TP Synchronisation

1. Concepts transversaux

Au cours de ce TP, certains concepts et sections de code ont été réutilisés quasi systématiquement. Ces éléments communs sont présentés dans cette section.

Adressage

À chaque fois qu'une absence d'adressage a dû être résolue (c.à.d. questions 2.1 et 4.1), elle l'a été de la manière suivante :

1. Une variable nommée `panId` (type : `uint8_t`) est initialisée avec la valeur `30` (choisie arbitrairement). Cette valeur constitue le premier champ des trames émises (voir ci-après).
2. À chaque réception d'une trame, un test est effectué sur le premier champ de cette dernière. Si le `panId` correspond, la trame est traitée. Sinon, un appel à la primitive `plmeRxEnableRequest()` est effectué afin de pouvoir lire la trame suivante.

```
if (rxBuffer[0] == panId) {
    // Traitement de la trame...
}
```

Note : Pour mettre en place un adressage basique, on aurait pu utiliser une version sur-définie de la méthode `init(uint32_t shortAddrAndPanId)`. `shortAddrAndPanId` est composé d'une adresse sur 16-bits et d'un PanId sur 16-bits. L'adresse est positionnée sur les bits de poids faible. Néanmoins, sur conseil du professeur, nous sommes partis sur la technique plus simple décrite précédemment.

Format des trames

Ce TP a nécessité la définition de deux types de trames : `DATA` et `ACK`, dont les formats sont définis ci-dessous.

```
      0 ----- 8 -------- 16 ------- *
DATA: | panID | seqNumber | payload |
ACK : | panID | seqNumber |
```

Construction et envoi de trames

Tous les *sketches* ou presque utilisent les mêmes fonctions (ou des variantes de celles-ci), qui suivent le même schéma.

a. `void XXXProtocol()`: Contient l'implémentation du protocole demandé par l'exercice. Cette fonction est appelée par loop().
b. `buildMsgXXX()`: Construit une trame de type DATA ou ACK.
c. `void sendFrame(uint8_t* frame, uint8_t length)`: Envoie la trame construite précédemment.

2. Prise en main du transceiver UWB

2.1. Prise en main des sketchs exemples

Les primitives utilisées ici sont détaillées dans le tableau ci-dessous.

Primitive	**Sémantique**	**Utilisé par :**
`Decaduino()`	Constructeur de la classe	Émetteur et récepteur
`decaduino.init()`	Initialise l'instance de la classe `Decaduino`, ainsi que le composant `DWM1000` sans filtrage des adresses (« *promiscuous mode* »). Retourne `true` si les deux éléments sont initialisés avec succès.	Émetteur et récepteur
`decaduino.engine()`	Traite les interruptions concernant le composant `DWM1000`. Doit être appelé régulièrement.	Émetteur et récepteur
`decaduino.pdDataRequest (uint8_t *buf, uint16_t len)`	Envoie une trame de taille `len` à partir du buffer `buf`. Renvoie `true` en cas de succès, `false` sinon.	Émetteur
`decaduino.hasTxSucceeded()`	Renvoie `true` si la dernière transmission a été effectuée avec succès, `false` sinon.	Émetteur
`decaduino.setRxBuffer (uint8_t *buf, uint16_t *len)`	Prépare un buffer de réception `buf` de taille `len`.	Récepteur
`decaduino.rxFrameAvailable()`	Indique si une trame a été reçue.	Récepteur
`decaduino.plmeRxEnableRequest()`	Passe en mode réception et accepte la prochaine trame.	Récepteur

La mise en place de l'adressage est présentée en page précédente.

2.2. Conception d'un premier protocole simple

L'implémentation de ce protocole est réalisée dans les *sketches* situés dans le répertoire `exercice2_2/`. Les formats de message utilisés sont définis plus haut dans ce document, avec un *payload* de taille arbitrairement fixée à `3`. La taille d'une trames de type DATA (*resp*. ACK) est donc de 5 octets (*resp*. 2 octets).

Le fonctionnement de ce protocole est le suivant :

[Émetteur]

1. Envoi d'un message.
2. Déclenchement d'un timer.
3. Si ACK reçu avant timeout, incrémentation du numéro de séquence et envoi du message suivant,
   Sinon, nouvelle tentative d'envoi du même message, jusqu'à 3 tentatives.

[Récepteur]

1. Réception d'un message.
2. Envoi d'un ACK. Pour simuler la perte de message et tester la fonctionnalité de timeout/retries, la variable `ack_lost` est incrémentée à chaque message reçu. Arbitrairement, un ACK n'est envoyé que pour un message sur cinq.

