ENSEEIHT - Département Sciences du numérique
1SN - Cours réseaux locaux

TP1 et TP2 : Méthodes d'accès

Katia Jaffrès-Runser

Mai 2021

Présentation du sujet

Contexte et objectif Dans le cours de réseaux locaux, différentes méthodes d'accès ont été présentées. Ces méthodes permettent de partager l'accès au médium des différents utilisateurs du même réseau local. Ces deux séances de TP ont pour objectif d'illustrer les méthodes d'accès les plus simples en les implantant sur deux capteurs équipés d'une interface de communication radio. Ces méthodes d'accès seront instrumentées afin de comparer leur efficacité en terme de taux de de succès trame.

Matériel fourni Vous utiliserez deux capteurs DecaWiNo (`https://wino.cc/decawino/`) composés d'un carte Teensy et d'une interface sans-fil DecaWave DW1000. La technologie de communication est l'Ultra-Wide Band (UWB) [1].

La taille maximale des trames émises est de 127 octets codés et il existe 3 débits possibles : 110kbps, 850kbps et 6.8Mbps. L'envoi d'une trame de 12 octets avec 10 octets de données utiles dure : 3.042ms, 380.3μs et 101.3μs pour les 3 débits respectivement. Le débit utilisé dans ce TP est 6.8Mbps.

La bibliothèque `DecaDuino` La bibliothèque `DecaDuino` permet d'émettre et recevoir des trames, de changer la fréquence de communication, etc. Elle est décrite dans la documentation suivante :
`https://www.irit.fr/~Adrien.Van-Den-Bossche/decaduino/index.html`

Cette interface de programmation s'appuie sur l'interface Arduino et offre un driver pour manipuler la couche physique UWB. L'environnement de développement est donc l'environnement Arduino, auquel deux modules extérieurs Teensy et DecaDuino ont été rajoutés. Tous les postes Linux sont équipés de l'environnement de programmation requis [2]

1 Prise en main de l'environnement.

Les deux séances de TP se font par binôme. Deux capteurs sont fournis, l'un étant utilisé pour émettre des trames et l'autre pour en recevoir. Vous utiliserez deux ordinateurs pour cela (un

1. La compréhension de la technologie UWB n'est pas un pré-requis à ce TP.
2. Si vous souhaitez installer la suite de développement Arduino+Teensy+DecaDuino, suivre les instructions suivantes : `https://wino.cc/tutorials/teensywino/teensywino-installing-software-development-tools/` (Arduino + Teensy) et `https://github.com/irit-rmess/DecaDuino` (librairie DecaDuino).

capteur par ordinateur).

1.1 Configuration initiale

Voici la marche à suivre pour éditer un programme qui pilote l'émission de trames sur un capteur et la réception de trames sur l'autre.

1. **Attention, lors de la première séance de TP**, il vous faudra télécharger la dernière version de la librairie DecaDuino sur `https://github.com/irit-irt/DecaDuino` et la décompresser dans le dossier `Arduino/libraries` de votre home.
2. Lancer l'environnement de développement Arduino depuis un terminal avec la commande : `arduino185`
3. Brancher les capteurs sur un port USB et vérifier dans le menu **Outils/types de cartes** que la carte **Teensy 3.2/3.1** est utilisée. La sélectionner dans le cas contraire.
4. Télécharger l'archive présentant des exemples de code sous Moodle (cours réseaux 1SN, partie réseaux locaux). La décompresser dans votre répertoire.
5. Ouvrir les deux Croquis (ou sketchs Arduino) qui permettent l'émission et la réception de trames qui se trouvent dans l'archive à partir de l'application Arduino :
   — `DecaDuinoSender` pour l'émission
   — `DecaDuinoReceiver` pour la réception
   Lire le code C des deux sketchs. Il n'existe que deux sous-programmes, l'un pour initialiser le capteur (`void setup()`) et l'autre pour décrire les instructions réalisées en boucle par le capteur (`void loop()`).
6. Configurer le port qui permet d'échanger les données entre le capteur et votre ordinateur en USB. Pour cela, dans le menu **Outils/USB type** choisir l'option **Serial** et dans le menu **Outils/Port** choisir l'option **/dev/ttyACM0 (Teensy)**. Attention, l'entier 0 peut valoir 1, 2, 3...

