Projet Tutoré

Réseaux de capteurs sans-fil : Configuration et test de connectivité IP

Master Technologie de l'Internet 1ère année
LABORDE Cédric - DELGADO Frédéric

encadrant : M. Pham

Mai 2010

Remerciements

Nous remercions tout particulièrement notre encadrant M. Pham d'avoir proposé ce sujet et de nous avoir prêté le matériel, et M. Cariou de nous l'avoir attribué.

Nous remercions également Mme Dufaur-Dessus, secrétaire du département informatique.

Table des matières

Table des figures

1 Introduction

L'objectif de ce TER est de configurer et tester la connectivité IP entre des capteurs sans-fil, puis de réaliser une petite application permettant de récupérer les données de ces capteurs en IP.

L'avantage de l'IP dans ce domaine, à l'echelle de notre projet, serait par exemple de pouvoir récupérer les informations émises par le ou les capteurs choisis (connaissant leur adresse IP) et d'accèder à ce réseau depuis internet.

Ce rapport sera divisé en deux parties principales. Nous présenterons dans la première une description du matériel ainsi que de l'environnement logiciel utilisé et de leurs avantages quant à la programmation des capteurs. La deuxième partie sera une analyse détaillée de la solution retenue afin de répondre à la problématique posée.

2 Présentation

Un capteur est un petit appareil autonome capable d'effectuer des mesures simples sur son environnement immédiat. Ces mesures peuvent tant concerner la température, la pression que la lumière ou le son. Son rôle est également de transformer l'état de ces grandeurs physiques observées afin qu'elles soient plus aisément manipulables, par exemple en un signal numérique utilisable par un ordinateur. L'interface peut alors être réalisée à l'aide d'une liaison filaire, ou alors, et comme cela se développe depuis plusieurs années, par ondes radio (par exemple en utilisant le protocole ZigBee dans notre cas) ce qui ouvre de nouvelles perspectives dans la création de réseaux de capteurs autonomes et permet une plus grande flexibilité. Cette absence de connexion physique implique alors des contraintes d'économie d'énergie du à leur autonomie énergétique, en faisant alors un véritable objectif. L'architecture physique se retrouve alors impactée par l'adoption d'une capacité mémoire limitée ainsi que par le choix du protocole de communication. Ces contraintes se retrouvent également prises en compte dans le choix du language utilisé ainsi que dans les techniques mise en œuvre comme nous le verrons plus tard.

L'utilisation de ces capteurs est depuis longtemps une réalité au sein de domaine tel que l'industrie automobile ou aéronautique, mais l'affranchissement de la connexion filaire de part les progrès dans les technologies du sans-fil permet d'étendre leur utilisation à une multitude d'autres applications. Ils répondent à l'émergence, ces dernières décennies, d'un besoin accru d'observer et de contrôler des phénomènes physiques et biologiques dans différents

domaines tel que :

- l'industrie
- la santé
- la sécurité
- etc...

Pour le magazine *Technology Review* du MIT [1], le réseau de capteurs sans fil est l'une des dix nouvelles technologies qui bouleverseront le monde et notre manière de vivre et de travailler.

3 Environnement de travail

Le modèle que nous avons utilisé lors de ce projet est le MicaZ (voir la figure 2 page 4) de la marque Crossbow. D'autres modèles existent chez ce fabriquant tel que le Mica2, l'Imote2 ou le TelosB.

3.1 MicaZ

L'ensemble des capteurs est basé sur une architecture semblable à celle représentée sur la figure 6 la page 18 [5].

On remarque que les capteurs sont constitués de quatre principaux groupes de composants ayant chacun leur propre rôle :

- **Processeur et mémoire** : composé du processeur qui effectue les traitements, de la mémoire ram qui stocke les données temporaires et de la mémoire flash qui contient le système d'exploitation.
- **Communication** : composé d'une antenne et d'un système de communication radio afin de pouvoir émettre et recevoir des signaux sans-fil.
- **Alimentation** : composé de la batterie fournissant l'énergie nécessaire au fonctionnement et assurant ainsi l'autonomie du capteur.
- **Interactions** : composé des interfaces, de système de capture, de LEDs et qui permet au capteur d'intéragir avec son environnement.

Le MicaZ est constitué d'une mémoire flash de 512 Kbytes permettant de stocker plus de 100 000 mesures ainsi que d'une mémoire flash de 128 Kbytes

[1] Massachusetts Institute of Technology

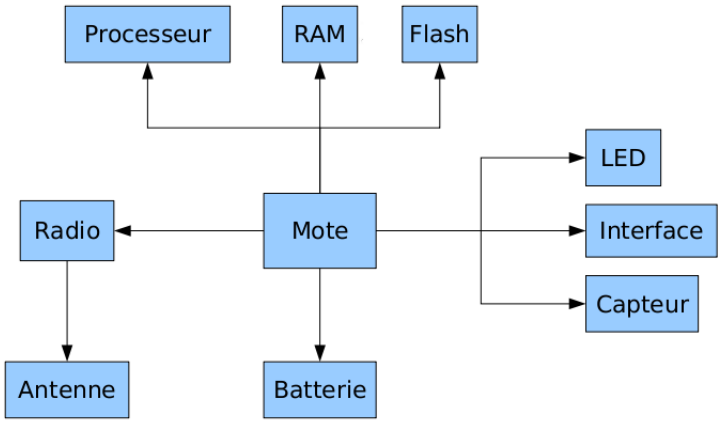


FIG. 1 – Architecture générale d'un capteur

contenant le programme flashé. Cette limite de la capacité est du aux raisons vues précédemment, inhérentes à l'autonomie énergétique.

Dans notre cas, il est alimenté par deux piles AA et fonctionne sur une plage théorique allant de 2,7 a 3,3 Volts. La portée de ce capteur peut atteindre une trentaine de mètres en intérieur et s'étendre jusqu'à 100 mètres en milieu ouvert. Ce type de communication convient pour les réseaux de type WPAN [2] dont la portée est de l'ordre de quelques dizaines de mètres. Celle-ci est gérée par un protocole de haut niveau permettant la communication entre des appareils autonomes restreint en mémoire et énergie : ZigBee.

