Introduction à Contiki (et 6LoWPAN)

Pierre Ficheux (pierre.ficheux@smile.fr)

Juin 2018

Contiki

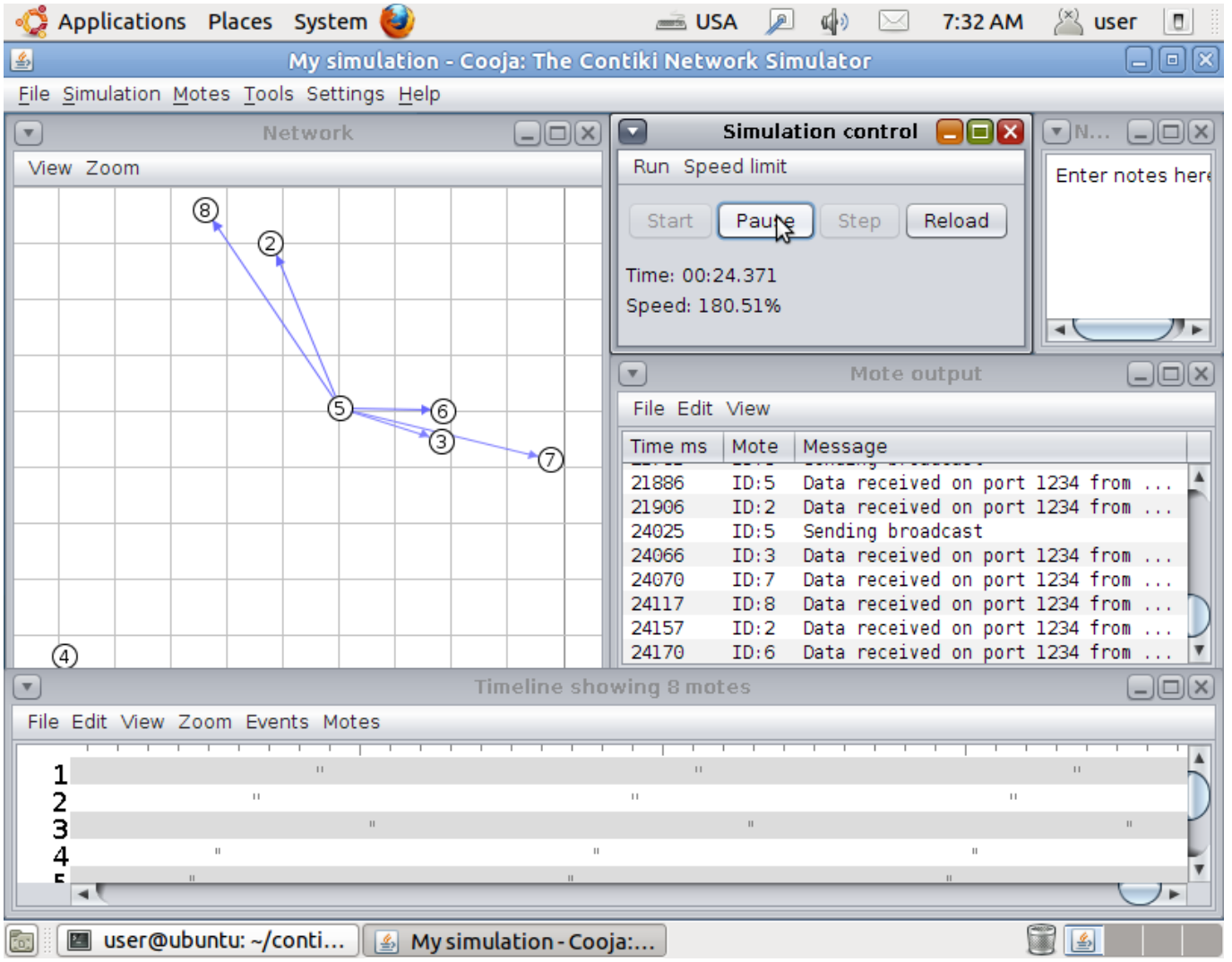
- OS libre pour l'Internet des objets
- Dédié aux micro-contrôleurs
- Diffusé sous licence BSD
- Dépôt sous Git
- Support IPv6, IPv4, 6LoWPAN, RPL, CoAP, etc.
- Support UART, leds, GPIO, et capteurs divers
- Programmation en C
- Compilation croisée ARM/x86 (`arm-none-eabi-gcc`)
- Test possible sur Linux/x86
- Assez peu de documentation si l'on compare avec Linux/Yocto → Doxygen
- Nombreux exemples fournis :-)

Contiki

- Apparu en 2003 (travaux IP sur CPU 8 bits – A. Dunkels)
- Configuration typique
  - 2 Ko de RAM
  - 40 Ko de ROM
- Réseau
  - La pile uIPv6 est certifiée IPv6 Ready Phase 1 par CISCO
  - Pile IP uIP publiée en 2011
  - Utilisée industriellement à travers le monde
- Émulateur (Cooja) + VM prête à l'emploi disponible
- Pas réellement un OS (à la Linux) mais un « exécutif » (un seul exécutable OS + application)
- Nombreuses macros et pas vraiment POSIX !

