Article précédent (/Hackable/HK-019/Decouvrez-LoRaWAN-et-creez-votre-passerelle-concentrateur)
Article suivant (/Hackable/HK-019/Construisez-un-emetteur-433-Mhz-pour-remplacer-vos-telecommandes)

CRÉEZ VOS MONTAGES ARDUINO COMMUNICANTS SUR LORAWAN

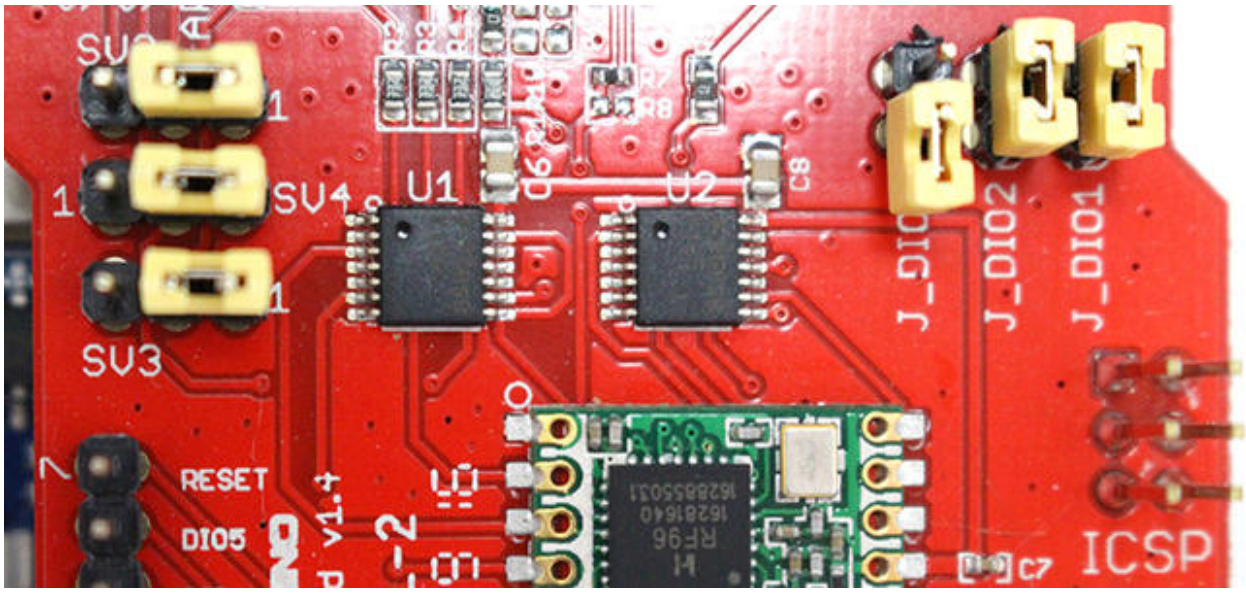
Hackable n° 19 (/Hackable/HK-019) | juillet 2017 | Bodor Denis (/auteur/Bodor-Denis)

Embarqué (/search/node?Domaines%5B0%5D=72457) **Radio et wireless (/search/node?Domaines%5B0%5D=72465)**

Dans l'article précédent, nous avons vu comment il était possible de déployer son concentrateur LoRaWAN avec TTN relativement simplement, mais ceci n'est toutefois pas indispensable. Si un concentrateur se trouve près de chez vous, vous pouvez l'utiliser pour vos projets et ainsi profiter, avec un investissement minimum, des bienfaits de LoRaWAN et des communications très longues distances pour vos projets.

Le principe même du fonctionnement de LoRaWAN est de permettre la mise en place d'un vaste réseau. Dans le cas de *The Things Network* (TTN), ce réseau utilise des concentrateurs installés par ses membres, qu'il s'agisse de produits clés en main, de kits comme nous l'avons fait dans l'article précédent ou de l'utilisation de modules et cartes achetés séparément et utilisés avec une Raspberry Pi ou tout autre ordinateur mono-carte.

Le fait de rejoindre le réseau communautaire TTN, ou un autre du même type, ne vous oblige en rien à déployer un concentrateur. Si vous avez la chance d'en avoir un à proximité, vous pouvez tout simplement l'utiliser pour votre projet. Tout ce qu'il vous faut alors c'est un module LoRa qui permettra à votre montage de contacter ce concentrateur pour échanger des données avec TTN, puis vos applications. Créer un compte TTN suffit donc, en plus du module LoRa, à créer un capteur relié à Internet, ou en d'autres termes, un objet connecté.



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_hopeRF1.jpg)

Le module qui équipe le shield Dragino Lora intègre une puce Semtech et divers composants annexes. Ce module peut être trouvé sur différents sites pour moins d'une dizaine d'euros, mais il ne peut être utilisé qu'en 3,3V. Sur le shield en revanche des convertisseurs de tension permettent une connexion à n'importe quelle carte Arduino.

1. DES ADRESSES, DES IDENTIFIANTS ET DE LA SÉCURITÉ

Un troupeau de concentrateurs installés par des particuliers, un réseau ouvert à tous, des communications radio en tous sens, des échanges sur Internet constants... LoRaWAN et *The Things Network* peuvent sembler être un véritable terrain de jeu pour qui voudrait écouter tout ce qui passe. Mais en réalité, la sécurité est au cœur de LoRaWAN, car les communications sont chiffrées entre un node et le réseau, mais également entre un node et l'application. Les données contenues dans les messages ne sont donc ni visibles du concentrateur, ni par le serveur réseau.

Pour protéger ces messages et s'assurer par la même occasion des identités de chaque intervenant, on utilise des clés et des identifiants. Un processus nommé « activation » utilise ces clés et informations pour s'identifier et se connecter au réseau TTN. Ces clés sont créées ou obtenues lors de l'enregistrement d'un concentrateur ou d'un node dans votre interface TTN (console).

Il est important de comprendre ces mécanismes de sécurité, car il vous faudra les utiliser pour créer les croquis de vos nodes envoyant des données avec LoRaWAN. Il existe deux types d'activations :

- OTAA pour *Over The Air Activation* est la façon la plus sûre de se connecter au serveur du fournisseur réseau LoRaWAN (TTN dans notre cas), c'est donc celle à préférer. Avec cette méthode, on utilise l'identifiant de l'application et sa clé associée pour négocier la connexion et on obtient en retour une adresse pour notre node. Les autres clés permettant le chiffrement sont négociées automatiquement ainsi que de nombreux paramètres de communication.

- ABP pour *Activation By Personalization*. Ici, les clés et les identifiants ne sont plus négociés dans un échange durant la connexion, mais sont déterminés lorsque vous enregistrez votre périphérique côté TTN. Ces informations doivent ensuite être utilisées dans votre croquis pour permettre la connexion. Comme le précise la documentation TTN, ceci peut paraître plus simple, car il n'y a pas de processus de connexion et de négociation, mais cela implique également une sécurité plus faible. De plus, vous pouvez rencontrer des problèmes à cause d'un autre mécanisme de sécurité.

Ce denier point est important et concerne le compteur de messages. Afin de garantir la sécurité du système, celui-ci doit s'assurer qu'un échange ne puisse pas être simplement rejoué (capturé et envoyé une seconde fois). Les messages chiffrés contiennent donc un numéro ou compteur qui permet au serveur de ne pas accepter une seconde fois un même message. Avec une activation OTAA, ce numéro fait partie de la négociation (en principe), mais avec ABP c'est à votre croquis de gérer ce mécanisme. En temps normal ceci est pris en charge par la bibliothèque Arduino, mais uniquement tant que votre montage n'est pas redémarré. En cas d'arrêt et de redémarrage, le compteur recommencera du début et le serveur va ignorer les messages pensant qu'il s'agit de copies. Une solution pour régler le problème est de désactiver cette fonction côté TTN, mais cela réduit grandement la sécurité.

Dans le mécanisme OTAA, on utilise l'identifiant de l'application (AppEUI) et la clé de l'application (AppKey). L'adresse du node (DevAddr), la clé de session réseau (NwkSKey) et la clé de session de l'application (AppSKey) sont générées à partir de l'identifiant et de la clé d'application durant la connexion. La première clé de session protège la communication entre le node et le réseau et la seconde étend cette protection jusqu'à l'application. Dans les deux cas, le concentrateur ne voit aucune des données qui circulent et dans le second, le serveur réseau lui-même n'a pas connaissance des données.

Le mécanisme ABP utilise également ces clés de session, mais elles ne sont pas négociées à la connexion, elles doivent alors figurer dans votre croquis et, bien entendu, correspondre aux mêmes informations présentes côté réseau lors de l'enregistrement d'un node.

Dans le cadre de cet article, nous nous en tiendrons à l'activation OTAA, plus sûre, plus simple et recommandée par défaut par TTN. La procédure d'activation ne concerne en effet pas seulement la sécurité, mais également la transmission d'informations spécifiques et en particulier les fréquences utilisées. Dans la partie précédente, j'ai précisé que ce qui fait un véritable concentrateur LoRaWAN est sa capacité à gérer 8 canaux de communication

et non un seul. Chaque canal possède une fréquence et des caractéristiques précises (débit, facteur d'étalement, rapport cyclique, bande passante). Trois de ces canaux sont définis par le standard LoRaWAN, mais les autres sont dépendants du réseau. Lors d'une activation OTAA, ces paramètres sont transmis au node lors de la connexion, mais dans le cas d'ABP ceci doit être configuré manuellement. Finalement, ABP est bien plus compliqué à mettre en œuvre et surtout, moins sûr...

En quelques mots donc : faites comme moi, tenez-vous-en à l'OTAA.

2. LE MATÉRIEL UTILISÉ

Il existe énormément de matériels permettant d'utiliser LoRa et donc de connecter un montage à LoRaWAN et TTN. Une simple recherche sur eBay des termes « *lora module* » retourne des dizaines de modèles différents, avec des prix variant entre 6€ et 30€. Les écarts de prix proviennent généralement du format proposé, de la qualité générale du module et du type d'antenne utilisable (si incluse).

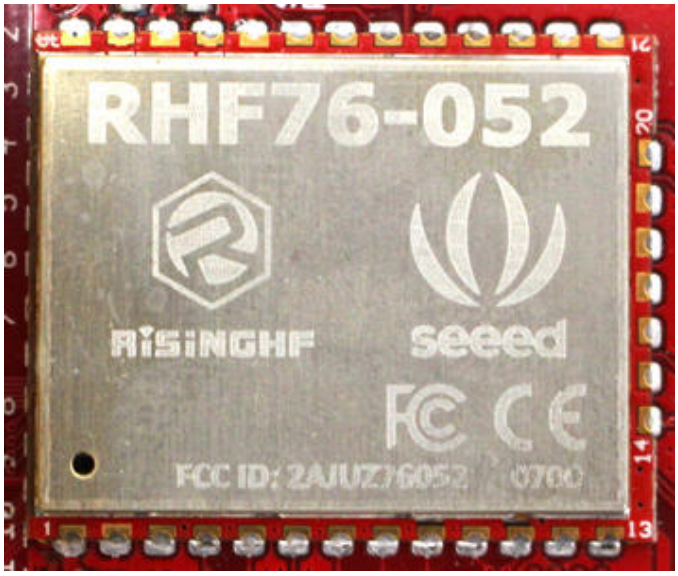
Pour mes expérimentations, j'ai opté pour une solution plus simple que l'utilisation de modules d'entrée de gamme en choisissant d'utiliser un shield Arduino (Dragino LoRa Shield) et une carte compatible Arduino intégrant une puce LoRa (Seeeduino LoRaWAN avec GPS). Les deux produits sont construits autour de la puce Semetch SX1276. Dans le cas du shield Arduino, il s'agit d'un module HopeRF RFM95W prenant place sur le circuit imprimé et pour la carte, il s'agit d'un module RHF76-052AM de RisingHF. Le module HopeRF RFM95W est le type de produit qu'on trouve généralement pour quelques euros en ligne.

Semtech est le seul fabricant de puce LoRa/LoRaWAN et quel que soit le matériel que vous allez acheter, c'est avec une telle puce que votre croquis va dialoguer pour émettre et recevoir des données. Deux principaux modèles sont généralement utilisés pour les nodes, le SX1272 et le SX1276. Fort similaires, ils se différencient en particulier par la gamme de fréquences pouvant être utilisées avec, dans le premier cas, une plage de 850 Mhz à 1 GHz et dans le second cette même plage, plus la bande des 169 Mhz et des 433 Mhz. Dans la plupart des cas, en utilisant la bande 868 Mhz (Europe), le choix entre SX1272 et SX1276 est secondaire dans un premier temps. Ce n'est qu'avec une certaine expérience et une bonne connaissance du fonctionnement du système qu'il devient important de prendre cela en considération.

L'utilisation d'un shield est une bonne solution, mais il faudra faire très attention à la carte Arduino utilisée. En effet, l'usage de LoRaWAN est généralement lié au déploiement de capteurs divers. Rappelons au passage que LoRaWAN est prévu pour une communication longue distance (en kilomètres), avec un faible débit, des données succinctes et une faible consommation énergétique, et non pour les transmissions en temps réel, la voix ou l'image, ou encore pour le contrôle à distance. Autrement dit, si vous optez pour LoRaWAN c'est surtout parce que vous voulez déployer des capteurs...

Bien entendu, qui dit capteur dit bibliothèques et code pour gérer ces capteurs. Tout ceci, en plus de la gestion LoRaWAN, du chiffrement et du format de données consomme une place non négligeable en mémoire. À titre d'exemple, un simple capteur humidité/pression/température/lumière sur base Arduino UNO avec le shield Dragino occupera, après compilation, quelques 87% des 32Ko de flash intégrés dans le microcontrôleur (voire 110% avec certaines versions des bibliothèques).

On comprend mieux alors pourquoi la carte Seeeduino LoRaWAN avec GPS est construite non pas autour d'un Atmel AVR, mais d'un Atmel ATSAMD21G18 (comme l'Arduino Zero) à cœur ARM Cortex-M0+ avec 256 Ko de flash et 32 Ko de mémoire vive. Cette carte, intégrée au kit Seeed pour LoRaWAN vaut une quarantaine d'euros (35€ en version sans GPS, le même prix qu'une Zero), mais intègre, en plus, un contrôleur de charge pour accu lipo et un connecteur dédié. Faire de cette carte un node autonome sera donc un jeu d'enfant et ne nécessitera rien d'autre qu'un accu adapté avec un connecteur JST2.0. Un dernier détail très important concernant cette carte concerne les tensions utilisées : maximum 3,3V, cette carte n'est pas tolérante au 5V !



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_rising1.jpg)

Le module utilisé par la carte Seeeduino est très différent de celui du shield. Lui aussi intègre un circuit intégré Semtech, mais également un microcontrôleur STM32 qui sert d'interface de communication. La carte Seeeduino ne dialogue pas directement avec la puce LoRa en SPI, mais avec le STM32 via une liaison série, comme un modem.

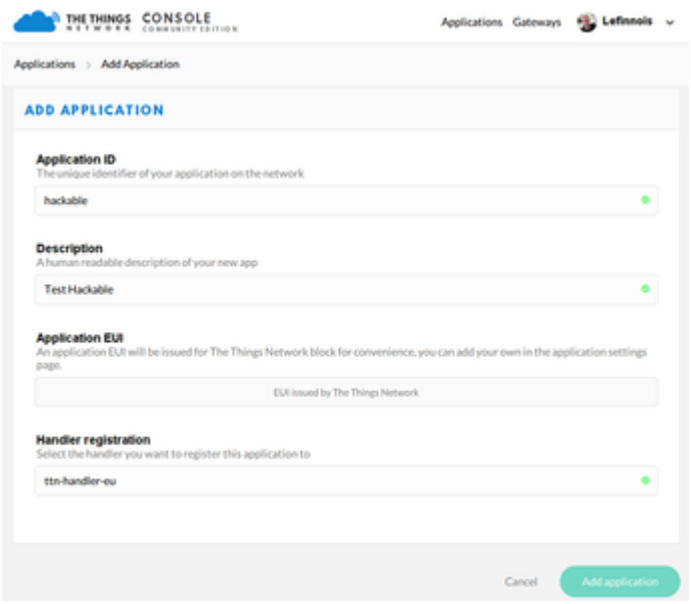
L'espace alloué pour cet article, déjà bien long, est insuffisant pour traiter à la fois du shield Dragino et de la carte Seeeduino. Cette dernière a tout de même fait l'objet d'expérimentations et vous trouverez le croquis commenté correspondant dans le dépôt GitHub du numéro (https://github.com/Hackable-magazine (https://github.com/Hackable-magazine)). Je ne cacherai pas une nette préférence pour le shield Arduino plutôt que pour la carte Seeeduino malgré la possibilité d'utiliser un accu facilement : la documentation est trop concise, la bibliothèque associée de piètre qualité et le contrôle de la communication et de la configuration du module trop limité à mon goût.