La machine servant à traiter les données ne possédant aucun moyen de communiquer directement avec le capteur (en radio), il est alors nécessaire d'utiliser un second capteur de même nature ainsi que d'une carte d'interface. Le modèle fournit pour la réalisation de ce projet est un MIB520 (voir la figure 3 la page 4).

Le second capteur vient alors se clipser sur cette carte qui sera reliée a la machine par le port USB. Le couple MIB520 + micaz sert alors de passerelle entre le micaz distant et l'ordinateur. Cela permet la communication des

[2] Wireless Personal Area Networks

FIG. 2 – Photographie d'un MicaZ

FIG. 3 – Photographie d'un MIB520

données mais également la programmation des capteurs.

3.1.1 ZigBee

De nombreux protocoles de liaison sans fil ont été crées avant l'apparition de ZigBee tel que le wifi ou le bluetooth, mais leur utilisation dans certains cadres d'application était impossible, par exemple dans le domaine des réseaux de capteurs, du fait de la lourdeur de ces protocoles. Dés 1998, des ébauches de réseau de type ZigBee sont apparus, s'inspirant de la technologie bluetooth, afin de combler cette nécessité [2]. La spécification initiale de ZigBee propose un protocole lent dont le rayon d'action est relativement faible, mais nécessitant nettement moins de ressource que le wifi ou le bluetooth et dont la fiabilité est assez élevée, le prix de revient faible et la consommation considérablement réduite. ZigBee s'appuie alors sur le protocole 802.15.4 défini par l'IEEE[3] et annoncé quelques années plus tard. Celui-ci est adapté aux réseaux dits LR WPAN [4] , c'est à dire nécessitant une faible consommation, ayant une faible portée ainsi qu'un faible débit. 802.15.4 est utilisé par de nombreuses implémentations basées sur des protocoles propriétaires ou sur IP. La ZigBee alliance s'occupe désormais de concevoir les spécifications de cette technologie.

3.2 TinyOS

TinyOS est un système d'exploitation open-source, conçu par des chercheurs de Berkeley [1], pour des réseaux de capteurs sans-fil. Son architecture est basé sur une association de composant s'appuyant sur le langage NesC[5]. Le NesC est un langage orienté composant, syntaxiquement proche du C.

[3] L'Institute of Electrical and Electronics Engineers est une organisation à but non lucratif. L'IEEE est constituée d'ingénieurs électriciens, d'informaticiens, de professionnels du domaine des télécommunications, etc. L'organisation a pour but de promouvoir la connaissance dans le domaine de l'ingénierie électrique (électricité et électronique). L'IEEE joue un rôle très important dans l'établissement de normes. Ceci est fait par la IEEE Standards Association.

[4] Low Rate Wireless Personal Area Network

[5] network embedded system C

TinyOS s'appuie sur un fonctionnement événementiel, ceci, combiné avec une programmation orienté composant, permet de réduire la taille du code nécessaire à sa mise en place et respecte ainsi les contraintes émises par les capteurs en terme d'économie de mémoire et d'énergie.

Un composant correspond à un élément matériel (LEDs, timer, ...) et peut être réutilisé dans différentes applications.

Un autre grand intérêt à utiliser TinyOS est sa bibliothèque de composants très complète implémentant des protocoles réseaux, des pilotes de capteur, des outils de capture , et plus particulièrement intéressante pour notre TER, une pile IP : Blip, que nous détaillerons plus loin.

3.2.1 Allocation de la mémoire

Le problème de la mémoire est récurant dans le domaine des réseaux sans-fils de capteur, il est donc important de bien la gérer. TinyOS ne propose pas de d'allocation dynamique ni de pointeur de fonction , de plus il ne dispose pas non plus de protection de la mémoire ce qui rend le système vulnérable aux corruptions de mémoire.

Dans le cadre d'une distribution minimal, TinyOS n'occupe pas plus de 400 octets dans la mémoire. Mais 4ko supplémentaire sont nécessaire pour :

- **La pile** servant de mémoire temporaire au fonctionnement du système : empilement et le dépilement des variables locales.
- **Les variables globales** réservant un espace mémoire pour le stockage de valeurs accessible depuis des applications différentes.
- **La mémoire libre** pour le reste du stockage temporaire.

3.2.2 Fonctionnement

Les composants déclarent des tâches, des commandes ou des évènements.

- **les taches** sont des travaux de "longue durée". Lors de l'appel d'un tache cette dernière est ajouté dans une file de type FIFO[6]. Les tâches s'exécutent dans l'odre de la file et en entier car TinyOS ne dispose pas de mécanisme de préemption entre les tâche.
- **les commandes** sont des exécution d'une fonctionnalité précise dans un autre composant
- **les événements** sont les équivalents logiciels aux interruptions matérielles, ils sont prioritaires par rapport aux tâches et peuvent donc les interrompre.

Lorsque la file d'attente est vide, cela signifie qu'aucune tâche n'est exécuté, et TinyOS met en veille le capteur, afin d'économiser la batterie.

[6]First In First Out : premier entré premier sorti

L'ordonnanceur dispose donc d'une file d'attente FIFO (disposant d'une capacité de 7 tâches) et de deux niveaux de priorité (bas pour les tâches et haut pour les événements).

TinyOS peut être installé partir d'un environnement Windows ou bien Linux. Pour le TER nous utilisons une image VMWare[7] d'une distribution Ubuntu 9.04 disposant d'un environnement TinyOS 2.1.0 entièrement fonctionnel, ainsi que le logiciel de programmation Eclipse pourvu du plugin Yeti 2[8].

3.2.3 Blip

Blip, conçu par Berkeley, est l'implementation dans TinyOS d'un certain nombre de protocoles basé IP [3], dans notre cas nous utiliserons blip pour une connectivité IPv6. tinyOS et blip donnent la possibilité de créer de réseaux IP multi-sauts entre différents capteurs utilisant des protocoles de communication différent.

En raison de l'evolution rapide de l'IETF[9] et des normes IEEE , blip n'est actuellement pas entièrement conforme aux normes, mais offre une importante interopérabilité avec d'autre réseaux IP.

L'utilisation de la librairie blip doit être déclarée lord de la compilation.