Simulateur Cooja

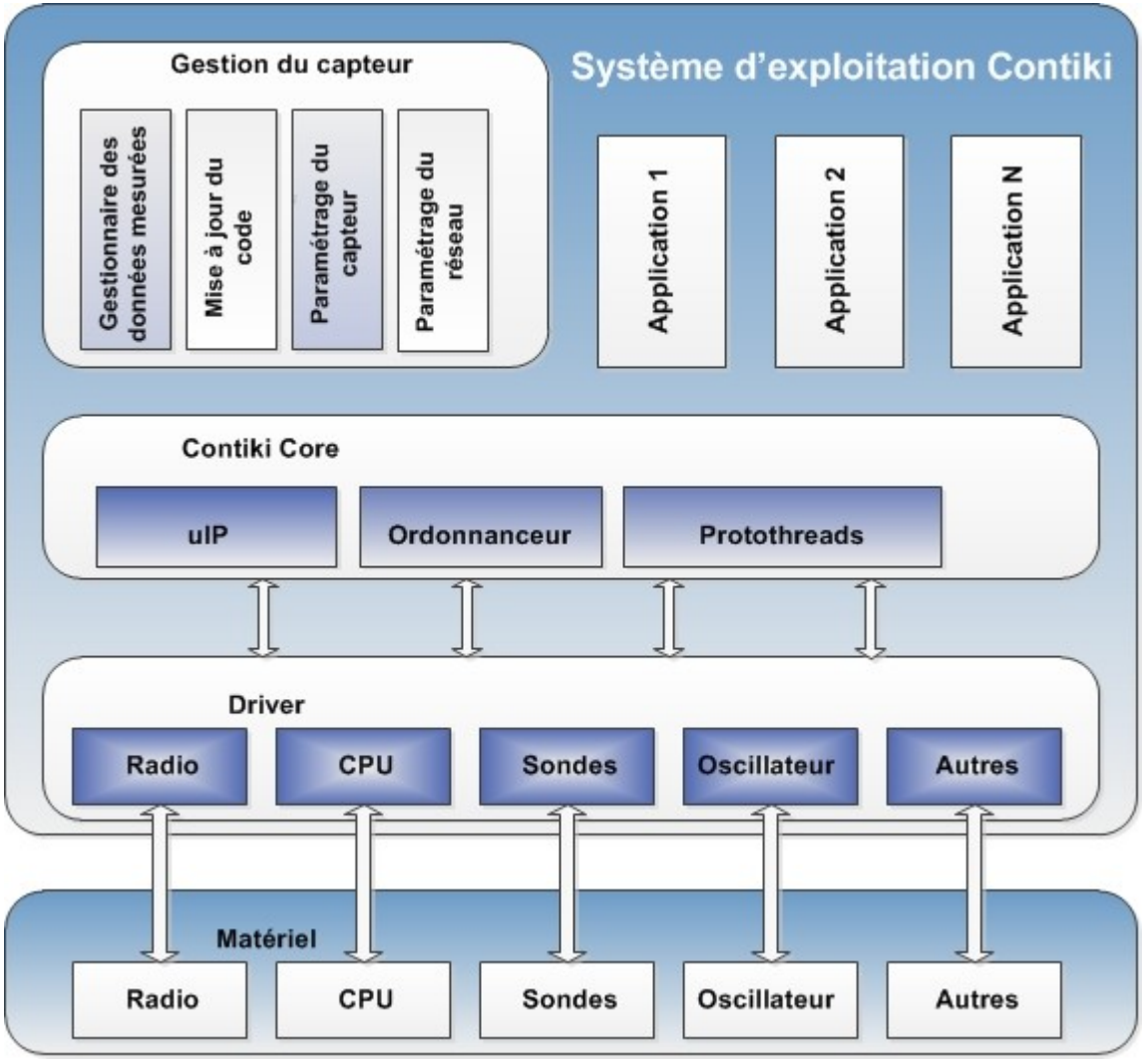
- Un nœud (capteur) connecté est appelé « mote »
- Simulation des nœuds et du réseau



Équipe de développement

- Contiki est développé par une équipe mixte industrie/université
- Le développeur principal est Adam Dunkels, Swedish Institute of Computer Science (SICS)
- L'équipe Contiki se compose de seize développeurs de différentes sociétés :
  - SICS
  - SAP AG
  - Cisco
  - Atmel
  - NewAE
  - TU Munich
- Activité – modeste - de la liste = 5 messages / jour

Architecture



Principaux répertoires

`app`	Applications Contiki (utilisables dans d'autres applications) → APPS += <application> dans Makefile
`cpu`	Fichiers spécifiques par architecture (arm, avr, 6502, cc26xx-cc13cc, etc.)
`dev`	Pilotes périphériques spéciaux (MMC, radio, capteurs divers)
`examples`	Exemples Contiki (+++)
`platform`	Fichiers spécifiques et pilotes (srf06-c26xx)
`core`	Principales fonctions Contiki (IPv6/IPv4, processus, threads, timers, etc.)
`tools`	Outils (dont émulateur Cooja)
`doc`	Documentation Doxygen → $ cd docs && make