3. TTN ET ENREGISTREMENT DE L'APPLICATION ET DU NODE

Pour connecter un node au réseau TTN, vous devez bien entendu posséder un compte. Si vous avez déployé un concentrateur LoRaWAN, vous pouvez utiliser le compte déjà prévu à cet effet sans le moindre problème.

Une fois connecté à la console TTN (https://console.thethingsnetwork.org/ (https://console.thethingsnetwork.org/)), vous devez avant toutes choses créer une application. Ceci n'est pas une application dans le sens habituel du terme (comme une application Windows ou Android par exemple), mais davantage l'aspect applicatif de votre projet. C'est l'endroit où devront aboutir vos données. Nous verrons plus loin comment connecter cette facette à quelque chose qui ressemble à une application en utilisant Cayenne.

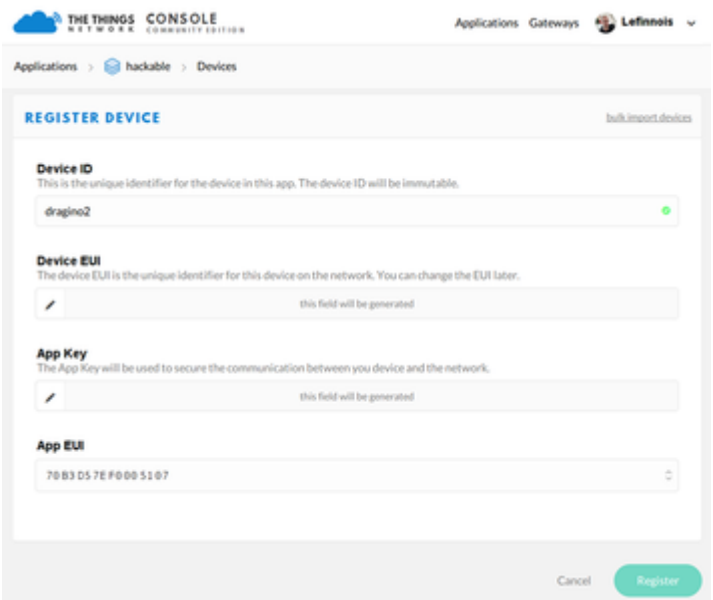
En ajoutant une application, vous serez invité à renseigner quelques éléments dont l'ID de l'application (alphanumérique minuscule uniquement, pas de « - » ou de « _ » consécutifs ou en début/fin), une description et le *handler* pour l'enregistrement (le serveur européen ici). L'identifiant unique de votre application (AppEUI) sera généré automatiquement par le réseau :



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/ttn_crea_app1.png)

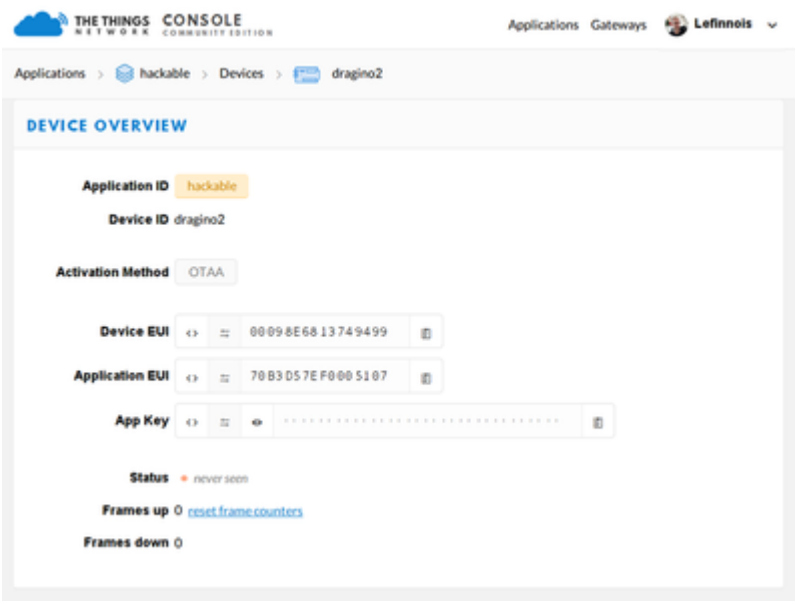
Cet identifiant unique apparaîtra dans le résumé une fois l'enregistrement validé. Vous avez maintenant le socle sur lequel va reposer votre ou vos nodes/périphériques (*devices*) et c'est dans une application que vous allez l'enregistrer. Dans le résumé de l'application, en haut à droite, vous trouverez un bouton ***Device*** vous affichant la liste des nodes gérés par l'application. Cliquez sur ***Register device*** pour en ajouter un nouveau.

Dans le formulaire, précisez un ID pour votre node (son nom répond aux mêmes impératifs typographiques que l'ID de l'application), c'est un simple nom qui n'est pas utilisé dans l'activation. Le ***Device EUI*** (DevEUI), en revanche, est un identifiant unique sur le réseau, vous pouvez le préciser vous-même ou tout simplement laisser le formulaire en choisir un à votre place en cliquant sur l'icône à gauche du champ de saisie. Un message précisera « *This field will be generated* » (« ce champ sera généré »). Cliquez ensuite sur ***Register*** et le tour est joué :



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/ttn_crea_dev1.png)

La page qui est affichée ensuite résume les informations pour ce node/périphérique :



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/ttn_vue_dev1.png)

La méthode d'activation par défaut est OTAA. Vous pouvez la changer en cliquant sur ***Settings*** en haut à droite, mais je ne couvrirai pas ABP ici comme précisé plus haut. En OTAA tout ce dont nous avons besoin pour composer notre futur croquis Arduino c'est l'identifiant unique du périphérique (DevEUI), l'identifiant unique de l'application (AppEUI) et la clé de l'application (AppKey), tout le reste sera généré lors de la connexion.

Pour vous faciliter les choses, l'interface web propose de formater ces informations de manière à les rendre directement utilisables dans un croquis en C/C++. C'est l'objet de l'icône « <> » à gauche des champs transformant, par exemple `00098E6813749499` en `{ 0x00, 0x09, 0x8E, 0x68, 0x13, 0x74, 0x94, 0x99 }`.

Notez également la seconde icône présentant une double-flèche permettant d'influer sur l'ordre des octets qui composent la donnée. En fonction des plateformes, modules et bibliothèques utilisées, l'ordre peut être différent et en cliquant sur cette icône vous pouvez donc réordonner les octets avant de les copier dans le presse-papier avec l'icône à droite du champ. Dans le cas du montage Arduino à base du shield Dragino, reposant sur la bibliothèque LMIC :

- l'AppEUI est en LSB (*Least Significant Byte*, le premier octet dans le sens de la lecture courante est celui valant le plus dans une valeur sur plusieurs octets) ;

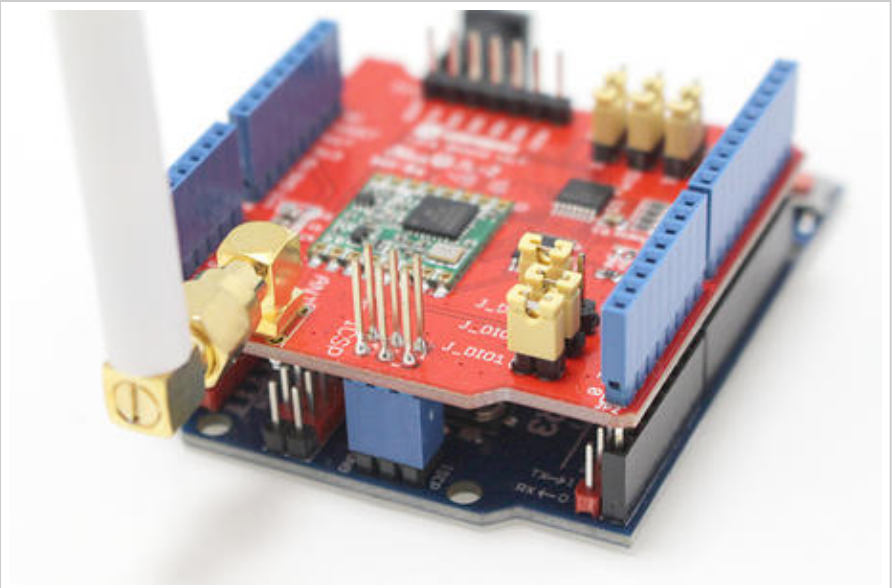
- le DevEUI est également en LSB ;

- l'AppKey est en MSB (*Most Significant Byte*), l'ordre inverse.