Structure d'adressage

La pile IP fournit la couche réseau une interface de datagramme IPv6. Afin de pouvoir programmer des sockets, deux structures de données sont très importantes :

- **struct in6_addr** qui définit le datagramme IP
- **struct sockaddr_in6** qui inclut 'struct in6_addr' et le port afin de définir une socket

```
#include <ip.h>

struct in6_addr
  {
    union
      {
      uint8_t    u6_addr8[16];
```

[7] http://nap.cse.bgu.ac.il/~ariksa/UbunTOS_NAP.html

[8] plugin TinyOS 2 pour Eclipse, gérant notamment le NesC

[9] L'Internet Engineering Task Force est un groupe informel, international, ouvert tout individu, qui participe l'élaboration de la plupart des nouveaux standards pour Internet.

```
      uint16_t u6_addr16[8];
      uint32_t u6_addr32[4];
      } in6_u;
#define s6_addr                 in6_u.u6_addr8
#define s6_addr16               in6_u.u6_addr16
#define s6_addr32               in6_u.u6_addr32
  };

struct sockaddr_in6 {
  uint16_t sin6_port;
  struct in6_addr sin6_addr;
};

/* parse a string representation of an address */
void inet_pton6(char *addr, struct in6_addr *dest);

/* stringify a packed ipv6 address */
int  inet_ntop6(struct in6_addr *addr,
char *buf, int cnt);
```

Mise en œuvre

Afin de pouvoir communiquer avec le réseau IP de capteurs, il est nécessaire d'installer un routeur de bord sur la machine connectée au capteur. Celui-ci fera le lien entre le réseau de la machine et le réseau de capteur. Les étapes d'installations de ce routeur de bord sont détaillé dans le tutoriel que nous avons rédigé (voir annexe 7.1 à la page 21. La configuration du routeur est contenue dans un fichier. Elle est chargée au démarrage du routeur sur la machine, elle comprend notamment son adresse IPv6 et le canal sur lequel vont communiquer les capteurs.

3.3 Le nesC

Le langage nesC est le langage utilisé par tinyOS, il permet de déclarer deux types de composants [4] :

- les modules, contenant le code d'un composant
- les configurations qui permettent de faire le liens entre différents modules

Les interfaces décrivent les commandes et évènements proposés par les composants. Il est possible de les redéfinir.

3.3.1 La compilation croisée

Un compilateur croisé[10], est un compilateur capable de produire un code objet pour une exécution sur un environnement différent de celui ou la compilation est effectué. C'est le cas lorsque nous compilons un programme conçu pour les capteurs micaz.

4 Implémentation

L'application qui nous à été demandé d'implémenter devait répondre à deux impératifs :

- récupérer des données en provenance d'un capteur
- utiliser le protocole IP

De plus une interface serait souhaitable afin de visualiser au mieux, les informations reçues.

4.1 Fonctionnement général

L'application se compose de deux parties, une embarquée écrite en NesC et une partie que nous appellerons cliente, écrite en Java, qui se lancera depuis la machine. Le client Java envoie une requête via une socket UDP et demande ainsi à récupérer des données (ici le voltage du capteur), il se met donc en attente de réception. Lorsque le capteur reçoit une demande, il effectue dix captures à intervalles réguliers, les stocke dans un tableau puis les envoie au client afin qu'il puisse les traiter. Tant que l'application n'est pas fermée, le client demande des données régulièrement afin de mettre à jour la courbe affichée dans l'interface.

Du fait que nous utilisons un protocole non fiable, UDP, il se peut que des paquets n'arrivent pas à leur destination et soient ainsi perdus. Cela pourrait entrainer un blocage du programme et un arrêt complet de la communication. Le scénario détaillé ci-dessus représente le comportement idéal de l'application mais il n'est pas atteignable sur une longue durée.

Pour remédier à ce problème un timer est nécessaire afin de déterminer si le client aurais du recevoir une réponse ou non.

Le client initie toujours la communication par une demande de données puis lance immédiatement le timer. Le capteur récupère alors cette demande ce qui a pour conséquence de lui faire effectuer les dix lectures consécutives.

