Raspberry pi3 Communication Bluetooth série

1 Introduction

Bluetooth est une norme de communication permettant l'échange bidirectionnel de données à très courte distance (quelques dizaines de mètres) en utilisant des ondes radio UHF comme support de transmission dans la bande de fréquence de 2.400 à 2.483 Ghz.

Cette bande est divisée en 79 canaux de 1 Mhz au total. Pour transmettre des datas, la technologie Bluetooth utilise le FHSS (Frequency Hopping Spread Spectrum).

L'origine de l'appellation Bluetooth fait référence à un roi danois Harald « Dent bleue » (dû à son goût immodéré pour les mûres) qui aurait unifié les différents royaumes nordiques à la fin du Moyen Âge, de même que Bluetooth SIG s'est créé autour d'intérêts communs entre différents fabricants de matériels.

Pour pouvoir expérimenter la liaison Bluetooth, on doit bien évidemment disposer de deux périphériques équipés d'une puce bluetooth. Ensuite, deux cas peuvent se présenter, soit les périphériques se connaissent déjà, soit ils sont inconnus l'un pour l'autre. Au cours de cet exposé nous utiliserons une raspberry Pi 3 et un un smartphone Android.

Sur le Raspberry Pi 3, nous n'avons besoin de rien de spécial car le Bluetooth est présent par défaut sur la carte.

2 Interfaçage entre couches matérielles et logicielles (HCI)

La couche HCI fournit une méthode uniforme pour accéder aux couches matérielles. Son rôle de séparation permet un développement indépendant du matériel et du logiciel.

la méthode de communication est uniformisée quelque soit le type de matériel auquel on a affaire via la couche **HCI** (**H**ost **C**ontroller **I**nterface). Cette interface est exhaustivement définie dans la norme Bluetooth. Elle est composée de commandes et d'événements. Le périphérique envoie des commandes à la puce bluetooth et reçoit des événements en retour.

hciconfig est utilisé pour configurer les périphériques Bluetooth. L'utilitaire **hciconfig** est presque l'équivalent de ifconfig pour une carte réseau ethernet.

hciX est le nom d'un périphérique Bluetooth installé sur le système.

Si hciX n'est pas indiqué, hciconfig donne le nom et les informations de base sur tous les

périphériques Bluetooth installés sur le système.

Si hciX est donné sans commande, il affiche uniquement les informations de base sur le périphérique hciX.

Les informations de base sont :

- le type d'interface,
- l'adresse BD,
- la MTU ACL,
- la MTU SCO,
- les indicateurs (active UP, initiée, en cours d'exécution RUNNING, brute, numérisation de page activée,PSCAN scan de requête activé, requête, authentification activée, cryptage activé).
- Le nombre d'octets reçus RX et envoyés TX

```
root@raspPi3plus:~# hciconfig
hci0:   Type: Primary  Bus: UART
        BD Address: B8:27:EB:D6:F8:2B  ACL MTU: 1021:8  SCO MTU: 64:1
        UP RUNNING PSCAN
        RX bytes:22671 acl:489 sco:0 events:980 errors:0
        TX bytes:30081 acl:990 sco:0 commands:214 errors:0
```

Chaque périphérique bluetooth est identifié par un nom **hci0** et par une adresse, sur 6 octets. Pour le pi3 ci-dessus: **B8:27:EB:D6:F8:2B** Elle peut être vue comme l'équivalent des adresses MAC des périphériques sur un réseau. Cette adresse est utilisé au niveau de la couche matériel baseband.

