Communication asynchrone sécurisée dans un MANet de Raspberry Pi & radio nRF24L01

Connexion du composant nRF24L01 au Raspberry PI

nRF24L01	Raspberry PI	*Usage*
IRQ	GPIO17	**Utilisée !**
CE	GPIO22	*8ème en partant du haut*
VCC	3.3v	*juste en dessous*
MO	GPIO10	*juste en dessous*
MI	GPIO9	*juste en dessous*
SCK	GPIO11	*juste en dessous*
GND	GND	*juste en dessous*
CSN	GPIO7	*juste à côté*

Interruption & nRF24L01

Le composant nRF24L01 est capable de déclencher une IRQ sur le matériel auquel il est connecté, afin de l'avertir des 2 situations suivantes :

- ▷ envoi réussi ;
- ▷ trame reçue et disponible ;
- ▷ échec de l'envoi à l'issu du nombre total de réémission.

Gestion de l'IRQ sur le Raspberry Pi grâce à la bibliothèque wiringPi

Au niveau connectique, nous connecterons la broche IRQ du composant nRF24L01 à la broche « GPIO Pin 17 » du Raspberry. L'IRQ sera détectée par un « *front descendant* » sur cette broche :

```
1 if (wiringPiSetup () < 0) {
2    fprintf (stderr, "Unable to setup wiringPi: %s\n", strerror (errno));
3    return 1;
4 ...
5 if ( wiringPiISR (IRQ_PIN, INT_EDGE_FALLING, &interruption) < 0 ) {
6    fprintf (stderr, "Unable to setup ISR: %s\n", strerror (errno));
7    return 1;
8 }
```

Configuration logicielle pour le projet

Installation de la bibliothèque de gestion du composant nRF24L01 et de l'IRQ :

```
xterm
$ git clone https://git.p-fb.net/pef/RF24IRQ.git
$ cd RF24IRQ
$ cd wiringPi
$ ./build
$ cd ../NRF24
$ sudo make install
$ cd examples
$ make
$ sudo ./chat_rf
```

Pour l'installation de la bibliothèque « PyCrypto » :

```
xterm
$ sudo apt-get install python-dev
$ sudo apt-get install python-setuptools
$ sudo easy_install pip
$ sudo pip install pycrypto
```

Création d'un programme de « communication » entre Raspberry PI & nRF24L01

Nous allons écrire un programme de dialogue entre deux terminaux mobiles où chaque terminal pourra communiquer avec l'autre de manière asynchrone.

Grâce à l'utilisation de l'IRQ, nous pouvons concevoir le programme de la manière suivante :

1. le composant radio nRF24L01 est configuré :
   - choix du canal de communication ;

Canal	Frequence (Mhz)	Description
0 à 82	2400 à 2482	autorisé mais bruité (conflits avec les réseaux WiFi Bluetooth etc.)
83 à 99	2483 à 2499	non autorisé pour les mobiles
100	2500	restreint
101 à 119	2501 à 2519	autorisé
120 à 125	2520 à 2525	restreint

   En cas d'utilisation de 2Mbps, la transmission utilise 2 canaux simultanément.
   - de la puissance d'émission ;
   - du nombre de tentative de réémission en cas d'échec ;
   - des adresses source et destination (pour la bibliothèque RF24, le lien est un « pipe ») ;

```
// Setup for GPIO 22 CE and CE1 CSN with SPI Speed @ 8Mhz
RF24 radio(RPI_V2_GPIO_P1_15, RPI_V2_GPIO_P1_26, BCM2835_SPI_SPEED_8MHZ);

// Radio pipe addresses for the 2 nodes to communicate.
const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

int main(int argc, char** argv)
{
  // sets up the wiringPi library
  if (wiringPiSetup () < 0) {
      fprintf (stderr, "Unable to setup wiringPi: %s\n", strerror (errno));
      return 1;
  }
  // Setup and configure rf radio
  radio.begin();
  radio.setPayloadSize(32);
  radio.setRetries(5,15);
  radio.setChannel(0x6e);
  radio.setPALevel(RF24_PA_LOW);
```

2. le traitement de l'IRQ est positionné sur une fonction ne traitant que de la réception d'un message ;

3. le programme effectue une boucle sur l'entrée clavier pour lire un message en provenance de l'utilisateur :
   - bascule du composant radio nRF24L01 en mode envoi ;
   - envoi du message et gestion des acquittements/réémissions automatiques grâce au mode « *Shockburst ™* » du composant ;
   - retour dans le mode réception ;

4. lors de la réception d'une IRQ liée à la réception d'un message :
   - récupération du contenu du message et écriture de son contenu sur le canal non bufferisé `stderr`.

```
void interruption(void)
{
  char message[33];

  if (!en_ecoute) return;
  radio.read(message, sizeof(char)*32);
  radio.startListening();
  message[32] = '\0';
  // Verify if it's a message
  if (message[0] != '\0')
        fprintf(stderr,"%s", message);
}
```