[10] en anglais cross compiler

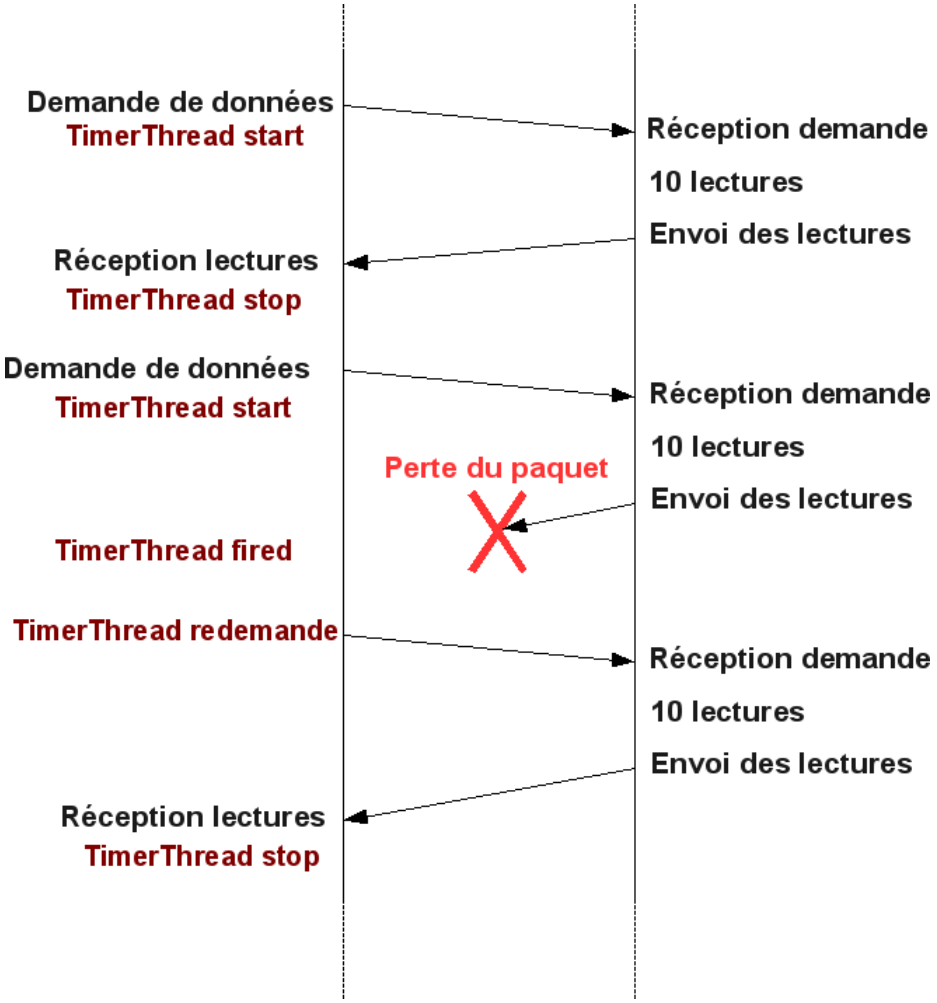


FIG. 4 – Schema du fonctionnement général de l'application

Une fois ce nombre atteint, un message est envoyé en retour contenant l'ensemble des dix données lues. Le client le réceptionne alors, stoppe le timer démarré précédemment, puis effectue les traitements pour afficher ces informations.

En parcourant le schéma 4.1 à la page 10, nous remarquons alors une nouvelle demande de la part du client. Il lance le timer puis se met en attente de la réponse. Le capteur, ayant reçu la requête, effectue 10 captures puis émet ces données. Cette fois-ci celles-ci n'arrivent pas à destination et sont perdues. Le client, par l'intermédiaire de son timer qui aurait du se couper lors de la réception, va effectuer une nouvelle demande ce qui va réamorcer

le processus de communication qui ne sera ainsi jamais bloqué. On remarque également que cette solution est valable si un message est perdu du client vers le mote.

Nous allons maintenant voir en détails l'implémentation de chaque partie.

4.2 Partie embarquée

Le fonctionnement de la partie embarquée est relativement simple, le capteur est initialement en attente de demande de donnée. Lorsque un paquet est reçu, un timer périodique est lancé afin d'effectuer dix mesures avant de les envoyer. La figure **??** à la page 16 en présente un shema de fonctionnement général.

Le code est divisé en deux fichiers distincts : IPOscilloscopeC.nc et IPOscilloscopeP.nc. Le premier contient la définition du ou des composants qui seront utilisés par l'application déployée sur le capteur alors que le deuxième comprend les modules, les interfaces utilisées ainsi que le code qui sera exécuté.

4.2.1 IPOscilloscopeC.nc

Les composants déclarés dans ce fichier sont :
- MainC - Interface du système avec la séquence de boot TinyOS.
- LedsC - Permet l'utiliation des leds
- DemoSensorC() as Sensor - Capteur de démonstration auquel on associe le nom Sensor
- IPOscilloscopeP - Composant défini dans notre fichier IPOscilloscopeP.nc
- TimerMilliC() as Timer0 - Timer en milliseconde nommé Timer0
- IPDispatchC - Envoi de messages en fonction de l'entête suivant des paquets IP
- UdpSocketC() as Echo,Status - Deux sockets UDP du nom de Echo et Status

Il faut ensuite associer ces composants au composant IPOscilloscopeP afin de pouvoir les utiliser dans le code de celui-ci. Cela est effectué de la manière suivante :

```
IPOscilloscopeP.Boot -> MainC;
IPOscilloscopeP.Leds -> LedsC;
IPOscilloscopeP.RadioControl -> IPDispatchC;
IPOscilloscopeP.Echo -> Echo;
IPOscilloscopeP.Status -> Status;
```

```
IPOscilloscopeP.Timer0 -> Timer0;
IPOscilloscopeP.Read -> Sensor;
```

Nous aurons ainsi accès à toutes les fonctionnalités proposées par ces composants.

4.2.2 IPOscilloscopeP.nc

Ce fichier va donc contenir la définition et l'implémentation du module IPOscilloscopeP. La librairie blip est incluse dans le programme, afin de bénéficier des structures IPv6 et des sockets UDP. On commence par annoncer, dans la clause uses, toutes les interfaces qui seront utilisées.

```
    interface Boot;
    interface SplitControl as RadioControl;
    interface Read<uint16_t>;
    interface UDP as Echo;
    interface UDP as Status;
    interface Leds;
    interface Timer<TMilli> as Timer0;
```

Viens ensuite l'implémentation du module qui sera le code exécuté par le capteur. Nous avons vu dans le chapitre consacré à TinyOS que son fonctionnement est basé sur des évènements. Notre programme sera un ensemble de fonctions réagissant chacune à un certain évènement. Nous allons les présenter selon l'ordre dans lequel elles se déclenchent suivant le schéma de fonctionnement vu précédemment.

On déclare dans un premier temps des variables globales nécessaires à l'application :

```
/*timer lance ou non*/
   bool timerStarted;

   /*adresse machine cliente*/
  struct sockaddr_in6 *ipfromto;

  /*structure contenant les lectures*/
  oscilloscope_t local;

/*nombre de lecture, 0 a NREADINGS  (ici 10 lectures)*/
  uint8_t reading;
```

Le premier évènement à être déclenché correspond au démarrage du capteur et l'action associée se nomme Boot.booted

```
event void Boot.booted() {

/*on lance le systeme de communication radio*/
  call RadioControl.start();

/*le timer n'est pas lance*/
  timerStarted = FALSE;

/*On lie la socket UDP Echo au port 7001
et la socket Status au port 7*/
  call Echo.bind(7001);
  call Status.bind(7);

}
```

Les éléments nécessaires à la communication ont maintenant été initialisé, le capteur est prêt à recevoir un message sur sa socket status, ce qui aura pour effet, une fois cela réalisé, de déclencher la fonction Status.recvfrom() associée à la réception d'un message.

```
event void Status.recvfrom
(struct sockaddr_in6 *from, void *data,
uint16_t len, struct ip_metadata *meta) {

/*paquet recu*/

/*LED verte allumee*/
report_received();

/*recuperation de l'adresse IP du client*/
ipfromto=from;

/*nombre de lecture remis a 0*/
reading=0;

/*lancement d'un timer periodique de 50ms*/
call Timer0.startPeriodic(50);
timerStarted = TRUE;
```

```
}
```