« In a nutshell »

- Chargement des sources

```
$ git clone https://github.com/contiki-os/contiki.git
$ cd contiki
$ git submodule init
$ git submodule update
```

- Compilation + test « Hello World » pour x86 (native)

```
$ cd examples/hello-world
$ make
...
$ ./hello-world.native
Contiki-3.x-3330-g719f712 started with IPV6, RPL
Rime started with address 1.2.3.4.5.6.7.8
MAC nullmac RDC nullrdc NETWORK sicslowpan
Tentative link-local IPv6 address fe80:0000:0000:0000:0302:0304:0506:0708
Hello, world
```

Show me the code !

```
#include "contiki.h"
#include <stdio.h> /* For printf() */

PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);

PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  printf("Hello, world\n");

  PROCESS_END();
}
```

Plate-forme réelle (SensorTag)

- Cortex M3 CC2650 (48MHz, 128KB flash, 8KB RAM)
- Flash externe 512KB pour OTA et/ou stockage
- Faible consommation (10 mA active, 100 uA en veille)
- Fonctionne sur pile bouton
- Nombreux capteurs
- Radio 802.15.4 + Bluetooth Low Energy (BLE)
- Prix = 30 € (TI, Mouser)
- Développement avec sonde DevPack (15 €)

Test « Hello World »

- Installation du compilateur croisé (Ubuntu)

```
$ sudo apt-get install gcc-arm-none-eabi
```

- Compilation

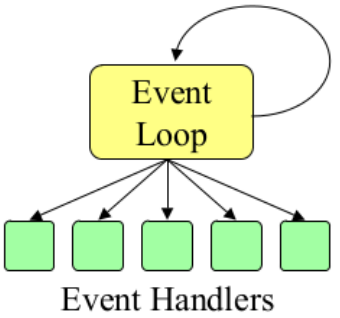
```
$ make TARGET=srf06-cc26xx BOARD=sensortag/cc2650 CPU_FAMILY=cc26xx
```

- Chargement sur le SensorTag
  - Ouverture `minicom` sur `/dev/ttyACM0` (console 115200, 8N1 + CR-LF = Ctrl-A-Z u)
  - Programmation avec Uniflash

```
Starting Contiki-3.x-3330-g719f712
With DriverLib v0.47020
TI CC2650 SensorTag
IEEE 802.15.4: Yes, Sub-GHz: No, BLE: Yes, Prop: Yes
 Net: sicslowpan
 MAC: CSMA
 RDC: ContikiMAC, Channel Check Interval: 16 ticks
 RF: Channel 25
 Node ID: 19078
Hello, world
```

Threads vs events

- De nombreux OS sont multithread
  - Programmation complexe, synchronisation nécessaire
  - Chaque thread doit avoir sa pile + son contexte
  - Indispensable pour TR et préemption
- Contiki utilise un modèle *événementiel*
  - Blocage de l'exécution en attente d'un événement
  - Event-handler exécuté par le noyau, jusqu'au retour
  - Pas de préemption
  - Durée d'exécution faible (lecture capteur HW)
  - `XNextEvent()` sur X11 (GUI) !
  - Utilisation des « protothreads »



Protothreads

- Modèle simplifié de thread
- Les protothreads partagent le contexte d'exécution
- Basé sur des macros
- Pas de pile → pas de variables locales !
- Portable (écrit en C)
- Attentes bloquantes sans changement de contexte
- Sous licence BSD
- Principales fonctions

  `PT_THREAD()`, `PT_BEGIN()`, `PT_END()`, `PT_WAIT_UNTIL ()`, etc.
- Création du thread d'un processus (`core/sys/process.h`)

```
#define PROCESS_THREAD(name, ev, data)                          \
static PT_THREAD(process_thread_##name(struct pt *process_pt,   \
                                       process_event_t ev,      \
                                       process_data_t data))
```

Processus Contiki

- Un bloc de contrôle

  `PROCESS(name, str_name)`
- Un protothread

  `PROCESS_THREAD(name, ev, data)`
- Démarrage automatique des processus

  AUTOSTART_PROCESSES(&name1, &name2, etc)
- Contrôle du processus

  `PROCESS_BEGIN()`

  `PROCESS_END()`

  `PROCESS_EXIT()`

  `PROCESS_WAIT_EVENT()` / `PROCESS_YIELD()` → attente événement

  `PROCESS_WAIT_EVENT_UNTIL()` → avec condition

  `...`

Timers

- timer + stimer
  - l'application vérifie si le temps a expiré
  - timer utilise l’horloge système
  - stimer utilise la seconde comme base de temps
- etimer (le plus utilisé ?)
  - attendre une période de temps
  - le reste du système peut continuer de fonctionner ou entrer en économie d'énergie
- ctimer
  - utilise une fonction « callback »
- rtimer
  - tâches « temps réel », préemption des processus
  - pour le code critique (radio)

Exemple etimer

```
#define SECONDS 2
int cnt;
static struct etimer et;

while(1) {
    // displays every 2 seconds
    etimer_set(&et, CLOCK_SECOND * SECONDS);

    PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);

    printf("Hello world %d!\n", cnt++);
}
```

leds

- Identifiées par `LEDS_RED`, `LEDS_BLUE`, etc. (+ `LEDS_ALL`)
- Configuration des leds

```
leds_toggle(leds)
leds_on(leds)
leds_off(leds)
```

```
leds_blink(void)
```

