A) Récupérer Contiki-3.0

Deux solutions : soit récupérer l'image disque complete (instant-contiki) à faire fonctionner dans uen virtual box.

Soit récupérer les sources (attention aux dépendances!!)

Soyons fous : récupérons les sources !

```
 git clone https://github.com/contiki-os/contiki.git contiki-3.0
cd contiki-3.0/
git submodule update –init
cd examples/sky
```

B) Premières approches

Consultez le fichier /contiki/examples/hello-world/hello-world.c

Le fichier source contient des macros :
PROCESS
AUTOSTART_PROCESSES
PROCESS_THREAD
PROCESS_BEGIN
PROCESS_END

Que sont les PROTOTHREADS ?
Quel est le rôle de chacune des macros listées ?

C) Compilation :

Contiki est prévu pour plusieurs plateformes. Cependant, il est possible de tester le code natif pour vérifier le bon fonctionnement (si vous n'avez pas besoin du réseau). Lorsqu'on compile, on obtient un binaire dans le nom contient la plateforme visée.
Attention : pour travailler plus facilement avec les TelosB, utilisez les sources de contiki 3.0 directement sur la machine (voir sur ma page web pour le téléchargement)

Il vous faut aussi : le compilateur, et la libc correspondant à l'architecture msp430 (pour telosB), et Atmel (pour micaz).

```
aptitude install gcc-msp430  msp430-libc
aptitude install gcc-avr avr-libc
aptitude install ant
```

Expliquer l'ensemble de ces commandes : pour chacune d'elles, à quoi sert elle ? Qu'est ce qui est installé ?

D) Premier essai avec l'exemple hello-world

```
make
./hello-world.native
```

Si on veut du réseau, on peut imposer la cible minimal.net

```
make TARGET=minimal-net
./hello-world.minimal-net
```

Pour compiler ce programme pour un TELOSB, utilisez la cible sky

```
make TARGET=sky
```

Le code obtenu peut être utilisé sur des motes émulés dans cooja, ou transféré sur un vrai Telos en appelant :

```
make TARGET=sky hello-world.upload
```

Pour activer IPV6, n'oubliez pas de définir la constante : UIP_CONF_IPV6=1

Pour lancer sous l'émulateur simulateur Cooja

```
make TARGET=cooja hello-world-example.csc
```

Expliquez ce qui se passe ?
Que sont 1 et 2 ? cliquez dessus pour en apprendre plus.
Expliquez pourquoi il a fallu faire les aptitudes install.
A quoi sert Cooja ?
Pourquoi dit on que Cooja est un émulateur/simulateur ? Qu'est ce qui est émulé et simulé ?
Dans quelle mesure Cooja est il fiable ? (à votre avis, quel est son intérêt ?)

Pour plus d'informations sur l'utilisation de Contiki, consultez les fichiers README qui sont à la racine du projet (Contiki-3.0). Il est possible d'avoir des détails spécifiques ensuite dans le répertoire des « examples » que vous voulez tester.

E) Ecrire ses propres programmes

Contiki peut être programmé en C, et utilise les librairies standard du C (#include<stdio.h>). On peut écrire ses propres programmes en s'inspirant des exemples fournis (dans /examples/. Attention, les variables sont déclarées en static (Protothreads : **if in doubt, do not use local variables inside a protothread!** *documentation*)

1) accès aux leds :

```
#include "leds.h"
...
leds_on(LEDS_RED);
leds_off(LEDS_GREEN);
leds_toggle(LEDS_RED);
leds_off(LEDS_ALL);
```

Ecrire un programme qui allume les 3 leds

2) timers

un timer est piloté par une structure etimer.
La commande etimer_set(*etimer, durée) permet d'armer le timer
On attend le timer avec le macro PROCESS_WAIT_EVENT_UNTIL(etimer_expired(*etimer))

Ecrire un programme qui fait clignoter les trois leds, une à une, toutes les secondes.

3)Gestion des capteurs

Boutons : (attention lors de l'utilisation sur TelosB, il y a un reset et un bouton user)
#include "dev/button-sensor.h"
…
SENSORS_ACTIVATE(button_sensor);
PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event &&data == &button_sensor);

Ecrire un programme qui allume et éteint les leds selon l'appui sur le bouton

Ecrire un programme qui allume ou éteint les leds toutes les 4 secondes ou selon l'appui du bouton.

Capteurs de lumiere : Cherchez comment l'activer et accéder aux résultats.
Ecrire un programme qui affiche, toutes les secondes, la luminosité ambiante.
Sachant que le sensirion SHT11 est le circuit utilisé pour capter la température et l'humidité, ajoutez à votre programme ces deux mesures.
NOTA : Attention, ce circuit n'est disponible que sur TelosB (sky). Que se passe t'il si vous essayez de compiler un tel programme sans indiquer de TARGET ?

F) Accès Réseau

Sky websense : HTTP

Cet exemple montre un serveur web qui tourne au dessus d'IPv6 sur les TelosB, et qui donne accès aux valeurs mesurées par les capteurs. Un border router RPL (qui organise l'arbre RPL) connecte le réseau de capteurs à l'Internet.