La commande **hcidump** lit et affiche les données HCI brutes d'une communication Bluetooth.

```
pi@raspPi3plus:~ $ sudo apt-get install bluez-hcidump

pi@raspPi3plus:~ $ hcidump
HCI sniffer - Bluetooth packet analyzer ver 5.43
device: hci0 snap_len: 1500 filter: 0xffffffff
> HCI Event: Connect Request (0x04) plen 10
    bdaddr 34:14:5F:2F:9F:BA class 0x5a020c type ACL
> HCI Event: Command Status (0x0f) plen 4
    Accept Connection Request (0x01|0x0009) status 0x00 ncmd 1
> HCI Event: Connect Complete (0x03) plen 11
    status 0x00 handle 11 bdaddr 34:14:5F:2F:9F:BA type ACL encrypt 0x00
> HCI Event: Command Status (0x0f) plen 4
    Read Remote Supported Features (0x01|0x001b) status 0x00 ncmd 1
...
```

3 La couche L2CAP

Le L2CAP est le protocole minimal d'échange de données de la spécification Bluetooth. On peut écrire des applications Bluetooth utilisant le L2CAP pour communiquer entre elles, via des dispositifs Bluetooth.

Les fonctions de L2CAP sont :

- **Multiplexer** des données entre différents protocoles de couche supérieure.
- **Segmentation** et **ré-assemblage** des paquets.
- Gestion de la transmission unidirectionnelle des données de multidiffusion vers un autre groupe de périphériques Bluetooth.

La commande **L2ping** envoie une demande d'écho L2CAP à l'adresse MAC Bluetooth indiquée en notation hexadécimale pointée.

```
pi@raspPi3plus:~ $ sudo l2ping -c 5 44:6D:6C:15:BE:C0
Ping: 44:6D:6C:15:BE:C0 from B8:27:EB:D6:F8:2B (data size 44) ...
0 bytes from 44:6D:6C:15:BE:C0 id 0 time 6.18ms
0 bytes from 44:6D:6C:15:BE:C0 id 1 time 30.88ms
0 bytes from 44:6D:6C:15:BE:C0 id 2 time 14.87ms
0 bytes from 44:6D:6C:15:BE:C0 id 3 time 23.61ms
0 bytes from 44:6D:6C:15:BE:C0 id 4 time 8.58ms
5 sent, 5 received, 0% loss
pi@raspPi3plus:~ $
```

4 Appairage avec les autres périphériques

Le périphérique souhaitant initier la communication doit faire une recherche de périphériques bluetooth à l'écoute dans son environnement. On appelle cela la phase d'inquiry. Au terme de cette recherche, le périphérique initiateur est en possession d'une liste de périphériques.

Il est nécessaire de rendre le raspberry pi visible pour les autres périphérique bluetooth.
La commande `hciconfig` hci0 piscan permet de le rendre découvrable et connectable .

```
pi@raspPi3plus:~ $ sudo hciconfig hci0 piscan
```

A partir de cet instant le raspberry pi3 plus est visible sur le smartphone. L'appairage est possible à partir du smartphone.

bluetoothctl est un outil de contrôle du Bluetooth que nous allons utiliser pour obtenir la liste des devices apairés et connectés.

```
pi@raspPi3plus:~ $ bluetoothctl
```

```
[NEW] Controller B8:27:EB:D6:F8:2B raspPi3plus [default]
[NEW] Device 34:14:5F:2F:9F:BA Galaxy A5 (2016)
[bluetooth]#
```

l'écran ci-dessus montre que le teléphone Galaxiy A5 (2016) est appairé.

Si on tente ensuite de refaire le même appairage à partir du raspberry pi3 on obtient le message d'erreur suivant :

```
[bluetooth]# pair 34:14:5F:2F:9F:BA
Attempting to pair with 34:14:5F:2F:9F:BA
Failed to pair: org.bluez.Error.AlreadyExists
[bluetooth]#
```