2 Envoi et réception de trames simple.

Téléverser le code sur les capteurs A vous de lancer maintenant le sketch en émission sur un ordinateur, et celui en réception sur l'autre. Les trames émises comportent 10 octets de données utiles. On gardera cette taille pour tout le TP. Pour exécuter un croquis, aller dans le menu **Croquis** ou utiliser l'icône qui représente un flèche horizontale.

Question 1. Qu'observez-vous ?

Observer le taux de succès trames. Vous noterez que les deux sketchs comptent le nombre de trames émises et reçues. Ces valeurs sont ensuite envoyées sur le port série avec l'instruction `Serial.print(txNb)` ou `Serial.print(rxNb)`.

Il est possible d'observer ces valeurs à l'aide du *Moniteur série* ou du *Traceur série*. Ils sont accessibles dans le menu **Outils**.

Note : Attention, pour que la communication se passe bien, il faut que le Moniteur série soit configuré avec la vitesse de modulation identique à celle mentionnée dans le code à l'instruction :

```
Serial.begin(115200); // Init Serial port speed
```

de la procédure `void setup()`.
Dans cet exemple, la vitesse de modulation est de 115200 bauds. La configuration de la vitesse du port série se fait dans le menu **Outil/Moniteur série**.

Question 2. Qu'observez-vous ?

Question 3. Calculer un taux de succès trame. On rappelle que le taux de succès trame est le rapport entre le nombre de trames reçues et le nombre de trames envoyées.
Qu'en déduire ?

3 Isoles les communications.

Pour isoler les communications de chaque binôme, nous allons créer un canal de communication virtuel. On utilisera pour cela un entier entre 0 et 255, unique dans le réseau, qui identifie une paire de capteurs communicants. Ce numéro de 'canal virtuel' est écrit au début de la trame, et permet au destinataire de ne sélectionner que les trames qui le concernent.

3.1 Création du canal virtuel

Voici les étapes à réaliser pour cela :

- Choisir un entier qui représentera de façon unique votre binôme.
  A cet effet, nous vous suggérons de prendre l'adresse d'un des deux capteurs que vous possédez (adresse écrite sur la gommette) pour identifier votre canal virtuel car elles sont uniques.
- Dans la trame émise, écrire dans l'octet 0 votre identifiant de canal virtuel.
- En réception, ne traiter que les trames reçues qui comportent cet identifiant de canal dans l'octet 0.

Question 4. Calculer à nouveau le taux de succès trame. Qu'observez-vous ?

Question 5. On pourrait aussi définir un canal virtuel avec la paire d'entiers `(addr_source, addr_dest)`, avec `addr_source` et `addr_dest` les adresses des capteurs source et destination.
Quels sont les avantages d'une telle solution ?

3.2 Période d'émission

La charge du réseau n'est pas très importante. Une trame est émise toutes les 3 secondes.

Question 6. Réduire la période d'émission des trames et observer à nouveau le taux de succès. Trouver la période pour laquelle on observe une baisse significative du taux de succès (de l'ordre de 20% quand tous les émetteurs sont actifs).

Note 1 : il est possible d'utiliser la fonction `delayMicroseconds()` de la bibliothèque Arduino.
Note 2 : l'envoi de messages depuis le port série vers l'ordinateur peut se bloquer s'il y a trop d'envois. Il est bon de limiter les données remontées vers l'ordinateur sur le port série si vous n'observez plus de traces d'exécution.
Note 3 : on conservera la période trouvée pour la suite du TP.

4 Méthode d'accès statique

La méthode d'accès illustrée depuis le début de ce TP n'est autre qu'ALOHA. Nous proposons dans cette partie de définir une méthode d'accès statique de type FDMA (Frequency Division Multiple Access).

