Bluetooth

Qu'est ce que c'est ?

Bluetooth est un protocole sans fil défini par le SIG Bluetooth (Special Interest Group). Ce groupe a été créé en 1998 par de grandes entreprises du secteur des télécommunications (Ericsson, Nokia, Toshiba, Intel, IBM) et compte aujourd'hui plus de 2000 membres dont Agere, Microsoft, Motorola ou encore 3com.

Bluetooth peut être vu comme un USB sans fil. Il sert à créer des PAN (Personnal Area Network), c'est à dire en langage civilisé raccorder des équipements proches les uns des autres.

Quelques exemples de périphériques qui peuvent être raccordés via bluetooth: Ordinateurs personnels, PDA, téléphones, imprimantes, claviers, souris, oreillettes ...

Un réseau bluetooth s'étend généralement sur une dizaine de mètres, pas plus. Au delà, on choisit une mise en réseau plus classique, quitte à faire appel au WiFi si on ne veut définitivement pas utiliser de câbles.

Bluetooth est naturellement destiné à être présent sur des périphériques nomades, donc fonctionnant sur batterie. Le protocole a été pensé dans ce sens et inclut des solutions pour minimiser la consommation d'énergie.

Bluetooth utilise des ondes radio pour sa communication, dans la bande 2.4GHz, appelée ISM (Industrial Scientific and Medical) band, libérée dans la plupart des pays du monde et ce sans avoir à s'acquitter de droits de licence, dans la plage 2.4-2.4835GHz. Cette plage étant librement utilisable, elle est encombrée par de nombreux autres protocoles (WiFi, HomeRF parmi les plus connus). Elle se trouve également polluée par les fours à micro-ondes et les moteurs de certains scooters :).

Pour contrer ces possibilités d'interférences, la plage de fréquence disponible est découpée en canaux. Bluetooth utilise ces canaux aléatoirement et en change fréquemment (toutes les 625ms en mode connecté, toutes les 312.5ms lors d'une tentative de connexion). Un paquet non arrivé à destination à cause d'une interférence sera donc renvoyé sur une fréquence différente et aura ainsi des chances d'arriver à bon port. C'est ce qu'on appelle le frequency hopping (sauts de fréquence), Bluetooth l'utilise sur 79 canaux différents.

Bluetooth met également l'accent sur la sécurité des échanges. La sécurisation passe par deux étapes, l'identification et l'authentification, via un échange de code PIN (Personnal Identification Number) puis le cryptage éventuel de données via la création de clés de liaisons partagées par les deux périphériques. Ceci s'appelle le Pairing ou couplage.

Enfin, au même titre que l'USB, la norme Bluetooth définit des profils (Oreillette, Accès au LAN, transfert de fichier, ...). L'utilisation de profils permet notamment la distribution d'un pilote par profil (USB storage) et non par matériel (carte vidéo). Ces derniers correspondent aux services que peut offrir un périphérique bluetooth donné. La norme définit également un protocole de recherche de services distants et de déclaration des services locaux.

Un réseau bluetooth comporte toujours un maître et des esclaves (jusqu'à sept esclaves par maître). Ce réseau s'appelle un piconet. Le maître d'un piconet peut également avoir le rôle d'esclave dans un autre piconet, on peut ainsi agrandir la portée et le nombre de périphériques du réseau. On parle alors de scatternet.

Scénario d'utilisation

Pour utiliser Bluetooth, on doit bien évidemment disposer de deux périphériques équipés d'une puce bluetooth. Ensuite, deux cas se présentent, soit les périphériques se connaissent déjà, soit ils sont inconnus l'un pour l'autre.

Périphériques inconnus

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.

Le périphérique initiateur peut alors faire une recherche de services sur les périphériques distants. Il présente ensuite la liste à l'utilisateur ou se connecte tout seul au périphérique distant adapté.

Les périphériques peuvent optionnellement être couplés via l'échange d'un code PIN mais ce n'est pas obligatoire.

Périphériques connus

Si deux périphériques sont couplés, la connexion peut être effectuée immédiatement. Toutefois, dans ce cas, on ne vérifie pas la disponibilité du périphérique distant et la tentative de connexion peut durer plusieurs dizaines de secondes avant de renvoyer un état d'erreur.

Une fois la connexion établie, les périphériques font ce qu'ils ont à faire et, à l'issue du travail, n'importe lequel des deux peut demander la déconnexion.