Écran obtenu pendant la phase d'appairage avec le téléphone 44:6D:6C:15:BE:C0

```
pi@raspPi3plus:~ $ bluetoothctl
[NEW] Controller B8:27:EB:D6:F8:2B raspPi3plus [default]
[NEW] Device 34:14:5F:2F:9F:BA Galaxy A5 (2016)
[NEW] Device 44:6D:6C:15:BE:C0 Anne-Marie BRUYERE
[CHG] Device 44:6D:6C:15:BE:C0 Modalias: bluetooth:v0075p0100d0200
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001105-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 0000110a-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 0000110c-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001112-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001115-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001116-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 0000111f-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 0000112d-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 0000112f-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001132-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 UUIDs: 00001200-0000-1000-8000-00805f9b34fb
[CHG] Device 44:6D:6C:15:BE:C0 ServicesResolved: yes
[CHG] Device 44:6D:6C:15:BE:C0 Paired: yes
```

```
pi@raspPi3plus:~ $ bluetoothctl
[NEW] Controller B8:27:EB:D6:F8:2B raspPi3plus [default]
[NEW] Device 34:14:5F:2F:9F:BA Galaxy A5 (2016)
[NEW] Device 44:6D:6C:15:BE:C0 Anne-Marie BRUYERE
[bluetooth]#
[CHG] Device 34:14:5F:2F:9F:BA Connected: yes
[Galaxy A5 (2016)]#
```

lorsqu'un device se connecte l'invite de commande donne le nom du device connecté dans l'exemple ci-dessus nous avons un Galaxy A5 (2016) qui vient de se connecter au raspberry pi3 plus.

```
[CHG] Device 34:14:5F:2F:9F:BA Connected: no
[bluetooth]#
```

A la déconnexion l'invite de commande redevient [bluetooth]#
pour quitter bluetoothctl

```
[bluetooth]# quit
[DEL] Controller B8:27:EB:D6:F8:2B raspPi3plus [default]
```

4 Ajout du profil SP

Bien que le raspberry pi soit visible il n'est pas possible de communiquer via RFCOMM en émulant une liaison série classique. Il est nécessaire d'ajouter le profil SP. Pour ce faire éditer ce fichier :

```
sudo nano /etc/systemd/system/dbus-org.bluez.service
```

Ajoutez l'indicateur de compatibilité, "-C" ou "--compat", à la fin de la ligne "ExecStart =".
Ajoutez une nouvelle ligne après cela pour ajouter le profil SP. add SP ou add --channel=22 SP
si vous souhaitez le service Serial Port (SP) associer au channel 22 :
Les deux lignes devraient ressembler à ceci:
Avec ces lignes, nous établissons une communication série (SP).

```
[Service]
Type=dbus
BusName=org.bluez
ExecStart=/usr/lib/bluetooth/bluetoothd -C
ExecStartPost=/usr/bin/sdptool add SP
NotifyAccess=main
...
```

Enregistrez le fichier et redémarrez.

5 Établissement d'une liaison série avec RFCOMM

RFCOMM est un protocole situé au dessus du protocole L2CAP. Il a pour but l'émulation de liaisons série RS232 entre deux dispositifs Bluetooth. (jusqu'à soixante canaux de connexions simultanées par périphérique Bluetooth à la fois). RFCOMM fournit un flux de données simple et fiable, similaire au protocole TCP.

rfcomm est aussi un utilitaire en ligne de commande pour établir une connexion série.

rfcomm est utilisé pour configurer, gérer et inspecter la configuration RFCOMM du sous-système Bluetooth dans le noyau Linux. Si aucune commande n'est donnée ou si l'option -a est utilisée, rfcomm affiche des informations sur les périphériques RFCOMM configurés.

rfcomm [options] < cmd > < dev >

L'option **wath** permet d écouter les connexions entrantes sur un canal RFCOMM spécifié.
Si aucun canal n'est spécifié, le canal 1 sera utilisé, toutefois, si cmd est donné, le canal doit être spécifié avant cmd., la commande cmd sera exécutée dès qu'un client se connecte. Quand le processus fils se termine ou le client se déconnecte, la commande prend fin. Les occurrences de {} dans cmd seront remplacées par le nom du périphérique utilisé par la connexion exemple /dev/rfcomm0 . Cette commande peut être interrompu par la séquence de touches CTRL-C.