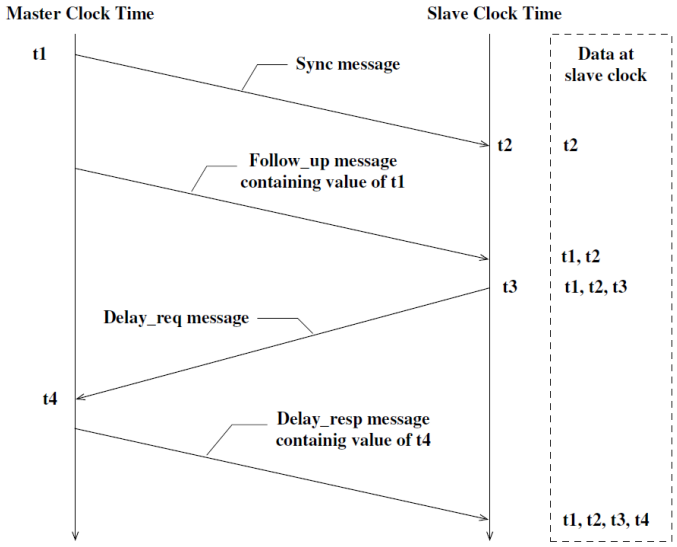
3. Synchronisation MAC

3.1. Implémentation d'une synchronisation en étoile

Les primitives nécessaires à la mise en place de ce protocole sont les mêmes que décrites plus haut, auxquelles s'ajoutent certaines primitives de marquage temporel et d'encodage, décrites dans le tableau ci-dessous.

Primitive	**Sémantique**	**Utilisé par :**
`decaduino.printUint64 (uint64_t msg)`	Affiche un entier de taille 64bits sur la console.	Master et slave
`decaduino.getLastRxTimestamp()`	Retourne le moment de la dernière réception d'une trame sous la forme d'un entier sur 64 bits.	Master et slave
`decaduino.getLastTxTimestamp()`	Retourne le moment de la dernière émission d'une trame sous la forme d'un entier sur 64 bits.	Master et slave
`decaduino.encodeUint64 (uint64_t src, uint8_t *dst)`	Encode l'entier `src` de taille 64 bits dans le tableau d'entier de taille 8bits `dst`.	Master
`decaduino.decodeUint64(uint8_t *dst)`	Construit un entier de taille 64 bits à partir d'un tableau d'entier de taille 8 bits.	Slave

Description du protocole



Formats des messages

Les messages transmis par ce protocole respectent les formats suivants :

```
            0 ----- 8 -------- 16 ---- 80
Sync:       | panID | seqNumber |
Follow_up:  | panID | seqNumber |  t1  |
Delay_req:  | panID | seqNumber |
Delay_resp: | panID | seqNumber |  t4  |
```

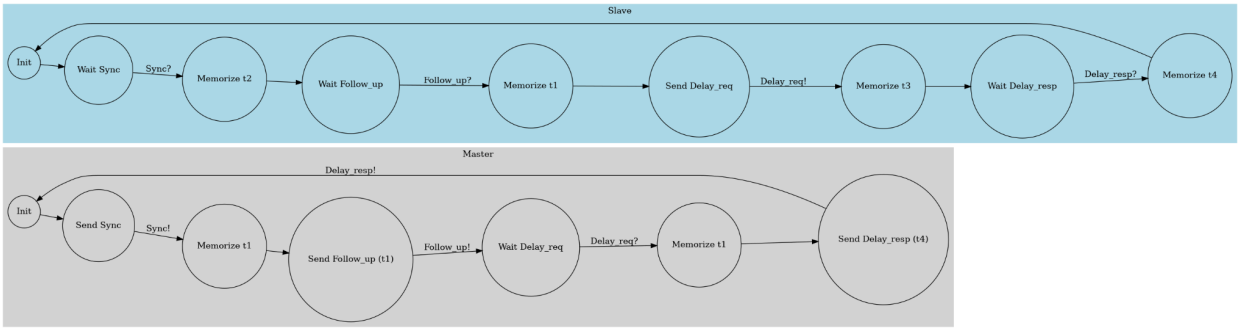
Calcul de l'offset sur le noeud du slave

```
MasterSlave_diff = t2 - t1
SlaveMaster_diff = t4 - t3
Offset = (MasterSlave_diff  - SlaveMaster_diff ) / 2
```