- Lecture (retourne un masque)

```
leds_get(void)
```

Capteurs

- Boutons, température, accélération, luminosité, etc. (dépend de la plate-forme)
- Bouton (événement)

```
SENSORS_ACTIVATE(button_sensor);
PROCESS_WAIT_EVENT_UNTIL(ev == sensors_event && data
== &button_sensor);
```

- Température (lecture)

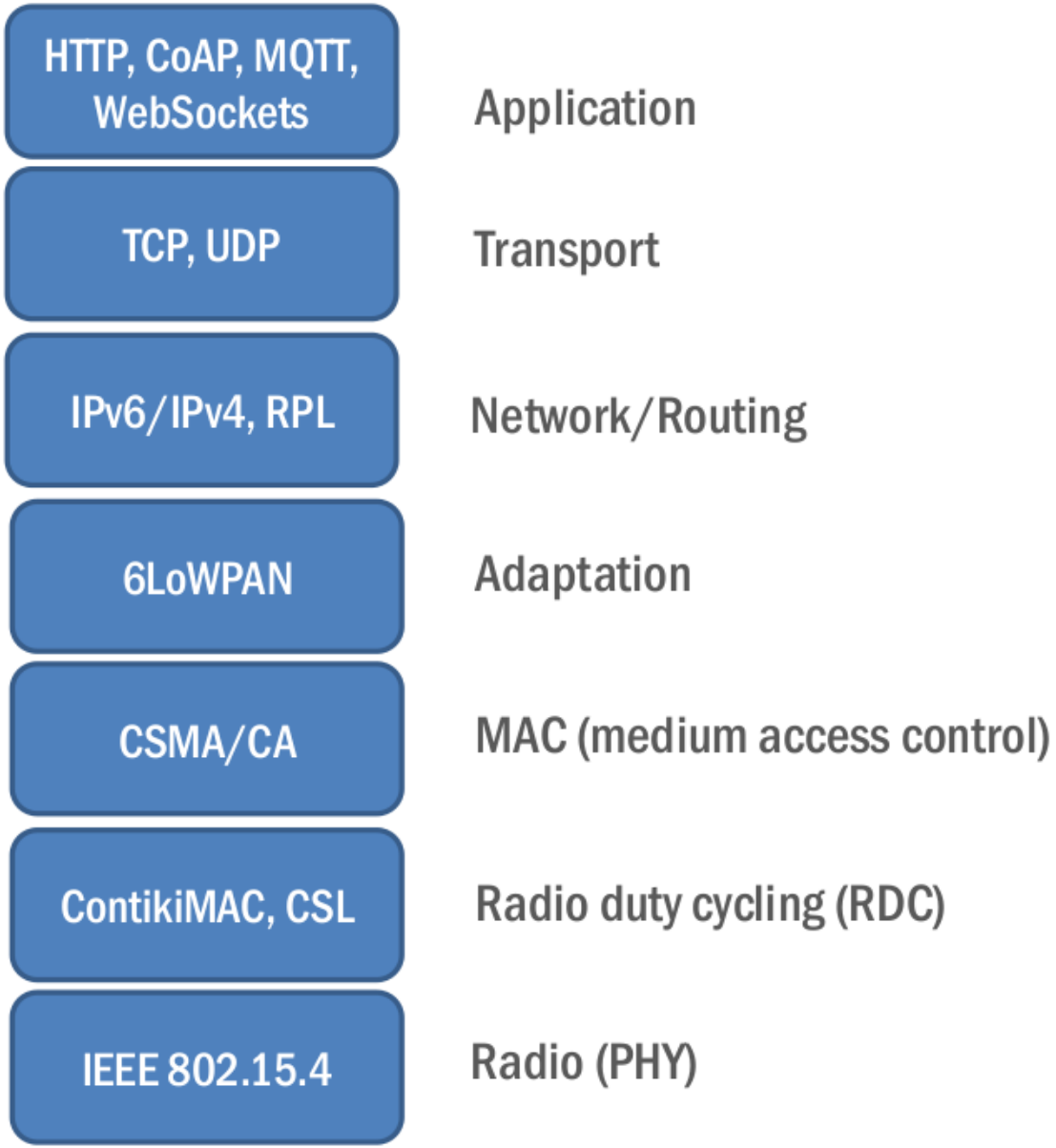
```
SENSORS_ACTIVATE(tmp_007_sensor);
temp = tmp_007_sensor.value(TMP_007_SENSOR_TYPE_ALL);
temp = tmp_007_sensor.value(TMP_007_SENSOR_TYPE_AMBIENT);
temp = tmp_007_sensor.value(TMP_007_SENSOR_TYPE_OBJECT);
```

