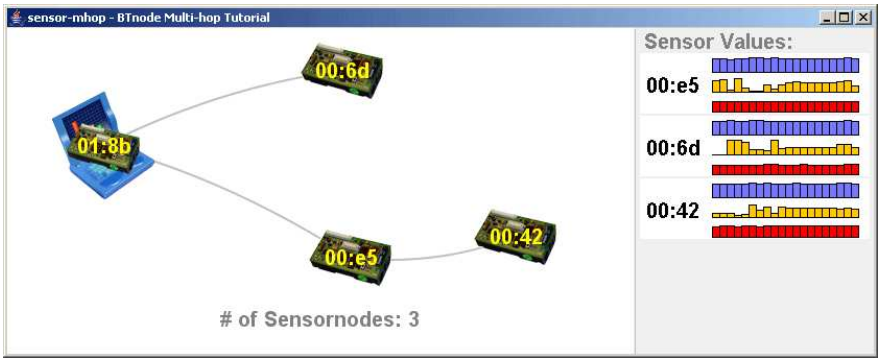
Chapter 9

Multihop Networking with Bluetooth



9.1 Introduction

In this chapter, we introduce the Bluetooth multi-hop layer of the BTnut System Software. With this step-by-step tutorial we guide the reader through the individual tasks that are required for initializing a connected Bluetooth scatternet and for sending data over multiple hops. The exercises are based on a simple example application in which multiple sensor nodes form a connected network, sample sensor data with an external sensor board, and send the sampled data over the multi-hop network to a central server.

9.2 Requirements

In order to complete the exercises of this chapter, the reader should be familiar with the BTnode tool chain and build system (Chapters 4 and 2).

Following material is required:

- BTnode development kit
- Teco Particles “ssmall” sensor board
- Server-node and server-application for visualization (provided)

9.3 Background: BTnut stack

Figure 9.1 shows the standard Bluetooth protocol stack and a BTnut protocol stack. The main difference is that the BTnut stack has a simplified *l2cap*-layer and additional multi-hop services.

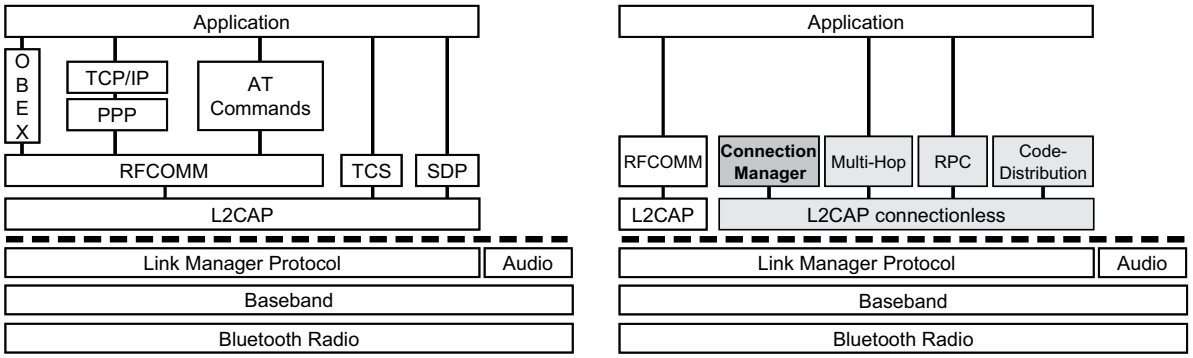


Figure 9.1: Standard (left) and BTnut (right) Multi-hop Protocol Stack.

The *l2cap*-layer has been introduced on page 54. For multi-hop data transport we use *l2cap-connectionless* data transport, which is much simpler than *l2cap-connection oriented* data transport since it no channels need to be set up.

The *l2cap-connectionless*-layer basically acts as a service multiplexer. Higher layer services can register themselves to this layer with a *Protocol Service Multiplexer (PSM)*-number. This layer then dispatches incoming data packets to the corresponding higher services according to the PSM-number in the packets.

The following services are registered to the *l2cap-connectionless*-layer:

Connection Manager: The connection manager is responsible for the formation and maintenance of a connected topology. It has a distributed topology control algorithm that controls the discovery and connecting of devices.

Multi-hop: The multi-hop service performs multi-hop routing and forwarding and provides an API for sending and receiving packets.

RPC: The RPC service implements a remote procedure call. Procedures can be registered to the RPC layer and can then be called from a remote node. After the execution, a result-packet is automatically routed back to the caller.

Code Distribution: With the code distribution service, the nodes can be updated with new software over-the-air.

There are dependencies between these services. Thus, they must be initialized in the correct order.

9.4 Multi-hop Services Initialization

For initializing the multi-hop services, we start with a code template, that basically looks like the *bt-cmd* application. We first need to include the API's of the services and configure some definitions. In order to do so, we create a separate file called `defs.h` that contains the definitions of the bluetooth and btnode API, and the new definitions of the multi-hop services including the connection manager and the RPC service. At the end of this file, we define the PSM numbers of the services. These service numbers must be the same on all communicating nodes (comparable to TCP/UDP port numbers).

In the next step we create the main application file `sensor-node.c` that includes `defs.h`.

defs.h:

```
// standard includes
#include <stdio.h>
#include <io.h>
#include <dev/usartavr.h>
#include <sys/thread.h>
#include <sys/timer.h>

// btnode includes
#include <hardware/btn-hardware.h>
#include <led/btn-led.h>
#include <terminal/btn-terminal.h>

// bluetooth stack
#include <bt/bt_hci_api.h>
#include <bt/bt_acl_defs.h>
#include <bt/bt_psm.h>
#include <bt/l2cap_cl.h>

// mhop includes
#include <mhop/mhop_cl.h>
#include <mhop/mhop_con_info.h>
#include <mhop/rpc.h>

// choose your connection manger here!
#include <cm/cm_tree.h>
// bluetooth class of device
#define CM_COD              931

#define MAX_NR_SERVICES     16
#define NR_BUFFERS          6

// service multiplexer definitions
#define CM_PSM              0x1003
#define MHOP_PSM            0x1005
#define RPC_PROC_PSM        0x1007
#define RPC_RESULT_PSM      0x1009
```

sensor-node.c:

```
#include "defs.h"
#include "program_version.h"

int main(void){
    // hardware init
    btn_hardware_init();
    btn_led_init(1);

    // init terminal app uart
    u_long baud = 57600;     // serial baud rate
    NutRegisterDevice(&APP_UART, 0, 0);
    freopen(APP_UART.dev_name, "r+", stdout);
    _ioctl(_fileno(stdout), UART_SETSPEED, &baud);
    btn_terminal_init(stdout, "[senso]$");

    // hello message
    printf("\n# ------------------------------------------------");
    printf("\n# Welcome to the BTnut Tutorial (c) ETH Zurich\n");
    printf("# program version: %s\n", PROGRAM_VERSION);
    printf("# ------------------------------------------------");

    printf("\nbooting bluetooth module.\n");
    btn_hardware_bt_on();

    // insert mhop services initialization here !

    // run terminal (never returns)
    btn_terminal_run(BTN_TERMINAL_NOFORK, 0);
    return 0;
}
```

Exercise 9.1. *Complete the* `sensor-node.c` *template with the following statements in order to initialize the multi-hop services:*

a) ***HCI Stack Initialization:***

```
struct btstack* bt_stack;
bt_stack = bt_hci_init(&BT_UART);
bt_acl_init(bt_stack, BT_HCI_PACKET_TYPE_DM3);
```

b) ***L2cap Initialization:***

```
bt_psm_t* psmux;
psmux = bt_psm_init(bt_stack, MAX_NR_SERVICES, NR_BUFFERS);
l2cap_cl_init(bt_stack, psmux);
```

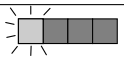
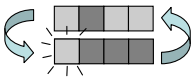
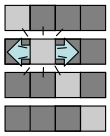
c) ***Connection Manager Initialization:***

```
con_mgr_init(bt_stack, psmux, CM_PSM, bt_hci_register_con_table_cb, CM_COD);
```

d) ***Mhop-API and RPC Initialization:***

```
mhop_cl_init(bt_stack, psmux, MHOP_PSM, NR_BUFFERS, con_mgr_register_con_table_cb);
rpc_init(psmux, 8, RPC_PROC_PSM, RPC_RESULT_PSM);
mhop_con_info_init();
```

Exercise 9.2. *Compile and upload the* sensor-node *application to your btnode. Inspect the four LEDs during the booting phase and interpret the state of the connection-manager with the following table:*

State	***LED Pattern***	***LEDs***
Initialization	*Heartbeat (blue LED)*	
No connection	*4-bit ID + Heartbeat*	
Inquiry	*Knight-Rider*	
Connected	*4-bit Tree-ID*	
Connectivity with base station	*green LED*	

When your node is connected with the base station, it should appear on the server visualization screen. Compare the last two bytes of the MAC address of your BTnode with the ones on the screen.

9.5 Sensor Data Sampling and Reporting

In the following exercises you will connect a sensor board to your BTnode, sample first one, then three sensors, and finally transmit a data packet containing your sensor values to a base station.

Exercise 9.3. ***Sample the Microphone:*** *The Teco Particles "ssmall" sensor board contains a number of sensors for measuring the temperature, luminance, sound and vibration. In the btnut software there exist library-functions for initializing and reading the sensors. We start with the microphone sensor:*

a) Attach the sensor board to the usbprog-board on your BTnode.

b) Include the microphone sensor API in your `defs.h`*:*

```
#include <teco_ssmall/micsampler.h>
#include <dev/adc.h>
```

c) Turn on the I/O power at the end of the Bluetooth initialization in your main routine:

```
btn_hardware_io_power(1);
```

d) Write a thread that first initializes the sensor and then periodically samples the sensor every 5 seconds. Print the sampled data to the terminal. For initializing use the following two functions in the given order:

```
ADCInit();
mic_init();
```

For sampling the sensor, use the function:

```
Reads loudest microphone value.
This function returns the loudest sampled microphone value since the last time it has been called.
The values range from 0 to 1023 (loudest).

u_short mic_read(void);
```

e) Test your program and the microphone sensor. If you snap your fingers over the sensor, a value above 1000 should be measured.

Exercise 9.4. ***Sample Luminance and Temperature:*** *In order to prepare for the next exercise, we read the three sensor values in a predefined structure.*

a) Include the defines for the luminance and temperature sensors:

```
#include <dev/twif.h>
#include <teco_ssmall/tsl2550.h>
#include <teco_ssmall/tc74.h>
```

b) Define a data structure for the sensor data:

```
typedef struct sensor_data_s {
        u_short mic;
        short light;
        u_char temp;
} sensor_data_t;
```

c) Complete the sensor thread with the initialization, and sensor sampling:

```
THREAD(sensorLoop, arg){
    TwInit(20); // init twi with slave addr

    while(tsl_init()) NutSleep(10000); // start light sensor

    ADCInit();
    mic_init(); // init mic

    sensor_data_t sensor_data;    // allocate data structure (see defs.h)
    sensor_data_t* data = &sensor_data;
    u_char channel0, channel1;
    signed int temp;
    for(;;){
        NutSleep(5000);
        data->mic = mic_read();             // sample mic
        tsl_read(&channel0, &channel1);     // sample light
        data->light = tsl_calculate_lux(channel0, channel1);
        tc_read(&temp);                     // read temperature
        data->temp = temp;
        printf("mic = %u, light = %d, temp = %u\n",
                data->mic,
                data->light,
                data->temp);
    }
}
```

d) Test this application on your BTnode.

Exercise 9.5. ***Sending packet with sensor-data to the base-station:***

a) Consult the BTnut API documentation for the usage of the function `mhop_cl_send_pkt` *which is located in* `btnut/btnode/include/mhop/mhop_cl.h`.

b) We first have to define the destination-address (MAC-address of the base-station). Ask the tutorial-leaders for the correct address and define it in the form:

```
// address of the base station (note: big endian)
bt_addr_t sink_addr = {0x8b,0x01,0x00,0x3f,0x04,0x00};
```

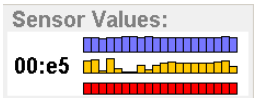
c) Add the PSM of this service to the `defs.h` *file:*

```
#define SENSO_PSM            0x100b
```

d) Insert in front of the printf-statement in the last exercise the command for sending the packet.

```
mhop_cl_send_pkt((u_char*)data,
                 sizeof(sensor_data_t),
                 sink_addr,
                 SENSO_PSM,
                 MHOP_CL_UNICAST, MHOP_CL_TTL_INFINITE);
```

e) Test your application on your BTnode and check whether the sensor data of your node is being received by the base-station. If everything is correct, you should see a small plot of your sensor data on the visualization screen.



Explanation *Access node (base-station)*: The program running on the access node has three parts:

1. the initialization of the mhop-services,
2. a discovery thread that periodically sends out request beacons for discovering nodes, and
3. the sensor data callback function, that processes the received data and prints it out to the terminal.

Part 1 is the same as for the sensor node. We therefore only discuss parts 2 and 3.
The discovery thread periodically calls the `coninfo` terminal-command, which has been registered with the statement `mhop_con_info_init()`.

```
THREAD(coninfoLoop, arg){
        while(1){
                NutSleep(10000);
                btn_terminal_process_cmd("coninfo");
        }
}
```

The `coninfo` command lets the node broadcast a "connection information request" to all connected nodes. Every node that receives this request sends an answer back with information about its direct connected neighbors.
Sensor data packets are sent by the sensor nodes with the PSM `SENSO_PSM`. In order to process these packets on the access node, a callback function is registered to this PSM. The following statement registers the function `sensor_data_cb` to the PSM `SENSO_PSM`:

```
bt_psm_service_register(psmux, SENSO_PSM, sensor_data_cb, NULL);
```

The `sensor_data_cb` callback function is defined as follows:

```
bt_acl_pkt_buf* sensor_data_cb(bt_acl_pkt_buf* pkt_buf,
                               u_char* data,
                               u_short data_len,
                               u_short service_nr,
                               void* cb_arg){
    u_char* source = mhop_cl_get_source_addr(pkt_buf->pkt);
    sensor_data_t* sensor_data = (sensor_data_t*) data;
    printf(":S %.2x:%.2x %u %d %u\n", (*(source + 1)), *(source),
            sensor_data->mic,
            sensor_data->light,
            sensor_data->temp);

    // free the received message
    return pkt_buf;
}
```

The sensor data callback prints out the the lower two bytes of the source address along with the sensor values. The visualization application parses this information and displays a plot.

Optional Exercise 9.6. *Consult the BTnut API about the usage of the registration function* `bt_psm_service_register` *which is defined in the file* `btnut/btnode/include/bt/bt_psm.h`.