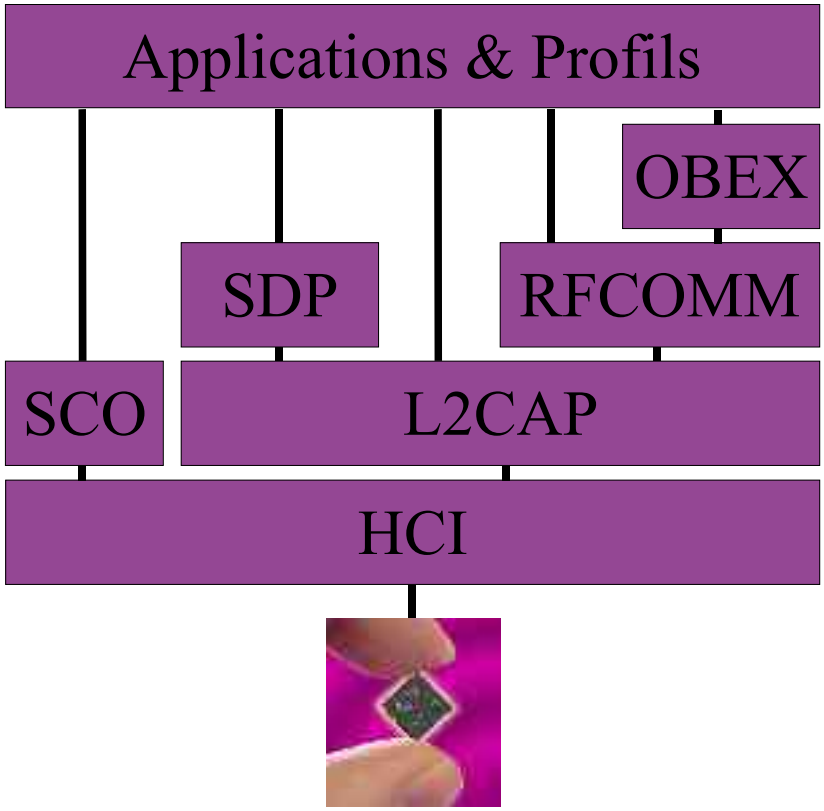
Pile Protocolaire

Chaque périphérique bluetooth est identifié par une adresse, sur 6 octets. Par exemple: 00:A0:96:1F:B6:93. Elle peut être vue comme l'équivalent des adresses MAC des périphériques sur un réseau.

En plus de cette adresse, on peut définir une “classe de périphérique”, codée sur 3 octets. Cette “classe de périphérique” nous informe sur le type de périphérique et éventuellement sur les services qu'il est susceptible de fournir.

Comme pour tout matériel, il faut un driver (pilote de périphérique) pour communiquer avec le module bluetooth. Ce dernier peut être accédé via un port USB, une liaison RS232, via deux UARTs ou toute autre méthode à disposition. Les trois premières possibilités étant les plus largement répandues et définies dans la norme au sections H:2, H:3 et H:4.

Une fois le « pilote » développé, on peut envoyer des octets à la puce bluetooth. Un langage est défini dans le protocole bluetooth, la HCI (pour Host Controller Interface). En utilisant ce langage, sont implémentées les couches supérieures de la « pile » bluetooth (L2CAP, RFCOMM, ...).



Pile Protocolaire – HCI

Une fois que l'on peut communiquer avec le matériel, la méthode de communication est uniformisée quelque soit le type de matériel auquel on a affaire via la couche HCI (Host Controller Interface). Cette interface est exhaustivement définie dans la norme Bluetooth. Elle est composée de commandes et d'évènements. Le périphérique envoie des commandes à la puce bluetooth et reçoit des évènements en retour.

Inquiry, la recherche

Exemple :

Ci-dessous se trouve un exemple d'échange pour une recherche de périphériques par défaut obtenu via le programme hcidump, disponible sous linux :

```
< 01 01 04 05 33 8B 9E 08 00
> 04 0F 04 00 01 01 04
> 04 02 0F 01 3D 86 14 D9 0A 00 01 00 00 0C 22 10 59 43
> 04 01 01 00
```

Décodage :

Les lignes commençant par < 01 sont les commandes envoyées, celles commençant par > 04 sont les évènements reçus. Les octets 01 et 04 définissant commandes ou évènements correspondent au protocole H:4, définissant les types d'échanges HCI lorsque la communication entre le système global et la puce bluetooth se font vie une liaison UART (série).

En utilisant la norme bluetooth, on peut décoder ces trames.

Commande 01 04 05 33 8B 9E 08 00 :

01 04: Commande de recherche de périphériques (Inquiry)
05: Taille des données qui suivent (5 octets)
33 8B 9E: Recherche de type global (par opposition au type local)
08: Temps pendant lequel attendre des réponses à la recherche en multiple de 1.28secondes
00: Nombre de réponses illimitées

Les opcodes définissant les différentes commandes bluetooth sont codées sur deux octets. Elles sont séparées en plusieurs familles et chaque famille fournit plusieurs commandes. On obtient l'opcode en combinant la famille (codée sur 6bits) et la commande (codée sur 10 bits).

La commande “Start Inquiry” (code 1) fait partie de la famille des “LINK CONTROL COMMANDS” (famille 1). Les deux octets de l'opcode contiennent donc les bits suivants:

00000100 00000001 en vert la famille (appelé l'OGF) et en rouge la commande (appelée l'OCF). Les octets sont affichés inversés, d'où un opcode 01 04.

Vient ensuite la taille en octet des paramètres de la commande, ici, 5.

Le type de recherche est ici global. C'est le type le plus utilisé. Ces 3 octets servent à limiter le type de périphérique découvert ou recherché. Si on utilise un code différent de celui utilisé ici, qui signifie recherche globale, on parle d'une recherche limitée. On ne trouvera alors que les périphériques ayant choisi de ne répondre qu'aux recherches faites avec ce même code spécifique. Inversement, un périphérique répond normalement à toutes les recherches utilisant le code global. Il doit être configuré si on souhaite qu'il réponde à des recherches limitées.

On voit ensuite que la recherche doit durer environ 10 secondes (8*1.28s=10.24s)

Le dernier paramètre sert à arrêter la recherche dès qu'on a trouvé un nombre donné de périphériques distants. Ici, on a utilisé 0, qui fait durer la recherche jusqu'à la fin du temps imparti.