- Désactivation

```
SENSORS_DEACTIVATE(tmp_007_sensor);
```

- Exemple SensorTag complet sur `examples/cc26xx`

Pile réseau Contiki

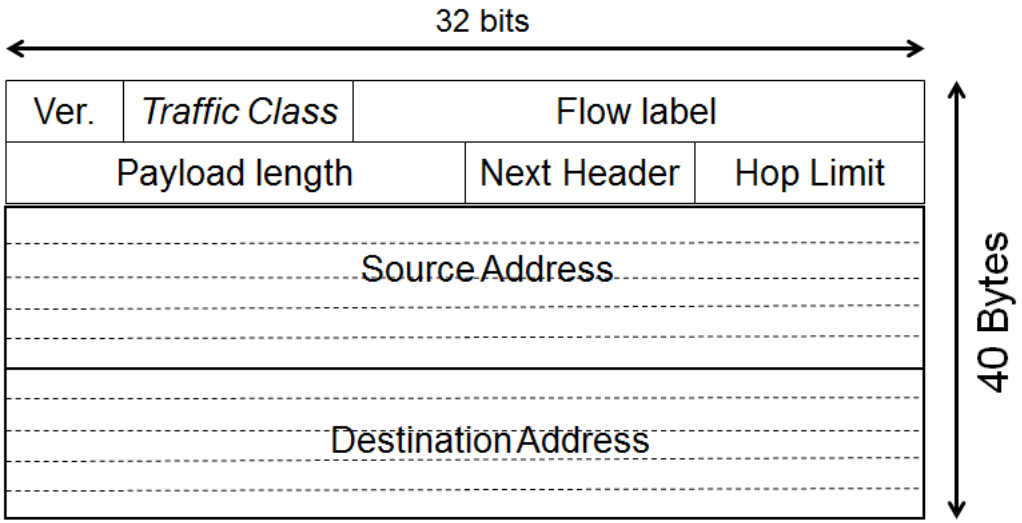


Histoire IPv6

- IETF, RFC 2460 (décembre 1998)
- Dans les années 90 :
  - Augmentation exponentielle de l'Internet
  - Augmentation du nombre d'entrées dans les tables de routage
- Allocation des adresses - janvier 1996
  - Classe A - 100.00%
  - Classe B - 61.95%
  - Classe C - 36.44%
- Prévisions d’épuisement des espaces d'adressage
  - Première alerte en 1994
  - Depuis le 1/2/2011 tous les blocs ont été attribués
  - Utilisation du NAT pour compenser !

Trame IPv6

- Les adresses sont sur 128 bits au lieu de 32 bits
- 2^{128} adresses au lieu de 4,294,967,296
- Étoiles observables dans le ciel → 2^{52}
- Approximativement 506 102 adresses par m^2 sur terre
- Environ 5×10^{28} adresses pour chaque habitant de la planète
- L'entête passe de 20 octets à 40 octets



Format des adresses

- Hexadécimal, octets sont groupés par 2 et séparés par « : »

  `FE80:0000:0000:0000:0224:D4FF:FEAD:0F54`
- Une (unique) suite de blocs composés uniquement de zéros peut être remplacée par « :: »

  `FE80::0224:D4FF:FEAD:0F54`
- Les zéros en début de blocs peuvent être omis

  `FE80::224:D4FF:FEAD:F54`
- Si nécessaire, on peut préciser l'interface

  `FE80::224:D4FF:FEAD:F54%eth0`
- Les adresses s'écrivent entre crochet si il y a ambiguïté

  `$ ssh user@[FE80::224:D4FF:FEAD:F54]`

Adresses spéciales

- Préfixe `2001:DB8:1::/48`
  - version compressée de 2001:0DB8:0001:0000:0000:0000:0000:0000
  - Les premiers 48 bits seront toujours 2001:0DB8:0001
- Lien local (toujours présent) → `FE80::/10`
- ULA (Unique Local Address) → `FD00::/8`
  - sur un même site, pas de routage (192.168.x.y en IPv4)
- Multicast → `FF00 ::/8`
- Adresse « loopback » `0:0:0:0:0:0:0:1` (`::1/128`)

Protocole NDP

- Auto-configuration = génération d'une adresse IPv6
- Récupération adresse d’un nœud via NDP (Neighbor Discovery Protocol, RFC 4861 2007)
- Découverte des voisins
  - résolution IPv6 → MAC (comme ARP avec IPv4)
- 4 type de messages
  - Router Solicitations (RS)
  - Router Advertisement (RA)
  - Neighbor Solicitation (NS)
  - Neighbor Advertisement (NA)
- Configuration des routes par défaut
- Propagation des paramètres de configuration

  → DHCPv6 également disponible !