Il est important de bien comprendre cet ordre des octets, car dans le cas contraire, votre node tentera de se connecter à TTN avec de mauvaises informations. C'est une erreur courante et la principale source de problème des utilisateurs. Notez également au passage que tout ceci n'a rien à voir avec le concentrateur, ce n'est pas lui, mais le réseau TTN qui négocie avec votre node, le concentrateur ne fait que relayer les informations entre un média (radio LoRa) et un autre (Internet). C'est un simple répétiteur.

Une dernière étape concernant l'application concerne le choix d'un format de données. En effet, vous pouvez envoyer tout ce qui vous plaît via LoRaWAN (du moment que c'est petit et ponctuel), mais si vous comptez utiliser ces informations vous devrez vous conformer à un format de données compris par votre application. Ici, notre application (au sens « destination finale des données ») sera la plateforme Cayenne. TTN se chargera alors de renvoyer les informations à Cayenne par le biais d'un mécanisme d'intégration. Nous configurerons cela plus tard, mais devons toutefois préciser comment sont encodées les données : au format Cayenne LPP (pour *Lower Power Protocol*).

Ceci se fait très simplement en cliquant sur ***Payload Formats*** sur la page de notre application et en précisant ***Cayenne LPP*** via le menu déroulant. Après enregistrement par un clic sur ***Save***, nos données seront traitées comme étant au format en question. Ceci signifie également que notre croquis Arduino devra encoder nos données dans ce format avant de les envoyer via LoRa. Notez également que le format, comme l'intégration, concerne une application dans son ensemble et donc tous les nodes qui y sont rattachés.



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_shieldSPI1.jpg)

Le shield Dragino Lora peut être utilisé avec plusieurs modèles de cartes Arduino. Dans le cadre d'une carte UNO, la connexion SPI entre le module et la carte se fait de la même manière via les broches standards et le connecteur ICSP. Avec un autre modèle de carte, il est possible de choisir, via cavaliers, comment se fait la connexion.

4. CRÉONS NOTRE NODE ARDUINO + LORA SHIELD

Tout est en place côté TTN, il ne nous reste plus qu'à utiliser ces informations pour créer notre node. Pour accéder au module LoRa sur shield Dragino, nous aurons besoin de la bibliothèque *Arduino-LMIC*. Celle-ci est installable depuis le gestionnaire de bibliothèques, mais mieux vaudra l'installer manuellement depuis https://github.com/matthijskooijman/arduino-lmic (https://github.com/matthijskooijman/arduino-lmic), car celle proposée via l'IDE est moins à jour (et occupe davantage d'espace en flash).

Nous allons utiliser deux capteurs en guise d'exemple, interfacés en i2c :

- un module BME280 mesurant température, hygrométrie et pression atmosphérique, pris en charge par la bibliothèque éponyme de *Tyler Glenn* disponible via le gestionnaire de bibliothèques ;

- un module SI1145 mesurant les rayonnements ultraviolets et la lumière ambiante visible, pris en charge par la bibliothèque Adafruit SI1145 également installable par le gestionnaire.

Enfin, nous avons besoin d'encoder nos données au format Cayenne LPP et devrons installer, via le gestionnaire, les bibliothèques TheThingsNetwork. La majeure partie de cet élément ne nous sera pas utile puisque nous ne ferons usage que de la bibliothèque *CayenneLPP* qui s'y trouve. Toute la partie matérielle concerne en effet les modules LoRa Microchip RN2xx3 dont un modèle est intégré à la « *The Things Uno* », un dérivé d'Arduino Leonardo spécialement conçu pour et par TTN.

Notre croquis débutera donc avec une belle série d'inclusions :

```
#include <lmic.h>
#include <hal/hal.h>
#include <SPI.h>
#include <BME280I2C.h>
#include <Adafruit_SI1145.h>
#include <CayenneLPP.h>
```

On définira ensuite quelques macros :

```
// Altitude
```

```
#define ALT 209
// Calcul de la correction de pression ajustée
#define COR (ALT/8.0)
// Taille maximum des messages LPP
#define MAX_SIZE 100
```

Les deux premières nous permettent de calculer, sur la base de la pression atmosphérique lue et de l'altitude où nous nous trouvons, une pression équivalente au niveau de la mer. Comme le précise un site de Météo France, en moyenne, la pression atmosphérique diminue de 1 hPa tous les 8 mètres, nous devons donc ajouter des hectopascals pour obtenir une valeur équivalente à celles utilisées conventionnellement en météo. La seconde macro fixe la taille maximum des messages au format LPP (c'est un maximum, les messages sont bien plus petits que cela).

On peut ensuite déclarer les variables et objets utilisés dans le croquis :

```
// On déclare les capteurs
Adafruit_SI1145 uv = Adafruit_SI1145();
BME280I2C bme;
// Identifiant tâche
static osjob_t sendjob;
// Émission toutes les 5 mn
const unsigned tx_interval = 300;
```

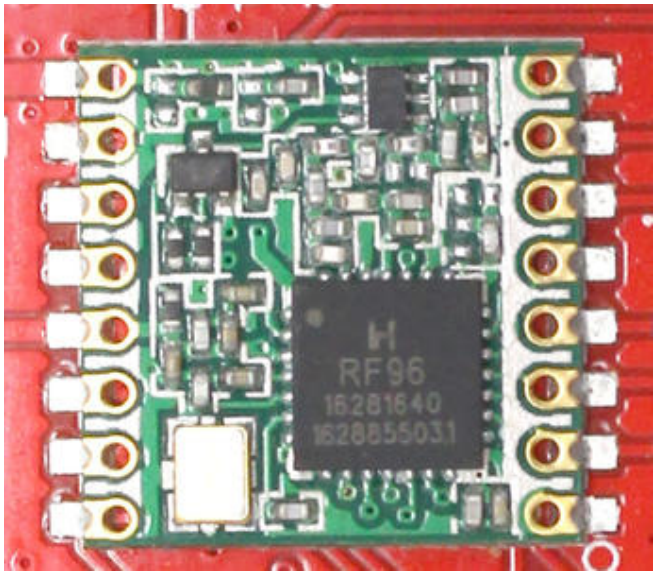
`uv` représente le module Silicon Labs SI1145 et **`bme`** le capteur Bosch BME280. **`sendjob`** représente une tâche (un *job*) devant être effectué et plus précisément la tâche consistant à envoyer notre message. La logique en œuvre dans la bibliothèque LMIC consiste à préparer une tâche d'envoi de message, accompagnée des données utiles et de fixer son exécution dans le temps. On ne pilote donc pas directement l'envoi comme on pourrait le faire avec une communication série ou autre. Enfin, **`tx_interval`** est l'intervalle en secondes entre les envois de messages, celui-ci est utilisé pour programmer la prochaine tâche une fois que la précédente est terminée.

Il faut ensuite s'occuper de la connexion physique :

```
// Brochage
const lmic_pinmap lmic_pins = {
    .nss = 10,
    .rxtx = LMIC_UNUSED_PIN,
    .rst = 9,
    .dio = {2, 6, 7},
};
```

La bibliothèque LMIC n'est pas utilisable qu'avec le shield Dragino, mais également avec n'importe quel module reposant sur une puce Semtech de la famille SX1272 ou SX1276 et donc le kit d'évaluation Semtech et les modules HopeRF RFM92 et RFM95. Il est possible de l'utiliser avec presque n'importe quel matériel, à l'exclusion des modules plus « évolués » comme ceux de Microchip communiquant sur une liaison série et intégrant un microcontrôleur en complément (comme le module RHF76-052AM équipant la Seeeduino LoRaWAN et utilisant un STM32-L0 en plus de la puce SX1276).

Une puce Semtech utilise une liaison SPI pour la communication avec votre montage, mais également des broches supplémentaires DIO0 à DIO2, une ligne de reset, une ligne CS (alias NSS) comme pour n'importe quelle liaison SPI et éventuellement une broche RXTX permettant de sélectionner le mode d'utilisation de l'antenne. Dans le cas du shield Dragino, RXTX n'est pas utilisé, CS pour sélectionner le périphérique est sur 10 et le reset est sur 9. En mode LoRa, seuls DIO0 et DIO1 sont utilisés, respectivement reliés aux broches 2 et 6. Notez que trois broches doivent être déclarées, car le module peut être utilisé pour une communication autre qu'en LoRa et que la bibliothèque LMIC (mode FSK) supporte ce mode.



