TP IoT LAB

Thibaut EHLINGER et Benjamin HERB
Université de Strasbourg

Mardi 13 décembre 2016

Dans ce rapport, nous allons vous présenter notre travail concernant le TP2 d'IoT. Pour rappel, il fallait dans un premier temps modifier certains paramètres (PanID, canal radio) et compiler une version de `Contiki OS` compatible avec les nœuds de la plate-forme FIT/IoT-lab. Ensuite il fallait flasher les six nœuds qui nous étaient attribués avec le binaire obtenu. Puis, nous avons dû : mettre en œuvre RPL ; analyser le déploiement du réseau avec le *sniffer* Foren6 et étudier le *Collect Tree Protocol (CTP)*. Nous allons vous présenter les différentes étapes de ce TP.

1 Contiki OS

Contiki OS est un OS open source spécialisé dans les microcontrôleurs à basse consommation d'énergie reliés à un réseau. Dans un premier temps, nous avons modifié le code de Contiki OS. Pour modifier le canal il fallait modifier `RF2_XX_CHANNEL` dans le fichier `platform/openlab/radiorf2xx.c`. Nous lui avons affecté la valeur **17**. Pour le PanID, nous avons fait un `define IEEE802154_CONF_PANID` et nous avons utilisé l'ID `0x0007`, dans le fichier `contiki/platform/openlab/contiki-openlab-conf.h`.

Pour compiler Contiki OS, il fallait utiliser la commande `make` avec la bonne cible :

```
$ make TARGET=iotlab-m3
```

Une fois le binaire généré, nous l'avons téléversé sur nos contrôleurs avec

```
$ node-cli -i 56476 -l strasbourg,m3,2+30-34 -up tutorial_m3.elf
```

2 RPL

Une fois nos contrôleurs dotés d'un OS, nous avons mis en œuvre RPL, un protocole de routage IPV6 prévu pour les WSN (*Wireless sensor network*). RPL construit un DODAG (*Destination Oriented Directed Acyclic Graph*) orienté vers le BR (*Border router*) de notre réseau.

Pour information, notre sous-réseau a pour adresse IPv6 `2001:660:4701:f0a6/64`. Notre *border-router* est le nœud numéro **`m3-2`**. Pour déployer notre réseau nous avons dû :

1. lancer `tunslip6` sur le serveur SSH ;

   ```
   $ sudo tunslip6.py 2001:660:4701:f0a6::1/64 -L -a m3-2 -p 20000
   ```

2. affecter au nœud `m3-2` son rôle de *border-router* ;

   ```
   $ node-cli --update ~/border-router.iotlab-m3 -l strasbourg,m3,2
   ```

3. récupérer l'adresse du *border-router*. Il s'agissait de `2001:660:4701:f0a6::a685`

4. déployer un serveur HTTP sur tous les autres nœuds du réseau :

```
$ node-cli --update ~/http-server.iotlab-m3 -l strasbourg,m3,30-34
```

Ci-dessous le texte lu sur l'interface web du BR :

```
Neighbors
fe80::a786
fe80::b286
fe80::a885
fe80::b287
fe80::b086
Routes
2001:660:4701:f0a6::a786/128 (via fe80::a786) 16711327s
2001:660:4701:f0a6::b286/128 (via fe80::b286) 16711398s
2001:660:4701:f0a6::a885/128 (via fe80::a885) 16711398s
2001:660:4701:f0a6::b287/128 (via fe80::b287) 16711397s
2001:660:4701:f0a6::b086/128 (via fe80::b086) 16711396s
```

Et le texte lu sur l'un des autres nœuds :

```
 Neighbors

fe80::a685 PREFERRED
fe80::b286
fe80::b287
fe80::b086
fe80::a885

Default Route

fe80::a685

Routes
```

3 Foren6

Dans cette section, nous allons déployer Foren6 sur l'un des nœuds de notre réseau dans le but de visualiser la construction du graphe RPL. Foren6 est un outil d'analyse non-intrusif de réseaux 6LowPan . Pour visualiser les données, nous avons téléchargé et compilé le code issu du dépôt `https://github.com/cetic/foren6` :

