Embedded Systems FS 2016

Lab 4: Multi-Hop Bluetooth Chat

Discussion Dates: 27.04.2016 / 04.05.2016

Introduction: The Bluetooth Protocol Stack

Bluetooth is a wireless communication standard for devices operating in the 2.4 GHz band. Nodes form piconets consisting of 1 master and at most 7 slaves. All nodes in a piconet use the same frequency hopping scheme. Connections are either one-to-one or between the master and all slaves (broadcast).

On a BTnode, the Atmega128 microcontroller communicates with the Zeevo ZV4002 Bluetooth radio according to the principles defined by the Host Controller Interface (HCI) in the Bluetooth standard. In this session you will familiarize with the Bluetooth Protocol Stack and the Bluetooth services available on the BTnodes.

![Diagram of Bluetooth Protocol Stack](image.png)

Figure 1: Comparison between the the standard and the BTnut Bluetooth protocol stack.

Fig. 1(a) shows the layer hierarchy of the standard Bluetooth protocol stack. The most important blocks for this lab session are the following:

- **Link Manager Protocol (LMP):** controls the radio link between two devices.
- **Host Controller Interface (HCI):** defines a common standardized interface between the Bluetooth host (e.g., the Atmega128 microcontroller) and the Bluetooth controller (e.g., the baseband controller and the Bluetooth radio).
- **Logical Link Control and Adaptation Protocol (L2CAP):** provides an abstract interface for data communication.
Compared to the standard version of the protocol stack, BTnut provides a simplified L2CAP layer and additional multi-hop services, as shown in Fig. 1(b). A simple Asynchronous Connection-Less (ACL) version of the L2CAP layer acts as a service multiplexer. Higher layer services (e.g., Multi-Hop) can register themselves with a Protocol Service Multiplexer (PSM)-number: the L2CAP connectionless layer dispatches the incoming data to the corresponding higher layers according to the PSM-number embedded in the packets. The following higher layer services are used in this exercise session:

- **Connection Manager**: forms and maintains a connected topology, manages discovery and connection of devices.
- **Multi-Hop**: performs multi-hop routing and forwarding, provides an API for sending and receiving packets.

**Task 1: One-to-One Transmission of a Text Message**

The goal of this task is to establish a one-to-one connection and exchange text messages between a pair of BTnodes. You are given the template code for two functions, `send_msg()` and `receive_msg()`. `send_msg()` is registered as the function that is executed when you run the `send` command in a terminal: by typing `send <h>` in the terminal, this command executes `send_msg()` and passes the given connection handle `h` as an argument to the function. You are required to complete this function with the following parts: define the message you want to send, check if it has a proper length, construct the corresponding packet with the proper headers, and print the message details (message string, message length, and connection handle).

```c
// send_msg() function: sends a predefined text message with a given connection handle
void send_msg(char *arg) {
    // handle variable
    int handle;

    // read handle value inserted by the user (use sscanf)

    // message to send
    char msg[] = "Hello there!";

    // compute the length of the message (including the terminating null character)
    // use u_short strlen(const char *str)

    // check if the length of the message is valid
    if (msg_len > MAX_MSG_LEN) {
        // the message is too long: notify the user and abort transmission
    }

    // the packet has the following format:
    // |---------------|---------------|-------------|
    // | HCI header   | L2CAP header | message     |
    // | |msg_len|channel|
    // |-------------------|---------------|-------------|
    // | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ...
    // |-------------------|---------------|-------------|
    // (the HCI header will be set directly by the bt_hci_send_acl_pkt() function)
    char packet[HCI_HEADER_LEN + L2CAP_HEADER_LEN + msg_len];

    // set the L2CAP header into the packet (little endian format):
    // set the msg_len part
    packet[HCI_HEADER_LEN] =
```

```c
// BTNUTDIR/ex/lab4/lab4-task1/lab4-task1.c
```
receive_msg() is registered as a callback function: it is called by BTnut whenever a new message is received from another BTnode. You are required to complete this function with the following parts: retrieve the details of the message (message string, message length, and connection handle) from the received packet and print them on the terminal.

```
struct bt_hci_pkt_acl *receive_msg(void *arg, struct bt_hci_pkt_acl *pkt, bt_hci_con_handle_t handle, u_char pb_flag, u_char bc_flag, u_short len, u_long t_arrive) {
    // set packet to point to the received pkt struct
    char *packet = (char *)pkt;
    // the packet has the same format as in the send_msg() function
    // retrieve the message length from the packet
    char msg[msg_len];
    // copy the text message from the packet to the msg
    // use strcpy(char *destination, const char *source)
    // print information about the received message
    return pkt;
}
```

### Task 1.1: Discover Bluetooth Devices

1. After completing the code in `lab4-task1.c`, compile and upload it to your BTnode.
2. Start Minicom to connect to the serial console of the BTnode.
3. When the BTnut terminal is ready to receive commands, type `bt inquiry sync` to start an inquiry and show the addresses of discovered Bluetooth devices. Bluetooth addresses have format `xx:xx:xx:xx:xx:xx`, where each `x` corresponds to an hexadecimal digit.

### Task 1.2: Connect with Another BTNode

Connect with the desired BTNode (e.g., the BTnode of a neighboring group) by typing `bt con <addr>`, where `addr` is the Bluetooth address of the device you want to connect with. If the connection is successfully established, the node prints the corresponding connection handle, which is required to send messages over this connection. If an error occurs, try to run the `bt con <addr>` command again, or try to connect with another BTNode. Pay attention that if node 1 has already established a connection with node 2, trying to connect from node 1 with node 2 always returns an error since the connection is bidirectional and is already existing. You can also type `bt contable` to print a table containing the open connections and the corresponding connection handles. Type `bt discon <h>` to interrupt the connection correspondent to handle `h`. 
Task 1.3: Send a Text Message to Another BTnode

Test your implementation of the `send_msg()` function by typing `send <h>`. It should print the correct details (message string, message length, and connection handle) of the message you have just sent. If the group using the paired BTnode has already completed the implementation of the `receive_msg()` function, you should also see your message correctly printed on their terminal.

Task 1.4: Receive a Text Message from Another BTnode

Test your implementation of the `receive_msg()` function by asking a group to connect with your BTnode and send you a message. It should print the correct details (message string, message length, and connection handle) of the message they have sent you.

Task 2: Multi-Hop Chat

In this task we make use of the multi-hop service provided by BTnut to establish a Bluetooth multi-hop network and send messages to a specific node, the base station. Upon the reception of a message at the base station, a Java GUI displays it on the video projector. Users can interact by sending several messages to the base station, as if they were using a chat client.

You are given the template code for function `chat()`, which is registered as the function that is executed when you run the `chat` command in a terminal: by typing `chat <msg>` in terminal, this command executes `chat()` and passes the given message `msg` as an argument to the function. In contrast to the previous task, you can specify the message directly in the command line (i.e., it is not defined within the function). You are required to complete the `chat()` function with the following parts: copy the message given in the command line into a local string, compute the length of the message, and print the message details (message string and message length).

```c
// chat() function: sends a given text message to the base station
void chat(char *arg) {
// copy (at most) the first MAX_MSG_LEN characters of the given message into msg
// use strncpy(char *destination, const char *source, u_short len)

// compute the length of the message (including the terminating null character)
// use u_short strlen(const char *str)

// send the packet to the base station using multi-hop services
mhop_cl_send_pkt((u_char *)msg, msg_len, sink_addr,
    MESSAGE_PSM, MHOP_CL_UNICAST, MHOP_CL_TTL_INFINITE);

// print information about the sent message
}
```