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_shield_jumper1.jpg)

Les cavaliers SV2, SV4 et SV3 sur le shield Dragino permettent de choisir une connexion SPI via les broches 11, 12 et 13, ou via le connecteur ICSP. Dans le cas d'Arduino UNO, ceci n'a pas d'importance, ces broches étant reliées entre elles. Les cavaliers J_DIO5, J_DIO2 et J_DIO1 déterminent la connexion des broches DI05, DI02 et DI01 du module LoRa aux broches 8, 7 et 6 de la carte Arduino. DI00 est branché « en dur » à la broche 2.

Tout est maintenant matériellement configuré et nous pouvons enfin nous occuper des identifiants et autres informations obtenues après enregistrement de notre node sur TTN :

```
// Identifiant d'application
// Format little-endian/LSB
static const u1_t PROGMEM appeui[8] = {0x07,0x51,0x00,0xF0,0x7E,0xD5,0xB3,0x70};
void os_getArtEui (u1_t* buf) { memcpy_P(buf, appeui, 8);}

// Identifiant de node/périphérique
// Format little-endian/LSB
static const u1_t PROGMEM deveui[8] = {0x99,0x94,0x74,0x13,0x68,0x8E,0x09,0x00};
void os_getDevEui (u1_t* buf) { memcpy_P(buf, deveui, 8);}
// Clé de l'application
// Format big-endian/MSB
static const u1_t PROGMEM appkey[16] =
  {0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F };
void os_getDevKey (u1_t* buf) { memcpy_P(buf, appkey, 16);}
```

Voilà qui peut être déroutant à première vue puisqu'on déclare des fonctions plutôt que d'en utiliser. Ceci vient du fait que la bibliothèque appelle ces fonctions pour paramétrer la communication LoRaWAN, d'où la présence de « **`get`** » dans leurs noms et non « **`set`** ». Nous créons donc des fonctions qui copient ces données en mémoire à partir du contenu de la variable déclarée juste dessus. Nous configurons ainsi l'identifiant de l'application, celui du node et la clé de l'application. Tout ceci est copié/collé depuis la page d'information du node dans la console TTN (attention à l'ordre des octets !).

Une autre fonction doit être déclarée par nos soins pour assurer le fonctionnement du croquis. Celle-ci est destinée à gérer les différents événements qui peuvent survenir dans l'exécution de nos demandes par le module LoRa :

```
// Gestion des événements
void onEvent(ev_t ev) {
    Serial.print(os_getTime());
    Serial.print(": ");
    switch(ev) {
        case EV_SCAN_TIMEOUT:
            Serial.println(F("EV_SCAN_TIMEOUT")); break;
        case EV_BEACON_FOUND:
            Serial.println(F("EV_BEACON_FOUND")); break;
        case EV_BEACON_MISSED:
            Serial.println(F("EV_BEACON_MISSED")); break;
        case EV_BEACON_TRACKED:
            Serial.println(F("EV_BEACON_TRACKED")); break;
        case EV_JOINING:
            Serial.println(F("EV_JOINING")); break;
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            LMIC_setLinkCheckMode(0); break;
        case EV_RFU1:
            Serial.println(F("EV_RFU1")); break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED")); break;
        case EV_REJOIN_FAILED:
            Serial.println(F("EV_REJOIN_FAILED")); break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (avec attente RX)"));
            if(LMIC.dataLen) {
                // données reçues ?
                Serial.print(F("Data RX: "));
                Serial.write(LMIC.frame+LMIC.dataBeg, LMIC.dataLen);
                Serial.println();
            }
            // Planifier la prochaine émission
            os_setTimedCallback(&sendjob, os_getTime()+sec2osticks(tx_interval), do_send);
            break;
        case EV_LOST_TSYNC:
            Serial.println(F("EV_LOST_TSYNC")); break;
        case EV_RESET:
            Serial.println(F("EV_RESET")); break;
        case EV_RXCOMPLETE:
            Serial.println(F("EV_RXCOMPLETE")); break;
        case EV_LINK_DEAD:
            Serial.println(F("EV_LINK_DEAD")); break;
        case EV_LINK_ALIVE:
            Serial.println(F("EV_LINK_ALIVE")); break;
         default:
            Serial.println(F("inconnu")); break;
    }
}
```

Ces événements sont générés depuis l'intérieur de la bibliothèque LMIC via des appels à **`reportEvent()`**, elle-même appelant notre fonction ici présente. La plupart de ces événements ne sont traités qu'à titre d'information, pour suivre sur le moniteur série le déroulement des opérations. **`EV_TXCOMPLETE`** est particulier, il

est généré dès qu'un message a été envoyé et nous l'utilisons pour pouvoir planifier un nouvel envoi avec **`os_setTimedCallback()`**. On précise en argument la tâche dont il s'agit, le moment où elle doit être exécutée (maintenant + **`tx_interval`**) et la fonction utilisée pour générer les données du message. Chaque fin de transmission provoquera donc la création d'une nouvelle tâche, exécutée plus tard.

Ceci nous conduit donc naturellement à l'écriture de **`do_send()`**, la fonction formant la partie « active » de notre node :

```
// Notre tâche
void do_send(osjob_t* j){
  // variable pour les capteurs
  float temp, hum, pres, uvindex, lum;
  // Communication déjà en cours ?
  if (LMIC.opmode & OP_TXRXPEND) {
    Serial.println(F("OP_TXRXPEND, pas d'envoi"));
  } else {
    // lecture des capteurs
    bme.read(pres, temp, hum, true, B001);
    uvindex = uv.readUV()/100.0;
    // Attention ! Il ne s'agit pas de lux comme avec un TSL2561
    lum = uv.readVisible();
    // Affichage des valeurs
    Serial.print("uv: ");
    Serial.print(uvindex);
    Serial.print("\tlum: ");
    Serial.print(lum);
    Serial.print("\ttemp: ");
    Serial.print(temp);
    Serial.print("\thum: ");
    Serial.print(hum);
    Serial.print("\tpress: ");
    Serial.println(pres+COR);
    // Composition du message
    lpp.reset();
    lpp.addTemperature(0, temp);
    lpp.addRelativeHumidity(1, hum);
    lpp.addBarometricPressure(2, pres+COR);
    lpp.addLuminosity(3,(int)uvindex);
    lpp.addLuminosity(4,(int)lum);
    // Enregistrement du message à envoyer
    LMIC_setTxData2(1, lpp.getBuffer(), lpp.getSize(), 0);
    Serial.println(F("Packet queued"));
  }
}
```

Pour relever les informations pertinentes, nous utilisons les méthodes de chaque bibliothèque prenant en charge les capteurs. Le point important est cependant tout autre : la construction de notre message au format Cayenne LPP. Nous commençons par réinitialiser notre variable **`lpp`** avec la méthode **`reset()`** afin de partir sur un message vide. Nous ajoutons ensuite, élément par élément, les données collectées en spécifiant un canal en premier argument (celui-ci sera utilisé plus tard sur Cayenne).

Enfin, nous utilisons **`LMIC_setTxData2()`** pour définir les données à envoyer en précisant où elles se trouvent (**`lpp.getBuffer()`** retourne l'adresse) et leur taille. Comprenez bien que ceci ne provoque pas l'envoi, nous ne faisons que « charger » le contenu du futur message, un peu comme un boulet dans un canon.

À présent que les macros, variables, objets et fonctions de notre croquis sont en place, nous pouvons nous occuper de **`setup()`**

```
// configuration
void setup() {
```

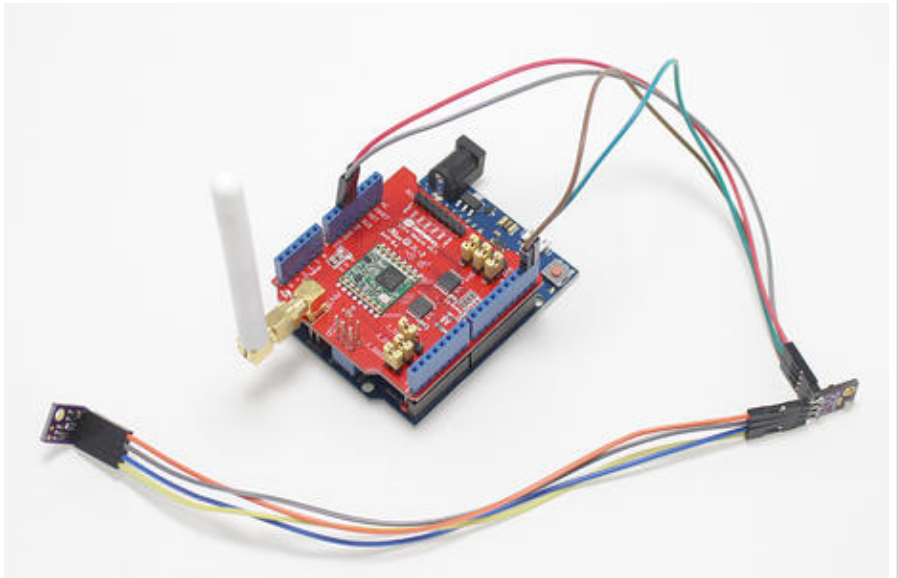
```
  Serial.begin(115200);
  Serial.println(F("Go Go Go !"));
  // initialisation capteurs
  while (!bme.begin()) {
    Serial.println("Erreur BME280! !");
    delay(2000);
  }
  while (!uv.begin()) {
    Serial.println("Erreur SI1145 !");
    delay(2000);
  }
  // LMIC init
  os_init();
  LMIC_reset();
  // Démarrage tâche
  do_send(&sendjob);
}
```