3.1 Mise en place de Foren6

Pour lancer l'interface graphique de Foren6 sur l'ordinateur nous avons utilisé les commandes :

```
$ sudo apt-get install libqt4-dev qt4-qmake cmake make libexpat1-dev tshark libpcap0.8-dev
    libc6-dev g++ gcc
$ git clone https://github.com/cetic/foren6.git
$ cd foren6
$ make run
```

Puis nous avons téléchargé le code de la partie Foren6 à exécuter sur le noeud. Avant de le compiler, il fallait modifier une ligne du code afin de configurer le bon canal radio :

```
// channel between 11 and 26
static uint8_t channel = 17;
```

Ensuite, nous avons compilé ce code spécifiquement pour les nœuds de la plateforme IoT-LAB :

```
$ mkdir build.m3 && cd build.m3 && cmake .. -DPLATFORM=iotlab-m3
```

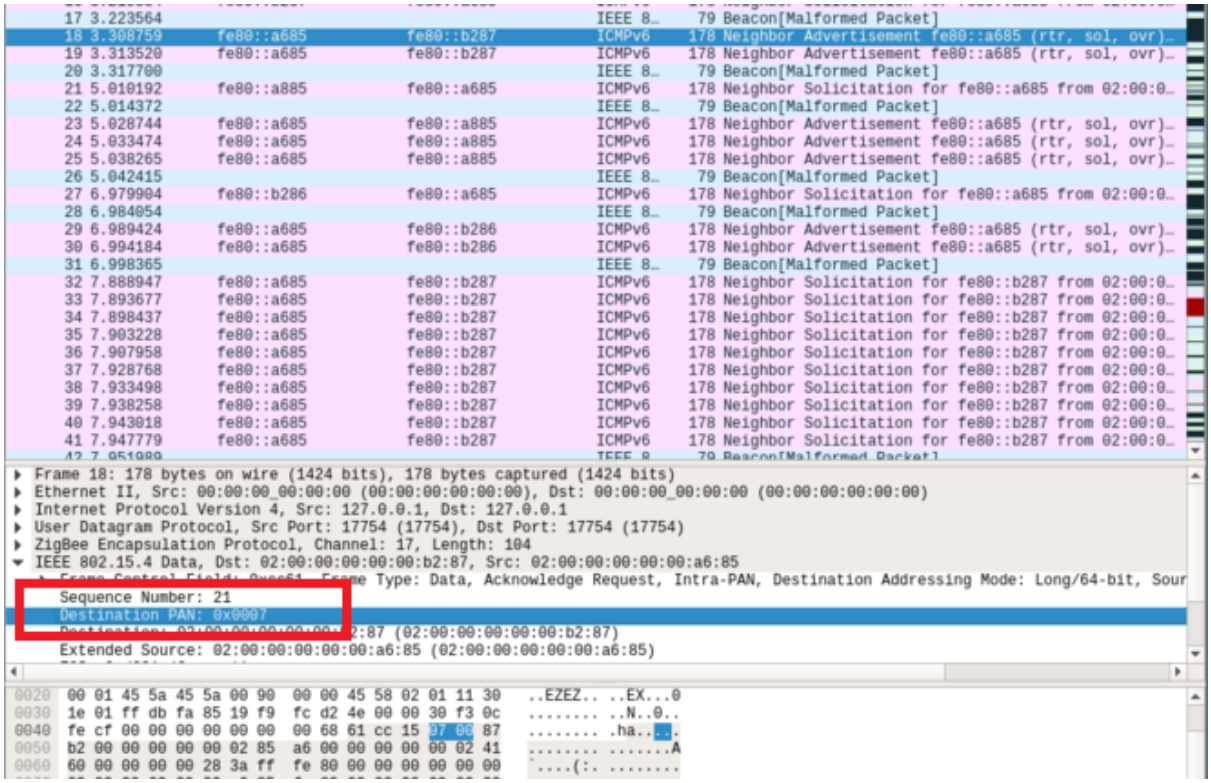

Figure 1 – *Pan ID (encadré rouge), trouvé grâce à Wireshark.*

Il a fallu ensuite relier le *sniffer* à l'interface graphique, puis relier le stream à un pseudo tty avec `socat` :

```
$ ssh -i /home/ben/.ssh/id_rsa 2016striot7@strasbourg.iot-lab.info -L 2001:m3-34:20000
$ socat TCP4:127.0.0.1:2001 pty,link=/tmp/mytty1,raw
```

Cela nous a permis d'obtenir une trace, dans `tty1`, pour visualiser notre graphe dans l'interface graphique Foren6.