Les diodes du capteur étant la seule interactivité possible avec celui-ci, nous allons les utiliser afin d'afficher les différentes étapes et les possibles erreurs survenues durant l'exécution.

```
void report_problem(){ call Leds.led0Toggle();}//rouge
void report_sent(){ call Leds.led1Toggle();}//orange
void report_received(){ call Leds.led2Toggle();}//vert
```

On allume la diode verte dès que l'on reçoit un paquet. L'adresse IP de la machine cliente peut être récupéré dans le paramètre from de la fonction. On la garde alors dans une variable ipfromto afin de l'utiliser pour envoyer les données. On met ensuite la variable reading à 0 puis on lance un timer de 50ms périodique, c'est à dire qui se répétera tant qu'il n'est pas arrêté.

Une fois ce timer arrivé à son terme, un évènement est déclenché appelant la fonction Timer0.fired

```
event void Timer0.fired() {

    if (!timerStarted) {
      call Timer0.startPeriodic(50);
      timerStarted = TRUE;
    }

    if (reading == NREADINGS)
      {
    call Status.sendto(ipfromto, local.readings ,
     sizeof(local.readings));
    call Timer0.stop();
    report_sent();
    reading=0;
      }
                else{
    if (call Read.read() != SUCCESS)
      report_problem();
      }
  }
```

Si le nombre de lecture effectuée est égal à 10, alors on envoie un paquet contenant les données à l'adresse récupérée lors de la réception puis on on stoppe le timer. On allume la led jaune signifiant qu'un paquet a été envoyé

puis on réinitialise le nombre de lecture effectuée à zéro. Sinon on effectue une lecture ce qui déclenchera l'événement readDone.

```
event void Read.readDone(error_t result, uint16_t data)
{
    if (result != SUCCESS)
      {
        data = 0xffff;
        report_problem();
      }
    local.readings[reading] = data;
    reading++;

  }
```

Les données lues sont placées dans un tableau et reading est incrémenté afin de remplir le tableau et non d'écraser les valeurs déjà présentes.

4.3 Partie Java

4.3.1 Client

Le client est basé sur l'utilisation des sockets UDP java, dont le fonctionnement dans le contexte IPv6 est semblable à celui d'IPv4. Le programme se lance en indiquant l'adresse IP du capteur auquel on demande des données.

La première action effectuée est le lancement de l'interface à l'intérieur d'un thread. On initialise ensuite la socket avec l'adresse transmise au programme. On entre alors dans une boucle commençant par l'envoi d'un paquet via la socket précédemment initialisée puis on lance un nouveau thread nommé Timer_thread.Ce timer permet de résoudre le problème de perte de paquets, en effet, si un paquet est perdu le client sera bloqué en attente d'un paquet qui n'arrivera jamais. Le capteur effectue ces mesures en 500ms, nous avons donc choisi de faire attendre le timer deux fois plus longtemps, c'est à dire 1s. Une fois le timer arrivé à son terme, on estime que le paquet n'arrivera jamais et le thread renvoi une demande. On répète cette étape toute les seconde jusqu'a ce qu'un message soit reçu. Dès la réception d'un paquet, le thread timer est stoppé.

Les valeurs récupérées ayant été sérialisé, il est nécessaire de les désérialiser afin de les manipuler. Cette tâche est réalisée grâce aux fonctions fournies par la classe OscilloscopeMsg, et en particulier get_readings(). Une formule