La partie intéressante est celle se situant, bien entendu, après l'initialisation des capteurs. On initialise le support LoRa/LoRaWAN avec **`os_init()`** qui lui-même, dans la bibliothèque initialise tous les éléments de la communication (mémoire, matériel, radio), puis on déclenche un reset pour supprimer toutes données en attente de traitement et stopper toutes actions en cours. Enfin, comme l'envoi répétitif de messages se base sur une tâche planifiée par la fin d'un précédent envoi, il est nécessaire d'appeler une première fois **`do_send()`**.

Le reste du croquis tient en une simple fonction **`loop()`** se contentant d'appeler constamment la fonction de gestion générale :

```
void loop() {
  // boucle de gestion
  os_runloop_once();
}
```

Une fois le croquis programmé dans la carte, son exécution devrait rapidement découler à l'arrivée d'un premier message sur TTN. Dans le moniteur série, « **`EV_JOINING`** » devrait apparaître, puis « **`EV_JOINED`** » et enfin « **`Packet queued`** », puis « **`EV_TXCOMPLETE`** ». En vous rendant sur la console TTN, sur la page de votre node (*device*), un clic sur ***Data*** devrait alors montrer ce premier message, puis des suivants (ceux-ci apparaissent également dans ***Data*** de l'application). Il en va de même, si vous avez et utilisez votre propre concentrateur, à la différence que le contenu n'est pas lisible, car chiffré.



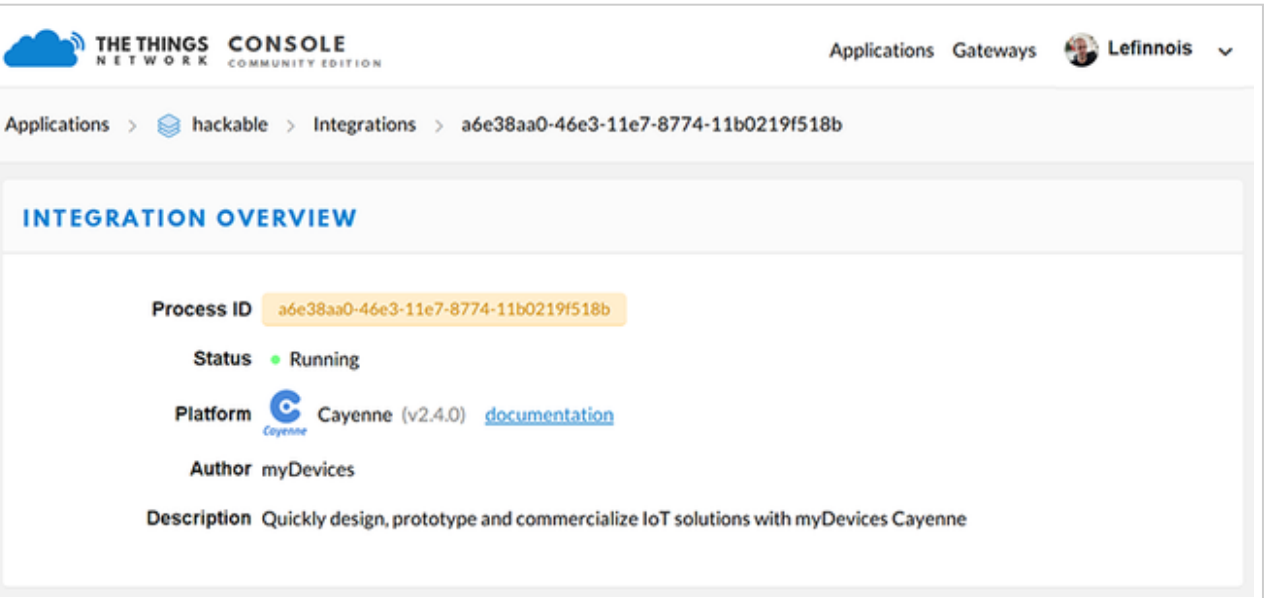
(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_montage1.jpg)

Notre montage de test est relativement simple et utilise deux modules capteur en i2c. Le premier relève température, humidité relative et pression atmosphérique et le second l'indice UV et une indication de l'intensité de lumière visible. Ce sont ces informations que nous allons envoyer périodiquement à notre application via LoRaWAN et Internet.

5. LES DONNÉES SONT LÀ, FAISONS-EN QUELQUE CHOSE

Sur la page de l'application et du périphérique côté console TTN, nous pouvons voir les données décodées et en clair. Sur ce point, il faut bien comprendre qu'une partie de la console TTN présente également le côté « Application » d'un réseau LoRaWAN et de ce fait, les données sont déchiffrées. Ceci vous permettra de valider le contenu de vos messages, comme ici, en lisant directement les valeurs présentes dans les données LPP.

Vous conviendrez cependant avec moi que ceci n'est pas très attrayant pour le commun des mortels. Pour réellement impressionner vos amis ou vos collèges, il est préférable de présenter ces informations de façon un peu plus évidente en utilisant une intégration. En vous rendant sur la page de votre application dans la console TTN, cliquez sur ***Integration*** et ***Add integration***. Là vous aurez le choix entre plusieurs options vous permettant de « faire quelque chose » avec les données des messages. À cette date, quatre options sont possibles et nous allons utiliser la plus simple : Cayenne.



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/ttn_integ_cayenne1.png)

Le lien entre TTN et Cayenne se fait par le biais d'une « intégration » (au sens mathématique du terme), un joli mot pour dire que les données sont « intégrées » ou accumulées quelque part.

Cayenne MyDevice est un site et une application Android/iOS permettant de centraliser la gestion des objets connectés. Vous pouvez y afficher et stocker des données de capteurs, piloter des montages connectés et analyser leur fonctionnement. Cayenne n'est pas spécifique à LoRaWAN, mais permet une gestion d'objets connectés de plusieurs façons (Wifi, Ethernet, 3G, etc.). LoRaWAN avec TTN n'est qu'un moyen parmi plusieurs. Notez également que Cayenne n'est pas le seul site à proposer ce genre de choses, mais il est fort pratique pour obtenir un premier résultat présentable.

Vous devrez commencer par créer gratuitement un compte sur https://cayenne.mydevices.com (https://cayenne.mydevices.com) et ce faisant vous arriverez directement sur une page vous permettant de sélectionner un périphérique pour créer un projet. Choisissez alors ***LoRa*** (en bêta actuellement), ***The Things Network*** (à gauche) et dans la liste à droite choisissez ***Cayenne LPP*** puis renseignez les informations utiles :

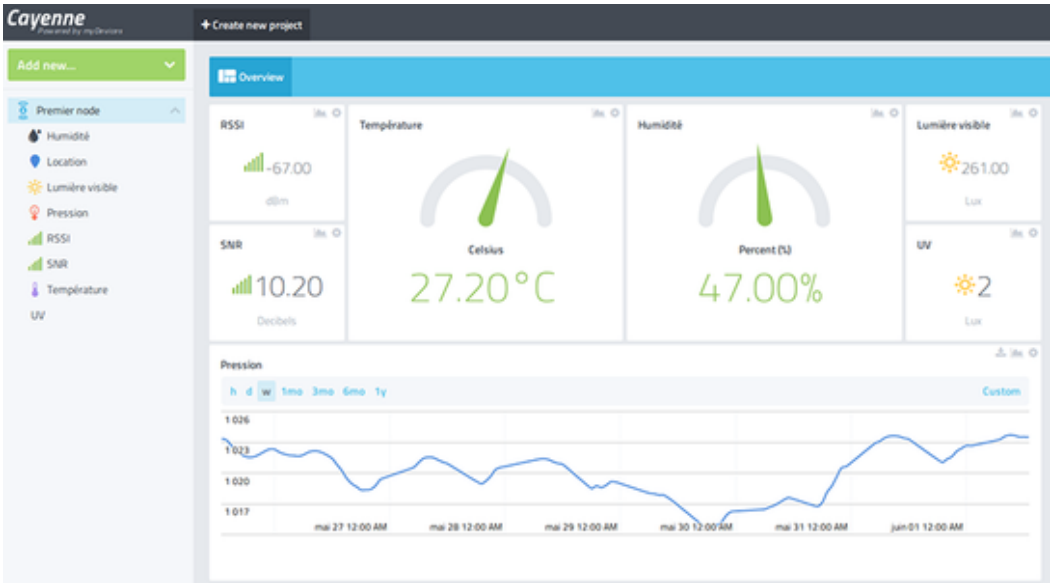
- ***Name*** : le nom du périphérique/node ;

- ***DevEUI*** : l'identifiant unique du node, à copier/coller depuis la console TTT ;

- ***Activation Mode*** : le mode d'activation du node, ici seul ***Already Registered*** (déjà enregistré) est utilisable, ça tombe bien parce que c'est le cas ;

- ***Location*** : choisissez ***This device doesn't move*** et spécifiez une adresse pour le placer sur la carte OpenStreetMap.