Encapsulation de « `chat_rf` » dans Python

Il est possible de contrôler le programme de communication asynchrone conçu précédemment dans un programme Python pour lui permettre de contrôler les messages en entrée comme en sortie du module radio, grâce au code suivant :

```
#!/usr/bin/python

import subprocess, os, pty

p = subprocess.Popen(["./chat_rf"],stderr=subprocess.PIPE,stdin=subprocess.PIPE,buf
size=0)

pid = os.fork()

if pid :
        while(1):
                ligne = p.stderr.readline()
                print ligne.rstrip('\n')
else :
        while(1):
                ligne = raw_input()
                p.stdin.write(ligne+'\n')
```

On utilise le canal de sortie d'erreur qui est non bufferisé.

Travail

1 – Vous étudierez et répondrez aux questions suivantes :

a. le canal par défaut est indiqué par la valeur `0x6e`, à quelle fréquence correspond-il ?

b. est-ce que le canal de communication est partagé ou non entre tous les terminaux présents dans la salle de TP ?

c. sur l'usage des adresses d'émission et de réception des messages ;
   - ◇ le taux d'erreur et la sensibilité ont-ils un rapport avec la bande passante et la vitesse de transmission ?
   - ◇ existe-t-il une possibilité de *broadcast* ?
   - ◇ comment est réalisée la gestion des collisions ?

d. La gestion des bascule « mode émetteur »/« mode récepteur » :
   - ◇ est-elle nécessaire ?
   - ◇ est-elle immédiate ?

e. les possibilités de routage entre raspberry :
   - ◇ réseau « *single hop* » *vs* « *multi-hop* » ;
   - ◇ portée radio/découverte des voisins/relais des échanges.

Vous vous aiderez de la documentation du composant nRF24L01 accessible à `http://p-fb.net/fileadmin/_migrated/content_uploads/nRF24L01_Product_Specification_v2_0.pdf`

2 – Vous comparerez le protocole de communication du « `chat_rf` » avec celui du « `send_receive` » proposé dans la fiche précédente :

a. Comment l'IRQ est-elle déclenchée par le composant radio nRF24L01 sur le Raspberry Pi ?

b. Quelles améliorations apportent l'utilisation d'une IRQ pour l'écriture du programme « `chat_rf` » ?

c. Quels sont les avantages du protocole de communication du programme « `chat_rf` » ?

3 – Le contenu du message transmis par le nRF24L01 est **chiffré** maintenant à l'aide d'AES-CTR :

a. Expliquez en quoi l'utilisation du mode CTR est *nécessaire* dans le contexte de communication réseau.

b. Est-ce que cette méthode de chaînage permet de faire de :
   - ◇ l'authentification ?
   - ◇ l'intégrité ?

c. Comment peut-on échanger les clés de chiffrement entre deux appareils ?

4 – Vous écrirez un programme Python permettant de faire des échanges sécurisés :

a. vous définierez un format de message (les messages sont limitées à 32 octets) contenant :
   - ◇ le nom de l'expéditeur : 4 octets ;
   - ◇ le nom du destinataire : 4 octets ;
   - ◇ le contenu du message sur au plus 16 octets ;

b. Écrivez un programme Python permettant de tester l'envoi et la réception de vos messages avec le format indiqué en combinaison avec le programme « `chat_rf` ».

c. Améliorez votre programme Python précédent pour échanger des messages chiffrés.

Attention

Pour permettre l'échange du message chiffré (binaire) entre le programme Python et le « chat », votre message devra être encodé en base64 :

xterm

```
>>> import base64
>>> a=base64.encodestring('toto')
>>> a
'dG90bw==\n'
>>> base64.decodestring(a)
'toto'
```

Exemple de procédure de création de message chiffré :
- ◇ limitation des messages à 16 octets à cause du codage en base64 (taille augmentée de 30%)
- ◇ utilisation d'une opération AES en mode CTR, « *counter* », utilisant un secret connu des deux interlocuteurs ;
  Le mode CTR permet de rendre le chiffrement dépendant du numéro de ce compteur : pour une même clé le chiffrement sera différent pour chaque valeur du compteur.

Exemple d'utilisation du chiffrement par block AES en mode compteur :

```
#!/usr/bin/python
from Crypto.Cipher import AES

# le compteur doit etre donne sur 16 octets au travers d'une fonction (ou 24 ou 32)
# le secret doit faire 16 octets
cipher=AES.new('ma cle super secrete d'au moins 16 caracteres'[:16],
   AES.MODE_CTR,counter=lambda : "0"*16)
# le chiffre est une sequence de 16 octets
chiffre=cipher.encrypt("A"*16)
```