Table 61: DW1000 supported UWB channels and recommended preamble codes

Channel number	Centre frequency (MHz)	Bandwidth (MHz)	Preamble Codes (16 MHz PRF)	Preamble Codes (64 MHz PRF)
1	3494.4	499.2	1, 2	9, 10, 11, 12
2	3993.6	499.2	3, 4	9, 10, 11, 12
3	4492.8	499.2	5, 6	9, 10, 11, 12
4	3993.6	1331.2 *	7, 8	17, 18, 19, 20
5	6489.6	499.2	3, 4	9, 10, 11, 12
7	6489.6	1081.6 *	7, 8	17, 18, 19, 20
N.B.	For correct operation of the DW1000 the software must take care to only allow selection of those preamble codes appropriate for the configured PRF.			

* The DW1000 has a maximum receive bandwidth of 900 MHz

FIGURE 1 – Canaux fréquentiels UWB disponibles sur le DW1000. Source : DW1000 user manual

4.1 Canaux fréquentiels UWB

La technologie UWB permet d'utiliser des canaux fréquentiels orthogonaux pour répartir les communications. La Figure 1 montre qu'on peut utiliser 6 canaux fréquentiels avec le DW1000, chaque canal étant définit par la fréquence centrale du canal et sa large de bande. Par exemple, le canal 1 a une fréquence centrale de 3.4944 GHz et occupe une bande fréquentielle d'environs 500MHz. Cela signifie que le spectre du signal se trouve entre 3.4944-0.250 GHz et 3.4944+0.250 GHz, soit dans l'intervalle [3.244MHz, 3.7444MHz].

Question 7. Combien de canaux au maximum ne se recouvrent pas ? Quels sont-ils ?

4.2 Répartition des canaux.

Il est possible de connaitre le canal d'émission actuel avec la fonction `uint8_t getChannel()` et de configurer le canal avec `bool setChannel(uint8_t channel)`. Il est aussi possible de changer le code utilisé pour définir le préambule avec les fonctions `bool setTxPcode(uint8_t pcode)` en émission et `bool setRxPcode(uint8_t pcode)` en réception. On notera qu'il faut choisir le code de préambule qui correspond au canal choisi. Ici, l'interface radio est utilise les codes de préambules pour un fréquence de répétition de 16MHz.

Question 8. Répartir les canaux orthogonaux entre les binômes et observer le taux de succès avec la période trouvée dans la partie 3.2. Pour cela, il faudra modifier vos sketchs Arduino.

5 Mécanisme de backoff

Dans cette partie, on s'intéresse à améliorer la méthode d'accès ALOHA en introduisant une mécanisme de backoff avant l'émission de chaque trame. Pour cela, on tirera un temps d'attente aléatoire, multiple de la durée d'émission T d'une trame. Pour cela, on tire un entier k dans une fenêtre de taille $[0, W[$ (W est exclu de l'intervalle) et on n'émet la trame que si le temps d'attente kT s'est écoulé.

Question 9. Implanter cette attente aléatoire à l'émetteur. Faire varier la taille W entre 2 et 2^{10} et observer son impact sur le taux de succès.

Note : Dans cette partie, on s'assurera que tous les émetteurs fonctionnent sur *le même* canal 1. On gardera aussi la période d'émission choisie à la partie 3.2.

6 Pour aller plus loin

On pourrait :

- Implanter des mécanismes d'acquittement :
  - l'émetteur attend un message ACK du destinataire pendant au plus TIMER millisecondes avant d'émettre la trame suivante
  - introduire un mécanisme d'acquittement avec du piggybacking
  - utiliser une mécanisme à fenêtre pour faire en plus du contrôle de congestion.
- Implanter un mécanisme RTS/CTS, pour réserver le canal pendant un certain temps.
- Implanter un polling où un capteur Maitre interroge les capteurs pour leur offrir un temps de parole.
- Implanter un TDMA, où un capteur Maitre envoie des trames balise pour délimiter une supertrame.

7 Documentation

Voici quelques liens utiles :

- Envoyer une donnée sur le port série.
  `https://www.arduino.cc/en/Serial/Print` `https://www.arduino.cc/en/Serial/Println`
- Pour accéder à la documentation de la classe DecaDuino
  `https://www.irit.fr/~Adrien.Van-Den-Bossche/decaduino/class_deca_duino.html`
- Générer un entier aléatoire en C pour Arduino
  `https://www.arduino.cc/reference/en/language/functions/random-numbers/random/`