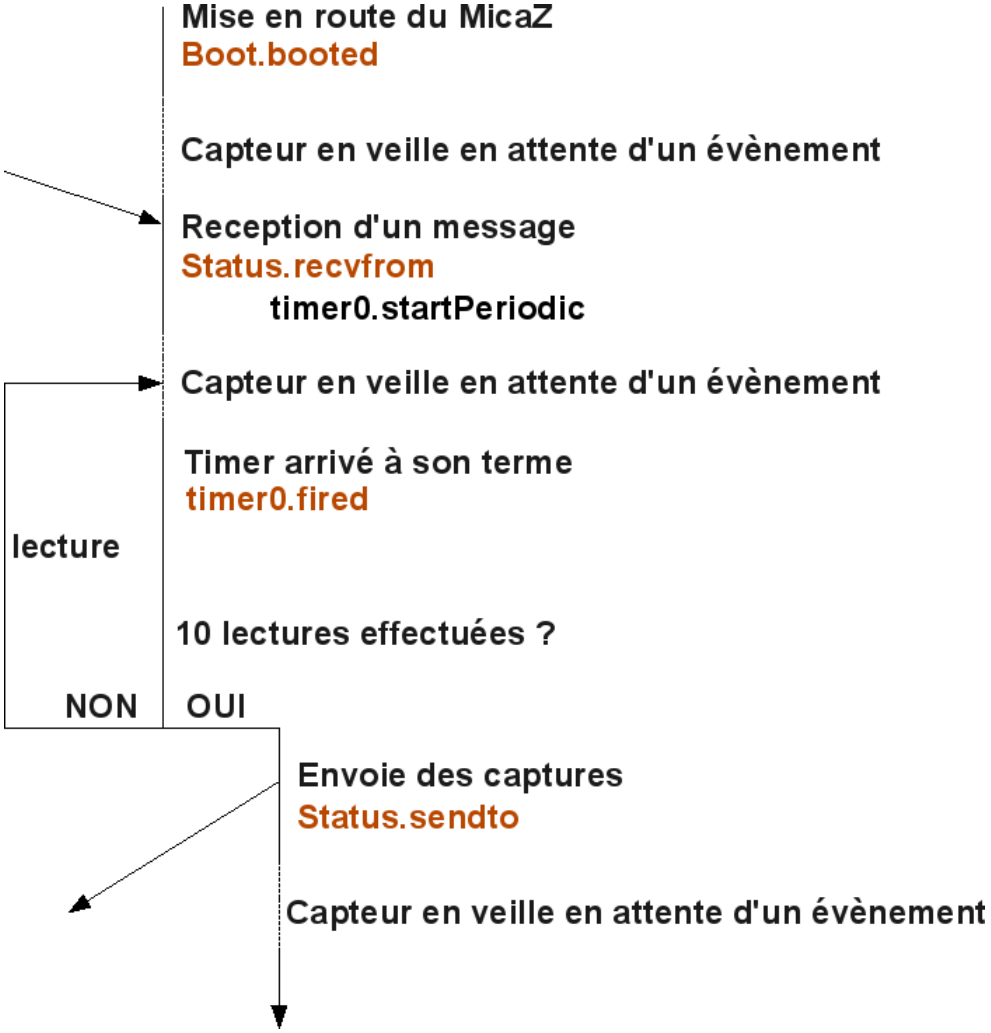


FIG. 5 – Schema du fonctionnement de la partie embarquée

doit ensuite être appliqué afin d'obtenir les valeurs correspondant réellement aux valeurs physiques. Elle n'est valable que pour la grandeur que nous capturons, dans notre cas le voltage

Voici la formule : voltage=Vref * 1024 / valeur où Vref=1.223V

On peut alors transmettre ces valeurs à l'interface afin qu'elles soient affichées, ce qui est effectué via la fonction me.setData. On met à jour l'affichage par la fonction window.newData() ;

4.3.2 Interface

L'interface de notre programme java est une adaptation de celle du programme Oscilloscope déjà présent dans les exemples d'applications de tinyOS. Il a ensuite fallu le modifier afin de l'adapter à l'affichage de nos données.

La fenêtre est constituée de différents éléments :

- **La zone principale** qui contient le graphe représentant les données ainsi qu'un axe des abscisses gradué
- **La zone de gauche** permet d'identifier le capteur émettant les données ainsi que de lui associer une couleur. Cette couleur sera ainsi reporté sur le graphe représentant les valeurs émises par ce capteur.
- **La zone du bas** est constitué d'un bouton nommé Clear Data permettant de remettre à zéro les données, ce qui a pour effet de nettoyer le graphe de toutes données déjà affichées. Un autre bouton du nom de Default, adjacent au précédent, permet de réinitialiser la configuration de l'interface. Cela remet l'echelle des X ainsi que des Y la même valeur qu'au lancement du programme. On retrouve ensuite un slider permettant de fixer l'échelle des abscisses dans une plage de valeur comprise entre 50 et 12800, et doublant à chaque graduation. Enfin, en bas à droite de la fenêtre se trouve un zone de saisie destiné à entrer l'échelle voulue de l'axe des ordonnées. Le texte saisie doit être de la forme " x - y " avec x et y compris entre 0 et 65535 et y strictement supérieur à x.

Comme dit précédemment, nous avons modifier l'interface d'origine afin de l'adapter aux données que nous voulions afficher. Celle-ci était en effet faites de façon à ne pouvoir afficher que des entiers. Or, nous avions besoin d'une précision supplémentaire au travers de l'utilisation de nombre flottants. Ceci a été ajouté en modifiant le type des valeurs manipulées au sein du programme.

Cette adaptation aux nombres à virgule nous a ensuite amener à repenser l'affichage des données et en particulier à l'axe des Y. La précision minimale des graduations de cet axe n'était que de l'ordre de l'unité, du fait que l'on affichait seulement des entiers. Dans notre cas, il devenait alors difficile d'évaluer approximativement mais avec assez de précision la valeur du nombre flottant affiché. Nous avons ajouté un niveau de graduation supplémentaire, correspondant au dixième, seulement visible lorque l'échelle des Y est de 1 (exemple : "1-2","10-11","2000-2001"). Le bouton Default, abordé précédemment, a lui aussi été ajouté pour un soucis d'ergonomie. Une zone de texte, dédié à saisir l'intervalle temporel de capture des données à destination du capteur a été enlevé, cette fonction ayant été retiré.

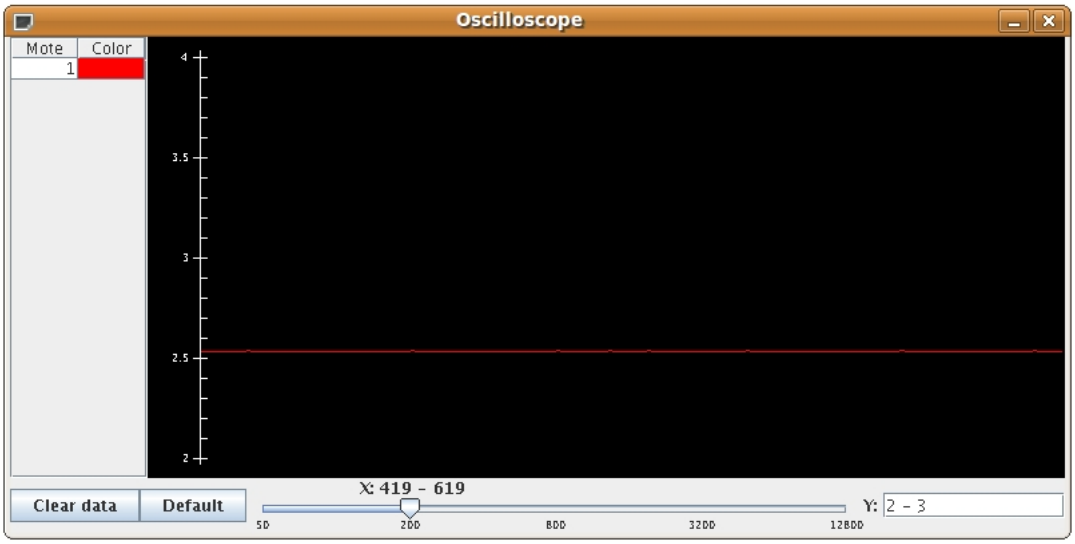


FIG. 6 – Capture d'écran de l'interface Java

5 Conclusion

Au cour de ce projet, il nous à été demandé de configurer et tester la connectivité IP entre des capteurs micaz, puis nous devions implémenter un petit programme permettant de récupérer des données au moyen de cette connexion IP. Il a fallut pour cela prendre en main beaucoup de nouvelles technologies, comme TinyOS et le nesC.

Nous avons également conçu un tutoriel afin d'expliquer pas à pas la mise en place et le test d'une simple connectivité IP entre micaz, puis par la même occasion comment utiliser notre programme afin que d'autres personnes puissent, s'ils le souhaitent, continuer notre travail pour, par exemple, récupérer des données plus pertinentes que le voltage.

La mise en œuvre de ce projet nous a permis de découvrir un nouveau domaine, une nouvelle manière de programmer et de concevoir une application, avec des contraintes techniques et matérielles très importantes.

Ce projet été particulièrement intéressant par le fait que les réseaux de

capteurs sans-fil sont vraiment en pleine expension de nos jours, mais encore trop peu connu des personnes extérieurs à ce domaine.

Lors de nos expérimentations nous ne disposions que d'une machine et de deux capteurs, dont un servant de passerelle. Il serait peut être intéressant par la suite d'étendre ce projet à un nombre de capteur beaucoup plus important, afin de couvrir une plus grande zone à étudier. Ainsi l'adressage IP des différents capteurs permettrait de diviser logiquement la zone en différents réseaux indépendants. Mais le plus grand intérêt serait la connexion de ces réseau de capteur sans-fil avec internet.

6 Bibliographie - Webographie

Références

[1] Site officiel de tinyos. http://www.tinyos.net.

[2] Site officiel de zigbee. http://www.zigbee.org.

[3] Tutoriel officiel blip. http://docs.tinyos.net/index.php/BLIP_Tutorial.

[4] D.Gay, D.Culler, and P.Levis. *nesC Language Reference Manual*. 2002.

[5] Bouillaguet Mathieu and Valero Mathieu. Os pour réseaux de capteurs, 2006. http://docs.tinyos.net/index.php/BLIP_Tutorial.

7 Annexes

7.1 Tutoriel

Voici le tutoriel que nous avons écrit afin de facilité la configuration et le test de connectivité IP pour notre matériel, nous avons également expliqué comment se servir de notre application. Ceci afin de pouvoir plus facilement poursuivre notre travail.

Tutoriel : installation et configuration de BLIP TinyOS-2.1.0

Ce tutoriel à été réalisé par des étudiants de 1ère année master Technologie de l'Internet de l'université de Pau dans le cadre d'un projet sur les réseaux de capteurs sans-fil, encadré par M.Pham professeur à l'UPPA.

Afin de pouvoir bénéficier d'un environnement complet et opérationnel, vous pouvez télécharger l'image VMware sur le site d'Arik Sapojnik à l'adresse suivante :
http://nap.cse.bgu.ac.il/~ariksa/UbunTOS_NAP.html.
Cette image contient :

- Ubuntu 9.04 (deux utilisateurs: root et nap)
- TinyOS-2.1.0
- Un environnement TinyOS fonctionnel
- Eclipse 3.5.1 muni du plugin TinyOS

Ce tutoriel est conçu pour le matériel suivant :

- L'image VMware présentée précédement
- Une carte d'interface MIB520 connectée en USB
- Des capteurs CrossBow modèle MicaZ

Pour toutes différences d'environnement ou de matériel, vous pouvez adapter ce tutoriel en conséquence.

Installation du routeur de bord

L'application fournissant une interface IEEE 802.15.4 et gérant l'IP se trouve dans le dossier /opt/tinyos-2.1.0/apps/IPBaseStation. Il faut l'installer sur le capteur qui restera connecté en USB grâce au MIB520 afin d'établir un pont entre la machine et le noeud.

```
$ cd /opt/tinyos-2.1.0/apps/IPBaseStation
```

```
$ make blip install mib520,/dev/ttyUSB0 micaz
```

Cette opération compile le programme et flash le capteur directement. Il est également possible de seulement compiler le programme puis de flasher sans le recompiler en operant comme suit.

```
$ make blip micaz
$ make blip reinstall mib520,/dev/ttyUSB0 micaz
```

Une fois IPBaseStation installé, il faut compiler le driver de routage (serial forwarder), pour cela procédez comme suit:

```
$ cd /opt/tinyos-2.1.0/support/sdk/c/sf
$ ./bootstrap.sh
$ ./configure
$ make
```

Si cette étape echoue, il se peut qu'il manque les paquets automake et autoconf sur votre distribution.

Répétez ensuite cette étape pour le driver.

```
$ cd /opt/tinyos-2.1.0/support/sdk/c/blip
$ ./bootstrap.sh
$ ./configure
$ make
```

Si toutes ces étapes ont réussi, alors le driver pour le routeur de bord est bien construit. Vous pouvez le lancer en allant dans /opt/tinyos-2.1.0/support/sdk/c/blip et en tapant :

```
$ sudo driver/ip-driver /dev/ttyUSB1 micaz
```

Configuration du routeur de bord

Lors du démarrage du routeur de bord, sa configuration est chargée depuis le fichier /opt/tinyos-2.1.0/support/sdk/c/blip/serial_tun.conf.

```
# Before you can run the adaptation layer and router, you must
# configure the address your router will advertise to the subnet, and
# which network interface to proxy neighbor advertisements on.
#

# set the debug level of the output
# choices are DEBUG, INFO, WARN, ERROR, and FATAL
# log DEBUG
# set the address of the router's 802.15.4 interface. The interface
# ID must be a 16-bit short identifier.
addr fec0::64
# the router can proxy neighbor IPv6 neighbor discovery on another
# interface so that other machines on the subnet can discover hosts
# routing through this router. This specifies which interface to proxy
# the NDP on.
proxy lo

# which 802.15.4 channel to operate on. valid choices are 11-26.
channel 15
```

Ici l'adresse IPv6 du routeur sera donc fec0::64 (adresse réduite) et opèrera sur le canal 15.
Le canal du routeur doit être identiques aux canaux des noeuds pour pouvoir communiquer.