3.2 Utilisation de Foren6

Nous avons d'abord *flashé* le sniffer sur un nœud qui n'est pas notre BR (ici, le noeud `m30`) :

```
$ node-cli -i 56200 -l strasbourg,m3,30 --update-profile sniffer
$ sniffer_aggregator -l strasbourg,m3,30 -o m3-30.pcap
```

Une fois la capture lancée, on *reset* les nœuds, mise à part le *sniffer*, afin d'être sûr que le *sniffer* soit actif lors de l'initialisation du graphe.

```
$ node-cli reset --exclude strasbourg,m30
```

Malheureusement nous n'avons pas réussi à visualiser la topologie de notre réseau avec Foren6. Par conséquent, pour trouver notre PanID nous sommes passés par Wireshark. Comme prévu, c'était le `0x000007`. La figure 1 montre l'analyse de la capture avec Wireshark qui nous a permis de trouver notre PanID. La figure 2 nous montre l'exécution du script dans notre réseau. On y voit notre que notre graphe est composé d'une racine et de 5 feuilles, il n'y a donc aucun nœud intermédiaire dans notre topologie.

4 Stack RIME

Afin d'avoir un puits et cinq noeuds, nous avons compilé deux firmwares différents. Pour cela nous avons modifié `example-collect.c`.

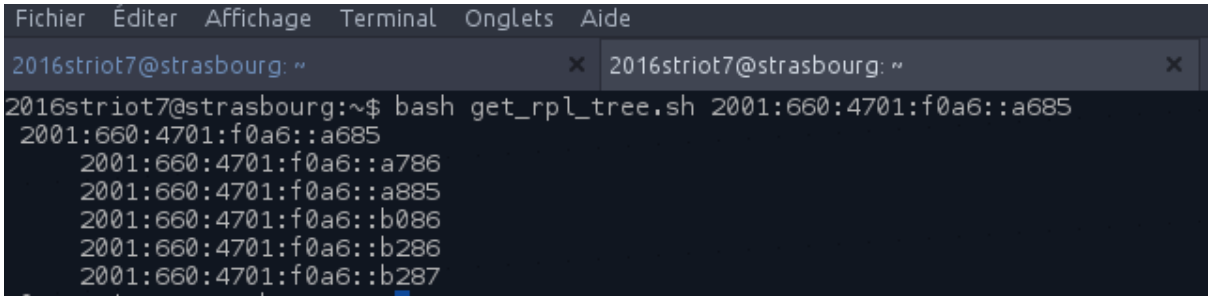

Figure 2 – *Exécution du script* `get_rpl_tree.sh` *dans notre réseau.*

```
#DEFINE SINK 1

[...]

if(SINK) {
  printf("I am a sink\n");
  collect_set_sink(&tc, 1);
}
```

Pour créer le firmware `example-collect-sink.iotlab-m3` nous avons donc mis la valeur `SINK` à 1 et pour le firmware `example-collect-node.iotlab-m3` la valeur à 0.

Puis pour mettre en place une fréquence d'émission à 5 secondes, nous avons modifié cette partie de `example-collect.c` :

```
if(etimer_expired(&periodic)) {
  etimer_set(&periodic, CLOCK_SECOND * 5);
  etimer_set(&et, random_rand() % (CLOCK_SECOND * 5));
}
```

Enfin nous avons pu compiler puis flasher le *firmware* sur les différents noeuds :

```
make TARGET=iotlab-m3

node-cli -i 56200 -l strasbourg,m3,30-34 -up example-collect-node.iotlab-m3
node-cli -i 56200 -l strasbourg,m3,2 -up example-collect-sink.iotlab-m3
```

5 Conclusion

Dans ce TP nous avons donc rapidement pris en main la plate-forme FIT/IoT-lab. Nous avons appris à compiler un OS et à déployer un protocole de routage adapter au WSN ainsi constitué. Enfin, Foren6 devait nous permettre d'observer les échanges entre agents du réseau, mais cela n'a pas marché comme prévu.

Cette expérience fût enrichissante, car nous avons découvert une plate-forme de grande ampleur dédiée à la recherche en IoT, et la documentation fournie pour la prendre en main est extrêmement complète.