Tester le sky websense sous Cooja

1. lancer COOJA et charger le fichier de configuration "example-sky-websense.csc"[1]

```
make TARGET=cooja example-sky-websense.csc
```

Explorez les différents objets accessibles (clic droit sur l'écran Network).

> Combien y 'a til de types d'objets différents ? Quels sont les codes installés sur ces nœuds ? A quoi servent ces codes ?

2. Se connecter au réseau simulé sur COOJA via tunslip6:

```
make connect-router-cooja
```

3. Vous pouvez maintenant accéder à n'importe quel nœud de la simulation avec un navigateur web :

Router: http://[aaaa::0212:7401:0001:0101]/

Node 2: http://[aaaa::0212:7402:0002:0202]/

Pour tester l'exemple avec des vrais TelosB

1. Compiler et télécharger le code sur chaque Telos (connecté via USB)

1 Attention : Contiki est écrit pour de nombreuses plateformes : cette version pourrait ne pas compiler pour un appareil aussi contraint que le TelosB. Dans ce cas, essayez de trouver comment économiser de précieux octets !! (nota : cherchez un peu ce que peut être nullrdc pour la MAC de CONTIKI :-))

```
make TARGET=sky sky-websense.upload
```

2. Compiler et télécharge le code sur le TelosB chargé de faire le border router

```
cd ../rpl-border-router
make TARGET=sky border-router.upload
```

3. Utiliser ce dernier en tant que routeur (en le laissant connecté à l'USB)

```
make connect-router
```

4. Rebooter le routeur et récupérer l'adresse

```
make TARGET=sky login
Appuyez sur  le bouton du Telos B.
A quoi sert cette commande make... login ? A quoi sert l'appui sur le bouton ?
(consultez le code source)
```

5. Accéder aux nœuds via votre navigateur web

G) Utilisation des démonstrations CoAP

Dans le simulateur Cooja

Dans er-rest-examples, lancez

```
make TARGET=cooja server-only.csc
```

Lancer ensuite, dans un autre terminal, le border router :

```
make connect-router-cooja
```

A quoi sert ce border-router ? Que permet il ?

Interactions avec l'objet :

Installez le plugin copper (Cu) pour firefox :
Ensuite, connectez vous avec votre navigateur sur **coap://[aaaa::0212:7402:0002:0202]/**
Lancer une découverte.

Pour chaque ressource (à gauche), vous allez pouvoir selectionner, puis lancer une des 4 commandes REST (en haut de l'écran). Le résultat, et/ou les messages d'erreurs, apparaitront dans les différentes zones de votre navigateur.

Dans firefox, allez dans core, et faites un GET
Testez un POST sur toggle. Et un GET, ou un DELETE.

Dans la partie test, vous allez pouvoir là encore tester les différentes commandes.

Sur quelle ressource le Observe est il implémenté ?
Ajouter dans la simulation 3 nouveaux nœuds.
Relancez la simulation, et vérifiez que vous accèdez bien à ses nouveau objets.

Sur de vrais TelosB

Pour compiler, même punition que précédemment. Pour compiler et installer le binaire sur l'objet via le port USB : (attention, il faut être root).

```
make  er-example-server.upload TARGET=sky
```

Pour récupérer l'IPv6 :

make TARGET=sky login (cela va vous permettre de scanner le port USB)
Reset du TelosB (via le bouton reset), et vous verrez l'adresse IP sur le scan du port USB

On crée ensuite le border router

```
cd ../ipv6/rpl-border-router/
make TARGET=sky border-router.upload
```

Le code est installé, on peut maintenant activer le routage entre votre PC et ce TelosB qui fera le relai avec le réseau de capteur

```
make connect-router
```

Grace à l'adresse IP récupérée plus haut, vous pouvez maintenant accéder à l'objet via Firefox (n'oubliez pas les crochets : *coap://[aaaa::0212:7400:146e:da98]*)

Allumez la diode. Observez le bouton, et utilisez le afin de récupérez l'événement dans Firefox

Si vous avez le temps, tester une bibiothéque afin d'écrire des programmes clients
Récupérez le code sur le site http://coapy.sourceforge.net/
Testez l'accès aux services CoAP

H) Programmation Réseau

Ecrire un programme qui permet sur un nœud d'allumer ou d'éteindre une led, et un autre qui permet d'envoyer le message de commande d'allumage à ce nœud lorsqu'on appuie sur le bouton.

Ecrire un programme « client » qui envoit des messages à un programme serveur, dont le contenu est numéroté (par exemple 1, 2, 3 etc...) Le programme serveur qui reçoit ce message allumera ou éteindra une led en fonction de la parité du contenu

Ecrire un programme qui renvoie le contenu du message en indiquant son propre compte. L'objectif est de vérifier la qualité de la communication et d'avoir une idée des pertes. Ainsi, le programme emetteur saura combien de ses messages ont été perdus, et combien de réponses. Votre programme devra donc afficher les différents totaux : messages émis, messages reçus par l'interlocuteur, et réponses de l'interlocuteur.

Faire une version proxy :
Un nœud central (le sink par exemple) est chargé de recevoir les messages de tous, et de les redispatcher selon le destinataire voulu. Vérifier si l'ajout d'un nœud « pivot » augmente les erreurs de transmission.