Evènement 0F 04 00 01 01 04 :

0F: Indication d'état d'une commande
04: Taille des données qui suivent (4 octets)
00: Commande actuellement en cours
01: Nombre de paquets pouvant être envoyés à la puce avant le prochain évènement
01 04: code de la commande à laquelle cet évènement fait référence

L'évènement 0F signifie “état de la commande”. On trouve ensuite la taille des données qui suivent (ici 4 octets).

Vient ensuite le statut de la commande. Ici, 0 signifie que la commande est en cours (pending).

Le quatrième paramètre nous indique combien de commandes on peut envoyer à la puce bluetooth avant de recevoir une autre autorisation.

Enfin, l'évènement rappelle l'opcode de la commande à laquelle il se rapporte.

Evènement 02 0F 01 3D 86 14 D9 0A 00 01 00 00 0C 22 10 59 43 :

02: Nouveau périphérique trouvé
0F: Taille des données qui suivent (15 octets)
01: Nombre de périphériques
3D 86 14 D9 0A 00: Adresse (à l'envers, du périphérique ayant répondu)
01 00 00: Trois paramètres (Page Scan Repetition Mode, Page Scan Period Mode et Page Scan Mode)
0C 22 10: Classe du périphérique distant (A l'envers)
59 43: Décalage d'horloge

L'évènement 02 est utilisé pour signaler la réponse d'un ou plusieurs périphérique(s) à la recherche.

Comme pour tous les autres évènements, le second octet contient la taille des données qui suivent (ici 15 octets).

L'octet suivant contient le nombre de périphériques ayant répondu contenu dans cet évènement.

On trouve ensuite l'adresse bluetooth du périphérique qui répond, celle-ci est inversée et correspond à l'adresse habituellement notée 00:0A:D9:14:86:3D.

Les trois paramètres qui suivent seront utilisés lors d'une éventuelle demande de connexion.

Les trois octets suivants sont la classe du périphérique distant. Ici, on apprend que l'on a affaire à un téléphone de type smartphone supportant les échanges d'objets (images, vCards).

Les deux octets suivants représentent le décalage d'horloge entre les deux périphériques, il pourra être répété lors d'une demande de connexion pour accélérer cette dernière. Pour savoir si ce décalage est valide, on vérifie que le bit de poids faible est à 1, comme c'est le cas ici.

Evènement 01 01 00 :

01: Recherche de périphériques terminée
01: Taille des données qui suivent (1 octet)
00: Recherche terminée normalement

L'évènement 01 signale la fin d'une période de recherche.

Le statut 00 signale que l'opération s'est passée sans souci particulier.

La connexion ACL

Une fois un périphérique trouvé, on s'y connecte. La connexion avec un périphérique est identifiée par un handle. Voyons comment s'établit une connexion au niveau HCI:

```
< 01 05 04 0D 3D 86 14 D9 0A 00 18 CC 01 0059 43 01
> 04 0F 04 00 01 05 04
> 04 03 0B 00 28 00 3D 86 14 D9 0A 00 01 00
...
> 04 05 04 00 28 00 13
```

Commande 05 04 0D 3D 86 14 D9 0A 00 18 CC 01 00 59 43 01 :

Pour demander une connexion on utilise l'opcode 05 04 suivi de 13 (0D) octets de paramètres.

Tout d'abord, on doit donner l'adresse du périphérique distant auquel on veut se connecter (en retournant les octets). Ici, l'appareil trouvé lors de la recherche, d'adresse 00:0A:D9:14:86:3D.

Les deux octets suivants permettent de spécifier le type de paquet que l'on pourra utiliser sur cette connexion. Bluetooth peut utiliser 6 types de paquets de tailles différentes. Selon le paquet utilisé le débit obtenu sera plus ou moins élevé. On spécifie les types de paquets utilisables en les combinant via des OU logiques. Dans cette demande on spécifie que l'on accepte l'utilisation les types de paquets DM1, DH1, DM3, DH3, DM5 et DH5, c'est à dire, tous les paquets de type “données”. Il existe en effet 3 autres types de paquets, non utilisables avec cette commande, qui servent dans les liens de type SCO.

Tableau récapitulatif des différents types de paquets:

DM1	0x0008
DH1	0x0010
HV1	0x0020
HV2	0x0040
HV3	0x0080
DM3	0x0400
DH3	0x0800
DM5	0x4000
DH5	0x8000

On répète ensuite, si possible, les paramètres Page Scan Repetition Mode et Page Scan Mode obtenus pendant la phase de recherche, dans la réponse du périphérique distant.

Vient ensuite le décalage d'horloge, lui aussi reçu lors de la recherche.

Ces deux paramètres ne sont pas tenus d'être exacts mais s'ils sont fournis et corrects, le temps de connexion s'en trouve diminué.

Le dernier octet indique si le périphérique demandant la connexion (qui deviendra maître, par défaut) acceptera par la suite un changement de rôle.

Suite à l'envoi de cette commande, on reçoit un évènement OF indiquant le statut de la commande (en cours, ici).

Evènement 03 0B 00 28 00 3D 86 14 D9 0A 00 01 00 :

On obtient, après un laps de temps variable, l'évènement indiquant si la connexion a réussi ou non (Evènement 03). Après la taille des données (0B) on reçoit le statut de la connexion (00 signifie que la connexion est établie) et le handle de la connexion au niveau HCI, ici 28 00, qui vaut 40.

Après ce handle est répété l'adresse du périphérique auquel on est à présent connecté.