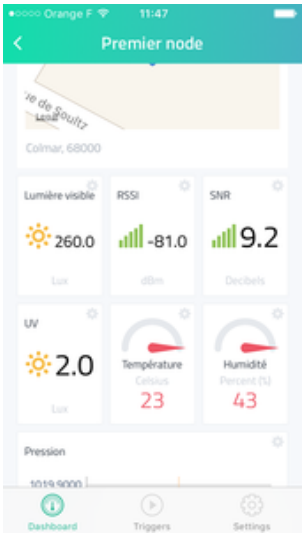
Valider le formulaire et le *Dashboard* devrait apparaître vous présentant une carte avec votre node. Revenez ensuite sur la console et choisissez une intégration avec Cayenne. Le « *Process ID* » doit être la longue chaîne de caractères se trouvant dans l'URL du dashboard Cayenne, juste après « /lora/ », vous devez copier/coller ce texte puis choisir ***Default key*** dans ***Access Key*** avant de valider. Ceci fait, TTN commencera automatiquement à relayer les données des messages de votre node à Cayenne et les informations devraient alors apparaître dans votre dashboard.



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/cayenne_node1.png)

Dès que les informations de votre premier node arrivent sur Cayenne via TTN et un concentrateur LoRaWAN, vous devrez les voir apparaître dans votre dashbboard. Vous pourrez alors configurer la manière dont ces informations doivent être présentées...

Ce que vous voyez apparaître dans votre navigateur ce sont les mesures faites par votre node, envoyées via les airs en LoRa à un concentrateur LoRaWAN, transitant par TTN pour finalement arriver sur Cayenne. Si vous installez l'application Android ou iOS, vous verrez les mêmes informations, présentées peu ou prou de la même façon...



(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/CayenneIOS1.png)

Cayenne n'est pas spécifique à LoRaWAN, à TTN ou même au Web. C'est une solution prenant également la forme d'une application Android ou iOS (iPhone) destinée à représenter les informations de plusieurs types de capteurs communiquant de différentes façons.

6. LIMITATIONS, USAGE RAISONNÉ ET SERVEUR PRIVÉ

Nous avons couvert ici la base de l'utilisation de LoRaWAN et avec l'article précédent (et un certain budget) vous devriez être en mesure de déployer vos nodes avec ou sans concentrateur déjà en place dans votre voisinage. TTN n'est pas le seul fournisseur existant, mais il est à mon sens le plus intéressant, car réellement communautaire.

Mais avec cette notion de partage des concentrateurs et cette liberté d'utilisation vient également l'importance d'un comportement raisonnable. Le fonctionnement même de LoRaWAN, les canaux, le débit, le rapport cyclique pour chaque canal et le nombre de nodes supportés par chaque concentrateur, implique une politique de partage claire et stricte. L'élément de base pour cet usage raisonné est le « temps dans l'air » (*air-time*), autrement dit le temps total de communication radio entre un node et un concentrateur. La règle de base est de ne pas dépasser 30 secondes sur une période de 24h.

Le calcul de ce temps quotidien n'est pas simple puisqu'il dépend de la taille des messages et de la vitesse de transmission lui même dépendant de la qualité du signal et donc de la distance entre le node et le concentrateur. Le débit est déterminé par la notion de facteur d'étalement ou *spreading factor* en anglais. Noté de SF7 à SF12, plus le facteur d'étalement est important, plus la distance peut être grande, mais plus la communication est lente, et donc de ce fait, le temps « dans l'air » est important. À titre d'exemple avec un message de 10 octets et un facteur d'étalement SF12, la limite d'utilisation raisonnée est de 20 messages par jour, mais sera de 500 en SF7.

(https://connect.ed-diamond.com/sites/default/files/articles/hackable/hk-019/83081/loranode_hat1.jpg)

Dragino propose également un hat pour Raspberry Pi utilisant le même module LoRa, accompagné d'un module GPS complémentaire (livré sans antenne). Ce type de carte additionnelle permet de créer un node, mais également un pseudo-concentrateur LoRaWAN à un seul canal (qui ne sera donc pas à proprement parler un concentrateur compatible).

La communication *downlink* (du concentrateur vers un node) est encode plus réduite puisque la limite est fixée à 10 messages par jour. Tout ceci permet d'assurer le support de 1000 nodes par concentrateur et un fonctionnement optimal du réseau. Si cela ne vous convient pas, rien ne vous empêche de vous passer de TTN et de faire fonctionner votre propre serveur réseau, TTN met à disposition les sources et les documentations utiles pour cela. Mais vous perdrez alors l'idée même derrière LoRaWAN et la construction d'un réseau complet communautaire, alors que le but est justement de permettre à des utilisateurs ne pouvant pas installer un concentrateur de bénéficier de LoRaWAN dans leur zone.

Côté application, nous avons vu qu'il était possible très simplement d'utiliser l'intégration pour présenter les données dans une interface web ou une application mobile avec Cayenne. Là encore, de nombreuses alternatives existent aussi bien sous la forme de services qu'avec des solutions « maison » via une intégration HTTP (les données sont envoyées avec des requêtes `GET` ou `POST` à un serveur web).

Enfin, un point que je n'ai pas du tout abordé, car l'idée était ici de ne se pencher que sur LoRaWAN, est le fait de tout simplement communiquer entre deux points en utilisant LoRA. Un exemple de ce type d'usage est disponible avec la bibliothèque LMIC et pourra avantageusement remplacer une autre forme de communication comme l'utilisation de module nRF24L01.

LoRa et LoRaWAN sont deux technologies qui gagnent rapidement du terrain et se popularisent à vitesse grand V, tout autant que TTN partout dans le monde. Au point qu'il est tout à fait raisonnable de penser que ceci deviendra le standard de fait avec une couverture importante d'ici peu de temps. Vous êtes, comme moi, aux premières loges et avez désormais les bases pour vous lancer...

Étiquettes : LoRA (/search/node?field_ct_article_tags_target_id_selective%5B0%5D=72487)

< Article précédent (/Hackable/HK-019/Decouvrez-LoRaWAN-et-creez-votre-passerelle-concentrateur)
Article suivant > (/Hackable/HK-019/Construisez-un-emetteur-433-Mhz-pour-remplacer-vos-telecommandes)

AU SOMMAIRE DU MÊME NUMÉRO

SUR LE MÊME SUJET

Maîtriser la sécurité de son accès Internet avec OpenWRT (/Linux-Pratique/LP-117/Maitriser-la-securite-de-son-acces-Internet-avec-OpenWRT)

Linux Pratique n° 117 (/Linux-Pratique/LP-117) | janvier 2020 | Kerma Gérald (/auteur/Kerma-Gerald)

Réseau (/search/node?Domaines%5B0%5D=72450) **Sécurité (/search/node?Domaines%5B0%5D=72467)** **Embarqué (/search/node?Domaines%5B0%5D=72457)**

Nous allons voir dans cet article comment installer le système GNU/Linux embarqué de type OpenWRT. OpenWRT est un système GNU/Linux pour les matériels embarqués et pour les matériels de types routeurs et box. Il constitue donc la part essentielle pour se libérer des GAFA. C'est un système léger, rapide et performant pour administrer et contrôler son accès Internet. Ce système est optimisé pour la gestion des ressources et ...