Automate

```
[Master] (Init) --> (Send Sync) --[Sync!]--> (Memorize t1) --> (Send
Follow_up (t1)) --[Follow_up!]--> (Wait Delay_req) --[Delay_req?]-->
(Memorize t1) --> (Send Delay_resp (t4)) --[Delay_resp!]--> (Init)
[Slave] (Init) --> (Wait Sync) --[Sync?]--> (Memorize t2) --> (Wait
Follow_up) --[Follow_up?]--> (Memorize t1) --> (Send Delay_req)
--[Delay_req!]--> (Memorize t3) --> (Wait Delay_resp) --[Delay_resp?]-->
(Memorize t4) --> Init
```

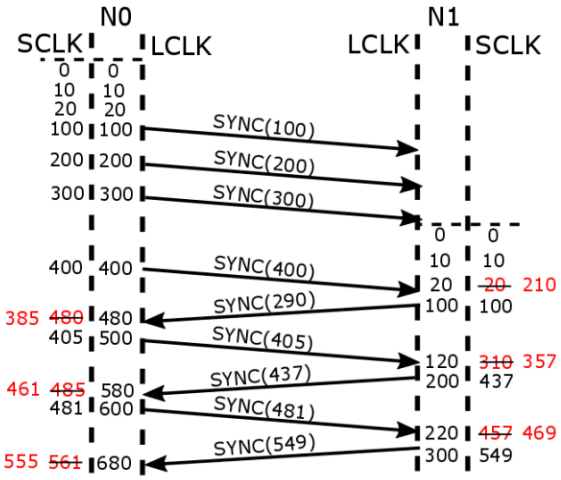
Notes : Une version « lisible » de ces automates se trouve dans le dossier contenant le code (un fichier `.dot` et un `.png`). En outre, la mise en forme du fichier dot a été peaufinée avec l'aide de Gemini 2.5.



3.2 Implémentation d'une synchronisation SISP

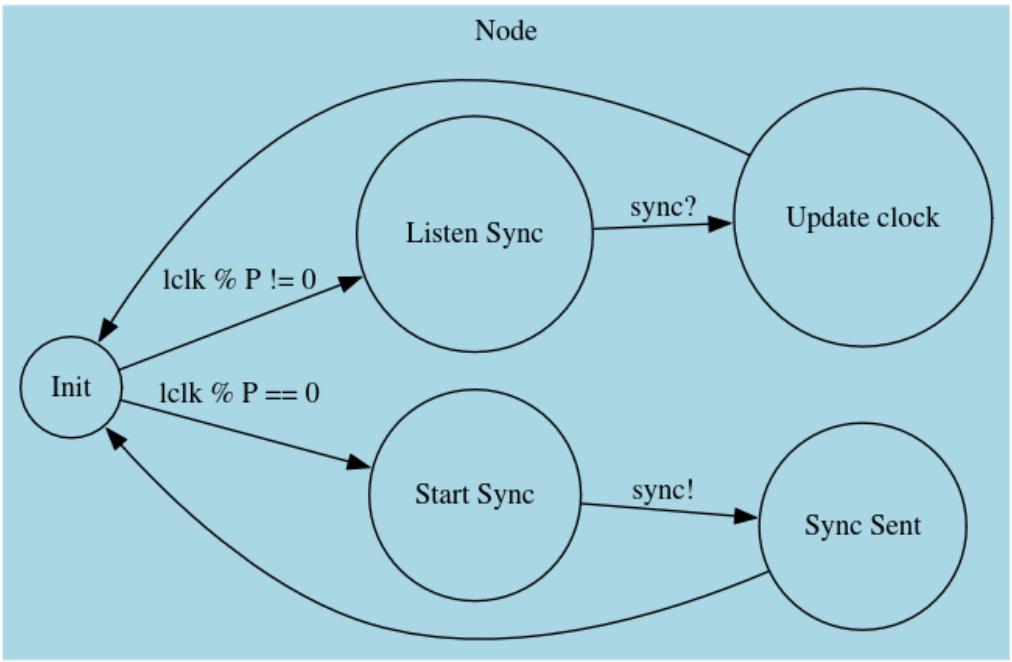
Primitives nécessaires
Il n'y pas de primitives autres que celles déjà décrites dans les parties précédentes.

Description du protocole



À intervalles réguliers, un nœud envoie un message SYNC contenant son horloge synchronisée (c.à.d. `sclk`). À tout autre moment, lorsqu'un nœud reçoit un message SYNC, il met à jour son horloge synchronisée avec la valeur de la moyenne entre cette dernière et la valeur reçue.

Automate



4. Ranging (synchronisation fine)

4.2 Implémentation de 2M-TWR

Voici les nouvelles primitives utilisées pour l'implémentation de ce protocole :

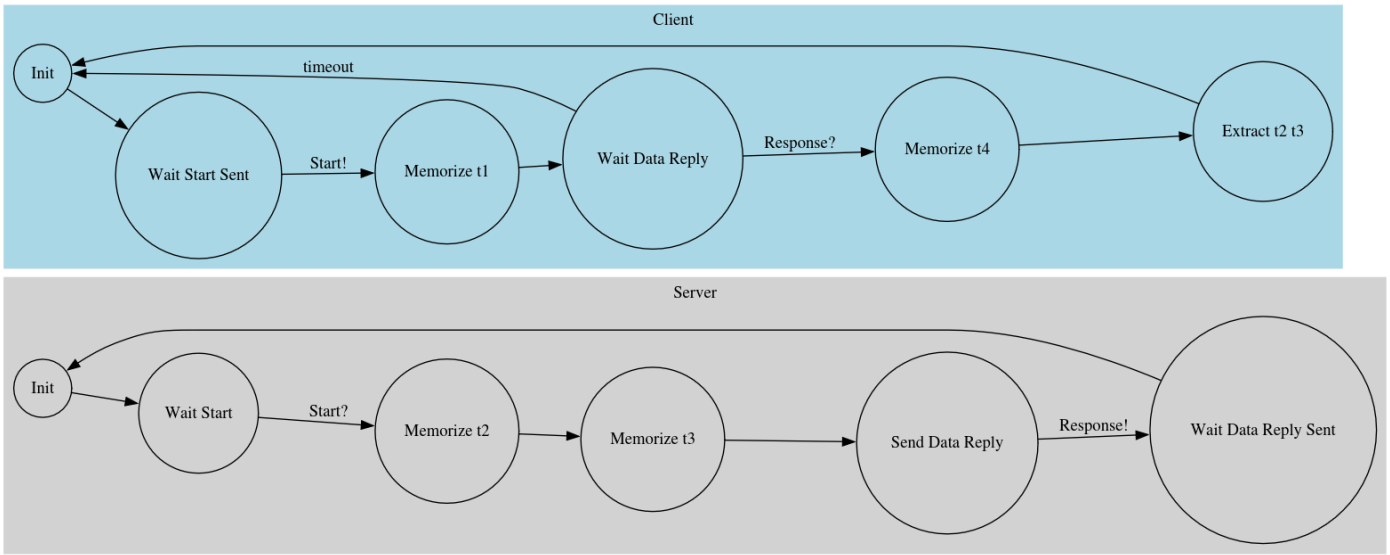
Primitive	Sémantique	Utilisé par :
`uint64_t = decaduino .alignDelayedTransmission(uint64_t wantedDelay);`	Retourne un timestamp utilisable dans "pdDataRequest" pour réaliser une transmission retardée.	Master
`decaduino.pdDataRequest(uint8_t * buf, uint16_t len, uint8_t delayed, uint64_t time);`	Réalise une transmission envoyée à l'instant "time" indiqué lorsque "delayed" vaut "true".	Master

Calcul de la distance (donné dans l'exemple)

```
// Sans prendre en compte la dérive
tof = (((t4 - t1) & mask) - ((t3 - t2) & mask))/2;
distance = tof*RANGING_UNIT;

// En compensant la dérive
tof = (((t4 - t1) & mask) - (1+1.0E-6*decaduino.getLastRxSkew())*((t3 - t2)
& mask))/2;
distance = tof*RANGING_UNIT;
```

Automates



4.3 Implémentation de N-TWR

Le code du serveur n'a pas évolué pour cette partie : à la réception d'un message du client, le serveur prédit `t3` et envoie sa réponse à l'instant `t3` dans un message contenant les valeurs de `t2` et de `t3`.
Côté client, le code a été modifié pour écouter plusieurs réponses.
Pour cela, on revient dans l'état `TWR_ENGINE_STATE_WAIT_DATA_REPLY` après chaque calcul de distance avec un serveur, tant que toutes les réponses n'ont pas été reçues et qu'il n'y a pas de *timeout*. Nous avons également créé des tableaux pour stocker les différentes valeurs de `t2`, `t3` et `t4`.

```
uint64_t t1, t2[NB_SERVER], t3[NB_SERVER], t4[NB_SERVER];
```

Automates