L'octet suivant spécifie le type de connexion, 00 pour une connexion SCO, 01 pour une connexion ACL. Les connexions SCO sont utilisées pour les données multimédia (audio, vidéo, ...). Les connexions ACL sont utilisées pour les données. Il s'agit ici d'une connexion ACL, visant à transmettre des données. Une connexion ACL est nécessaire entre deux périphériques, même si ceux ci veulent ensuite créer une liaison SCO pour l'échange de données audio, comme dans le cas d'une oreillette et d'un téléphone par exemple.

Enfin, le dernier octet précise type de cryptage utilisé pour cette connexion (00 signifiant “pas de cryptage”).

Après réception d'un tel évènement, on dispose d'une connexion ACL, élément indispensable à la connexion bluetooth entre deux périphériques.

On peut ensuite envoyer des paquets de données, pour, par exemple créer une connexion L2CAP ou une connexion SCO.

En résumé, la couche HCI du protocole bluetooth permet la communication avec la puce, le paramétrage local, la mise en place des connexions entre dispositifs et le paramétrage de ces dernières. C'est également grâce à lui que l'on peut mettre en place l'authentification et le cryptage des connexions et gérer les modes d'économie d'énergie partagés.

On peut envoyer des paquets de données sur une liaison ACL mais la norme ne prévoit pas de pouvoir établir une communication grâce à ces derniers. On doit donc pour cela les utiliser pour implémenter un protocole supérieur tel que le L2CAP.

Evènement 05 04 00 28 00 13 :

Cet évènement (05) signale une déconnexion.

Il y a 4 octets de paramètres.

Le premier (00) signifie que la déconnexion est effective.

Les deux octets suivants indiquent la connexion ACL pour laquelle la déconnexion est intervenue.

Le dernier octet indique la raison de la déconnexion. 13 signifie “Other End Terminated Connection: User Ended Connection”, c'est à dire une déconnexion normale, dont le périphérique distant a pris l'initiative. Si le périphérique local avait été à l'origine de cette déconnexion, cet octet aurait eu la valeur 16.

Pile Protocolaire – 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.

C'est également en utilisant le L2CAP que sont implémentées les plus hautes couches du protocole bluetooth tels que le SDP (pour les protocole de recherche de services) ou RFCOMM (qui a pour but l'émulation d'une liaison série entre deux dispositifs bluetooth).

Le L2CAP est un protocole relativement simple à mettre en oeuvre. Bien qu'il existe un mode non connecté au L2CAP pouvant être utilisé pour l'envoi de paquets en diffusion (broadcast) sur le réseau bluetooth, on l'utilise majoritairement en mode connecté.

Une connexion L2CAP est définie par :

- Le handle de la connexion ACL sous-jacente. Ce handle est attribué par les couches basses du protocole. C'est sur ce handle la couche L2CAP envoit ses paquets.
- Deux identifiants de “canal” L2CAP, un identifiant sur le périphérique local et un identifiant sur le périphérique distant. Ses canaux sont utilisés par les couches supérieures pour envoyer des trames. Le canal local de chaque périphérique est attribué par la couche L2CAP de chauqe périphérique.
- Un PSM (Protocol Service Multiplexer). Le PSM identifie le type de données transitant sur cette connexion L2CAP. Il existe quelques valeurs prédéfinies (01 pour SDP, 03 pour RFCOMM, par exemple). Le PSM à utiliser peut également être choisi arbitrairement ou être fourni par le biais du protocole SDP.
- La configuration de la connexion (MTU notamment mais aussi, Qualité de service “QoS” ou temps avant transfert des données en tampon “flush timeout”)

Un canal spécial est utilisé pour la mise en place, la configuration, le test et la fermeture des connexions L2CAP, il s'agit du canal 1. Le canal 2, quant à lui est réservé au mode non connecté. Les canaux supérieurs à 64 (40 en hexadécimal) sont libres pour l'utilisation “normale”.

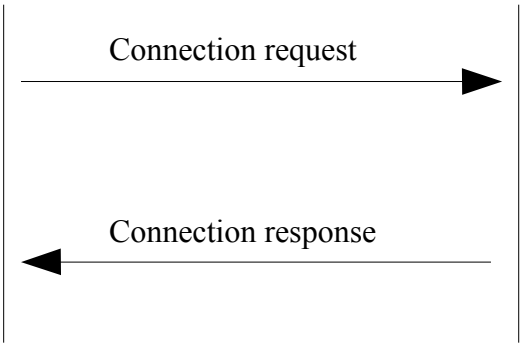
Le cycle de vie d'une connexion L2CAP est composé d'une connexion, d'une configuration de la connexion, d'échanges de données et d'une déconnexion. Ces phases sont détaillées ci-dessous:

Connexion

La connexion se décompose en une requête et une réponse.

La requête contient le PSM à utiliser et le numéro de canal qui sera utilisé pour cette connexion par le périphérique envoyant cette requête.

La réponse contient le numéro de canal utilisé pour cette connexion par le périphérique envoyant cette réponse, le numéro de canal utilisé sur le périphérique initiateur est ensuite répété. Deux octets indiquent la réponse (soit la connexion est acceptée, soit elle est mise en attente, soit elle est refusée). Deux octets supplémentaires permettent de spécifier la raison de la mise en attente éventuelle.



Configuration

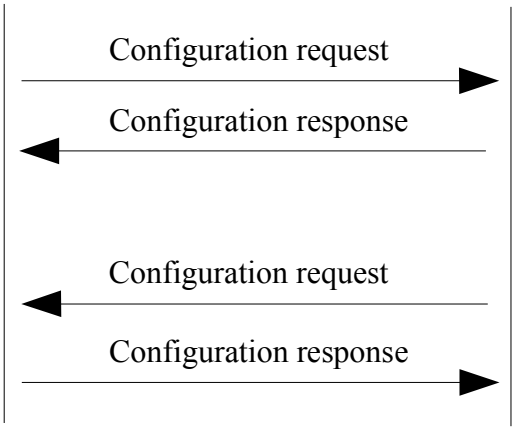
La configuration se fait en un minimum de deux échanges de deux requêtes/réponses.

Un des périphériques envoie une requête. L'autre envoie une réponse acceptant ou non ces paramètres. Si un des paramètres est refusé par la réponse, cette dernière peut contenir des valeurs qui auraient été acceptées., le périphérique ayant envoyé la requête peut en renvoyer en seconde pour accepter ces nouveaux paramètres ou mettre fin à la connexion, en cas de refus.

Le même échange intervient alors en échangeant les rôles.

La raison de cette « double configuration » est de permettre la configuration de la communication dans les deux sens. Chaque échange requête/réponse ne fixant les paramètres que pour un « sens » de communication.

On peut ainsi, par exemple, utiliser un mtu plus important dans un sens que dans l'autre, pour des raisons de tailles de buffers de réception et d'émission différentes.



Echanges de données

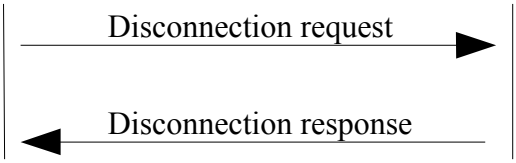
Une fois la connexion établie (connexion et configuration effectuée), les périphériques peuvent s'envoyer des trames contenant des données.

Le format des trames de données est le même que celui des trames de commandes et est explicité plus loin.

Déconnexion

La déconnexion, comme la connexion est composée d'une requête et d'une réponse.

La connexion ACL peut être ou non supprimée après une déconnexion L2CAP mais ce n'est pas automatique.



Exemple:

Entête HCI :

Avant de décortiquer un échange au niveau L2CAP, on doit se familiariser avec le format des trames de données qui transitent au niveau HCI ...

Ces trames sont préfixées par un 02 dans le transport H:4, déjà évoqué et qui est utilisé pour l'affichage des dumps sous linux, via l'outil hcidump.

Le format général d'une trame est le suivant:

Handle ACL	flags	Longueur des données	Données
12 bits	4 bits	16 bits	n bits
02 hh hf ll ll dd dd dd dd dd			

hhh représente le handle de la connexion ACL utilisée. Elle est codée sur 12 bits.

f représente deux flags de 2 bits chacun. Il s'agit du flag BC pour broadcast, la plupart du temps, ce dernier vaut 00, qui signifie qu'on envoie les données d'un périphérique à un autre, en mode point-à-point, sans diffusion (broadcast). L'autre flag est PB (packet boundary) qui est utilisé pour spécifié s'il s'agit d'un début de paquet ou paquet entier (valeur 10) ou de la suite d'un paquet précédent (01).

ll ll spécifie le nombre d'octets de données qui suivent.

La série de dd sont les octets de données. Dans le cas qui nous intéresse, ces données seront formatées selon le protocole L2CAP.

Format des trames L2CAP :

Les trames L2CAP ont toutes la même forme. Elles commencent par un entête de 4 octets (deux octets pour la taille des données et deux octets pour l'identifiant de canal utilisé).

Longueur	Canal	Données
16 bits	16 bits	n bits

```
ll ll cc cc dd dd dd dd ....
```

ll ll représente la taille des données (la série de dd)

cc cc représente l'identifiant de canal utilisé.

Dans le cas de paquets de données, tous les dd dd de la trame ci-dessus sont des octets de données.

Dans le cas de paquets de commandes les données sont composées :

1. du code de la commandes (sur un octet)
2. d'un identifiant qui devra être répété dans la réponse (sur un octet)
3. de la taille de ce qui suit (sur deux octets)
4. des paramètres de la commande

On peut ainsi envoyer plusieurs commandes à la suite, dans la même trame en répétant ce schéma.

Connexion L2CAP :

```
< 08 00 01 00 02 01 04 00 cc ee 40 00
> 0c 00 01 00 03 01 08 00 50 00 40 00 00 00 00 00
```

On envoie d'abord la demande de connexion (08 octets de données), sur le canal de commande (ccee). Ces 8 octets de données sont composées de l'identifiant de la commande de demande de connexion (02), d'un identificateur de commande (01), de la taille des paramètres (04 octets) du PSM à utiliser (0100) et de l'identifiant du canal alloué à cette connexion sur le périphérique local (4000).

La réponse (de 12 octets) arrive sur le canal de commande (0100). Elle est composée de l'identifiant de la commande de réponse à une demande de connexion (03) et rappelle l'identificateur de commande (01). Cette réponse utilise huit octets (0800) de paramètre. Elle informe de l'identifiant de canal alloué sur le périphérique (5000), rappelle celui du périphérique initiateur de la connexion (4000) et envoie les octets signifiant un bon déroulement de la connexion (quatre fois 00).

Configuration L2CAP :

```
< 08 00 01 00 04 02 04 00 50 00 00 00
> 16 00 01 00 05 02 06 00 40 00 00 00 00 00 04 03 08 00 40 00 00 00 01 02 4e 00
< 0a 00 01 00 05 03 06 00 50 00 00 00 00 00
```

On envoie une demande de configuration (8 octets, sur le canal 1). L'identifiant de commande est 4 est l'identificateur de commande passe à 2 (la connexion utilisait 1). Les 4 octets qui suivent rappellent le numéro de canal alloué pour la connexion sur le périphérique distant et deux octets nul qui sont des drapeaux utilisés en cas de fragmentation de paquets (ce qui n'est pas le cas ici). La configuration est vide, c'est à dire qu'on demande à utiliser paramètres les paramètres par défaut.