Capteur de glucose connecté : comment ça marche ? (/MISC/MISC-106/Capteur-de-glucose-connecte-comment-ca-marche)

MISC n° 106 (/MISC/MISC-106) | novembre 2019 | Apvrille Axelle (/auteur/Apvrille-Axelle)

Sécurité (/search/node?Domaines%5B0%5D=72467) **Embarqué (/search/node?Domaines%5B0%5D=72457)**

L'IoT est souvent vue comme un ensemble de gadgets sympathiques, mais pas forcément utiles (ni bien sécurisés). Qu'en est-il de l'IoT médicale, comme ce capteur de glucose connecté ? ...

La liberté jusqu'au cœur du processeur avec RISC-V (/Hackable/HK-031/La-liberte-jusqu-au-coeur-du-processeur-avec-RISC-V)

Hackable n° 31 (/Hackable/HK-031) | octobre 2019 | Marteau Fabien (/auteur/Marteau-Fabien)

Électronique (/search/node?Domaines%5B0%5D=72452) **Embarqué (/search/node?Domaines%5B0%5D=72457)**

RISC-V est un jeu d'instructions 32 bits libre, développé initialement par l'université de Berkeley. Ce jeu d'instructions (ISA pour Instruction Set Architecture) est maintenant soutenu par une fondation regroupant quasiment tous les grands noms de l'industrie informatique. Dans cet article, nous allons décrire succinctement le concept de RISC vs CISC, puis nous expliquerons les bases du jeu d'instructions avec un peu de code ...

Petites antennes réalisées par impression additive : de la conception à la visualisation des diagrammes de rayonnement (en vrai et en virtuel) (/Hackable/HK-031/Petites-antennes-realisees-par-impression-additive-de-la-conception-a-la-visualisation-des-diagrammes-de-rayonnement-en-vrai-et-en-virtuel)

Hackable n° 31 (/Hackable/HK-031) | octobre 2019 |

Friedt Jean-Michel (/auteur/Friedt-Jean-Michel)- - Carry Emile (/auteur/Carry-Emile)- - Testault Olivier (/auteur/Testault-Olivier)

Radio et wireless (/search/node?Domaines%5B0%5D=72465)

Les antennes de petites dimensions sont un sujet qui a toujours été à la mode auprès des ingénieurs, désireux de faire rayonner un signal électromagnétique par un conducteur de dimensions aussi réduites que possible (penser « faire tenir une antenne dans un téléphone portable »). Le problème a été abordé très tôt, alors que les émissions sub-MHz, donc avec des longueurs d'onde de plusieurs kilomètres, étaient courantes [1]. Alors ...

Analysez et décodez les messages d'un collier de « dressage » (/Hackable/HK-030/Analysez-et-decodez-les-messages-d-un-collier-de-dressage)

Hackable n° 30 (/Hackable/HK-030) | juillet 2019 | Bodor Denis (/auteur/Bodor-Denis)

Code (/search/node?Domaines%5B0%5D=72464) **Radio et wireless (/search/node?Domaines%5B0%5D=72465)**

Certaines personnes inventent tout et n'importe quoi, d'autres achètent tout et n'importe quoi et enfin, d'autres encore font tout et n'importe quoi. Dans la catégorie « au top sur tous les tableaux » (sarcasme), je vous présente le collier de dressage canin, censé permettre aux propriétaires de chiens de soi-disant les éduquer, sans lever leurs grosses fesses du canapé et sans réellement chercher à apprendre comment on ...

Démarrez avec MicroPython (/GNU-Linux-Magazine/GLMF-228/Demarrez-avec-MicroPython)

GNU/Linux Magazine n° 228 (/GNU-Linux-Magazine/GLMF-228) | juillet 2019 | Lonkeng Toulepi Stéphane (/auteur/Lonkeng-Toulepi-Stephane)

Embarqué (/search/node?Domaines%5B0%5D=72457)

Pour mettre sur pied une preuve de concept ou un prototype électronique, il faut habituellement choisir une carte de développement et ensuite acquérir les outils de développement logiciel du fabricant. Ces logiciels s'appuient généralement sur les langages bas niveau comme le C, dont la maîtrise n'est pas accessible à tout le monde. MicroPython est a été mis sur pied pour permettre l'utilisation d'un langage de programmation haut ...

PAR LE MÊME AUTEUR

MODULE INTERFACE I2C POUR ÉCRAN LCD (/HACKABLE/HK-032/MODULE-INTERFACE-I2C-POUR-ECRAN-LCD)

Hackable n° 32 (/Hackable/HK-032) | janvier 2020 | Bodor Denis (/auteur/Bodor-Denis)

Électronique (/search/node?Domaines%5B0%5D=72452)

Les afficheurs LCD alphanumériques disposant d'une interface compatible HD44780 (composant Hitachi à l'origine) se pilotent tous de la même façon et peuvent avoir différentes caractéristiques et tailles : une ligne de 8 caractères, quatre lignes de 20 caractères, deux lignes de 16 caractères, etc., tantôt avec rétroéclairage, tantôt sans. ...

VITE FAIT : CRÉER UN THERMOSTAT D'AMBIANCE PROGRAMMABLE (/HACKABLE/HK-032/VITE-FAIT-CREER-UN-THERMOSTAT-D-AMBIANCE-PROGRAMMABLE)

Hackable n° 32 (/Hackable/HK-032) | janvier 2020 | Bodor Denis (/auteur/Bodor-Denis)

Domotique (/search/node?Domaines%5B0%5D=72455)

Dans ma nouvelle maison, j'ai découvert les joies du chauffage au fioul et les limitations d'un système de régulation de la température intérieure le plus simpliste qui soit. La simplicité a ses avantages, et le fioul aussi, mais lorsqu'on regarde sa facture, on se rend rapidement compte que cette simplicité a un coût, qui peut être important. Pour régler le problème, j'ai décidé de faire rapidement évoluer mon installation, avec l'aide d'une ...

ÉDITO (/HACKABLE/HK-032/EDITO)

Hackable n° 32 (/Hackable/HK-032) | janvier 2020 | Bodor Denis (/auteur/Bodor-Denis)

De Noël à mars... Avez-vous remarqué qu'il y a un climat de crise énergétique ambiant en ce moment ? Tout devient économe en énergie, les ampoules à filament ont laissé place aux leds dans les rayons des magasins et petit à petit dans les rues, de manière générale les choses énergivores sont devenues le mal incarné car, comme dirait la pub, « ce n'est pas Versailles ici ». ...

ARDUINO MKR VIDOR 4000 : UN ARDUINO PAS COMME LES AUTRES (/HACKABLE/HK-031/ARDUINO-MKR-VIDOR-4000-UN-ARDUINO-PAS-COMME-LES-AUTRES)

Hackable n° 31 (/Hackable/HK-031) | octobre 2019 | Bodor Denis (/auteur/Bodor-Denis)

Électronique (/search/node?Domaines%5B0%5D=72452)

Les microcontrôleurs existaient bien avant l'arrivée des cartes Arduino. La révolution provoquée par l'initiative italienne n'a pas été technologique, mais pédagogique. C'est le fait de totalement démocratiser et de rendre accessible le développement sur microcontrôleurs qui a mis le mot « Arduino » sur toutes les lèvres. Aujourd'hui, Arduino récidive avec un autre domaine de l'électronique numérique et s'attaque à une ...

ÉDITO (/HACKABLE/HK-031/EDITO)

Hackable n° 31 (/Hackable/HK-031) | octobre 2019 | Bodor Denis (/auteur/Bodor-Denis)

Dans la vie, il y a ceux qui savent être « open » et ceux qui ne savent pas. ...

CONVERTISSEUR HDMI VERS VGA (/HACKABLE/HK-031/CONVERTISSEUR-HDMI-VERS-VGA)

Hackable n° 31 (/Hackable/HK-031) | octobre 2019 | Bodor Denis (/auteur/Bodor-Denis)

Électronique (/search/node?Domaines%5B0%5D=72452) **Audio/Vidéo (/search/node?Domaines%5B0%5D=72453)**

L'intérêt premier d'un convertisseur HDMI vers VGA est de permettre l'utilisation de matériels plus anciens avec des cartes et équipements modernes. Le cas typique ici est, bien entendu, de recycler un vieil écran VGA comme moniteur pour une carte Raspberry Pi. Ceci ne sera pas nécessairement très intéressant pour une utilisation « desktop », mais sera parfaitement viable pour le jeu, l'émulation, la création d'un media center ou ...

1 (?page=0%2C0) 2 (?page=0%2C1) 3 (?page=0%2C2) 4 (?page=0%2C3) 5 (?page=0%2C4)
6 (?page=0%2C5) 7 (?page=0%2C6) 8 (?page=0%2C7) 9 (?page=0%2C8) Suivant › (?page=0%2C1)

()