementation: You can find sample code etc. at https://github.com/ibr-alg/wiselib/wiki/MQTTSN-client-implementation

On MQTT

- Introduction to MQTT – from the very, very beginning can be found at https://www.baldengineer.com/mqtt-introduction.html and https://www.baldengineer.com/mqtt-tutorial.html
- Step-by-step guides to MQTT https://www.survivingwithandroid.com/2016/10/mqtt-protocol-tutorial.html
- If you like to watch video go to https://www.youtube.com/watch?v=-KNPXPmx88E

Related topics

On REST and RESTFUL

- What is RESTFUL API (programming) https://stackoverflow.com/questions/671118/what-exactly-is-restful-programming
- REST, Roy Fielding's Ph.D thesis, 2000 "Architectural Styles and Design of Network Based Software Architecture" at https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm