Communications de données et d'images issues de la carte Fox par Bluetooth

G. Weisenhorn, E. Pamba Capo-Chichi, J.-M. Friedt

8 mai 2007

Résumé

Nous présentons ici l'évolution d'une plate-forme en cours de développement combinant autour d'un processeur de consommation modeste un certain nombre de capteurs (consommation électrique, position, température) et de modules de communication sans fil dans le but d'obtenir un outil de base pour la réalisation d'un réseau de capteurs mobiles. Nous abordons ici l'aspect le plus coûteux en ressources de calcul et de bande passante : la transmission d'images sans fil. Il s'agit en effet ici de l'application la plus gourmande en puissances électrique et de communication, justifiant l'utilisation d'un processeur puissant pour la compression et la transmission des données, au delà de laquelle la transmission de données scalaires (courant consommé, latitude et longitude du capteur, quantités physiques) n'apparaissent que comme des compléments à la charge associée à la communication d'une image. Nous avons sélectionné la communication par réseau bluetooth comme compromis entre une bande passante modeste mais aussi une consommation électrique raisonnable, un réseau de petites dimensions compatibles avec la géométrie du réseau que nous envisageons de réaliser, et surtout la disponibilité à faible coût de matériel supportant ce protocole.

1 Présentation

L'observation continue de notre environnement est une condition nécessaire si l'on vise à en comprendre l'évolution et en appréhender les modifications sur le long terme. La majorité des sites dignes d'intérêt n'est pas munie de source inépuisable d'énergie, est souvent située en environnement hostile pour un circuit électronique, et ne permet pas une communication filaire aisée. Nous désirons proposer une plate-forme formant un nœud d'un réseau sans fil pour l'observation de l'environnement, qu'il s'agisse d'acquisition de quantités physiques scalaires ou d'images plus riches en informations mais nécessitant beaucoup plus de ressources pour leur stockage et communications. Nous avons basé notre travail sur un assemblage de composants disponibles commercialement au grand public dans un souci de généralité de notre travail et de réduire le coût de chaque nœud, ainsi que pour faire évoluer notre application au même rythme que l'électronique grand public en profitant des gains en performance et en consommation associées aux évolutions technologiques.

2 La carte Fox

La carte Fox est un support pour le développement de systèmes embarqués, développée par ACME [1], sur lequel tourne un système d'exploitation GNU/Linux (noyaux 2.4 ou 2.6). Ce système fonctionne avec un microprocesseur Axis ETRAX 100LX fournissant 100 MIPS [2] pour une consommation de l'ordre de 1,5 W. Cet environnement, illustré dans la Fig.3 supporte de nombreuses applications telles que par exemple un serveur web, des interfaces réseau (ethernet, wifi, bluetooth, GSM), l'ensemble des développements se faisant sur des outils opensource (SDK for Linux Systems)...

La série de la carte avec laquelle nous travaillons est la carte Fox LX 416 (4 MB de flash et 16 MB de RAM). C'est sur cette carte que nous allons greffer les dispositifs qui nous intéressent pour notre projet, capteurs, écran LCD (Fig. 6), GPS, communication sans fil ...

La mise en place du SDK et les méthodes de compilation ont déjà été largement développées par ailleurs [3]. Noter cependant que les outils évoluent rapidement en ce moment : il faut donc régulièrement vérifier les mises à jour sur le serveur CVS de ACME et notamment les diverses options de compilation en fonction des déclinaisons de la carte Fox.

Nous utiliserons sur notre carte un noyau 2.6.15 avec la librairie C uClibc par souci d'économie de place. Les étapes de configuration sont les suivantes :

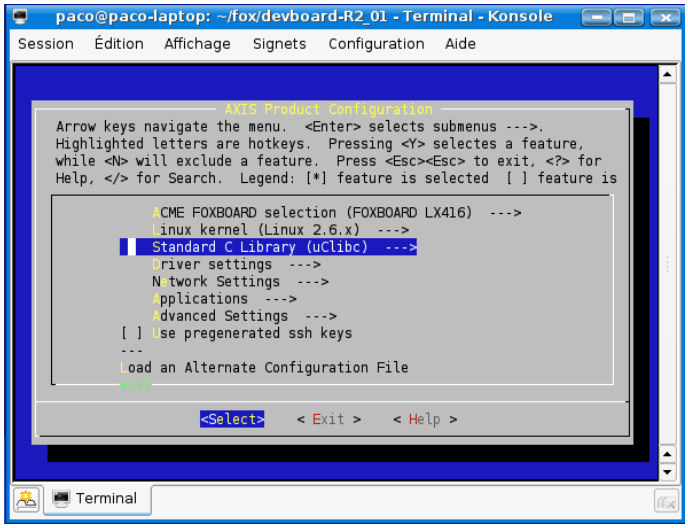

Fig. 1 – Le menu de configuration

Choisir le bluetooth dans Divers settings du menu principal (Fig. 1), *Attention les options Openobex et ussp-push peuvent avoir quelques problèmes pendant la compilation* Ensuite fermer votre menu sans oublier de sauvegarder (Fig.2) Pour générer l'image nommée `fimage` qu'on va flasher sur notre carte fox il nous suffit de taper les commandes suivantes :

```
./configure
make
```

Une fois `fimage` créée, ce fichier binaire est transféré par le réseau local (les cartes Fox ont par défaut l'adresse IP 192.168.0.90). La carte est mise en écoute

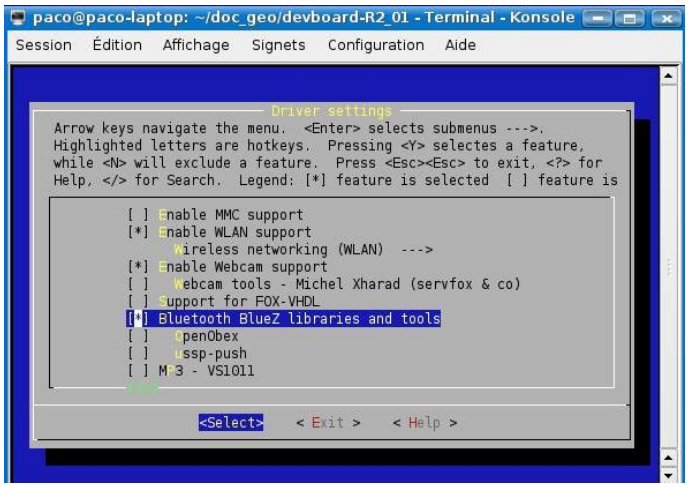

Fig. 2 – Le bluetooth dans le menu

de la nouvelle image en fermant le jumper J8 (Fig. 6) : le transfert se fait alors du côté PC par `. init_env`
`boot_linux -F -i fimage`

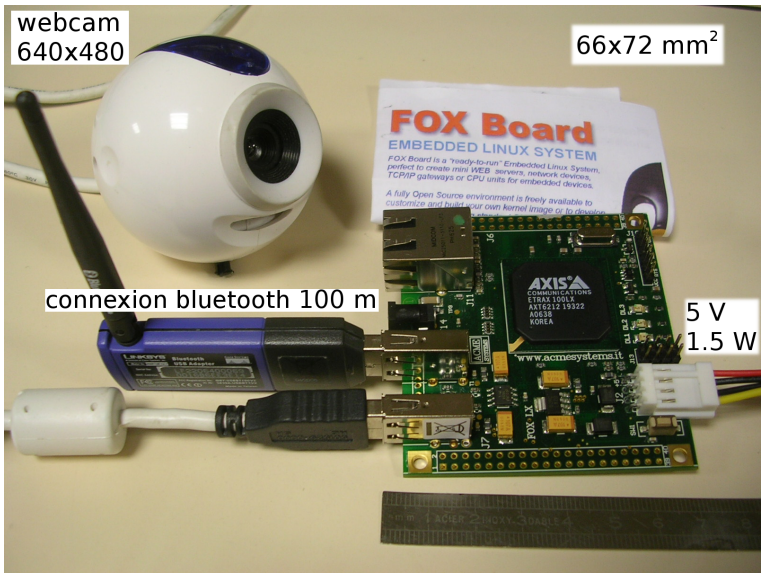

Fig. 3 – Un des circuits décrits dans cet article : la carte Fox est ici équipée d'une webcam QuickCam Zoom et d'une clé de conversion USB-bluetooth. La consommation globale de ce montage est de l'ordre de 370 mA sous 5 V une fois les périphériques activés.

Une fois le kit installé, la définition des variables d'environnement nécessaires à la cross-compilation des outils pour le processeur ETRAX s'obtiendra toujours

en se plaçant dans le répertoire `devboard-R_01` et en exécutant :
`. init_env`
Prepending "/..../devboard-R2_01/tools/build/bin" to PATH.
Prepending "/usr/local/cris/bin" to PATH.

3 Support du bluetooth

3.1 Les réseaux locaux sans-fil

Dans le cadre du réseau de capteurs, nous nous sommes interressés à un certain type de communication : les réseaux locaux sans-fil, ou WLAN [1]. Le domaine d'application des WLAN est : l'extension de LAN, l'interconnexion d'immeubles, l'accès nomade et les réseaux ad hoc. Le réseau adhoc désigne un réseau *peer-to-peer* pour établir une communication temporaire et immédiate. Cependant nous avons quelques contraintes à satisfaire :

- Débit : le protocole de contrôle d'accès (MAC) doit utiliser efficacement le support pour maximiser la taille des données transmises.
- Nombre de nœuds : un LAN sans fil doit pouvoir supporter un nombre important de nœuds
- Zone de service : la zone de couverture du réseau doit être maximale, la zone couvre en général une diamètre de 100 à 300 m.
- Consommation réduite : chaque nœud du WLAN est alimenté par une batterie, dont l'autonomie doit être la plus longue possible. Ainsi le protocole d'accès doit réduire au minimum les négociations entre les nœuds. Les implémentations typiques offrent la possibilité de passer en mode veille lorsque le réseau n'est pas utilisé, ou le contrôle de puissance qui permet d'empêcher les nœuds d'émettre plus de puissance que nécessaire.
- Fiabilité et sécurité des transmissions : un WLAN doit être insensible aux interférences, assurer la fiabilité des données échangées et un niveau de protection contre les écoutes.
- Fonctionnement non soumis à licence : il est préférable d'utiliser des produits peu onéreux pour construire un WLAN ne nécessitant pas de licence d'exploitation comme les bandes de fréquences ISM [2].

Actuellement, la gamme de produits de WLAN qui répondent à certaines de ces exigences sont le Wifi (IEEE 802.11x), le Bluetooth (IEEE 802.15) ou bien le Zigbee (IEEE 802.15.4).

La raison d'être du Zigbee, moins connu, est de proposer un protocole lent avec un rayon d'action très faible nommé LR-WPAN[3] avec une fiabilité assez élevée, un prix de revient faible et une consommation réduite.

Le ZigBee se retrouve naturellement dans des environnements embarqués où la consommation est un critère majeur. Cette technologie est comparable à la technologie Bluetooth en étant moins chère, mais pour aussi pour le moment moins disponible. ZigBee se base sur le standard IEEE 802.15.4 rédigé par la **Zigbee Alliance** qui a publié en juin 2005 la version ZigBee 1.0 en libre téléchargement [5].

[1] Wireless LAN
[2] Industrial, Scientific, Medical
[3] Low Rate Wireless Personal Area Network

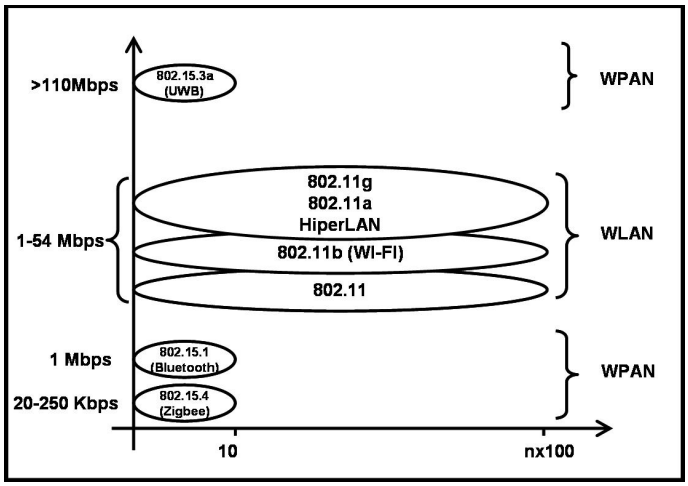

FIG. 4 – Divers protocoles de communication sans fil avec leur débit en ordonnée et ordre de grandeur de portée en abscisse [4]. Le Bluetooth de classe I que nous utilisons annonce un portée de 100 m.

3.2 Choix du Bluetooth

Notre choix s'est porté sur le Bluetooth [6, 7] dans la mesure où il s'agit d'une technologie plus utilisée que ZigBee offrant des possibilités qui remplissent assez bien les contraintes liées au réseau de capteurs.

Le Bluetooth permet des débits allant jusqu'à un peu moins de 1 Mbit/s avec une portée allant jusqu'à 100 m pour certains modèles de classe I et une faible consommation. La limitation majeure est la possibilité de fonctionner avec un maximum de 8 équipements (1 maître et 7 esclaves) formant un petit réseau appelé : *piconet*. Cette limite peut être levée en utilisant un état de connexion permettant à un esclave n'ayant pas besoin de participer à un piconet sans toutefois le quitter : *l'état park*. Ce mode offre l'avantage à un nœud de fonctionner en consommation réduite avec une très faible activité où seule l'horloge est active [4, 8].

En ce qui concerne la fiabilité, Bluetooth offre trois techniques de corrections d'erreurs prévues pour répondre à des besoins qui s'opposent selon le type de paquets Bluetooth lié au type de liaison utilisé (SCO [4] ou ACL) :

- Code de correction aval (FEC) de taux 1/3 consiste simplement à l'envoi de trois copies de chaque bit avec une logique majoritaire pour discerner la valeur du bit reçu en cas de différence.
- Code de correction de taux 2/3 utilise un code correcteur de blocs (codes de Hamming) dans un mot de code FEC peut corriger une erreur sur un bit et détecter une erreur double.
- Une fonction ARQ (Automatic Repeat Request) de demande automatique de retransmission permettant la détection d'erreurs, un acquittement positif pour les paquets sans erreur, retransmission après temporisation et

[4] Synchronous Connection Oriented : lien qui alloue une bande passante fixe pour une connexion point à point entre le maître et un esclave, ce type de paquet n'est jamais retransmis

un acquittement négatif puis retransmission.

En ce qui concerne le développement d'applications, nous allons utiliser le protocole **L2CAP** [5] qui contrôle la liaison entre les membres d'un réseau au support partagé. L2CAP assure un certain nombre de services en s'appuyant sur une couche inférieure pour le contrôle de flux et d'erreurs pour l'échange de données il utilise une liaison de type ACL sans réservation de bande passante, mais offre un débit maximal par l'emploi de paquets à 5 slots non-protégés (paquets de type DH5) avec une allocation de capacité asymétrique produisant 721 kb/s vers l'esclave et 57,6 kb/s vers le maître. [6] L2CAP permet trois types de canaux logiques :

- Sans connexion : ce type de canal unidirectionnel, identifié par un identifiant de canal (CID) de valeur 2, est employé pour du trafic broadcast du maître vers les esclaves
- Avec connexion : chaque canal est ici bidirectionnel avec une spécification de flux QOS assignée pour chaque direction et un CID pour chaque extrémité.
- Signalisation : ce type de canal permet l'échange de messages de signalisation entre les entités L2CAP avec un CID de 1.

Il est donc possible de créer un seul canal sans connexion et un seul canal de signalisation mais plusieurs canaux avec connexion.

4 Développements logiciels : exemple de client/serveur Bluetooth

Nous allons désormais introduire les grandes lignes de la communication Bluetooth au moyen d'un premier exemple de client-serveur dont l'objectif est l'échange d'informations associées à la qualité de la liaison radio. La programmation Bluetooth en espace utilisateur que nous réaliserons nécessite l'installation des librairies Bluez : `bluez` et `libbluetooth1-dev` [9].

Les structures définies dans la pile bluetooth (Fig.5) appelé HCI (Host Controller Interface) où sont implémentées les couches supérieures de la pile bluetooth (L2CAP, RFCOMM, ...) vont permettre d'effectuer la communication entre les périphériques. Ci-dessous le fichier `client.c` correspondant au client de notre PC qui enverra des messages sur notre carte fox qui affichera ces messages sur un écran LCD tout en renvoyant la qualité de la liaison radiofréquence (RSSI : *Received Signal Strength Indication*).

Nous choisissons dans un premier temps de placer le client sur le PC et le serveur au niveau de la carte Fox. Cette dernière, équipée d'un récepteur GPS dont les informations sont réactualisées toutes les secondes, cadencera les échanges une fois la liaison réalisée.

Les entêtes définissant les diverses structures nécessaires au protocole Bluetooth sont :

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <netinet/in.h>
```

[5] Logical Link Control/Adaptation Protocol

[6] Asynchronous Connectionless : liason multipoint de type commutation par paquets entre le maître et les esclaves du piconet

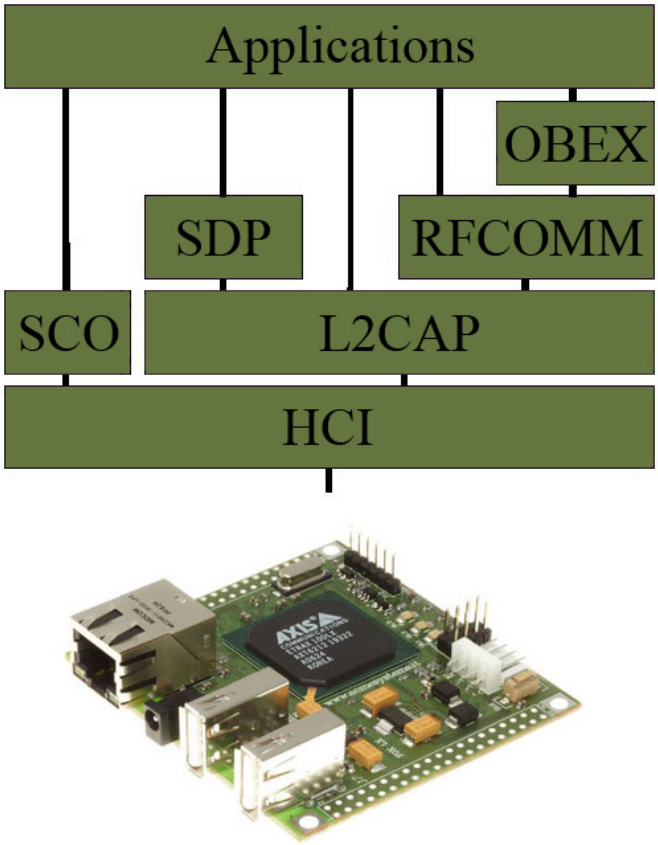

Fig. 5 – Pile protocolaire bluetooth

```
#include <bluetooth/bluetooth.h>
#include "hci.h"
#include <bluetooth/hci_lib.h>
#include <bluetooth/l2cap.h>
#include <sys/poll.h>

#define BUF_SZ 256

// psm en L2cap correspond au numeo de port en programmation TCP/UDP
static uint16_t psm = 0xaa79;
```

Le programme principal se charge alors d'initialiser la communication en recherchant les divers interlocuteurs disponibles et en se connectant avec le premier qui a été détecté.

```
int main () {
  int dev_id, num_rsp, length, flags;
  inquiry_info *info = NULL;
```

```
  bdaddr_t bdaddr;        //Bluetooth adresse
  char name[14]="[NOT FOUND]";
  char namelocal[14]="[NOT FOUND]";
  char *buff = (char *)malloc(sizeof(char) * BUF_SZ);
  char *buff2= (char *)malloc(sizeof(char) * BUF_SZ);
  int  sock;
  struct sockaddr_l2 addr;
//variables pour recuperer le RSSI
  struct hci_conn_info_req *cr;
  struct hci_request rq, rq2;
  get_link_quality_rp rp2;
  read_rssi_rp rp;
  uint16_t handle;
  int resultrssi=0; //variable = -1 lorque le rssi cause des pbs
  char test[256];

//initialisation des variables
//variable d'information sur la connexion tel que RSSI
  cr = malloc(sizeof(*cr) + sizeof(struct hci_conn_info));
  int i,dd;
  dev_id = 0;  // device hci 0
  length = 8;  // 10.24 seconds
  num_rsp = 10;// nbre maxi a detecter
  flags = 0 ;
  int k=0;

//recuperation du nom de la machine
  dd = hci_open_dev(dev_id);
  hci_local_name(dd, sizeof(namelocal),namelocal,0);
  printf("[MACHINE LOCAL] : %s \n", namelocal);
```

À l'aide de la commande `hci_inquiry`, on récupére les informations sur l'ensemble des périphériques visibles dans le tableau `info` de `num_rsp` éléments.

```
//on recupere le tout dans le tableau info
  num_rsp = hci_inquiry (dev_id, length, num_rsp, NULL, &info, flags);
  printf("[PERIPHERIQUE(S)  DETECTE] : %d \n\n", num_rsp);

//parcours du tableau pour tenter une connexion
  for (i=0 ; i<num_rsp ; i++ ) {
     baswap (&bdaddr, &(info+i)->bdaddr);
     printf ("[ADRESSE PERIPHERIQUE DETECTE] : %s\n", batostr(&bdaddr)) ;
```

Ici nous nous contentons de sélectionner le premier périphérique détecté, alors qu'ultérieurement une utilisation optimum de la bande passante et de l'énergie disponibles imposeront de sélectionner le voisin le plus utile pour le routage des paquets vers l'utilisateur [10].

```
//creation de la socket L2cap
  sock = socket(AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);
  addr.l2_family = AF_BLUETOOTH;
```

```
  addr.l2_psm = htobs(psm);
  baswap(&addr.l2_bdaddr,&bdaddr);
  k=hci_remote_name(hci_open_dev(dev_id),&(info+i)->bdaddr,sizeof(name),name,k);
  printf("\n[CONNEXION] : %s \n",name);
  getname(nomFichier);
  fichier =  fopen(nomFichier,"w");
  fprintf(fichier,"RSSI local %s\t Link quality\t RSSI distant %s\n",namelocal,name);
```

Afin de gérer la reconnexion du client en cas de perte de liaison, nous effectuons la boucle infinie sur la fonction `connect()` et testons à chaque communication la validité de la socket (test sur le retour de la fonction `read()`) : en cas de déconnexion, la socket n'est plus valide et `read()` renverra une valeur négative résultant en une tentative de reconnexion.

```
while (1) {
  int res = connect (sock, (struct sockaddr *)&addr, sizeof(addr));
  if (res<0) {
      printf("[ECHEC CONNEXION] %s \n",name);perror("connect");}
  else {
    printf("[CONNEXION VALIDE] : %s\n",name);
    while ( res>=0 ) {  // res = statut de la socket
        //TEST ENVOI RSSI
        bacpy(&cr->bdaddr, &addr.l2_bdaddr);
        ba2str(&cr->bdaddr, test);
        printf("%s\n", test);

        cr->type = ACL_LINK;
        if (ioctl(dd, HCIGETCONNINFO, (unsigned long) cr)<0)
           {resultrssi =-1;printf("pb ioctl\n");}

        handle = htobs(cr->conn_info->handle);
        memset(&rq, 0, sizeof(rq));
        rq.ogf    = OGF_STATUS_PARAM;
        rq.ocf    = OCF_READ_RSSI;
        rq.cparam = &handle;
        rq.clen   = 2;
        rq.rparam = &rp;
        rq.rlen   = READ_RSSI_RP_SIZE;

        if (hci_send_req(dd, &rq, 100)<0) {
          resultrssi=-1;printf("pb hci sen req\n");
        }

        if (rp.status) {resultrssi =-1;printf("rp status");}

        printf("[RSSI] : %d \n", rp.rssi);
        fprintf(fichier,"%d\t ",rp.rssi);

        //RECUPERATION LINK QUALITY
        memset(&rq2, 0, sizeof(rq2));
```

```
        rq2.ogf    = OGF_STATUS_PARAM;
        rq2.ocf    = OCF_GET_LINK_QUALITY;
        rq2.cparam = &handle;
        rq2.clen   = 2;
        rq2.rparam = &rp2;
        rq2.rlen   = GET_LINK_QUALITY_RP_SIZE;

        if (hci_send_req(dd, &rq2, 100) < 0)
          printf("HCI get_link_quality request failed");

        if (rp2.status)
          printf("HCI get_link_quality cmd failed (0x%2.2X)\n",rp2.status);
        printf("Link quality: %d\n", rp2.link_quality);

        fprintf(fichier,"%d                          ",rp2.link_quality);

        if (resultrssi!=-1) {sprintf(buff,"rssi : %d",rp.rssi);}
           } else {sprintf(buff,"pb rssi");}

        read(sock,buff2,BUF_SZ);
        res=read(sock,buff,BUF_SZ);
        printf (" [TRAMES GPS] :%s \n",buff2);
        printf (" [RSSI distant] :%s \n",buff);
        fflush(fichier);
      }  //FIN WHILE
    }
  } // fin de la boucle infinie de connexion = daemon
  } //FIN FOR
  fclose(fichier);close (sock);close(dd);free(cr);
  return 0;
}
```

Le protocole de communication se limite donc ici en l'attente de messages du serveur contenant la position (coordonnées GPS) et la qualité de la liaison.

Ce programme se linke avec la librairie `libbluetooth` par la compilation suivante : `gcc -o client client.c -lbluetooth` qui nécessite l'installation du paquet `libbluetooth2` sous Debian.

Du côté du serveur embarqué sur la carte mobile, les méthodes sont très similaires. Nous avons dû copier les fichiers `bluetooth.h`, `hci.h`, `l2cap.h` dans le répertoire courant car le compilateur CRIS est incapable de les localiser. Toutes les fonctions pour l'affichage sur l'écran LCD de type HD44780 (Fig. 3) sont obtenues à `http://www.acmesystems.it/?id=8021`. La fonction `lecture_rs232(fd,buffer)` ; lit sur le descripteur de fichier associé à `/dev/ttyS3` les trames GPS et bloque l'exécution du programme jusqu'à l'occurent de GPGGA : ainsi, un message n'est envoyé que une fois par seconde, taux de rafraichissement de l'information GPS.

```
#define BUF_SZ     256
#define RX_NULL     0
#define RX_TAG      1
```

```
#define RX_CONTENTS 2
static uint16_t psm = 0xaa77;

main(void) {
      printf("Serveur en attente...\n");
      int sock, client,fd;
      struct sockaddr_l2 addr, cliaddr;
      int addrlen;
//variables pour recuperer le RSSI
      struct hci_conn_info_req *cr;
      struct hci_request rq;
      read_rssi_rp rp;
      uint16_t handle;
      int resultrssi=0;//variable = -1 lorque rcuprer le rssi cause des pbs
      char test[256];

      int dd, dev_id;
      char rssi[256],buffer[255];
      int l,b;

      dev_id = 0;

      sock = socket(AF_BLUETOOTH, SOCK_SEQPACKET, BTPROTO_L2CAP);

      fd=tty_open("/dev/ttyS3"); // GPS

      addr.l2_family = AF_BLUETOOTH;
      bacpy(&addr.l2_bdaddr, BDADDR_ANY);
      addr.l2_psm    = htobs(timeserver_psm);
      b = bind(sock, (struct sockaddr *)&addr, sizeof(addr));
      l = listen(sock, 10);
      dd = hci_open_dev(dev_id);
      if (dd < 0) {perror("HCI device open failed");}
      cr = malloc(sizeof(*cr) + sizeof(struct hci_conn_info));

      while (1) {
        addrlen = sizeof(cliaddr);
        client = accept (sock, (struct sockaddr *)&cliaddr, &addrlen);
        if (client < 0) {
           printf("Erreur accept \n");perror("accept");return(0);
        }
        else {
          b = 1; l = 1;
          while ((b>=0) && (l>=0)) { // b,l : erreur de liaison
            bzero(cr,sizeof(cr));
            bacpy(&cr->bdaddr, &cliaddr.l2_bdaddr);
            ba2str(&cr->bdaddr, test);
            printf("Adresse peripherique distant %s\n", test);
            cr->type = ACL_LINK;
            if (ioctl(dd, HCIGETCONNINFO,(unsigned long)cr)<0)
```

```
            {resultrssi =-1;printf("pb ioctl\n");}
          handle = htobs(cr->conn_info->handle);

          memset(&rq, 0, sizeof(rq));
          rq.ogf    = OGF_STATUS_PARAM;
          rq.ocf    = OCF_READ_RSSI;
          rq.cparam = &handle;
          rq.clen   = 2;
          rq.rparam = &rp;
          rq.rlen   = READ_RSSI_RP_SIZE;

          if (hci_send_req(dd, &rq, 100) < 0)
          {resultrssi =-1;printf("pb hci sen req\n");}

          if (rp.status){resultrssi =-1;printf("rp status");}
          printf("RSSI return value: %d \n", rp.rssi);
          sprintf(rssi,"%d",rp.rssi);
          lecture_rs232(fd,buffer);
          b=write(client,buffer,strlen(buffer));
          l=write(client,rssi,sizeof(rssi));
          if(l<0 || b<0){shutdown(client,SHUT_RDWR);close(client);}
        }
       }
      } // boucle infinie de connect = daemon

       close(fd);close (client);close (sock);
       return 0;
}
```

Le serveur est compilé (`make cris-axis-linux-gnuuclibc && make`) et placé par `ftp` dans le répertoire `/mnt/flash` de la carte Fox (mémoire non volatile accessible en écriture) et lancé comme daemon au démarrage de la carte, par exemple à la fin de `/etc/init.d/rc` (lui aussi en mémoire non volatile accessible en écriture). Il faut au préalable avoir intialisé les modules bluetooth : `cdBluez-start && /mnt/flash/serveur`

4.1 Résultats expérimentaux

Un récepteur GPS ET301 est connecté au port série 3 de la carte Fox afin d'estimer la distance parcourue lors des mesures de portée de la communication Bluetooth. La cadence de rafraîchissement des données est imposée par la cadence du GPS qui fournit sa position une fois par seconde. Chaque fois que le GPS fournit une trame de position (trame GPGGA), nous lisons le RSSI local, le transmettons au PC fixe qui stocke de plus sa propre information de qualité de liaison. Nous obtenons donc une cartographie de la qualité de la liaison bluetooth en fonction de la position GPS du capteur mobile (Fig. 7). Les deux ordinateurs sont équipés de cartes de conversion USB-Bluetooth Linksys de classe I.

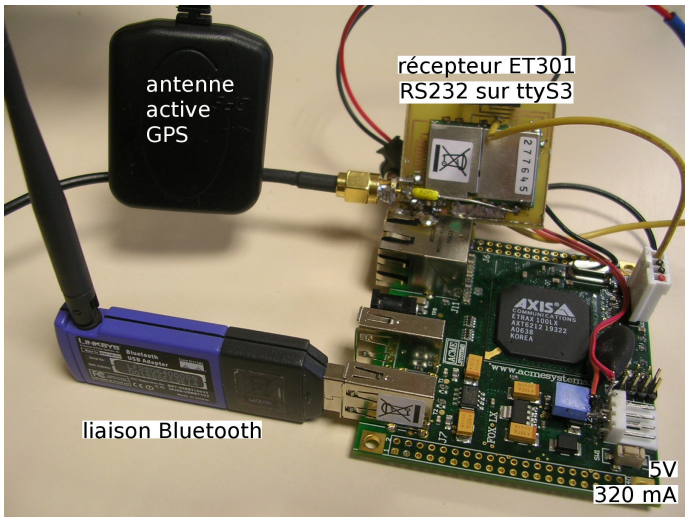

FIG. 6 – Notre Carte fox équipée d'un récepteur GPS utilisé pour transmettre la position du capteur mobile et l'état de la connexion (RSSI) par liaison sans fil avec le PC fixe.

5 Capture et transmission d'images

5.1 Modèles de webcam et convertisseurs USB-bluetooth

En ce qui concerne les webcams, nous avons pu obtenir plusieurs modèles différents au point de vue chipset et format d'images :

- Logitech, Inc. QuickCam Zoom (046d :08b3) chipset pwc format(yuv,raw)
- Labtec Elch2 (0 46d :0929) chipset Sunplus bridge spca561a sensor embedded format (bgbr)
- Creative Technology, Ltd Webcam Live!/Live! Pro chipset Zstar bridge Zc0301P sensor Hv7131c format (jpeg)

Pour le bluetooth, nous avons pu tester différentes classes de périphériques ayant des portées de 10 à 100 m. Le modèle ayant une portée de 100 mètres est une Linksys USB.

6 Support d'une webcam

Pour pouvoir utiliser une webcam sur une carte Fox, il est d'abord nécessaire de comprendre comment récupérer une image mais aussi savoir comment décoder l'image, ce qui varie entre les modèles testés et également en fonction des résolutions choisies.

Pour chaque modèle testé, la webcam est initialement testée sous GNU/Linux sur plate-forme x86 avant de porter le programme de capture sur la carte Fox. L'intérêt d'avoir un système GNU/Linux sur une carte embarquée présente donc des avantages indéniables. Les problèmes de portabilité rencontrés sont plus des problèmes de versions que d'architecture.

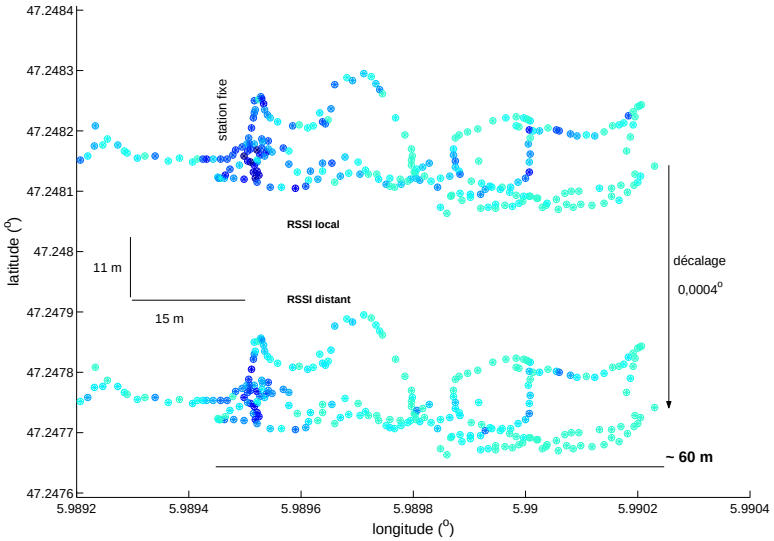

FIG. 7 – Qualité de la liaison bluetooth vue par le PC fixe (courbe du haut) et la carte Fox mobile (courbe du bas) en fonction de la position de cette dernière. La courbe de qualité de liaison de la carte Fox a été arbitraitement décalée de 0,0004 degrés pas souci de clareté du digramme. Un point est d'autant plus sombre que la liaison est de bonne qualité (RSSI élevé). La mesure a été faite en extérieur, en environnement dégagé à l'exception d'arbres, dans le parc de l'Observatoire de Besançon

.

6.1 Comment faire fonctionner une webcam sur la carte Fox ?

La carte Fox étant un système GNU/Linux, il est facile de tester le bon fonctionnement du modèle sur un ordinateur avant de tester sur la carte ce qui facilite grandement cette phase qui se révèle généralement complexe pour d'autres systèmes embarqués n'ayant pas le même OS. Sur le site de Acme Systems, un article explique comment faire fonctionner une Webcam Labtech Pro pour un noyau de la branche 2.4 (2.4.31). Nous allons ici nous intéresser à un autre modèle supporté dans le noyau officiel, la QuickCam Zoom controlée par le pilote `pwc` de Luc Saillard pour un noyau 2.6.

La configuration de l'environnement de travail se fait comme suit :

1. Lancer le menu de configuration du sdk `make menuconfig`
2. sélectionner le noyau pour la version 2.6.15
3. dans le menu `Driver settings`, activer uniquement `Enable Webcam support`
4. sauvegarder et quitter le menu, puis `./configure`

La configuration du noyau est nécessaire pour les modèles à notre disposition : il suffit d'activer quelques options si elles ne le sont pas déjà :

1. Activer le chargement de module
2. Activer le support du SCSI ainsi que `support de SCSI disk support`
3. Activer `Video For Linux` dans `Multimedia devices`
4. Activer dans `USB Support` dans la section `Multimedia USB devices` la webcam : pour nous `USB Philips Camera` pour la QuickCam Zoom
5. On peut maintenant lancer la compilation du sdk pour ensuite flasher avec notre nouvelle `fimage`.

Dans la perspective du réseau de capteurs, nous avons différents modèles de caméras. L'économie de mémoire est essentielle : pour éviter de flasher une image pour chaque modèle de caméra, nous avons choisi de créer une `fimage` avec les modules noyau communs à tous les modèles, *i.e.* `videodev.ko` et les modules liés à l'API `video4linux`. Puis pour chaque modèle de caméra, nous compilons à part les modules de la caméra utilisée, qui seront placés dans la partition de la flash accessible en écriture `/mnt/flash`. Après avoir flashé la carte, il suffit de charger les modules de la caméra avec un script shell puis vérifier le résultat avec la commande `dmesg`. L'automatisation de cette procédure peut, comme auparavant, se faire dans `/etc/init.d/rc`. La capture d'image de la caméra viendra dans la suite de cet article.

6.2 Format d'image et espaces colorimétriques (YUV, RGB, raw)

Avant de parler de la capture d'images, nous allons parler de la façon dont les webcams encodent une image. Sur le diagramme 8, on repère deux blocs distincts : le capteur CMOS et la puce appelé **Bridge Chip**. Un pilote de web-

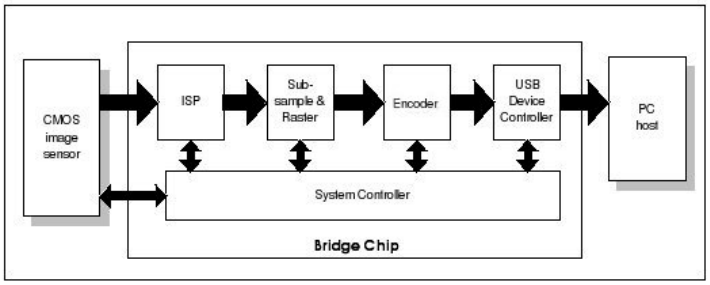

FIG. 8 – diagramme de fonctionnement d'une webcam

cam dialogue directement avec cette puce pour paramétrer le capteur CMOS. En ce qui concerne l'encodage de l'image, c'est aussi ce composant qui offre la possiblilité d'encoder une image directement en jpeg pour les webcams supportées par le pilote spca5xxx. D'autres webcams, telles que celles supportées par le pilote `pwc`, nous proposent un format YUV420 planar ou un format RAW selon la résolution de l'image. Dans ce cas, c'est un programme qui doit réaliser l'encodage de l'image. Nous allons détailler le modèle colorimétrique YUV et comment l'encodage a été réalisé.

6.2.1 Le modèle YUV

Ce modèle définit un espace colorimétrique sur 3 composantes : le premier est la luminance et les 2 autres représentent la chrominance, U pour la valeur de différence Rouge et V pour la valeur de différence Vert.

L'encodage d'une image en YUV provient du fait que l'œil humain perçoit mieux la luminance [7] comprise en 60 et 70, *i.e.* la couleur verte. C'est pourquoi dans ce modèle, on stocke de manière plus précise la luminosité par rapport aux niveaux de couleur : la couleur verte est suréchantillonnée par rapport au rouge et au bleu [11, 12, 13, 14].

6.2.2 Conversion YUV-RGB

Pour passer du modèle YUV au modèle RGB, une bibliothéque `libpwc` créée par L. Saillard permet de convertir facilement les images capturées par le pilote `pwc`. Sur le site [15] une multitude de formules de conversion d'un modéle à l'autre sont listées.[8] Dans `libpwc`, c'est la formule issue de la spécification TIFF 6.0 qui a été choisie :

$$Y = 0.29900 \times R + 0.58700 \times G + 0.11400 \times B[0...1]$$

$$Cb(V) = -0.16874 \times R - 0.33126 \times G + 0.50000 \times B[-0.5...0.5]$$

$$Cr(U) = 0.50000 \times R - 0.41869 \times G - 0.08131 \times B[-0.5...0.5]$$

La fonction `pwc_yuv420p_to_rgb` va nous servir à créer une image au format PPM dont voici le code :

```
void write_ppm(int num,char *buf,int width, int height)
{
  int yuv_buffer_size;
  void *yuv_data = NULL;
  int yuvlen;
  void *rgb_data;
  unsigned char *py, *pu, *pv;
  char filename[256];
  FILE *fp;

  yuv_buffer_size = (width * height * 3) / 2;
  yuv_data = malloc(yuv_buffer_size);
  memcpy(yuv_data,buf, yuv_buffer_size);
  rgb_data = malloc(width * height);
  yuvlen = yuv_buffer_size;
  py = yuv_data;
  pu = yuv_data +  yuvlen;
  pv = yuv_data + yuvlen + yuvlen/4;
  pwc_yuv420p_to_rgb(width, height, py, pu, pv, rgb_data);

  snprintf(filename,sizeof(filename),"pwc-%d.ppm",num);
```

[7] aussi appelé luma, niveau de luminosité

[8] Pour une conversion YUV-RGB, il est à noter que tous les modèles YUV n'ont pas le même espace colorimétrique.

```
  fp = fopen(filename,"w");
  if(fp == NULL){
    fprintf(stderr,"[err] process_frame %s\n",strerror(errno));
    return;
  }
  fprintf(fp,"P6\n%d %d\n255\n",320,200);
  fwrite(rgb_data,height,width,fp);
  fclose(fp);
  free(yuv_data);
}
```

6.2.3 Conversion YUV-JFIF

Dans le cadre du transfert d'images par Bluetooth, le temps de transfert d'un fichier au format PNM n'est pas négligeable. Nous avons décidé d'utiliser le format JFIF (JPEG File Interchange Format) qui convient parfaitement pour compresser des images du monde réel. Il est à noter que la conversion RGB-YUV utilise une autre formule qui est la suivante :

$$R = Y + 1.402 \times (Cr - 128)$$

$$G = Y - 0.34414 \times (Cb - 128) - 0.71414 \times (Cr - 128)$$

$$B = Y + 1.772 \times (Cb - 128)$$

L'espace colorimétrique pour le format JFIF est le même soit `JCS_YCbCr`. La façon dont l'image est compressée se divise en 2 fonctions :

- une fonction qui permet de paramétrer la taille et le taux de compression de l'image en remplissant une structure `struct jpeg_compress_struct` une fois pour toute dans une session de capture en continue.

```
void jpeg_init(int width, int height, int quality,
               struct jpeg_compress_struct *cinfo,
               struct jpeg_error_mgr *jerr,
               JSAMPIMAGE jimage, JSAMPROW y)
{
  int i;
  JSAMPROW u,v;

  cinfo->err = jpeg_std_error(jerr);
  jpeg_create_compress(cinfo);

  cinfo->image_width = width;
  cinfo->image_height = height;
  cinfo->input_components = 3;
  cinfo->in_color_space = JCS_YCbCr;
  jpeg_set_defaults(cinfo);

  cinfo->raw_data_in = TRUE;

  cinfo->comp_info[0].h_samp_factor = 2;
  cinfo->comp_info[0].v_samp_factor = 2;
```

```
  cinfo->comp_info[1].h_samp_factor = 1;
  cinfo->comp_info[1].v_samp_factor = 1;
  cinfo->comp_info[2].h_samp_factor = 1;
  cinfo->comp_info[2].v_samp_factor = 1;

  jimage[0] = malloc(height * 2 * sizeof(JSAMPROW));

  jimage[1] = jimage[0] + height;
  jimage[2] = jimage[1] + (height/2);

  u = y + width * height;
  v = u + width * height / 4;

  for(i = 0; i < height; ++i, y+=width) {
    jimage[0][i] = y;
  }
  for(i = 0; i < height/2; ++i, u+=width/2, v+=width/2) {
    jimage[1][i] = u;
    jimage[2][i] = v;
  }
  jpeg_set_quality(cinfo, quality, TRUE);
}
```

– une fonction qui réalise la compression pour écrire l'image dans un fichier.

```
void jpeg_write(int height, JSAMPIMAGE jimage,
                struct jpeg_compress_struct *cinfo,
                const char *fmt)
{
  JSAMPARRAY jdata[3];
  char filename[1024];
  FILE *outfile;
  time_t tt;
  struct tm *tm;
  int i;

  tt = time(NULL);
  if(tt == (time_t)-1) {
    perror("Failed to get time"); return; }
  tm = localtime(&tt);
  if(strftime(filename,1024,fmt,tm) == 0) {
    fprintf(stderr,"Error: resulting filename to long\n");
    return;
  }

  outfile = fopen(filename, "wb");

  jdata[0] = jimage[0];
  jdata[1] = jimage[1];
  jdata[2] = jimage[2];

  jpeg_stdio_dest(cinfo, outfile);
```

```
  jpeg_start_compress(cinfo, TRUE);

  for (i = 0;i < height;i += 2*DCTSIZE) {
    jpeg_write_raw_data(cinfo, jdata, 2*DCTSIZE);
    jdata[0] += 2*DCTSIZE;
    jdata[1] += DCTSIZE;
    jdata[2] += DCTSIZE;
  }

  jpeg_finish_compress(cinfo);
  fclose(outfile);
}
```

6.3 Capture d'images avec l'API `video4linux1`

Bien que dépréciée dans la version actuelle du noyau (2.6.21), la capture d'images en `video4linux1` reste une interface simple pour capturer une image d'une webcam [16]. Cette méthode a déjà été traitée dans un article de Pierre Ficheux [17].

Rappelons les grandes lignes pour développer un programme en espace utilisateur de capture utilisant cette interface de programmation. Globalement un programme de capture réalise les étapes suivantes pour capturer une image en laissant de côté le format de l'image :

- Ouverture du périphérique
- Lecture des capacités de la webcam
- Lecture de la liste des canaux
- Allocation de mémoire
- Capture d'image
- Fermeture du périphérique

L'interface V4L été conçue pour avoir une couche d'abstraction entre un périphérique vidéo et le système GNU/Linux. Le programme dialogue avec le pilote v4l au travers des `ioctl()`. Pour les spécificités de chaque périphérique, le pilote dispose d'`ioctl()` privés documentés dans les entêtes (.h) du pilote.

6.4 Pourquoi le passage Video for Linux 1 à Video for Linux 2 ?

L'API Video for Linux est le nom de l'interface de capture vidéo conçue pour proposer une interface unique permettant de supporter une large variété de périphériques. Une petite partie seulement est vraiment pour faire de la vidéo. L'API est composé de 5 interfaces :

- interface de capture, capture la vidéo d'un tuner ou d'une caméra
- une variante de l'interface de capture qui permet d'afficher plus rapidement les données vidéo du périphérique sans passer par le CPU (*interface overlay*).
- interface de sortie vidéo, pour piloter les périphériques qui fournissent une vidéo pour la transformer par exemple en signal pour TV.
- interface VBI, permet l'accès aux données transmises durant l'intervalle de blanc d'une vidéo.
- interface radio, pour accéder au flux audio FM et AM d'un tuner.

Peu après l'intégration de V4L1 dans le noyau, l'API fut critiquée pour être trop rigide. Bill Dirks proposa de nombreuses améliorations qu'on peut considérer comme le debut de V4L2. Les améliorations n'étaient pas juste une extention mais un remplacement de la première API.

Il a fallu environ 4 ans avant d'inclure officiellement ces modifications dans le noyau sous sa forme actuelle. Les principales différences entre les 2 APIs sont le renommage de `ioctl()` et le remaniment des structures de données, l'ajout de nouveaux standards vidéo,...

6.5 Capture d'images avec l'API `video4linux2`

La capture d'images avec `video4linux2` est possible pour les pilotes qui en implémentent l'interface, mais n'est pas encore adoptée par tous les pilotes [18]. Cependant les étapes pour utiliser l'API sont quasiment identiques :

- Ouverture du périphérique
- Lecture des capacités du périphérique
- Changement des caractéristiques de luminosité et balance de blanc
- Négotiation d'un format de données
- Négotiation d'une méthode d'entrées/sorties
- Boucle de capture
- Fermeture du périphérique

FIG. 9 – Exemple d'image capturée au moyen d'une webcam fonctionnant sous pwc installé sur carte Fox. La résolution de 160×120 pixels a été choisie pour optimiser la bande passante disponible. L'image a été capturée sur Zaurus SLC3200 sous GNU/Linux.

6.5.1 Ouverture et fermeture du périphérique

Le périphérique est accessible en ouvrant le fichier spécial `/dev/videoX` adéquat en notant l'ajout du drapeau supplémentaire `O_NONBLOCK` qui permet durant la récupération d'une image sur la file de retourner immédiatement si la file est vide.

```
void open_device(int *fd)
{
  struct stat st;

  if(stat(device_name,&st) == 1){
    fprintf(stderr, "Cannot identify '%s': %d, %s\n",
```

```
            device_name, errno, strerror(errno));
    exit(EXIT_FAILURE);
  }
  if(!S_ISCHR(st.st_mode)){
    fprintf(stderr, "%s is no device\n", device_name);
    exit(EXIT_FAILURE);
  }
  /* When the O_NONBLOCK flag was given to the open() function,
     VIDIOC_DQBUF returns immediately with an EAGAIN error code
     when no buffer is available */
  *fd = open(device_name,O_RDWR | O_NONBLOCK, 0);
  if(*fd == -1){
    fprintf(stderr, "Error while accessing device %s: %s\n",
            device_name, strerror(errno));
    exit(EXIT_FAILURE);
  }
}
```

La fermeture utilise la fonction `close()` sur le descripteur de fichier. Il est possible d'ouvrir plusieurs fois un périphérique V4L2 si le pilote le supporte, ce qui offre à l'utilisateur la possibilité d'avoir une application pour régler les paramètres de contraste, luminosité, balance de blancs . . .et une autre application qui capture le son et la vidéo comparable à une application mixer ALSA.

6.5.2 Lecture des capacités du périphérique

Comme V4L2 couvre un nombre important de périphériques, certaines fonctionnalités ne sont pas toujours utilisables. L'ioctl `VIDIOC_QUERYCAP` est disponible pour demander les fonctionnalités et la méthode d'entr'ee/sortie supportées.

```
int v4l2_init(int fd)
{
 struct v4l2_capability cap;

  if(ioctl(fd,VIDIOC_QUERYCAP,&cap) == -1){
    return -1;
  }
  fprintf(stderr,"[v4l2] driver info: %s %d.%d.%d \n"
          "[v4l2] device info: %s @ %s\n",
          cap.driver, cap.version >> 16) & 0xff,
          cap.version >>  8) & 0xff, cap.version & 0xff,
          cap.card,cap.bus_info);

  if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
    fprintf (stderr, "[v4l2]  is no video capture device\n");
  }else{
     fprintf (stderr, "[v4l2] video capture device\n");
  }

  /* check if we could use mmap */
```

```
  if(!(cap.capabilities & V4L2_CAP_STREAMING)){
    fprintf (stderr, "[v4l2] no streaming support\n");
  }else{
    fprintf (stderr, "[v4l2] streaming support\n");
  }

  if(!(cap.capabilities & V4L2_CAP_READWRITE)){
    fprintf(stderr,"[v4l2] no read/write i/o methods support\n");
  }else{
    fprintf(stderr,"[v4l2] read/write i/o methods support\n");
  }

  if(!(cap.capabilities & V4L2_CAP_ASYNCIO)){
    fprintf(stderr,"[v4l2] no asynchronous i/o methods support\n");
  }else{
    fprintf(stderr,"[v4l2] asynchronous i/o methods support\n");
  }
  return 0;
}
```

D'autres ioctls permettent de savoir le nombre, le type et le nom de l'entrée vidéo `VIDIOC_ENUMINPUT`.

```
#define MAX_FORMAT  32
...
 /*Enumerate image formats*/
  int nfmts,i;
  uint32_t *fourcc;
  struct v4l2_fmtdesc fmt[MAX_FORMAT];

  for (nfmts = 0; nfmts < MAX_FORMAT; nfmts++){
    fmt[nfmts].index = nfmts;
    fmt[nfmts].type  = V4L2_BUF_TYPE_VIDEO_CAPTURE;

    /* ioctl:VIDIOC_ENUM_FMT : On success 0 is returned, on
       error -1 and the errno variable is set appropriately
       EINVAL: The struct v4l2_fmtdesc type is not supported
               or the index is out of bounds.
    */
    if (-1 == ioctl(cam_fd, VIDIOC_ENUM_FMT, &fmt[nfmts]))
      break;
  }
  fprintf(stderr,"[v4l2] number of image formats: %d\n",nfmts);
  nbformats = nfmts;
  for(i = 0; i < nfmts ; i++){
    xioctl(cam_fd, VIDIOC_ENUM_FMT, &fmt[i], EINVAL);
    fourcc =  &(fmt[i].pixelformat);
    fprintf(stderr,"[v4l2] * %s [%c%c%c%c]  %s\n",
            fmt[i].description,
            fourcc[0],
            (fourcc[0] >> 8 & 0xff),
```

```
            (fourcc[0] >> 16 & 0xff),
            (fourcc[0] >> 24 & 0xff),
            ((fmt[i].flags) ? "compressed" : ""));
  }
```

6.5.3 Négotiation d'un format de données

Il est très important de soumettre un format de données avant d'initialiser les tampons d'entrée/sortie. Le format est soumis au pilote avec l'ioctl `VIDIOC_S_FMT` en utilisant la structure de données `struct v4l2_format` qui peut être modifiée si la taille est incorrecte. D'autres ioctls sont disponibles pour la négociation du format on retiendra `VIDIOC_G_FMT` pour avoir la liste des formats de données et `VIDIOC_TRY_FMT`.

```
 /* 0 raw / 1 yuv420planar */
  struct v4l2_format myfmt;
  int default_width = 320;
  int default_height = 200;
  if((nbformats != 0) && (numformat >= 0)){
    memset(&myfmt,0,sizeof(struct v4l2_format));
    myfmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
    myfmt.fmt.pix.width = default_width;
    myfmt.fmt.pix.height = default_height;
    myfmt.fmt.pix.pixelformat = fmt[1].pixelformat;
    myfmt.fmt.pix.field = V4L2_FIELD_INTERLACED;
    if(ioctl(cam_fd,VIDIOC_S_FMT,&myfmt) == -1){
      fprintf(stderr,"[err] set format error using default \n");
    }
    width = myfmt.fmt.pix.width;
    height = myfmt.fmt.pix.height;
    fprintf(stderr,"[info] capture format: width=%d height=%d\n",width,height);
  }
```

6.5.4 Boucle de capture

Nous allons expliquer comment capturer une image en utilisant la méthode `mmap`. Cette méthode est disponible seulement si le drapeau `V4L2_CAP_STREAMING` [9]dans le champ *capabilities* dans la structure `struct v4l2_capability` est positionné. Cette méthode permet de copier entre l'application et le pilote uniquement les pointeurs des tampons, les données ne sont pas copiées, ce qui convient parfaitement pour des applications de streaming vidéo.

Avant de capturer, il faut tout d'abord allouer les tampons pour le periphérique, généralement en mémoire DMA, avec la structure `struct v4l2_requestbuffers` avec l'ioctl `VIDIOC_REQBUFS` en précisant le nombre de frames et le type de tampon : dans notre cas `V4L2_BUF_TYPE_VIDEO_CAPTURE`. Cet ioctl permet également de modifier le nombre de tampons et de libérer les tampons.

Pour que notre application puisse accéder aux tampons du pilote dans son espace d'adressage, on utilise la fonction `mmap` : l'adresse des tampons à passer à mmap s'obtient avec l'ioctl `VIDIOC_QUERYBUF`, en particulier les champs

[9]Toutes les webcams testées supportaient cette méthode

`m.offset` et `length` de la structure `struct v4l2_buffer`. Attention, la mémoire alloué se fait en mémoire physique et donc ne peut pas être *swappée* : il ne faut surtout pas oublier de libérer ces tampons. Les pilotes V4L2 utilisent deux files de tampons, la file d'entrée et la file de sortie. Ce principe permet d'avoir une capture synchrone par rapport au périphérique et de ne pas se soucier de l'application qui peut être préemptée par un autre processus, ou simplement ralentie par l'écriture des données sur le disque. Le pilote nécessite tout le temps un nombre minimum de tampons dans la file d'entrée pour fonctionner. Par contre il n'existe pas de limite supérieure.

Initialement tout les tampons mappés en mémoire ne sont pas accessibles par le pilote : il faut mettre tout les tampons dans une file en utilisant l'ioctl `VIDIOC_QBUF` pour enfiler (par analogie pour défiler un tampon c'est l'ioctl `VIDIOC_DQBUF`). L'application attendra qu'un buffer soit rempli pour lire les données et remet le tampon dans la file quand les données ont été traitées. Pour commencer une capture, il suffit d'appeler l'ioctl `VIDIOC_STREAMON`, et `VIDIOC_STREAMOFF` pour arrêter la capture et vider également les tampons des deux files.

```
struct img_buf{
unsigned int img_length;
char *img_data;
};
struct v4l2_buffer *buf_v4l2;
struct img_buf *buf_pwc;
...
struct v4l2_requestbuffers reqbuf;
register unsigned int i;
enum v4l2_buf_type type;

memset (&reqbuf, 0, sizeof (reqbuf));
reqbuf.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
reqbuf.memory = V4L2_MEMORY_MMAP;
reqbuf.count = count;
if (ioctl(cam_fd, VIDIOC_REQBUFS, &reqbuf) == -1){
  if(errno == EINVAL)
    fprintf(stderr,"[err] mmap-streaming is not supported\n");
  else
    fprintf(stderr,"error ioctl:  VIDIOC_REQBUFS\n");
  exit(EXIT_FAILURE); // ugly
}
/* minimal buffers needed */
if(reqbuf.count < 2){ /* free buffer here */
  fprintf(stderr,"[err] no enough buffer memory in video device\n");
  exit(EXIT_FAILURE);
}
buf_v4l2 = calloc(reqbuf.count,sizeof(struct v4l2_buffer));
if (!buf_v4l2){
  fprintf(stderr,"[err] no enough buffer memory\n");
  exit(EXIT_FAILURE);
}
```

```
  buf_pwc = calloc(reqbuf.count,sizeof(struct img_buf));
  if (!buf_pwc){
    fprintf(stderr,"[err] no enough buffer memory\n");
    exit(EXIT_FAILURE);
  }
  for(i = 0; i < reqbuf.count ; i++){
    buf_v4l2[i].index = i;
    buf_v4l2[i].type = reqbuf.type;
    buf_v4l2[i].memory = reqbuf.memory;
    if (-1 == ioctl(cam_fd, VIDIOC_QUERYBUF, &buf_v4l2[i])){
       fprintf(stderr,"error ioctl:  VIDIOC_QUERYBUFS\n");
       exit(EXIT_FAILURE);
    }
    buf_pwc[i].img_length = buf_v4l2[i].length;
    buf_pwc[i].img_data = mmap(NULL /* start everywhere */,
                               buf_v4l2[i].length,
                               PROT_READ | PROT_WRITE /* required */ ,
                               MAP_SHARED /* recommended */ ,
                               cam_fd,buf_v4l2[i].m.offset);
    if(buf_pwc[i].img_data == MAP_FAILED){
      /* need to unmap and free the so far mapped buffers here...*/
      fprintf(stderr,"[err] mmap error %s",strerror(errno));
      exit(EXIT_FAILURE);
    }
  }
  /* Queue all mapped buffers */
  for(i = 0; i < reqbuf.count; i++){
    if(ioctl(cam_fd,VIDIOC_QBUF,&buf_v4l2[i]) == -1){
      fprintf(stderr,"error ioctl:  VIDIOC_QBUF");
    }
  }
  type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
  if (ioctl(cam_fd, VIDIOC_STREAMON, &type) == -1){
    fprintf(stderr,"VIDIOC_STREAMON");
    exit(EXIT_FAILURE);
  }
```

C'est à ce moment que la boucle de capture doit s'adapter selon le but de l'application : nous allons pour l'exemple simplement capturer 5 images et arrêter la capture.

```
void grabloop(int cam_fd)
{
  int nbloop = 5;
  while(nbloop-- > 0){
    for(;;){
      fd_set fds;
      struct timeval tv;
      int r;

      FD_ZERO(&fds);
```

```
      FD_SET(cam_fd,&fds);
      /* Timeout 1s */
      tv.tv_sec = 1;
      tv.tv_usec = 0;
      r = select(cam_fd + 1, &fds, NULL, NULL, &tv);
      if(r == -1){
        if(errno == EINTR)
          continue;
        perror("select");
      }
      if(r == 0){
        fprintf(stderr,"[err] timeout in grabbing loop\n");
        exit(EXIT_FAILURE);
      }
      if(read_frame(cam_fd))
        break;
      /* EAGAIN - continue select loop */
    }
  }
}
```

La boucle de capture s'articule autour d'un `select` avec un timeout de 1 seconde. Une fonction est appelée dès que des données sont disponibles, en suivant ce qui a été énoncé plus haut. La fonction `process_frame` s'occupe de traiter les données de l'image, sans aucun lien avec l'API Video4Linux2, comme par exemple l'écrire dans un fichier ou compresser dans un format jpeg...

```
int read_frame(int cam_fd)
{
  struct v4l2_buffer buf;

  memset(&buf,0,sizeof(struct v4l2_buffer));
  if (!buf_v4l2){
    fprintf(stderr,"[err] no enough buffer memory\n");
    exit(EXIT_FAILURE);
  }
  buf.type =  V4L2_BUF_TYPE_VIDEO_CAPTURE;
  buf.memory = V4L2_MEMORY_MMAP;
  /* dequeue a buffer */
  if(ioctl(cam_fd, VIDIOC_DQBUF, &buf) == -1){
    switch(errno){
    case EAGAIN:
      return 0;
    default:
       fprintf(stderr,"[err] VIDIOC_DQBUF\n");
       exit(EXIT_FAILURE);
    }
  }
  /* save this buffer into a file */
  process_frame(buf_pwc[buf.index].img_data,buf.length);
```

```
  /* enqueue again this buffer */
  if(ioctl(cam_fd, VIDIOC_QBUF, &buf) == -1)
    fprintf(stderr,"[err] VIDIOC_QBUF\n");

  return 1;
}
```

7 Envois d'images par Bluetooth

Dans cette partie, nous allons utiliser ce qui a été présenté dans les parties précédentes en réalisant une application client fonctionnant sur la carte FOX permettant de capturer une image de la transmettre par liaison Bluetooth à un autre nœud qui peut être soit une carte Fox, soit un PC ou un PDA où tourne le serveur pour recevoir des images. Notez que cette fois le serveur tourne sur le PC "fixe" tandis que les cartes Fox mobiles sont considérées comme des clients fournissant des images.

7.1 Principe de l'application

Le principe de l'application est la capture d'une image par la webcam et son envoi à un serveur actif dans le réseau. Voici un diagramme expliquant la démarche suivie par le client :

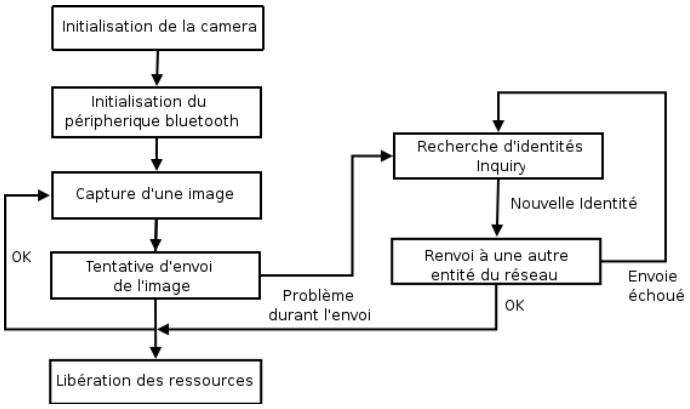

Fig. 10 – fonctionnement du client

Il est à noter que l'image n'est pas envoyée directement après sa capture : elle est d'abord convertie au format JPEG en mémoire puis des informations comme l'adresse Bluetooth de la carte FOX émettrice, le RSSI et la qualité radio de la connexion établie entre les 2 équipements sont écrites dans l'entête de l'image en employant la fonction `rdjpgcom()` de la `jpeglib`.

En ce qui concerne le serveur, il doit à son lancement détecter les équipements pour les enregistrer dans une structure de données adaptée. Le dialogue entre le serveur et le client suit un protocole très simple expliqué sur le schéma :

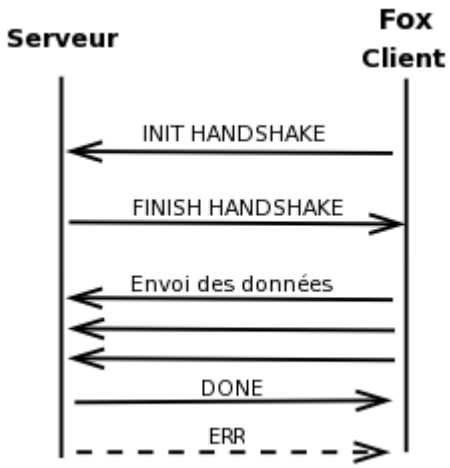

FIG. 11 – schéma du protocole

7.2 Envoi d'images en L2CAP

Les deux parties de l'application partagent une série de fonctions correspondant au protocole décrit plus haut en manipulant la structure `request_t` représentant notre requête de transmission d'images :

```
#define FILE_NAME_LENGTH 256
typedef struct {
  char filename[FILE_NAME_LENGTH];
  ssize_t content_length;
  int status;
} __attribute__((packed)) request_t;
```

Nous allons nous intéresser essentiellement à la fonction qui fait de l'écriture fragmentée de données mappées en mémoire en tenant compte du MTU[10] du module Bluetooth qui offre l'avantage de ne pas écrire de fichier sur la mémoire flash : celle-ci étant limitée en nombre d'écritures, il est judicieux de ne pas y placer de fichier temporaire régulièrement modifié.

```
int do_mapwrite(int fd,void *filefd, int filesize,int mtu)
{
  void* buf;
  int actual = -1,size,rsize,nbpacket,rest;

  size = filesize;
  buf = filefd;
  DEBUG("will send %d bytes\n", size);
  /* Send and fragment if necessary  */

  nbpacket = size/mtu;
```

[10]Maximum Transmit Unit

```
  while (nbpacket != 0) {
    actual = write(fd, buf, mtu);
    DEBUG("actual  %d bytes\n", actual);
    if (actual <= 0)
      return actual;
    nbpacket --;
    /* Hide sent data */
    buf += actual ;
    size -= actual;
  }
  if(filesize%mtu != 0){
    DEBUG("last packet size %d \n", filesize%mtu );
    rest = filesize % mtu;
    actual = write(fd, buf, rest);
  }
  return nbpacket;
}
```

7.2.1 Le client

Au lancement du programme client, celui-ci vérifie que le module Bluetooth est présent et bien activé à l'aide de l'ioctl `HCIGETDEVINFO` sur une socket raw ouverte pour dialoguer avec le *HCI* et en testant si le drapeau `HCI_UP` est bien positionné.

```
  int dev_id = 0; /* hci0 */
  struct hci_dev_info di = {dev_id: dev_id};

/* get device info that do not require to open device */
    s = socket(AF_BLUETOOTH, SOCK_RAW, BTPROTO_HCI);
    if(s < 0){
        perror("socket");
        exit(EXIT_FAILURE);
    }
    if(ioctl(s, HCIGETDEVINFO,(void *) &di)){
        printf("[ERR]| hci device is not available\n");
        exit(EXIT_FAILURE);
    }
    close(s);
    /* is this hci device up ?*/
    if(!hci_test_bit(HCI_UP, &di.flags)){
        printf("[ERR] bluetooth device is down\n");
        exit(EXIT_FAILURE);
    }
    ba2str(&di.bdaddr, btaddr);
    printf("[INFO] using %s (%s)\n",di.name, btaddr);
```

Ensuite une fonction est lancée pour initialiser la caméra et exécuter la boucle de capture (`grabloop`) qui après avoir capturé une image et compressé au format

JPEG appelle une fonction qui envoie l'image à une adresse Bluetooth donnée en paramètre ou détectée après avoir réalisé une recherche d'identités (inquiry) en appelant la fonction `hci_inquiry` (veiller à bien vider le cache avant).

```
void push_img(const char *filename, char *fmap, int fsize, const bdaddr_t *rdev)
{
  int rsck;
  request_t newreq, donereq;
  ssize_t bytes;

  if(fmap != MAP_FAILED)
  {
     rsck = do_connect(rdev);
     if( rsck < 0)
        return;
        newreq = new_request(filename, fsize);
        bytes = send_request(rsck, &newreq);
        fprintf(stderr,"[DEBUG] send a request for file:%d bytes\n",bytes);
        if(bytes < 0){
         fprintf(stderr,"[ERR] send_request error");
         close(rsck);
         return;
        }
        bytes = recv_request(rsck,&donereq);
        if(bytes > 0)
            fprintf(stderr,"[INFO] request accepted %s ( %d bytes)\n",
                donereq.filename, donereq.content_length);
        do_mapwrite(rsck, fmap, fsize, 400);
      /* OK or ERR ? */
      bytes = recv_request(rsck,&donereq);
      if(bytes > 0)
          fprintf(stderr,"[INFO] request done %s ( %d bytes)\n",donereq.filename, donereq.
      if(munmap(fmap, fsize) < 0){
          fprintf(stderr,"[ERR] Can't unmap file : %s (%d)\n", strerror(errno), errno);
          return;
      }
  }else{
      fprintf(stderr,"[ERR] Can't mmap file : %s(%d)\n",strerror(errno), errno);
  }
}
```

7.2.2 Le serveur

Le serveur commence en faisant la même chose que le client, c'est-à-dire vérifier si le module Bluetooth est présent et bien activé puis réalise une recherche d'identités avec le nom de l'identité très utile pour savoir si la carte Fox a bien démarré et lancé les modules noyau nécessaire au fonctionnement du Bluetooth.

```
 inquiry_info *info = NULL;
 /* The inquiry should last 8 * 1.28 = 10.24 seconds */
```

```
    printf("[INFO] Scanning ...\n");
    flags |= IREQ_CACHE_FLUSH; /* flush inquiry cache */
    num_rsp = hci_inquiry(dev_id, length, num_rsp, NULL, &info, flags);
    if (num_rsp < 0) {
        perror("Inquiry failed");
        exit(1);}
   dd = hci_open_dev(dev_id);
   for (i = 0; i < num_rsp; i++) {
        memset(name, 0, sizeof(name));
        if (hci_read_remote_name(dd, &(info+i)->bdaddr, sizeof(name), name, 100000) < 0)
            strcpy(name, "n/a");
        ba2str(&(info+i)->bdaddr, btaddr);
        printf("\t%s\t%s\n", btaddr, name);
    }
```

Ensuite comme un serveur classique, une fonction `img_listen` tourne indéfiniment en écoutant sur un PSM (Protocol Service Multiplexer) non réservé, dans notre cas `imgserver_psm = 0xaa77`. À la connexion d'un client, diverses informations sont lues par le serveur qui pourrait nous être utiles par la suite comme l'adresse du client, l'identifiant de connexion, la configuration de la connexion MTU et le temps avant le transfert des données en tampon.

```
static void img_listen(){
    struct sockaddr_l2 addr;
    struct l2cap_options opts; /* for incoming, outcoming mtu */
    unsigned int optlen;
    struct l2cap_conninfo conn;
    int sock_serv; /* socket server */
    int sock_newcon;  /* socket for new connection */
    int backlog = 10; /* length for waiting connections */
    char bastr[18];
    /* create socket */
    sock_serv =  socket(AF_BLUETOOTH,SOCK_SEQPACKET,BTPROTO_L2CAP);
    if(sock_serv < 0){
        exit(EXIT_FAILURE);
    }
    /* bind to our local address */
    addr.l2_family = AF_BLUETOOTH;
    addr.l2_psm = htobs(imgserver_psm) ;
    bacpy(&addr.l2_bdaddr,BDADDR_ANY);
    if(bind(sock_serv,(struct sockaddr *) &addr, sizeof(addr)) < 0){
        close(sock_serv);
        exit(EXIT_FAILURE);
    }
    /* get default l2cap options */
    memset(&opts,0,sizeof(opts));
    optlen = sizeof(opts);
    if(getsockopt(sock_serv,SOL_L2CAP,L2CAP_OPTIONS, &opts,&optlen) < 0){
        close(sock_serv);
        exit(EXIT_FAILURE);
    }
```

```
    fprintf(stderr,"[INFO]| L2cap options: imtu %d omtu %d\n",opts.imtu,opts.omtu);
    /* Listen for client connections */
    if(listen(sock_serv,backlog) < 0){
        close(sock_serv);
        exit(EXIT_FAILURE);
    }
    fprintf(stderr,"[INFO] Waiting for connection on psm %d\n",imgserver_psm);
    while(1){
      memset(&addr, 0, sizeof(addr));
      optlen = sizeof(addr);
      sock_newcon = accept(sock_serv, (struct sockaddr *)&addr, &optlen);
      if(sock_newcon < 0){
         fprintf(stderr,"[ERR] Accept failed : %s (%d)",
         strerror(errno),errno);
        continue;
      }
      /* get current options */
      memset(&opts,0,sizeof(opts));
      optlen = sizeof(opts);
      if(getsockopt(sock_newcon, SOL_L2CAP, L2CAP_OPTIONS, &opts, &optlen) < 0){
          close(sock_newcon);
          continue;
      }
      /* get connection infos */
      memset(&conn,0,sizeof(conn));
      optlen = sizeof(conn);
      if(getsockopt(sock_newcon, SOL_L2CAP, L2CAP_CONNINFO, &conn, &optlen) < 0){
        close(sock_newcon);
        continue;
      }
      ba2str(&addr.l2_bdaddr,bastr);
      fprintf(stderr,"[INFO] New connection from %s \n \t imtu %d\n  omtu %d\n \
        \t flush_to %d\n \t mode %d\n \t handle %d\n \t class 0x%02x%02x%02x\n",
        bastr, opts.imtu, opts.omtu, opts.flush_to ,opts.mode, conn.hci_handle ,
        conn.dev_class[2], conn.dev_class[1], conn.dev_class[0]);
        process_request(sock_newcon);
        close(sock_newcon); /* client disconnect */
  }
}
```

la fonction `process_request` se charge de dialoguer en suivant le protocole et de recevoir l'image pour l'écrire au final dans un fichier et valider la réception.

```
void recv_img(int sck, char *filename, ssize_t length, int inmtu)
{
  int fdfile, recv_length, rlen, wlen;
  char buf[inmtu];
  request_t req;

  recv_length = 0;
  fdfile = creat(filename,0664);
```

```
  while(recv_length != length){
    if(length - recv_length)
      rlen = read(sck, buf, inmtu-1);
    else
      rlen = read(sck, buf, length-recv_length);
    if(rlen == -1){
      printf("[ERR] read error transfert aborting\n");
      return;
    }
    wlen = write(fdfile, buf, rlen);
    if(wlen == -1){
      printf("[ERR] write error transfert aborting\n");
      return;
    }
    recv_length = recv_length + wlen;
    fprintf(stderr,"[DEBUG] receive %d bytes\n",wlen);
  }
  close(fdfile);
  fprintf(stderr,"[INFO] Receiving %d bytes\n", recv_length);
  strcpy(req.filename,filename);
  req.content_length = recv_length;
  req.status =  UNDEF;
  send_request_file_ok(sck, &req);
}
}
```

7.3 Envoi d'images en OpenOBEX

7.3.1 Le protocole OBEX

OBject EXchange est un protocole de l'**IrDA** pour simplifier l'échange de données binaires entre périphériques. Initialement créé pour les transports sur faisceaux infrarouges, il a été par la suite adapté à d'autre canaux à bande passante réduite, notamment l'USB et le Bluetooth. La conception du protocole et les fonctionnalités sont similaires au protocole HTTP dans le sens où le client utilise un mode connecté pour se connecter sur un serveur et recevoir et/ou envoyer des requêtes. Cependant ces deux protocoles sont différents sur plusieurs points :

- Transport : contrairement à l'HTTP qui se base sur la couche TCP/IP, Obex se place au-dessus de la couche RFCOMM pour le Bluetooth.
- Transmission binaire : les requêtes OBEX utilisent un format binaire type-longueur-données pour échanger des informations, ce qui facilite l'analyse grammaticale d'une requête pour des périphériques de puissance de calcul limitée.
- Support des sessions : avec OBEX, une connexion peut supporter plusieurs situations par exemple il est possible de continuer une transaction arrêtée brutalement dans le même état avant l'arrêt.

7.3.2 Utilisation de OpenObex sur la carte Fox

OpenOBEX est une bibliothèque qui implémente le protocole de session OBEX. Pour utiliser l'API OpenOBEX, il faut inclure l'entête `openobex/obex.h`. La bibliothèque est disponible dans le SDK, il suffit de le valider durant la configuration : il se trouve dans `Driver settings->OpenOBEX` en n'oubliant pas d'activer `Bluetooth BlueZ libraries and tools`. Cependant il nécessite quelques modifications des scripts de compilation pour utiliser `uclibc`, trouver la librairie bluez-libs et s'installer au bon endroit, c'est-à-dire dans notre cas `target/cris-axis-linux-gnuuclibc/`.

Nous allons décrire simplement la façon dont une communication OBEX sur du Bluetooth s'ordonne en citant les principaux appels de fonctions : pour en savoir d'avantage la documentation de OpenOBEX est une véritable mine d'or, sinon les sources de la bibliothèque qui contient des exemples très simples. Tout d'abord il faut créer une instance OBEX en appelant la fonction `OBEX_init` où il faut spécifier le type de transport, dans notre cas `OBEX_TRANS_BLUETOOTH`, et une fonction de rappel (callback) de gestion des évènements. Cette fonction retourne un identifiant de connexion (handle) qu'il faut utiliser pour toutes les autres fonctions de la bibliothèque. La fonction de gestion d'évènements doit avoir le prototype suivant : `void fct_rappel(obex_t *handle, obex_object_t *object, int mode, int event,int obex_cmd, int obex_rsp)`

- `handle` : identifiant de connexion
- `object` : structure representant un objet OBEX
- `mode` : OBEX_CLI pour un évènement d'un client et OBEX_SRV pour un évènement du serveur
- `event` : l'évènement reçu (voir `obex_const.h` pour avoir la liste complète)
- `obex_cmd` : commande OBEX selon le type d'évènement
- `obex_rsp` : réponse OBEX selon le type d'évènement

Voici la fonction de rappel du client :

```
void obex_event(obex_t *handle, obex_object_t *object, int mode, int event,
                int obex_cmd, int obex_rsp)
{
  switch (event)        {
  case OBEX_EV_PROGRESS:
    printf("Made some progress...\n");
    break;
  case OBEX_EV_ABORT:
    printf("Request aborted!\n");
    break;
  case OBEX_EV_REQDONE:
    client_done(handle, object, obex_cmd, obex_rsp);
    break;
  case OBEX_EV_REQHINT:
    switch(obex_cmd){
    case OBEX_CMD_PUT:
    case OBEX_CMD_CONNECT:
    case OBEX_CMD_DISCONNECT:
      OBEX_ObjectSetRsp(object, OBEX_RSP_CONTINUE, OBEX_RSP_SUCCESS);
      break;
    default:
```

```
      /* reject other commands */
      OBEX_ObjectSetRsp(object, OBEX_RSP_NOT_IMPLEMENTED,
                        OBEX_RSP_NOT_IMPLEMENTED);
      break;
    }
    break;
  case OBEX_EV_LINKERR:
    OBEX_TransportDisconnect(handle);
    printf("Link broken!\n");
    break;
  default:
    printf("Unknown event %02x!\n", event);
    break;
  }
}
```

Pour finir, la fonction `OBEX_HandleInput` équivalente à la fonction `select` permet de lire et traiter les données entrantes mais bloque si aucune donnée n'est disponible. Cette fonction doit être appelée si un évènement est attendu ou bien si un évènement a été envoyé.

7.3.3 Le client

Pour utiliser le protocole OBEX, le client doit connecter le transport pour le Bluetooth avec la fonction `BtOBEX_TransportConnect` en passant comme paramètre l'identifiant de connexion, l'adresse source (souvent `BDADDR_ANY`), l'adresse destination et le canal RFCOMM utilisé. Le client doit maintenant envoyer une requête OBEX `Connect` comme toutes autres commandes OBEX, puis une fois la requête finie le client ne doit pas oublier d'envoyer une commande `Disconnect` mais aussi déconnecter le transport avec `OBEX_TransportDisconnect`. Pour envoyer une requête OBEX comme `Connect` ou `Disconnect`, il faut créer un object OBEX en appelant `OBEX_ObjectNew` avec le nom de la commande puis rajouter des entêtes en utilisant `OBEX_ObjectAddHeader` et envoyer cette requête avec `OBEX_Request`. Un évènement `OBEX_EV_REQDONE` est envoyé dès que la requête est finie.

Voyons la requête d'envoi d'image, en notant que la fonction `wait_event` est simplement l'appel de la fonction bloquante `OBEX_HandleInput` pour s'assurer que la requête s'est bien effectuée :

```
void push_image(obex_t *handle, char *filename, char *fsize, char *fmap)
{
  obex_object_t *object;
  obex_headerdata_t hd;
  uint8_t *uname;

  object = OBEX_ObjectNew(handle, OBEX_CMD_PUT);
  uname_size = (strlen(filename)+1)<<1;
  uname = malloc(uname_size);
  OBEX_CharToUnicode(uname, (uint8_t *) filename, uname_size);
  hd.bs = uname;
  OBEX_ObjectAddHeader(handle, object, OBEX_HDR_NAME, hd, uname_size, 0);
```

```
  hd.bq4 = fsize;
  OBEX_ObjectAddHeader(handle, object, OBEX_HDR_LENGTH, hd, sizeof(uint32_t), 0);
  hd.bs = fmap;
  OBEX_ObjectAddHeader(handle, object, OBEX_HDR_BODY, hd, fsize,0);
  OBEX_Request(handle, object);
  wait_event(handle);
}
```

7.3.4 le serveur

Après avoir enregistré le transport, le serveur doit demander au transport de recevoir les connections entrantes avec la fonction `OBEX_ServerRegister` sur un canal non réservé. Quand une connexion survient, le serveur reçoit l'évènement `OBEX_EV_ACCEPTHINT` : dans notre cas on appelle `btobex_accept`.

Voici une fonction de rappel typique d'un serveur :

```
switch (event) {
 case OBEX_EV_REQ:       /* incoming request */
     switch(obex_cmd) {
     case OBEX_CMD_CONNECT:
     case OBEX_CMD_DISCONNECT:
     /* Response is already set to success by OBEX_EV_REQHINT event */
         break;
     case OBEX_CMD_PUT:
         put_image(object);
         break;
     }
     break;
 case OBEX_EV_REQHINT:/* new request is coming in */
     switch(obex_cmd) { /* Accept some commands! */
     case OBEX_CMD_PUT:
     case OBEX_CMD_CONNECT:
     case OBEX_CMD_DISCONNECT:
         OBEX_ObjectSetRsp(object, OBEX_RSP_CONTINUE, OBEX_RSP_SUCCESS);
         break;
     default:
         /* Reject any other commands */
         OBEX_ObjectSetRsp(object, OBEX_RSP_NOT_IMPLEMENTED,OBEX_RSP_NOT_IMPLEMENTED);
             break;
     }
     break;
 case OBEX_EV_REQDONE:
     if(obex_cmd == OBEX_CMD_DISCONNECT) {
       /* Disconnect transport here */
     }
     break;
 case OBEX_EV_LINKERR:
     /* error */
     break;
 default:
```

```
      break;
  }
```

Pour répondre à une requête suite à `OBEX_EV_REQ`, on utilise `OBEX_ObjectSetRsp` qui sera suivie de `OBEX_EV_REQDONE` en cas de succès. On traite la demande en lisant les entêtes à l'aide de `OBEX_ObjectGetNextHeader`. Par exemple pour `PUT` :

```
void put_image(obex_object_t *object)
{
  obex_headerdata_t hv;
  uint8_t hi;
  uint32_t hlen;

  const uint8_t *body = NULL;
  int body_len = 0;
  char *name = NULL;
  char *namebuf = NULL;

  while (OBEX_ObjectGetNextHeader(handle, object, &hi, &hv, &hlen))     {
        switch(hi){
          case OBEX_HDR_BODY:
                body = hv.bs;
                body_len = hlen;
                break;
                case OBEX_HDR_NAME:
                        if( (namebuf = malloc(hlen / 2)))       {
                                OBEX_UnicodeToChar((uint8_t *) namebuf, hv.bs, hlen);
                                name = namebuf;
                        }
                        break;
                case OBEX_HDR_LENGTH:
                        printf("OBEX_HEADER_LENGTH = %d\n", hv.bq4);
                        break;
                default:
                        printf("Skipped header %02x\n",hi);
                }
        }
        if(!body){
                printf("Got a PUT without a body\n");
                return;
        }
        if(!name){
                printf("Got a PUT without a name\n");
                return;
        }
        save_file(name, body, body_len);
        free(namebuf);
}
```

Suite à la réception d'une requête OBEX `Disconnect`, et seulement après avoir répondu au client, on peut déconnecter le transport.

7.4 Conclusion et perspectives

Nous avons présenté les briques de base pour la réalisation d'un réseau de capteurs : une plate-forme de calcul puissante, de consommation électrique modeste, équipée de modes de communications sans-fil et de capteurs. Nous avons présenté le développement de clients/serveurs d'une part pour le transfert de données scalaires qui nous ont permis de mesurer la qualité de la liaison radiofréquence avec la distance entre station fixe et capteur mobile, puis nous avons illustré notre capacité à communiquer des informations volumineuses en temps réel par le transfert d'images *via* une liaison Bluetooth, et ce en profitant des couches de laison L2CAP et OBEX.

Un certain nombre de points n'ont pas été traités, notamment la mise en communication de plusieurs cartes qui implique l'utilisation d'un protocole de routage avec une prise en charge de l'énergie ainsi que la capacité à reformer dynamiquement le réseau en fonction d'informations géographiques et d'autonomie des batteries.

8 Remerciements

E.P.C est doctorant au Laboratoire d'Informatique de Franche Comté (LIFC) sous la direction de H. Guyennet. G.W. a effectué ce travail dans le cadre d'un projet de Master Informatique en collaboration avec le LIFC, le Laboratoire de Physique et Métrologie des Oscillateurs (FEMTO-ST/LPMO) et l'association étudiante Projet Aurore à Besançon. J.-M.F est membre de l'association Projet Aurore, et ingénieur hébergé par le LPMO.

Références

[1] `http://www.acmesystems.it/`

[2] `http://developer.axis.com/`

[3] D. Bodor, *Linux embarqué sur carte ACME Fox*, GNU/Linux Magazine France Hors Série **27** (Octobre/Novembre 2006)

[4] William Stallings, *Réseaux et communication sans fil, 2ème Edition* (2005)

[5] Documentation sur ZigBee : `http://www.ifn.et.tu-dresden.de/Emarandin/ZigBee/`

[6] `http://www.xgarreau.org/`

[7] X. Garreau, *Bluetooth : aperçu de la spécification*, GNU/Linux Magazine France **78**, Décembre 2005

[8] M. Ilyas Ed., *The Handbook of Ad Hoc Wireless Networks*, CRC Press (2003)

[9] `http://www.bluez.org/`

[10] P. Santi, *Topology Control in Wireless Ad Hoc Sensor Networks*, John Wiley & Sons (2005)

[11] J.-M Friedt, S. Guinot *Introduction au Coldfire 5282* GNU/Linux Magazine France, **75** (Septembre 2005)

[12] Colour difference coding in computing `http://vektor.theorem.ca/graphics/ycbcr/`

[13] Color FAQ http://www.poynton.com/ColorFAQ.html

[14] Gamma FAQ http://www.poynton.com/GammaFAQ.html

[15] RGB/YUV Pixel Conversion : http://www.fourcc.org/fccyvrgb.php

[16] Video for Linux resources : http://www.exploits.org/v4l/

[17] P. Ficheux, *Programmation de l'API Video for Linux*, GNU/Linux Magazine France **72** (Mai 2005), disponible à http://pficheux.free.fr/articles/lmf/v4l/video4linux_img_final.pdf

[18] V4L2 API Specification : http://v4l2spec.bytesex.org/spec/book1.htm