La trame de réponse (22 octets sur le canal 1) contient une réponse de configuration (identifiant 05) bâtie sur le même principe de la demande (tous octets à 0) mais y ajoutant une réponse (deux octets à 00, signifiant « Success », et signifie une acceptation de la configuration.

On trouve immédiatement à la suite la demande de configuration pour l'autre direction. Elle est construite sur le même principe (rappel du canal, octets de continuation de paquets à 00 00) mais contient une directive de configuration à sa suite (01 02 4e 00). 01 signifie que le périphérique veut changer le MTU des trames qui lui sont destinées. 02 est la taille des paramètres qui suivent, en octets. Ces derniers fixent un MTU de 78 octets (4e00).

Le périphérique local envoie alors sa réponse à la requête de configuration et en accepte les paramètres.

A partir de ce moment là les périphériques peuvent s'envoyer des données.

Echange de données :

On ne connaît pas le protocole utilisé ensuite à priori qui est propre à chaque application. Il peut s'agir de l'implémentation d'un protocole des couches supérieures de bluetooth ou d'un protocole partagé par les deux périphériques.

Voici un exemple d'échange de données arbitraires :

```
< 03 00 50 00 02 04 05
> 03 00 40 00 02 05 04
```

On reconnaît les en-têtes L2CAP (2 octets de longueur, 2 octets de canal).

Chaque périphérique répète le numéro employé **par l'autre** pour identifier le canal de données en entête de trame, après la longueur des données transmises.

On peut supposer ensuite que les données sont composées d'un octet spécifiant la taille des données et que le rôle du périphérique distant est de renvoyer une version inversée des données transmises par le périphérique local.

Déconnexion L2CAP :

```
< 08 00 01 00 06 04 04 00 50 00 40 00
> 08 00 01 00 07 04 04 00 50 00 40 00
```

Le périphérique local demande une déconnexion (identifiant de commande 06, disconnection request). La commande porte le numéro identificateur 04 et utilise 4 octets de paramètres. Les deux premiers octets identifient le canal à fermer sur le périphérique distant. Les deux suivant identifient le canal à fermer sur le périphérique local.

Le périphérique distant répond via la commende utilisant l'identifiant 07 (Disconnection response) et répète les paramètres envoyés par le périphérique local.

Pile Protocolaire – SDP

Présentation

Le SDP (Service Discovery Protocol) définit le protocole utilisé pour permettre à un « client » d'interroger un « serveur » sur les « services » qu'il propose. La réponse contient spécifie si le service existe ou non. Si ce dernier existe, la réponse contient les prérequis à l'utilisation du services, ses « attributs ». Notamment, on peut déterminer quelles couches de la pile bluetooth seront utilisées pour l'utilisation de ce service et les paramètres à utiliser pour chacune (PSM pour L2CAP, Canal pour RFCOMM, ...).

Le SDP lui même nécessite une connexion L2CAP pour fonctionner. Le PSM réservé pour les communication liées au SDP est le 1.

Chaque périphérique bluetooth peut embarquer un « serveur » sdp et/ou être un « client ».

Dans une utilisation « classique » de bluetooth, on trouve, dans l'ordre chronologique :

- Une recherche de périphériques
- Une connexion ACL
- Une connexion L2CAP
- Une recherche SDP
- Une déconnexion L2CAP
- Une déconnexion ACL
- Une connexion ACL
- Les connexions supplémentaires nécessaire à l'utilisation du service « intéressant » (L2CAP/RFCOMM/Protocole de niveau supérieur comme TCP)
- L'utilisation du service
- La déconnexion des différentes couches, de la plus haute à la plus basse (par exemple TCP puis RFCOMM puis L2CAP puis ACL)

Théoriquement, une connexion différente de celle utilisée pour le SDP doit être crée pour l'utilisation d'un service trouvé par ce biais. (Norme Bluetooth 1.1, Partie E)

Définitions

Le SDP définit le format des enregistrements définissant les services. Un serveur SDP tient à jour une « base de données » locale contenant ces enregistrements (Service Record).

A chaque enregistrement est associé un « handle », codé sur 32 bits, le ServiceRecordHandle (Attribut 0 de chaque enregistrement dans un serveur SDP). Ce handle est propre à chaque serveur SDP. A titre d'exemple, un enregistrement ayant pour handle 0X3ac7 sur un serveur S1 ne représente par forcément le même service qu'un enregistrement ayant le même handle sur une serveur S2.

Exception: Chaque serveur SDP possède un enregistrement dont le handle est 0, qui définit le serveur SDP lui même et dont un des attributs fournit notamment la version du protocole utilisée par ce serveur. Par ailleurs, les handles inférieurs à 0x0000FFFF sont réservés.

Pour assurer que l'« état » de la base de données du SDP est valable, on définit une variable, que le serveur SDP met à jour à chaque modification de la base. C'est le ServiceDatabaseState. Une variable d'état équivalente existe pour chaque enregistrement, c'est le ServiceRecordState.

Chaque enregistrement est entièrement (uniquement) composé d'une liste d'« attributs ». Un attribut est composé d'un identifiant et d'une valeur.