Premier exemple d'utilisation

```
root@raspPi3plus:/home/pi# rfcomm watch hci0
Waiting for connection on channel 1
```

la Raspberry pi3 est en attente d'une demande de connexion. Dès qu'un périphérique demande une connexion un périphérique virtuelle **/dev/rfcomm0** est créé.

```
Connection from 34:14:5F:2F:9F:BA to /dev/rfcomm0
Press CTRL-C for hangup
```

Vous devez garder la commande de surveillance rfcomm en cours d'exécution.
Et dans une autre console, vous pouvez ouvrir **/dev/rfcomm0** pour envoyer et ou recevoir du texte (avec minicom par exemple)

```
root@raspPi3plus:/home/pi# minicom -D /dev/rfcomm0
```

Deuxième exemple d'utilisation en exécutant une commande à la connexion

```
root@raspPi3plus:/home/pi#  rfcomm watch 0 1 cat {}
```

Remarque : l'argument {} dans cat sera remplacé par le nom du périphérique utilisé par la connexion.

Pour tester, il suffit d'utiliser un client capable de se connecter sur notre périphérique. Il en existe de nombreuses applications Android (comme Bluetooth Terminal) pour faire cela.

Autre exemple avec le programme bluetooth
`root@raspPi3plus:/home/pi/ABC_du_C/11_Liaison_serie# rfcomm watch 0 1 ./bluetooth`

Si on lance la commande **rfcomm -a** dans un autre terminal

```
pi@raspPi3plus:~ $ rfcomm -a
rfcomm0: B8:27:EB:D6:F8:2B -> 34:14:5F:2F:9F:BA channel 1 connected [reuse-dlc release-on-hup tty-attached]
```

l'état de la connexion est affiché ainsi que le canal utilisé.

6 Ajout d'un service systemd

Voici la démarche à suivre pour transformer un programme en un service systemd pouvant être lancé automatiquement au démarrage du système.

Nous devons ajouter un nouveau service en créant un nouveau service nommé rfcomm.service

pour ce faire :

```
pi@raspPi3plus:~ $ sudo nano /etc/systemd/system/rfcomm.service
```

```
[Unit]
 Description=RFCOMM service
 After=bluetooth.service
 Requires=bluetooth.service

[Service]
 ExecStart=/usr/bin/rfcomm watch 0 1 /home/pi/bluetooth

[Install]
 WantedBy=multi-user.target
```

Avec ce service nouvellement créé, nous surveillons toutes les connexions bluetooth entrantes de l'extérieur. Une fois le client déconnecté, la commande "watch" redémarre l'écoute des nouvelles connexions entrantes.

Une fois le fichier de configuration de service crée, il faut l'activer pour qu'il soit pris en compte par le système et lancé à chaque démarrage. la commande suivante active le service

```
pi@raspPi3plus:~ $ sudo systemctl enable rfcomm
```

La prochaine fois que la raspberry va redémarrer, elle écoutera toutes les connexions bluetooth entrante.

Nota si vous ne voulez pas redémarrer, vous pouvez démarrer le service manuellement:

```
pi@raspPi3plus:~ $ sudo systemctl start rfcomm
```

On peut contrôler sa bonne marche avec les commandes suivantes :

```
pi@raspPi3plus:~ $ sudo systemctl status rfcomm
● rfcomm.service - RFCOMM service
   Loaded: loaded (/etc/systemd/system/rfcomm.service; enabled; vendor preset:e
   Active: active (running) since Sun 2018-12-30 21:13:04 CET; 13h ago
 Main PID: 299 (rfcomm)
   CGroup: /system.slice/rfcomm.service
           └─299 /usr/bin/rfcomm watch 0 1 /home/pi/bluetooth

déc. 31 08:13:21 raspPi3plus rfcomm[299]: Verrouillage effectué
déc. 31 08:13:21 raspPi3plus rfcomm[299]: /dev/rfcomm0 Vitesse : 9600
déc. 31 08:13:21 raspPi3plus rfcomm[299]: 6 caractères reçus : avant
déc. 31 08:13:21 raspPi3plus rfcomm[299]: 5 caractères reçus : stop
```

C'est fini.
Vous pouvez me laisser un commentaire philippe.simier@ac-nantes.fr