Exemple NDP

- Soit donné l'adresse MAC = `00:17:F2:EA:59:46`
- Création d'une adresse lien-local
  - `FE80::217:F2FF:FEEA:5946`
- Vérification d'unicité de l'adresse lien-local
  - message NS sans réponse
- Récupération du préfixe IPv6 du lien
  - RS/RA (ex: `2001:db8:42::/64`)
- Création de l'adresse globale
  - `2001:DB8:42::217:F2FF:FEEA:5946`
- Vérification d'unicité de l'adresse globale

6LoWPAN

- LoWPAN = IPv6 LoW Power wireless Area Networks
- Nom d'un groupe de travail de l'IETF
- Basé sur la compression d'en-tête de paquets IPv6 + fragmentation des trames
- Permet a ces paquets de transiter sur des réseaux de type 802.15.4
  - 802.15.4 → payload de 128 octets
  - IPv6 impose une MTU (Maximum Transmission Unit) minimal de 1280 octets (68 pour IPv4)

Contraintes 6LoWPAN

- 6LowPAN est contraint en ressources (faible CPU, mémoire et batterie)
- Le débit est limité (jusqu'à 250 kbits/s)
- La MTU d'un réseau 802.15.4 est de 128 octets
- L'entête de la couche MAC est de 25Bytes

  → Il ne reste que 102 octets pour les couches supérieures

  → La couche de sécurisation des données AES ajoute 21 octets supplémentaires

Trame 6LoWPAN

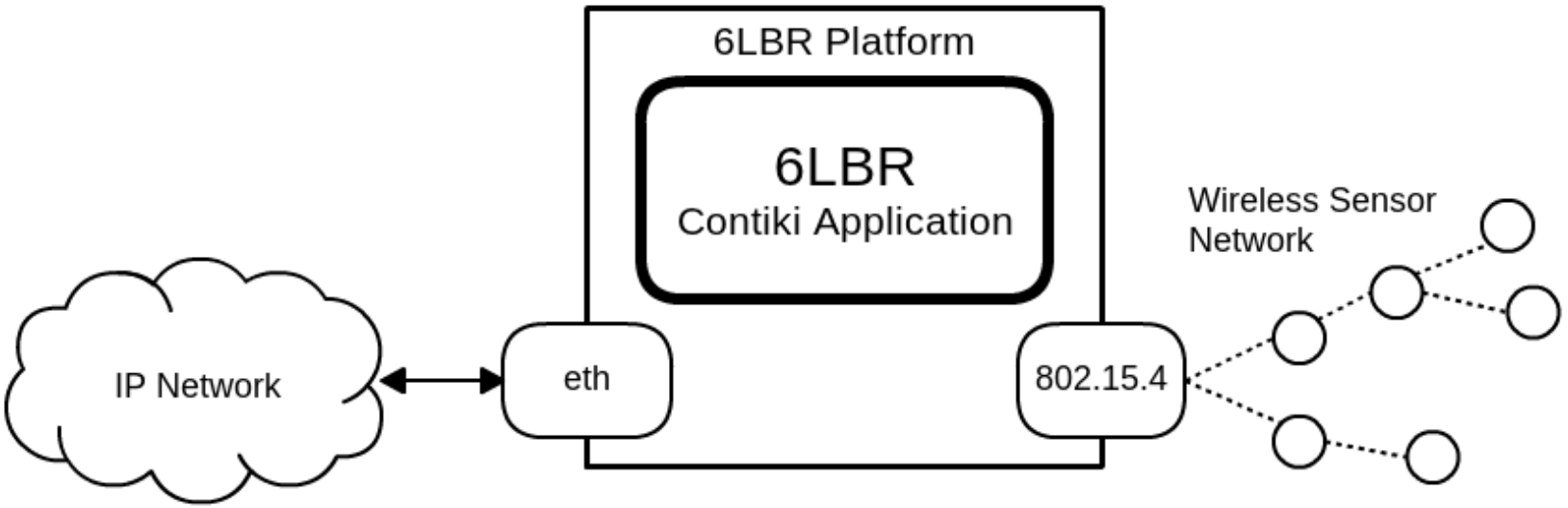
- Seulement 33 octets de payload en UDP et 21 en TCP.
- Fragmentation en plusieurs trames 802.15.4 + ré-assemblage → consommateur de ressources
- Plus de 30 % est pris par l'entête IPv6 (40 octets) → compression de l'entête
- Complexité du routage car nombre de nœuds est conséquent
- L'auto-configuration IPv6 (RFC 4862) est préconisée

Norme 802.15.4

- Protocole de communication standard IEEE pour les réseau sans fil
- Dédié au LR-WPAN → Low-Rate Wireless Personal Area Network)
- Faible consommation
- Faible débit
- Faible portée
- Utilisé dans différentes implémentations :
  - 6LoWPAN (ouvert, RFC 4919/4944)
  - ZigBee (propriétaire)

Configuration réelle

- Réseau local en 6LoWPAN
- Accès à Internet par un « border router » (LBR = Low power and lossy network Border Router)
- Nombreux services toujours en IPv4 (10 % de l'Internet en IPv6 en 2016)
- Exemple de 6LBR (sur Raspberry Pi + Yocto)

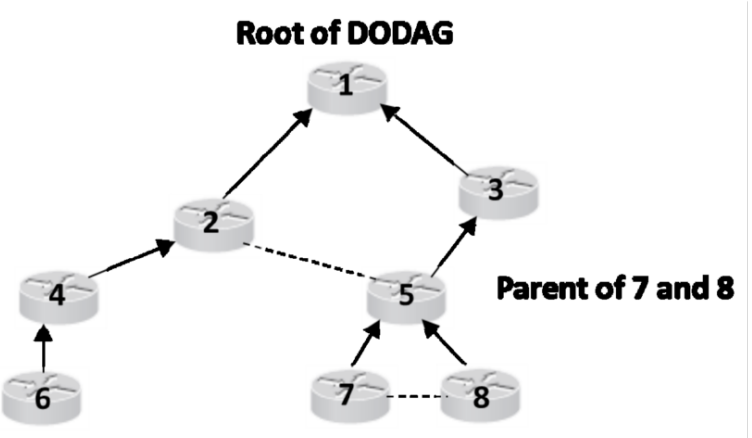


Routage (RPL)

- Contiki utilise le protocole RPL (Routing Protocol for Low power and lossy networks, RFC 6650)
- Basé sur un DAG (Directed Acyclic Graph)
- Le LBR de rang 1 est à la source du DAG construit par le protocole
- le LBR et tous les équipements de rang supérieurs forment un DODAG (Destination Oriented DAG)
- le LBR envoie un message d'information DIO (DODAG Information Object) en multicast
- Lorsqu'un équipement reçoit une nouvelle version de DIO il calcule son rang et propage son DIO
- Pour un nœud donné, tous les nœuds possédant un rang inférieur peuvent être parents

Message RPL

- RPL définit trois messages ICMPv6 :
- DIO → permet a un nœud de découvrir l'instance RPL + déduire sa configuration et sélectionner ses parents dans le DODAG
- DAG Information Sollicitation (DIS) → sollicite un DIO depuis un nœud RPL
- Destination Advertisement Object (DAO) → propage les informations de destination dans le DODAG



Test en réseau local

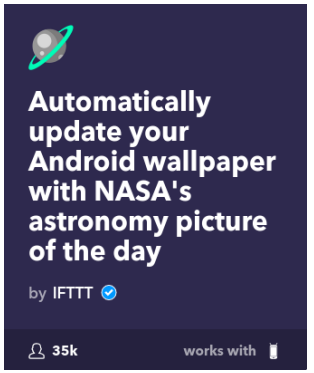
- Broadcast UDP

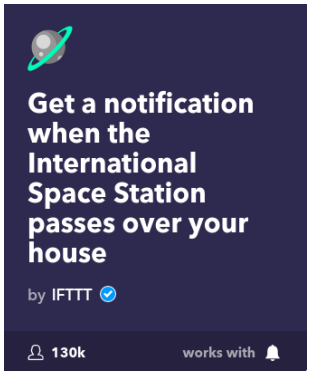
  `examples/ipv6/simple-udp-rpl/broadcast-example.c`
- Réception

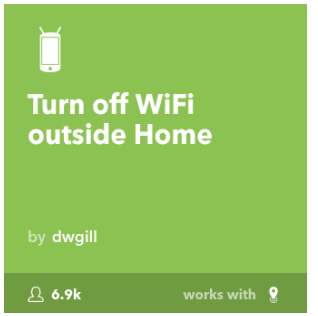
  `examples/ipv6/simple-udp-rpl/unicast-receiver.c`

Accès IFTTT

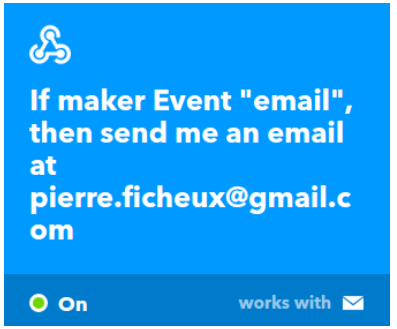
- IFTTT = If This then That (2011)
- Déclenchement d'action sur un « trigger » (Applet)







- Accès à une URL avec une clé (webhooks)



- Activité visible sur https://ifttt.com/activity

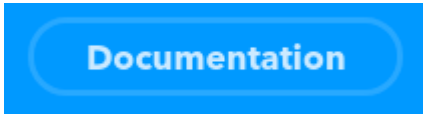
Test IFTTT (Linux)

- Déclaration d'une applet « webhooks »
  - *SI* accès URL *ALORS* envoi email
- Test requête GET

```
$ curl https://maker.ifttt.com/trigger/email/with/key/<key>
```

- Test requête POST (passage de paramètre JSON)

```
$ curl -X POST -H "Content-Type: application/json" -d '{"value1":"42"}'
https://maker.ifttt.com/trigger/email/with/key/<key>
```

→ voir https://ifttt.com/maker_webhooks puis 

- Mise au point avec http://httpbin.org

```
$ curl http://httpbin.org/ip
{
  "origin": "92.169.42.87"
}
```

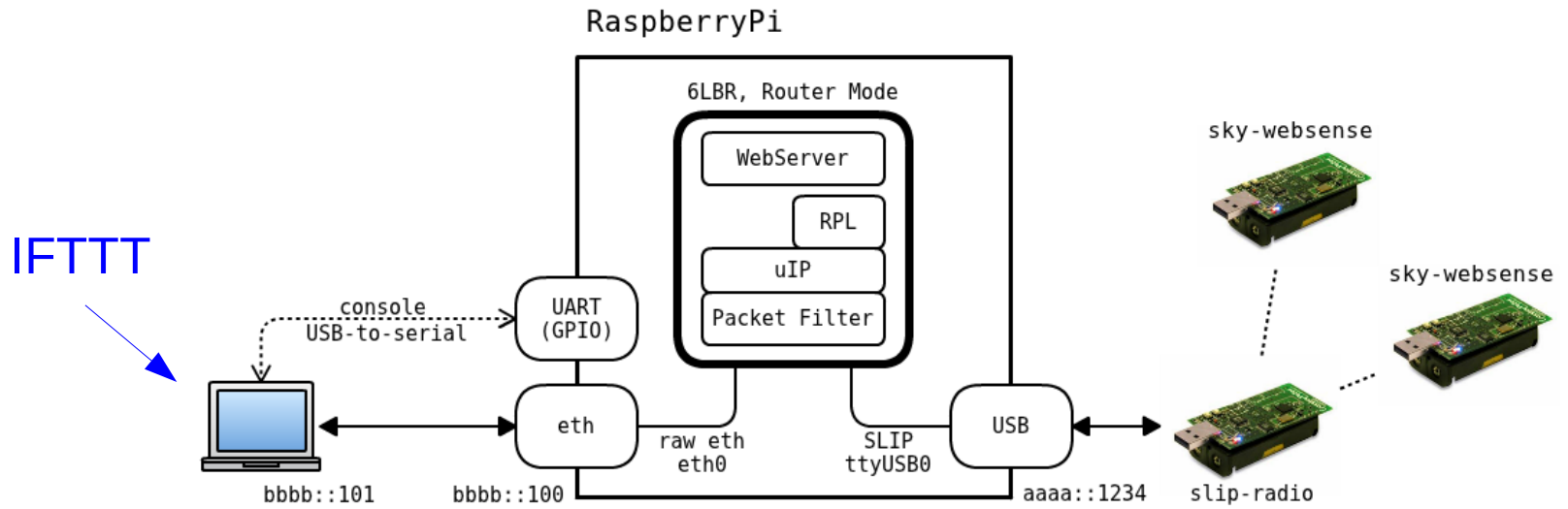
Border router 6LBR

- Projet initié par la société belge Cetic
- 6LBR est un « fork » de Contiki
- Solution de border router 6LoWPAN / RPL
- Fonctionne en tant qu'application sur une cible ou processus sur un hôte Linux
- Test Raspberry Pi + adaptateur 802.15.4 externe (protocole SLIP → `examples/ipv6/slip-radio`)
- Dernière version stable = 1.4.x (12/2016)
- Branche de développement = 1.5.x (01/2017)
- Configuration par serveur web local
- Layer Yocto disponible (thanks to Smile ECS) !

Accès IFTTT par 6LBR

- IFTTT fonctionne en IPv4
- Utilisation IP64 (IPv6 ↔ IPv4)

  The IP64 module lets an IPv6 Contiki network be connected to an IPv4 network without any additional configuration or outside software. It runs on the RPL root node and translates outgoing IPv6 packets into IPv4 packets and incoming IPv4 packets to IPv6 packets. (`core/net/ip64`)
- À activer dans la configuration 6LBR
- Adresse IPv6 dans `/var/log/6lbr.ip`



Références

- http://contiki-os.org/
- Test Cooja/simulation sur http://www.contiki-os.org/start.html
- http://anrg.usc.edu/contiki/index.php/Main_Page
- Doc Doxygen 2.6 http://contiki.sourceforge.net/docs/2.6
- https://github.com/marcozennaro/IPv6-WSN-book (+++)
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3574732
- SensorTag sur http://www.mouser.fr/search/ProductDetail.aspx?r=595-CC2650STK
- http://processors.wiki.ti.com/index.php/SensorTag_User_Guide
- DevPack sur http://www.mouser.fr/search/ProductDetail.aspx?r=595-CC-DEVPACK-DEBUG
- https://github.com/alignan/contiki/tree/iot-workshop/examples/zolertia/tutorial/01-basics
- Uniflash pour SensorTag sur http://processors.wiki.ti.com/index.php/Category:CCS_UniFlash
- https://fr.slideshare.net/salahecom/contiki-protothread
- Protothreads http://dunkels.com/adam/pt
- https://fr.slideshare.net/salahecom/contiki-protothread
- Doc API protothreads http://contiki.sourceforge.net/docs/2.6/a01802.html
- How protothreads really work http://dunkels.com/adam/pt/expansion.html ***
- http://anrg.usc.edu/contiki/index.php/Processes
- https://es.informatik.uni-freiburg.de/application/files/9414/4610/0089/2015_04_20_Contiki01.pdf
- https://github.com/cetic/6lbr/wiki
- http://cosy.univ-reims.fr/~lsteffenel/cours/DU-ASRE/UE5-IRI/Cours6/UE5-cours6-IPv6.pdf (+++)

Références

- https://fr.wikipedia.org/wiki/6LoWPAN
- https://github.com/cetic/6lbr/wiki/Example-1:-Hello-6LBR
- https://github.com/Openwide-Ingenierie/meta-6lbr
- http://www.linuxembedded.fr/2016/11/environnement-open-source-pour-les-objets-connectes/
- https://github.com/Openwide-Ingenierie/meta-6lbr
- Démo IFTTT + Contiki https://youtu.be/GXEg5gOEBEY