Quelques exemples d'identifiant:

- ServiceClassIDList (id 1): liste de types de service implémentés par le service représenté par l'enregistrement (représentés par des UUID, Universally Unique Identifier).
- ServiceID (id 3): service implémenté par le service représenté par l'enregistrement (représenté par un UUID, Universally Unique Identifier).
- ProtocolDescriptorList (id 4): protocoles nécessaires à l'utilisation du service représenté par cet enregistrement.
- LanguageBaseAttributeIDList (id 6): définit une liste de langue et les identifiants de base pour tous les attributs dépendant de la langue utilisée.
- ServiceName (id variable selon la langue utilisée, id de base pour la langue): nom du service représenté par cet enregistrement.
- ServiceDescription (id variable selon la langue utilisée, id de base pour la langue + 1): description du service représenté par cet enregistrement.
- ...

De plus, les identifiants d'attributs compris entre 0x000D et 0x01FF sont réservés.

Une valeur d'attribut est une suite d'octets dont la signification dépend de son identifiant.

Aussi bien les identifiants que les valeurs respectent un même codage, composé comme suit:

- Une valeur codée sur 5 bits, qui représente le type de la valeur de cet attribut, selon le tableau suivant:

0	Nil, type NUL
1	Entier non signé
2	Entier signé
3	UUID
4	Chaîne de caractères
5	Valeur booléenne
6	Séquence d'éléments
7	Liste d'éléments (un seul valide)
8	URL

- une valeur codée sur 3 bits spécifiant la taille des données

0	1 octet de données (ou 0 dans le cas du type nil)
1	2 octets de données
2	4 octets de données
3	8 octets de données
4	16 octets de données
5	La taille est contenue dans l'octet qui suit
6	La taille est contenue dans les deux octets qui suivent
7	La taille est contenue dans les 4 octets qui suivent

- les données elles mêmes

Protocole de communication

Comme je l'ai écrit précédemment, le protocole SDP utilise une connexion L2CAP avec le PSM 1. Le protocole SDP est entièrement contenu dans des trames de données L2CAP, en mode connecté.

Les échanges SDP passent par des PDU (Protocol Data Unit)

Un PDU SDP est constitué:

- d'un octet pour l'identifiant de PDU
- de deux octets identifiant la transaction
- de deux octets spécifiant le nombre d'octets de paramètres qui suivent
- des paramètres

ID PDU	ID Transaction	Longueur	Paramètres
8 bits	16 bits	16 bits	n bits

Le SDP utilise les identifiants de PDU suivants:

0 et 7 à FF	Réservés
1	SDP_ErrorResponse
2	SDP_ServiceSearchRequest
3	SDP_ServiceSearchResponse
4	SDP_ServiceAttributeRequest
5	SDP_ServiceAttributeResponse
6	SDP_ServiceSearchAttributeRequest
7	SDP_ServiceSearchAttributeResponse

A l'instar de l'identifiant de commande utilisé sur la canal 0 dans le protocole L2CAP, l'identifiant de transaction identifie une requête SDP. La réponse à une requête donnée doit répéter son identifiant.

Le « client » commence par rechercher un ou plusieurs services sur le « serveur » (ServiceSearchRequest & ServiceSearchResponse). Pour les services qui l'intéressent, le « client » demande la liste des attributs de l'enregistrement correspondant (ServiceAttributeRequest & ServiceAttributeResponse). Enfin, le « client » récupère la valeur des attributs qui l'intéressent (ServiceSearchAttributeRequest & ServiceSearchAttributeResponse).

Les recherches se font via des UUID, identifiant uniques codés sur 128 bits

A propos des UUID

Un UUID est codé sur 128 bits. Toutefois, la norme bluetooth utilise souvent des UUID codés sur 16 ou 32 bits ... Pour retrouver les 128 bits dans ce cas, on applique une des formules suivantes, selon la taille de l'UUID dont on dispose:

```
128_bit_value = 16_bit_value * 2^96 + Bluetooth_Base_UUID
128_bit_value = 32_bit_value * 2^96 + Bluetooth_Base_UUID
```

Le Bluetooth_Base_UUID vaut 00000000-0000-1000-8000-00805F9B34FB. Certains UUID sont fixés

par la norme bluetooth. On les retrouve dans les définitions des protocoles les utilisant ou dans les « bluetooth assigned numbers ».

Par exemple, le service Serial Port possède l'UUID sur 16bits suivant: 1101 (en héxadécimal). Ce qui correspond à l'UUID 00001101-0000-1000-8000-00805F9B34FB.

Pile Protocolaire – RFCOMM

RFCOMM est un protocole de remplacement de câble définissant l'émulation de ports série. RFCOMM utilise une partie de la norme TS 07.10 (aussi connue sous l'appellation ETSI TS 101 369).

Principe général

Pour établir une connexion RFCOMM entre deux périphériques et donc simuler une liaison par port série entre eux, on doit tout d'abord créer une liaison L2CAP. Le PSM réservé pour le RFCOMM est le 3.

RFCOMM permet d'émuler les signaux de contrôle d'une liaison RS232 (TD, RD, RTS, CTS, DSR, STR, CD, RI).

RFCOMM permet de simuler une ou **plusieurs** liaisons entre deux périphériques équipés d'une puce bluetooth. Le protocole a été défini pour supporter jusqu'à 60 liens virtuels.

Une liaison virtuelle est identifiée par un « dlci » (Data Link Connection Identifier) codé sur 6 bits mais dont on n'utilise que les valeurs comprises entre 2 et 61. Le dlci 0 correspond à un canal de contrôle permettant notamment l'établissement de nouveaux canaux virtuels et les valeurs 1, 62 et 63 sont inutilisables.