Installation d'un noeud

Une fois le routeur de bord installé et configuré, vous pouvez installer un noeud. Pour cela TinyOS fournit une application de test dans le dossier /opt/tinyos-2.1.0/apps/UDPEcho. Cette application permet notamment de tester le ping, afin de voir si le noeud est bien dans le réseau. Flashez donc le capteur comme précédement mais avec le programme UDPEcho.

```
$ cd /opt/tinyos-2.1.0/apps/UDPEcho
```

```
$ make blip install.ID mib520,/dev/ttyUSB0 micaz //tester
```

Ou bien :

```
$ make blip micaz
$ make blip reinstall.ID mib520,/dev/ttyUSB0 micaz
```

ID est l'identifiant du noeud, il doit être unique. L'ID est par defaut 1 s'il n'est pas précisé lors de l'installation

Test

Maintenant que le routeur de bord et un noeud sont installés, assurez-vous d'avoir lancé le routeur comme vu précedement. Puis testez la communication IP avec le capteur avec le ping6 ou tracrt6 (l'installation de paquets sera peut être nécessaire)

```
$ ping6 fec0::ID
```

Comme vous avez pu le remarquer l'adresse IPv6 du noeud est la concaténation du préfixe indiqué dans le fichier serial_tun.conf et de l'ID du noeud.

```
$ tracert6 fec0::ID
```

On remarque que pour atteindre fec0::ID on passe bien par fec0::64 le routeur de bord.

IP Oscilloscope

Pour aller un peu plus loin, nous vous proposons d'installer l'application que nous avons implémentée.

Telecharger IPOscilloscope

Une fois l'archive téléchargée, extrayez son contenu dans le dossier tinyos-2.1.0/apps. Cette application permet de récupérer le voltage du capteur en IP, une interface Java est fournie afin de visualiser en temps réel les données.
Cette application peut être flashée sur le noeud et nécessite que IPBaseStation soit installé sur le capteur relié à la machine. Une fois les capteurs prêts, lancez le driver et placez vous dans le dossier IPOscilloscope/java. Vous pouvez vous assurer que le ping fonctionne bien sur le noeud.Compilez le programme java et lancez l'application.

```
$ javac *.java
$ Client fec0::ID
```

Quelques liens utiles

Divers tutoriels sur le site de documentation de TinyOS :	http://docs.tinyos.net/index.php/TinyOS_Tutorials
La fiche technique des capteurs MicaZ :	http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf
Le site d'Arik Sapojnik avec l'image VMWare :	http://nap.cse.bgu.ac.il/~ariksa/UbunTOS_NAP.html

Nous contactez

Frédéric Delgado :	delgado.frederic@gmail.com
Cédric Laborde :	king_xaero@hotmail.fr

7.2 Fiche technique du MicaZ

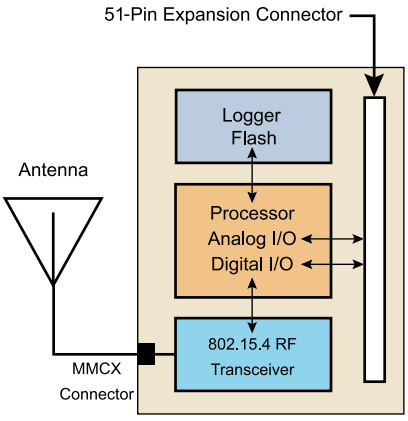
MICAz

WIRELESS MEASUREMENT SYSTEM

- 2.4 GHz IEEE 802.15.4, Tiny Wireless Measurement System
- Designed Specifically for Deeply Embedded Sensor Networks
- 250 kbps, High Data Rate Radio
- Wireless Communications with Every Node as Router Capability
- Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

Applications

- Indoor Building Monitoring and Security
- Acoustic, Video, Vibration and Other High Speed Sensor Data
- Large Scale Sensor Networks (1000+ Points)



MPR2400 Block Diagram

MICAz

The MICAz is a 2.4 GHz Mote module used for enabling low-power, wireless sensor networks.

Product features include:

- IEEE 802.15.4 compliant RF transceiver
- 2.4 to 2.48 GHz, a globally compatible ISM band
- Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security
- 250 kbps data rate
- Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
- Plug and play with Crossbow's sensor boards, data acquisition boards, gateways, and software

MoteWorks™ enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and a configuration.

Processor & Radio Platform (MPR2400CA)

The MPR2400 is based on the Atmel ATmega128L. The ATmega128L is a low-power microcontroller which runs MoteWorks from its internal flash memory. A single processor board (MPR2400) can be configured to run your sensor application/ processing and the network/radio communications stack simultaneously. The 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals. The MICAz (MPR2400) IEEE 802.15.4 radio offers both high speed (250 kbps) and hardware security (AES-128).

Sensor Boards

Crossbow offers a variety of sensor and data acquisition boards for the MICAz Mote. All of these boards connect to the MICAz via the standard 51-pin expansion connector. Custom sensor and data acquisition boards are also available. Please contact Crossbow for additional information.

Document Part Number: 6020-0060-04 Rev A

Processor/Radio Board	MPR2400CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	< 15 µA	Sleep mode
RF Transceiver		
Frequency band[1]	2400 MHz to 2483.5 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	-24 dBm to 0 dBm	
Receive Sensitivity	-90 dBm (min), -94 dBm (typ)	
Adjacent channel rejection	47 dB	+ 5 MHz channel spacing
	38 dB	- 5 MHz channel spacing
Outdoor Range	75 m to 100 m	1/2 wave dipole antenna, LOS
Indoor Range	20 m to 30 m	1/2 wave dipole antenna
Current Draw	19.7 mA	Receive mode
	11 mA	TX, -10 dBm
	14 mA	TX, -5 dBm
	17.4 mA	TX, 0 dBm
	20 µA	Idle mode, voltage regular on
	1 µA	Sleep mode, voltage regulator off
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes
[1]5 MHz steps for compliance with IEEE 802.15.4/D18-2003.
Specifications subject to change without notice

MIB520CB Mote Interface Board

Base Stations

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any MICAz Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510 or MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

Ordering Information

Model	Description
MPR2400CA	2.4 GHz MICAz Processor/Radio Board
WSN-START2400CA	2.4 GHz MICAz Starter Kit
WSN-PRO2400CA	2.4 GHz MICAz Professional Kit

Document Part Number: 6020-0060-04 Rev A