ble-udp-bridge Documentation - v1

Shawn Nock - Jaycon Systems

17 December 2015

Contents

Source Code

main.c

Read config file and setup global config struct. Cleans up ble socket upon exit.

ble.c

`ble_scan_loop`

Main loop of the program. Poll(3) the HCI socket for BLE advertisements and the `connected` UDP socket for acknowlegements from the server

- When an ack is received, update the ack timestamp.
- When BLE advert is received:

1. Find or create a `beacon_t` in the hash table (`beacon_find_or_add` @beacon.c:31)
2. Iterate the filter with new data, retrieve current distance estimate (`kalman` @kalman.c:15)

3. Update the beacon packet count

- When either of the above, or a poll(3) timeout

1. If it's time to send a report, add report callback and walk the hash table (`report_beacon` @report.c:67)
2. Send report if it contains data
3. If it's garbage collection time, add beacon_t expire cb and walk the hash ('beacon_expire @beacon.c:53)
4. If we haven't sent a report in a while, send a keepalive.

kalman.c

`kalman` (line 15)

This function implements the kalman smoothing of the rssi values. It's a 2D filter on RSSI with one hidden variable (RSSI')

report.c

`report_beacon` (line 67)

This is a hash_walk callback, called for every beacon_t in the hash. It takes a context variable that is unused. If the beacon_t has a packet count (>0), a beacon report is appended (in the format documented below) to the report buffer.

This function also manages the report buffer size, expanding as needed.

hash.c

`hash_find` (line 45)

Finds a given beacon_t based on content (uuid, major, minor)

`hash_add` (line 55)

Expects a heap allocated beacon_t, finds by content (or heap address). If it exists, the passed beacon_t is free'd and the existing object is returned.

If it's not found, the beacon_t is added to the appropriate place in the hashtable.

`hash_walk` (line 90)

Calls an array of functions on each beacon_t in the hash_table; passes a context argument to each function (from the args array), size arg is length of walker & args array.

beacon.c

Tunables

`HOSTNAME_MAX_LEN` 255

`MAX_NET_PACKET` 64

`REPORT_INTERVAL_MSEC` 500

How often to send reports

`MAX_BEACON_INACTIVE_SEC` 10

Free memory for any beacons quietfor this long

`MAX_ACK_INTERVAL_SEC` 40

Reopen UDP socket if we haven't heard from the server in for this long

`MAX_PATH_LOSS_DIGITS` 5

/* How many significant figures in rssi calibration */

`DEFAULT_PATH_LOSS_EXP` [3.2]

`GC_INTERVAL_SEC [(MAX_BEACON_INACTIVE_SEC / 2)]`

/* How often to check for inactive beacons */

`KEEP_ALIVE_SEC` [30]

`MAX_HASH_CB` 5

Packet Format

```
|---|----------------|...|
             Data ------^
         ^-- Listener Name (Wifi MAC Addr)
  ^--------- Header (3 bytes)
```

Header format

Byte 0x00: Protocol Version & Packet Type

The most significant 4 bytes of byte 1 identify the version of the protocol that the listener speaks.

```
 version = (uint8_t) byte[0] >> 4;
 packet_type = (uint8_t)(byte[0] & 0x0f);
```

Current Versions:

- 0x00: As shipped to TapMyLife

The least significant four bytes identify the packet type.

Current Packet Types:

- 0x00: Keepalive Packet
- 0x01: Data Packet

See Packet Types section for details.

Byte 0x01: Report/data unit size

In a Data packet, each report will have the same length. This length (in bytes) is the value in byte 2 of the packet header.

For a Keepalive packet this byte may be set to any value, it is ignored. It has no function, but is reserved for future use.

Byte 0x02: Listener identifier length

The length of the listener identifyer that follows.

```
listener_id_len = byte[2]
```

Byte 0x03–`(0x03+listener_id_len-1)`: Listener Identifier

In version 1 of the protocol the listener identifier is the hostname of the listener. With the custom hardware, this is set to the MAC address of the wifi interface.

Byte `(0x03+listener_id_len)`–`EOF`: Packet Data

Packet Types

Keepalive Packet

```
|---|-----------------|...|
                        ^-- Data (optional, ignored)
            ^-------------- Listener Name (Wifi MAC Addr)
  ^------------------------ Header (3 bytes)
```

Keepalive packets are sent by the listener if a data report hasn't been sent for `KEEP_ALIVE_SEC` (c3listener.h:24) . It's intended to allow us to distinguish between no beacon activity and a malfuctioning/missing listener.

Data Packet

```
|---|-----------------|-----|-----|...|-----|
         Beacon Report n ----------------^
         Beacon Report 2 ------^
         Beacon Report 1 ^
       ^ Listener Name (Wifi MAC Addr)
  ^----- Header (3 bytes)
```

Data packets are sent approximately every `REPORT_INTERVAL_MSEC` interval (c3listener.h:12). The report contains a beacon report section for every beacon heard during the interval since the previous data packet. Data packets are not sent if there was no beacon activity during the interval. The length of each report is given in the header in byte 0x01.

Beacon Report Format

```
|----------------|--|--|--|--|
        ^--------------------- UUID (16 bytes)
                 ^----------- Major (uint16_t, BE)
                    ^-------- Minor (uint16_t, BE)
                       ^----- Packet Count (uint16_t, BE)
                          ^-- Filtered Distance (uint16_t, BE, in cm)
```

All values are in network byte order (big endian). Packet count is the count number of packets since the last data packet. Distance is in centimeters. UUID, major, and minor are from the ibeacon standard.