La notion de dlci est divisée en deux. Le premier bit servant à identifier le périphérique selon qu'il est ou non l'initiateur de la session est les suivants servant à la définition du canal RFCOMM.

RFCOMM utilise pour le contrôle du flux un mécanisme de jetons. Tant qu'un périphérique a des « jetons », il peut envoyer des trames à l'autre. Chaque périphérique réaffecte des jetons à l'autre au fur et à mesure du traitement des données transmises.

Trames

Une fois la connexion L2CAP établie avec le PSM 3. Les périphériques doivent créer une session RFCOMM, c'est à dire créer le canal 0.

Pour créer un canal le périphérique initiateur utilise la commande SABM (Set Asynchronous Balanced Mode) avec un paramètre dlci = 0.

Le périphérique distant marque son acceptation en envoyant la commande UA (Unnumbered Acknowledgement).

On dispose alors du canal de contrôle du multiplexeur nécessaire à la session RFCOMM et on peut créer des canaux de données selon une procédure similaire.

Les paramètres du canal à créer sont négociés via un échange utilisant des trames de type PN CMD et PN RSP (Parameter Negotiation Command et Parameter Negotiation Response). Cet échange est obligatoire lors de l'établissement du premier canal de données et fortement recommandé, bien qu'optionnel pour les canaux suivants. En effet ce premier échange permet d'attribuer un nombre initial de crédits à chaque périphérique.

La création du canal est rigoureusement identique et utilise les trames de type SABM et UA.

Avant de pouvoir échanger des données, les périphériques s'échangent des trames de types MSC (Modem Status Command) spécifiant l'état des « signaux de contrôle ».

Les échanges de données ainsi que l'attribution de crédits se font ensuite via des trames UIH (Unnumbered information with header check).

Une fois la connexion RFCOMM établie, on dispose d'un port série. On peut donc l'utiliser pour « brancher »une souris, un modem, pour connecter une console ou l'utiliser dans une application utilisant une connexion série pour l'acquisition de données, par exemple.

Pile Protocolaire – Profils

A la sortie de Bluetooth 1.1, les profils suivants étaient définis :

- GENERIC ACCESS PROFILE
- SERVICE DISCOVERY APPLICATION PROFILE
- CORDLESS TELEPHONY PROFILE
- INTERCOM PROFILE
- SERIAL PORT PROFILE
- HEADSET PROFILE
- DIAL-UP NETWORKING PROFILE
- FAX PROFILE
- LAN ACCESS PROFILE
- GENERIC OBJECT EXCHANGE PROFILE
- OBJECT PUSH PROFILE
- FILE TRANSFER PROFILE
- SYNCHRONIZATION PROFILE

Aujourd'hui, le site du SIG recense pas moins de 56 profils.

Chaque profil est défini dans la norme Bluetooth. Les spécifications incluent le ou les protocoles nécessaires à son fonctionnement et la façon dont il doit être enregistré dans un serveur SDP.

Chaque application développée est l'équivalent d'un profil bluetooth, avec son protocole particulier et ses ressources nécessaires.

Les interfaces de programmation

Différents niveaux de développement

Le « développement bluetooth » peut prendre plusieurs formes. On peut développer des applications utilisant certaines couches de la pile ou développer des couches de la pile elle-même.

L'interface avec la puce bluetooth est la HCI. On peut déveopper tout et n'importe quoi en utilisant cette interface.

Packages nécessaires:

Vous aurez besoin pour pouvoir utiliser et développer des applications bluetooth sous linux des paquets suivants.

- bluez-libs
- bluez-utils
- bluez-hcidump
- bluez-pin
- bluez-firmware
- bluez-hciemu

Votre distribution les propose surement mais si ce n'est pas le cas, on peut les télécharger sur http://www.bluez.org/ et les compiler soi-même. Vous devrez aussi avoir activé le support pour bluetooth dans votre kernel.

Initialisation:

Une fois les prérequis obtenus, on peut faire connaissance avec la bête. Ci dessous se trouve le code pour initialiser et rendre découvrable et connectable un ordinateur équipé d'un dongle bluetooth sur son port usb.

```
ZazouMobile:/home/zazou# modprobe hci-usb
ZazouMobile:/home/zazou# hciconfig hci0 up piscan
ZazouMobile:/home/zazou# hciconfig hci0 -a
hci0:   Type: USB
        BD Address: 00:A0:96:1F:B6:93 ACL MTU: 128:8  SCO MTU: 64:8
        UP RUNNING PSCAN ISCAN
        RX bytes:113 acl:0 sco:0 events:12 errors:0
        TX bytes:35 acl:0 sco:0 commands:9 errors:0
        Features: 0xff 0xff 0x05 0x00 0x00 0x00 0x00 0x00
        Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
        Link policy:
        Link mode: SLAVE ACCEPT
        Name: 'bluez'
        Class: 0x000000
        Service Classes: Unspecified
        Device Class: Miscellaneous,
        HCI Ver: 1.1 (0x1) HCI Rev: 0x72 LMP Ver: 1.1 (0x1) LMP Subver: 0x72
        Manufacturer: Cambridge Silicon Radio (10)
```

On retrouve une interface semblable à celle fournie par l'outil ifconfig avec notamment l'adresse du périphérique, les statistiques de transmission et les paramètres propres à bluetooth.

Utilisation:

Développement: