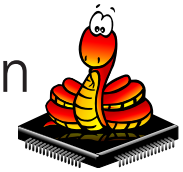
LoRa avec le Raspberry Pi Pico

S'amuser avec MicroPython



Mathias Claußen (Elektor)

Se servir de MicroPython sur un Raspberry Pi Pico est une solution sympathique pour s'initier à la programmation. En y ajoutant un module LoRa RFM95 de SeeedStudio et un capteur de température DS18B20, on peut créer en un tournemain un nœud LoRaWAN avec MicroPython. Et comme le câblage de composants sur des plaques d'essai peut manquer de fiabilité, nous vous proposons un circuit imprimé.

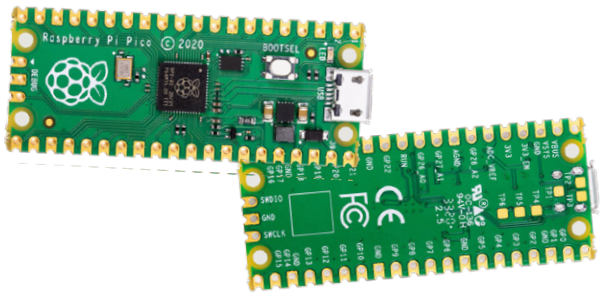


Figure 1. Le Raspberry Pi Pico.

Figure 2. Module RFM95.

Le Raspberry Pi Pico (**fig. 1**) est une nouvelle carte à microcontrôleur, équipée du RP2040. Cette puce est conçue et développée en interne par la Fondation Raspberry Pi et il s'agit donc de leur premier silicium. Après tout le battage dont elle a fait l'objet – sans oublier les doutes et la fascination qu'elle suscite – je me suis dit qu'il était temps de lancer quelques projets. La carte ne dispose pas du Wi-Fi, mais peut fonctionner avec un jeu de piles pour des applications portables, alors pourquoi ne pas combiner le RPi Pico avec un modem LoRa ? Le RFM95 de HOPERF (**fig. 2**) est un module bon marché et largement répandu qui a été utilisé dans de précédents projets d'Elektor, comme « LoRaWAN : décollage facile » [1]. Alors que dans cet article, nous utilisions un module STM32 BluePill, cette fois-ci nous avons retenu le RPi Pico.

Nous construirons un dispositif simple de mesure de la température avec un capteur DS18B20, qui enverra les données à une passerelle LoRaWAN qui les transmettra au nuage du réseau *The Things Network*. De là, nous récupèrerons les données avec un RPi (classique) grâce à la plateforme domotique Node-RED.

Nous réaliserons le nœud LoRa de mesure de la température sur une plaque d'essai, mais si vous préférez un circuit imprimé, cet article vous en fournira une ébauche, réalisée avec KiCad.

La **figure 3** montre le flux de données entre le RPi Pico et le serveur de *The Things Network*, via le réseau LoRaWAN. Vous pouvez voir qu'une passerelle se trouve au milieu pour traduire les données hertziennes en quelque chose qui peut être acheminé sur l'internet vers *The Things Network*. Comme nous sommes intéressés par la température transmise, nous devons traiter les données déposées dans *The Things Network*. Pour ce faire, nous allons les récupérer et les présenter sur une petite page web. La **figure 4** montre le flux de données de *The Things Network* vers un RPi qui créera une page web avec la température transmise actuelle ainsi qu'un graphique montrant les dernières valeurs reçues.

Le logiciel Node-RED se chargera de récupérer les données du réseau *The Things Network* et de produire une page web. C'est un outil qui permet de construire sous forme graphique des flux de données et de les traiter. Cela peut se faire simplement avec le navigateur de votre choix – il vous suffit d'installer le serveur Node-RED et d'ouvrir un navigateur

pour commencer. Node-RED a déjà été utilisé dans des projets d'Elektor et permet d'accéder rapidement au traitement et à la manipulation de données. Un livre d'initiation à Node-RED est disponible dans la boutique d'Elektor (voir **Produits** pour plus d'informations).

Les ingrédients de ce projet sont simples et figurent dans l'encadré **Matériel requis**. Assurez-vous également que vous êtes à portée d'une passerelle LoRaWAN qui acheminera vos données vers *The Things Network*. Si vous n'en avez pas à proximité et que vous souhaitez démarrer avec votre propre passerelle pour avoir une meilleure couverture LoRaWAN, jetez un œil à [3]. Il s'agit d'une solution tout-en-un que vous pouvez acheter dans le commerce. Vous pouvez également construire votre propre passerelle basée sur un RPi avec [4].

D'abord la plaque d'essai

Pour mettre au point notre montage, il sera d'abord réalisé sur une plaque d'essai. Cela permet d'effectuer des tests rapides et, comme vous pouvez le voir, avec un câblage simple (fig. 5). Pour connecter le modem LoRa au RPi Pico, nous avons besoin de quatre connexions pour l'interface SPI de base : MOSI, MISO, SCK et CS (*data in*, *data out*, *clock* et *chip select*). Cela permettra l'échange de données avec le modem LoRa. RESET et DIO0 sont nécessaires en plus pour commander le modem. Les broches DIO1 à DIO3 sont connectées et requises par certaines autres bibliothèques LoRa, comme la bibliothèque LMIC (si nous programmons le RPi Pico en C/C++). Nous pouvons choisir l'un des ports SPI dont dispose le RPi Pico et le connecter au modem LoRa RFM95. Il suffit également de connecter les broches RESET et DIO0 du module à l'une des broches GPIO du Pico. Pour obtenir un modem LoRa pleinement opérationnel, il faut également fixer une antenne.

Pour l'antenne, nous utiliserons un simple fil. Un morceau de fil de cuivre de 1 mm de diamètre suffit parfaitement. Pour calculer la longueur de l'antenne quart d'onde pour 868 MHz (gamme dans laquelle le module LoRa fonctionne), on a la formule : $\lambda/4 = (c_0/868\ \text{MHz})/4 = (299.792.458\ \text{m/s})/(868.000.000\ 1/\text{s}*4) = 0{,}08635\ \text{m} = 8{,}635\ \text{cm}$. Cette longueur s'applique pour la propagation dans le vide, mais dans le cuivre la vitesse de l'onde est plus faible, ce qui implique un facteur de raccourcissement. Avec un coefficient de 0,95, on obtient finalement une longueur approximative de 8,2 cm pour le fil de cuivre.

Bien que facultative, la résistance de rappel sur la ligne de réinitialisation s'est avérée nécessaire dans le passé. L'ajouter permet d'avoir un fonctionnement plus stable, car le RESET peut capter du bruit par les fils volants et réinitialiser le module de manière inattendue. Si vous travaillez sur votre propre carte LoRa, il peut être plus sûr de la prévoir, sans la câbler, quitte à la mettre plus tard à la main sur votre carte.

Pour le capteur de température, on utilisera une connexion One-Wire. Comme le nom One-Wire l'indique, une seule broche GPIO suffit pour l'échange de données. Le protocole peut être implémenté de façon purement logicielle, donc on peut utiliser n'importe quelle broche GPIO. Le protocole One-Wire nécessite la présence d'une résistance de rappel. Selon l'UC utilisée, il peut s'agir d'une résistance interne,

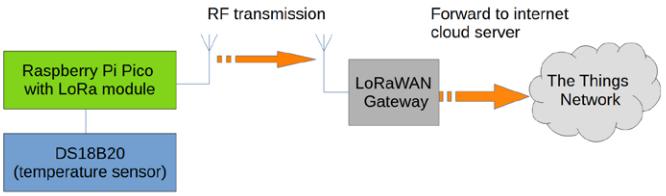


Figure 3. Flux de données vers The Thing Network.

Figure 4. Flux de données entre The Thing Network et la page web de l'utilisateur.

Matérie requis

- RFM95*
- Jeu de plaques d'essai*
- 9 fils de liaison mâle-mâle
- Carte BoB pour RFM95
- Connecteur à 2×20 broches
- Connecteur à 2×8 broches
- Capteur de température DS18B20
- Fil de cuivre de 10 cm (diamètre 1 mm

Les pièces marquées d'un * peuvent être commandées dans la boutique d'Elektor, voir **Produits**.

Pour la carte d'extension (BoB), des fichiers Gerber sont fournis [15] pour votre fabricant de circuits imprimés favori

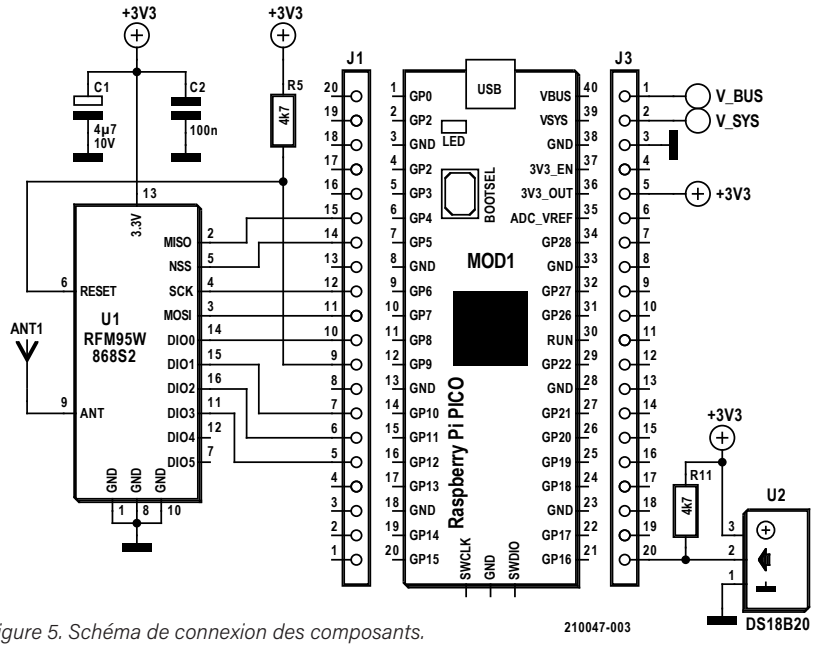


Figure 5. Schéma de connexion des composants.

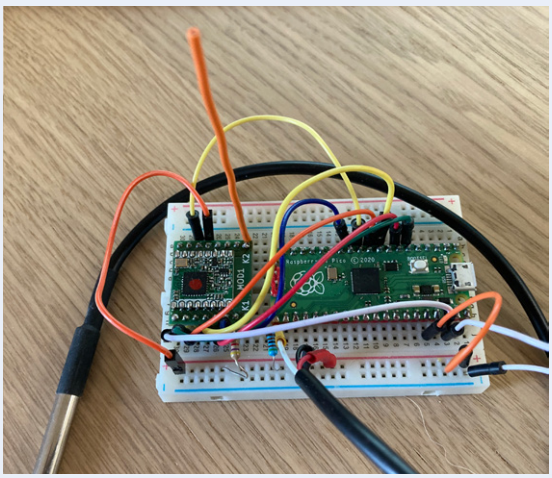


Figure 6. Installation sur une plaque d'essai.

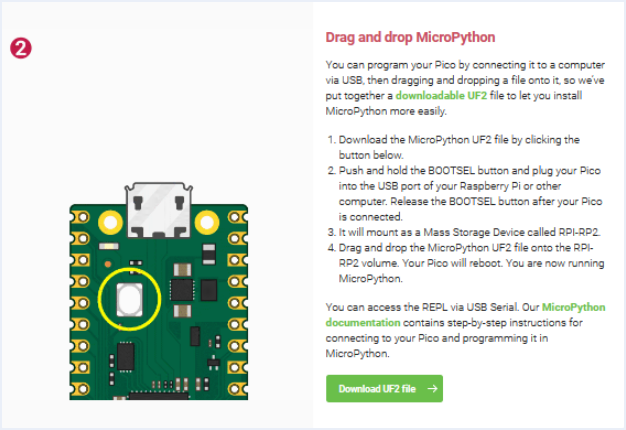


Figure 7. Téléchargement de MicroPython pour le Raspberry Pi Pico.

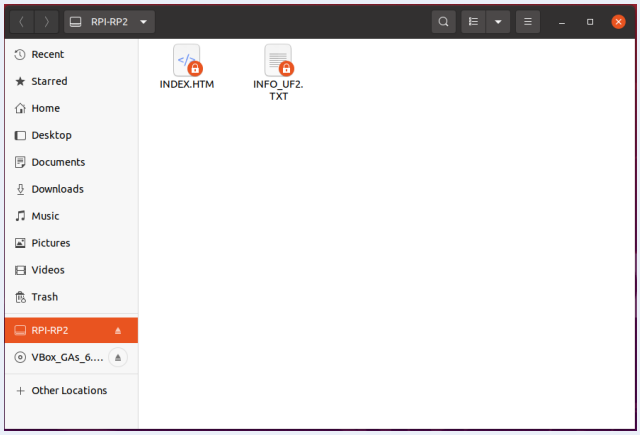


Figure 8. Raspberry Pi Pico en mode bootloader.

Figure 9. Thonny installé en version 3.3.4.

fournie par le bloc GPIO de votre UC, ou d'une résistance externe. Pour avoir un transfert de données fiable, une résistance de rappel externe est préférable.
Une fois tout le matériel installé, cela ressemblera à ce que vous voyez sur la **figure 6**, passons au codage.

MicroPython et LoRa

Nous avons déjà programmé en C/C++ pour d'autres projets LoRa, cette fois nous choisissons MicroPython. Le RPi Pico est une carte bien adaptée à l'enseignement, et l'utilisation de MicroPython facilitera encore un peu plus les choses. Si vous ne connaissez pas MicroPython sur le RPi Pico, consultez le livre de Gareth Halfacree et Ben Everard, *Get Started with MicroPython on Raspberry Pi Pico* (Raspberry Pi Foundation, 2021). Si vous souhaitez obtenir une copie papier, rendez-vous sur la boutique d'Elektor [5]. Si vous préférez un livre électronique, vous pouvez obtenir une copie gratuite en anglais sur [6].
La combinaison de LoRa et MicroPython a été réalisée sur d'autres plateformes, mais cela présente certaines difficultés. MicroPython est un langage interprété comme l'était le BASIC sur le C64 ou d'autres ordinateurs domestiques. L'accès s'en trouve facilité, mais au prix d'une surcharge de l'UC pendant l'exécution et donc certaines limitations.
La bibliothèque que nous utiliserons est une version légèrement modifiée de uLora par fantasticdonkey que l'on trouve sur GitHub à [7]. Cette bibliothèque est elle-même une variante de la bibliothèque TinyLoRa pour CircuitPython d'Adafruit. Les limitations mentionnent que nous ne pouvons que transmettre des données, sans rien recevoir en retour du LoRaWAN. Avec cette limitation, nous avons également dû utiliser l'APB (activation par personnalisation) pour l'authentification, ce qui signifie que la clé de session du réseau et la clé de session de l'application sont stockées dans notre code.
Comme ce n'est qu'un exemple pour montrer que l'on peut faire du LoRa avec MicroPython sur le Raspberry Pi Pico, ce n'est pas parfait, mais cela permet de débuter rapidement. Les fichiers modifiés nécessaires peuvent être téléchargés depuis la page GitHub d'Elektor [12]. Notez également que même si le logiciel semble fonctionner, la stabilité peut être un problème qui devra être résolu à un moment donné.

Installer MicroPython sur le RPi Pico

Le Raspberry Pi Pico est livré avec un chargeur de démarrage qui permet de changer facilement le micrologiciel en cours d'exécution. Comme nous aurons besoin de MicroPython, nous installons la dernière version disponible sur [8] (cf. fig. 7). Téléchargez le fichier UF2 depuis la page web et préparez votre RPi Pico à entrer en mode *bootloader*.
Pour entrer en mode *bootloader*, déconnectez le RPi Pico, appuyez sur le bouton BOOTSEL et reconnectez-le à votre ordinateur. Un nouveau périphérique de stockage de masse doit alors apparaître, comme sur la **figure 8**. Faites glisser le fichier UF2 téléchargé dans ce lecteur, ce qui installera la version courante de MicroPython.
Après un redémarrage, vous serez prêt à continuer, ou du moins votre RPi Pico le sera. Pour le développement, ce serait bien d'avoir un éditeur ou un EDI. L'étape suivante est la configuration de Thonny comme EDI Python.

Installer Thonny

L'installation de l'EDI Thonny sur une Ubuntu 20.04 récente ou même sur Windows ne prend que quelques clics. Pour Windows, vous pouvez récupérer un programme d'installation à l'adresse [9]. Pour Ubuntu, vous pouvez utiliser `wget -q -O - https://github.com/thonny/thonny/releases/download/v3.3.4/thonny-3.3.4.bash` pour obtenir le programme d'installation de la version 3.3.4 avec prise en charge du RPi Pico. Après avoir téléchargé le fichier, exécutez-le avec `bash thonny-3.3.4.bash`. Si l'installation réussit, vous aurez l'EDI Thonny en version 3.3.4 installé (cf. **fig 9**).

Après son premier démarrage, nous devons configurer le RPi Pico. Cela se fait à partir du menu avec *Outils → Options* qui ouvrira la boîte de dialogue de configuration (cf. **fig. 10**). Lorsque les outils et le matériel sont en place, il est temps de passer au code.

Mise en place du code

Pour LoRa, il est important de savoir dans quelle région du monde vous vous trouvez. La bande ISM utilisée peut être dans la gamme des 433 MHz, 868 MHz ou 915 MHz. Pour l'Europe, il s'agit principalement de 868 MHz alors qu'aux États-Unis c'est 915 MHz. Ce projet est conçu pour l'Europe, il utilisera donc le plan de fréquence 868 MHz, et si vous le reproduisez, assurez-vous de l'adapter à votre zone géographique. Comme la bibliothèque uLoRa est modifiée pour être utilisée sur le RPi Pico, vous devez ouvrir et enregistrer *ulora.py*, *ttn_eu.py*, *ulora_encryption.py* et *lora.py* sur votre RPi Pico dans l'EDI Thonny. Pour le fichier *ttn_eu.py*, s'il n'est pas applicable à votre zone, vous devez enregistrer le *fichier ttn_xx.py* correspondant sur votre carte. Cela fournira à la carte les bibliothèques requises, et un exemple simple que nous allons expliquer maintenant.

Dans le fichier *lora.py* se trouve la logique principale de ce projet. La **figure 11** détaille le déroulement du programme. Après l'initialisation, on recherche si un capteur de température est connecté au bus One-Wire. S'il n'y en a pas, après 120 s de sommeil, il y a une nouvelle tentative pour en trouver un. Si un capteur est présent, sa température est lue puis transmise. La façon dont la valeur est codée n'est pas très efficace en termes de charge utile, mais sera plus tard utile pour la démonstration. La **figure 12** montre les données transmises dans la console de The Things Network. Après une transmission, nous devons attendre 900 s pour respecter la politique d'utilisation équitable de The Things Network. Si vous souhaitez démarrer votre code automatiquement sur votre RPi Pico, enregistrez le fichier *lora.py* sous le nom de *main.py*.

Vous pouvez remarquer dans le code qu'une valeur est écrite dans un fichier appelé *current.txt*. Il s'agit du compteur de trame utilisé pour la transmission vers The Things Network. Si nous ne sauvegardions pas cette valeur, elle serait remise à zéro à chaque redémarrage du nœud,

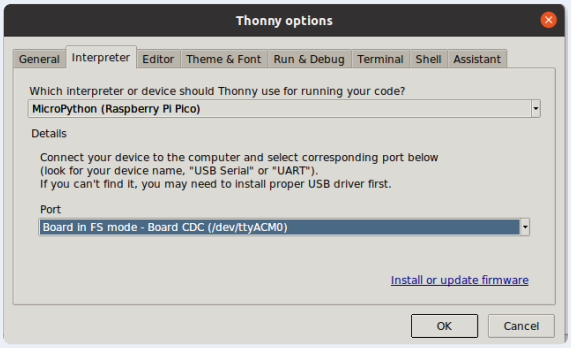


Figure 10. Paramètres de connexion pour le Raspberry Pi Pico.

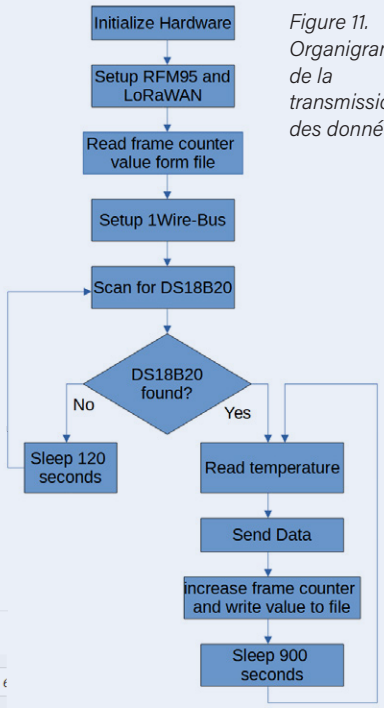


Figure 11. Organigramme de la transmission des données.

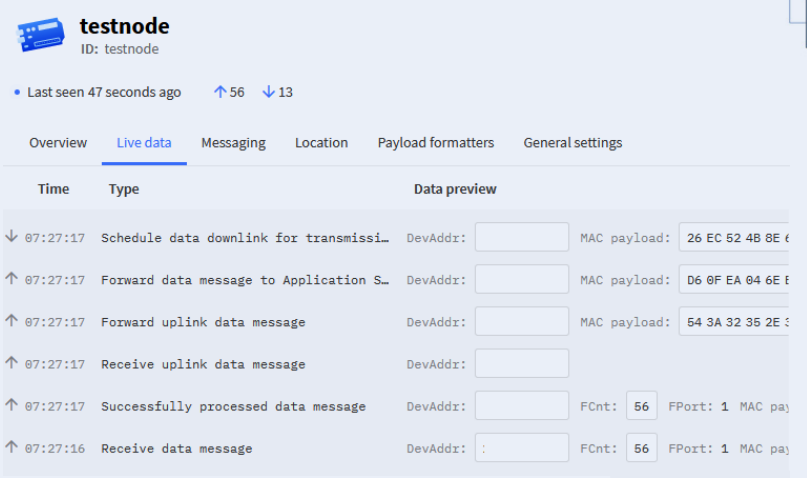


Figure 12. Données reçues à l'intérieur de la console TTN.

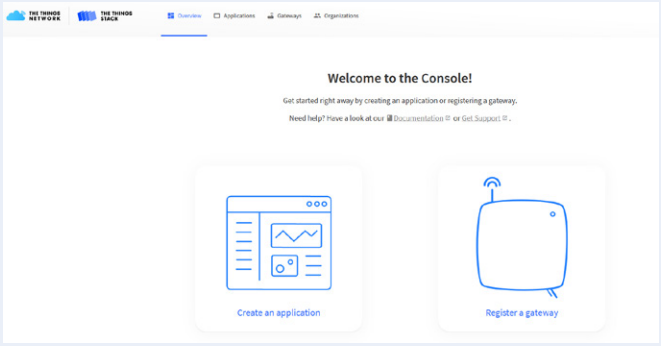


Figure 13. Créer une nouvelle application.

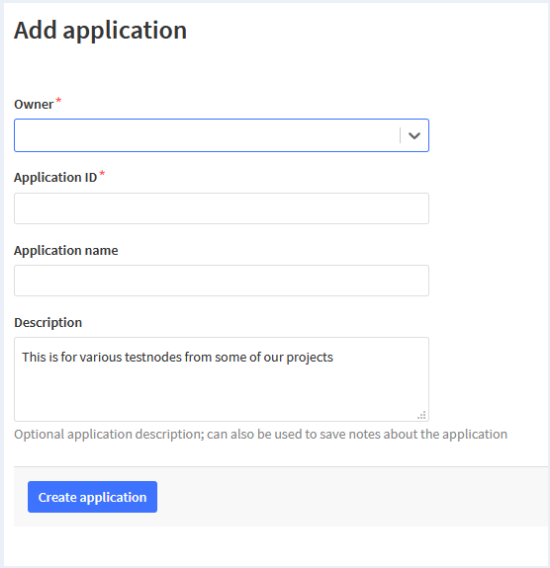


Figure 14. Assistant pour ajouter une nouvelle application.

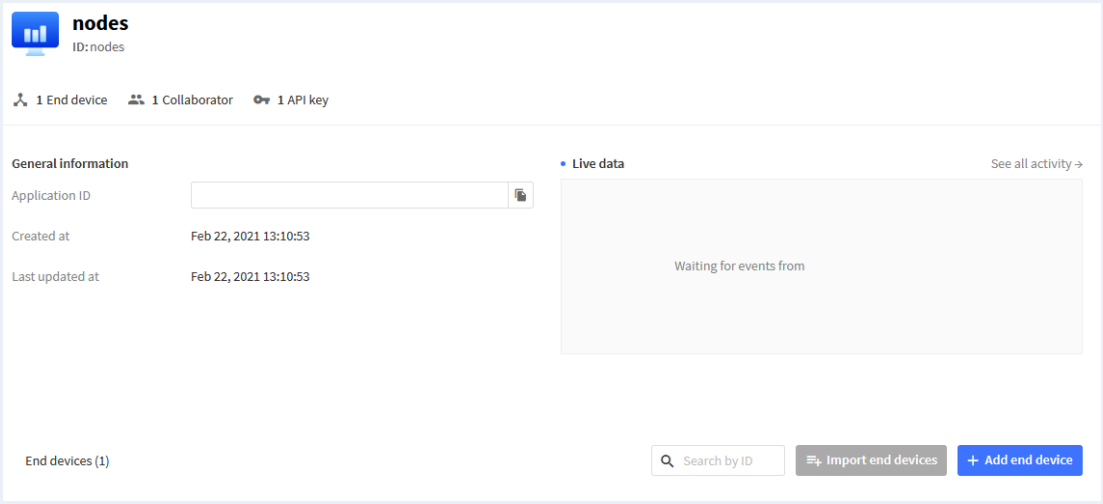


Figure 15. Ajouter un nouveau dispositif.

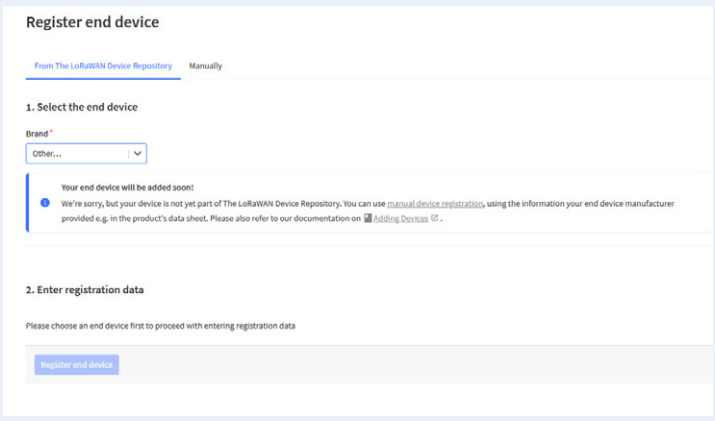


Figure 16. Assistant pour créer un nouveau nœud.

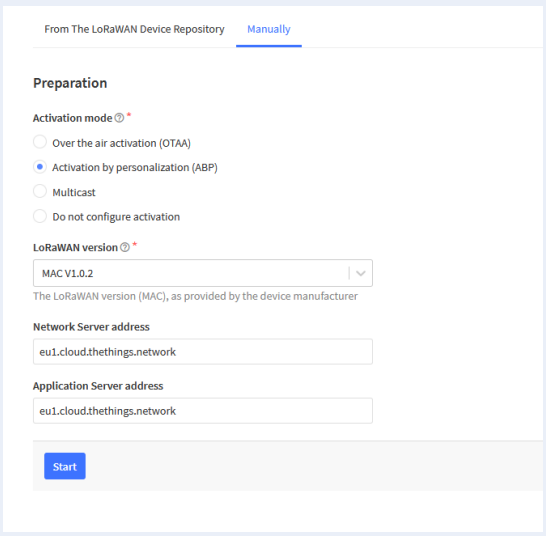


Figure 17. Paramètres du mode ABP.

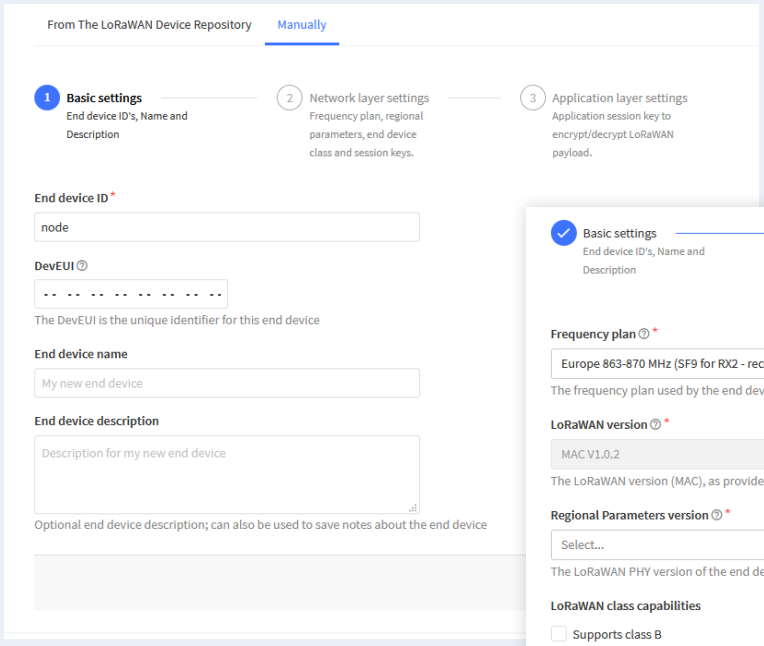


Figure 18. Paramétrage du nom et de l'EUI.

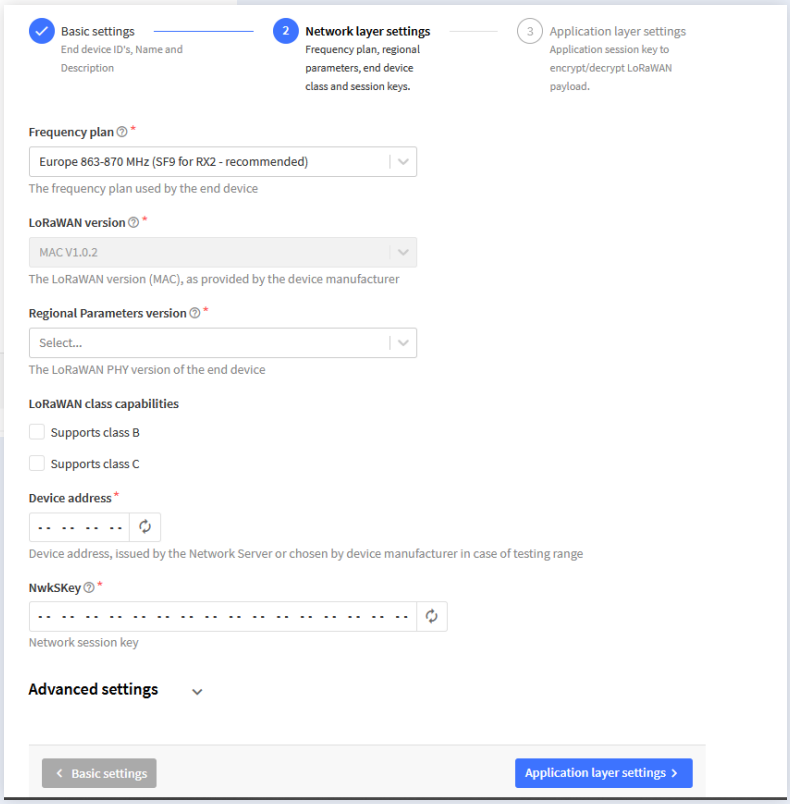


Figure 19. Paramètres du plan de fréquence.

par ex. après un changement de batterie. Si le compteur était remis à zéro à chaque redémarrage de l'appareil, les données transmises seraient rejetées par The Things Network pour des raisons de sécurité. Après chaque transmission, le fichier est mis à jour avec la valeur courante utilisée pour la recharger au redémarrage suivant. Ce n'est pas idéal et réduira la durée de vie de la mémoire flash. Si nous écrivons une nouvelle valeur toutes les 15 min, cela signifie 96 écritures par jour, ou 35.040 écritures par an, ce qui est loin d'être parfait et doit être amélioré.

Vers The Things Network et retour

De précédents articles expliquent comment configurer un nœud LoRa, mais je voudrais ajouter quelques mots à ce sujet. L'article précédent présentait la pile The Things Network dans sa version 2. La version 3 a récemment été annoncée, et dans certains endroits vous pouvez déjà migrer vos applications et passerelles dans cette nouvelle version et donc la console.

Pour que les données soient transmises, il faut ajouter quelques paramètres importants aux fichiers MicroPython fournis. Nous devons avoir un compte sur The Things Network pour utiliser l'infrastructure communautaire. Lorsqu'un compte a été créé, ou si vous souhaitez en créer un, vous pouvez vous rendre sur la pile de la version 3 en utilisant [10]. Après la connexion à notre compte, nous devons créer un nouveau nœud. Pour cela, nous devons d'abord configurer une application. Comme le montre la **figure 13**, cliquez sur *Create an application* et remplissez le formulaire de la **figure 14**. Sélectionnez le propriétaire de cette application, dans ce cas votre compte, et renseignez l'*Application ID*. Cliquez sur *Create application* pour terminer le processus.

Une nouvelle application ayant été configurée, nous pouvons y ajouter un nouveau nœud. Celui-ci fournira les informations d'identification que nous devons entrer dans le script MicroPython. Dans votre application nouvellement créée, sélectionnez *Add end device* pour lancer un assistant de création d'un nouveau nœud (**fig. 15**).

L'assistant (**fig. 16**) vous demandera de sélectionner un appareil. Comme le nôtre ne sera pas dans la liste, sélectionnez *Manually*. Comme sur la **figure 17**, choisissez *Activation by personalization* (ABP) et sélectionnez la version LoRaWAN pour notre MAC. Ici, vous pouvez choisir MAC V1.0.2. Sélectionnez *Start* pour accéder aux paramètres de base (**fig. 18**). Renseignez l'*End device ID*, un identifiant unique pour votre nœud. Les champs *End device name* et *End device description* sont utilisés pour vous permettre de distinguer ultérieurement les nœuds que vous avez créés. La page suivante (**fig. 19**) vous demande de sélectionner le plan de fréquences. Réglez-le en fonction de votre zone. Comme le nœud que nous réalisons se contente de faire du LoRaWAN pour transmettre des données, nous ne prenons pas en charge la classe B ou la classe C. Sélectionnez *Application layer settings* pour continuer. Générez une clé de session d'application (**fig. 20**) et terminez avec *Add end device*. Le nouveau dispositif créé apparaîtra (**fig. 21**). Nous avons besoin pour notre script MicroPython de la *Device address*, de la *NwkSKey* et de l'*AppSKey*. Celles-ci seront placées dans le fichier *lora.py*, où DEVADDR

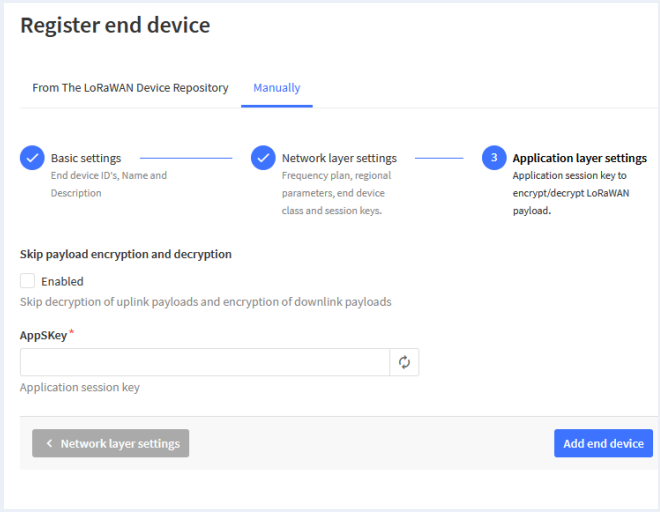


Figure 20. Génération de la clé d'application.

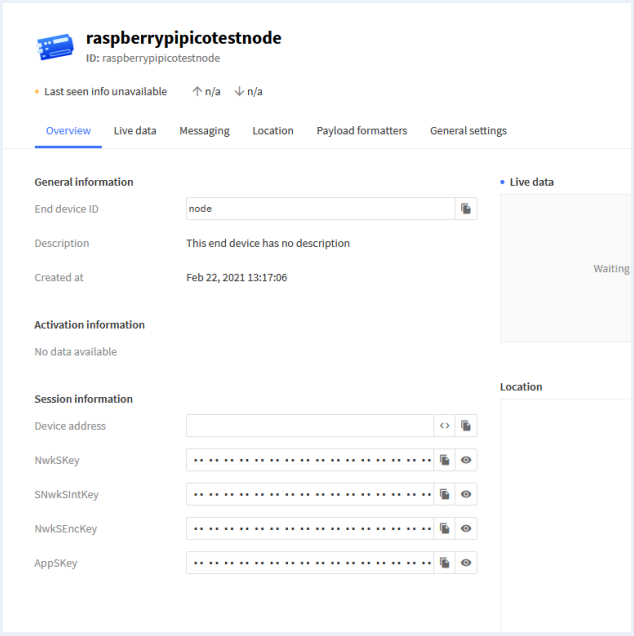


Figure 21. Informations sur le dispositif pour le nouveau nœud.

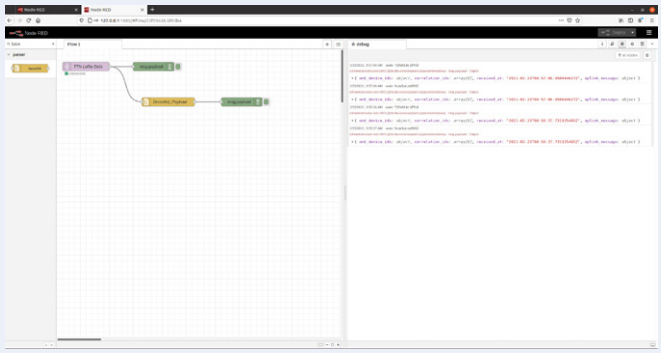


Figure 22. Flux de base Node-Red pour les données LoRaWAN.

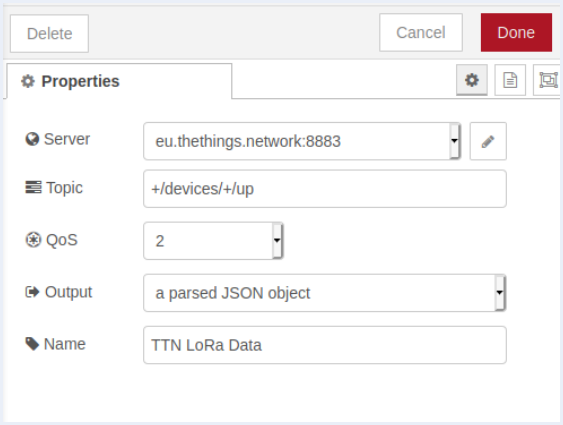


Figure 23. Paramétrage du serveur pour le nœud MQTT.

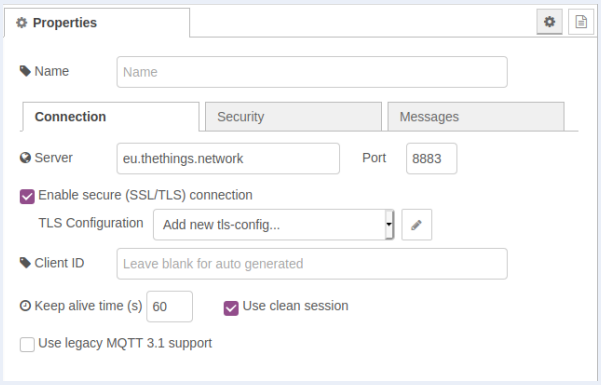


Figure 24. Paramètres du serveur et de SSL.

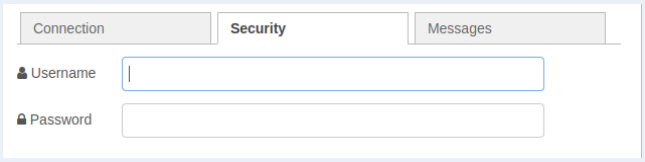


Figure 25. Paramétrage du nom d'utilisateur et du mot de passe.

contiendra l'adresse du dispositif, `NWKEY` contiendra NwkSKey et `APP` AppSKey. Si toutes les données sont entrées correctement, la transmission apparaîtra dans The Things Network.

Récupérer les données avec Node-RED

La commande suivante, dans un terminal ou via une connexion par SSH, permet d'installer Node-RED sur un Raspberry Pi :

```
bash <(curl -sL https://raw.githubusercontent.
com/node-red/linux-installers/master/deb/
update-nodejs-and-nodered
```

Pour que Node-RED fonctionne comme un service, vous devez également entrer et exécuter la commande suivante après son installation :

```
sudo systemctl enable nodered.service
```

Le guide d'installation complet est disponible ici [11]. Pour démarrer le service, exécutez la commande suivante dans un terminal : `sudo systemctl start nodered.service`.
Une fois que Node-RED est configuré et fonctionne, nous devons bâtir un flux comme celui de la **figure 22** pour récupérer nos données transmises. Bien que des nœuds pour The Things Network existent, sur un RPi vous devez utiliser la « connectivité » générique MQTT. L'utilisation des nœuds pour The Things Network sur un RPi peut entraîner des erreurs et des plantages de votre flux de données. Sans doute des problèmes de compatibilité avec les nœuds fournis, car ils fonctionnent bien avec du matériel à base de X86.
Pour le connecteur MQTT, nous sommes intéressés par les données que notre nœud va transmettre, appelées *payload*. La **figure 22** montre le flux complet de Node-RED pour récupérer les données que nous transmettons. La configuration requise commencera par le nœud MQTT. Ici, nous devons renseigner le serveur qui fournira nos données, et les informations d'identification.
La **figure 23** montre le *Serveur* et le *Topic* qui doivent être saisis. Un *Topic* peut être considéré comme un espace de conversation sur un sujet défini. De cette manière, seuls les messages pour lesquels il existe un intérêt sont fournis. Nous utilisons `+/devices/+/up` comme *Topic*. Cela signifie que nous sommes intéressés par tous les messages qui nous sont envoyés par les noeuds enregistrés dans notre application. Comme sortie, nous attendons un objet JSON décodé pour un traitement ultérieur. Dans l'étape suivante, il est nécessaire de définir les préférences pour les serveurs de The Things Network. Pour modifier les paramètres d'un serveur, cliquez sur le bouton crayon à droite du serveur. Une nouvelle boîte de dialogue s'affiche **(fig. 24**). Comme nous fonctionnons sur la nouvelle pile V3, utilisez *eu1.cloud.thethings.network* et le port *8883*. Assurez-vous que l'option *Enable secure (SSL/TLS) connection* est cochée. Sous l'onglet *Security* (cf. **fig. 25**), vous devez entrer un nom d'utilisateur et un mot de passe. Comme nom d'utilisateur, utilisez le nom de l'application que nous avons créée précédemment dans la console The Things Network.
Obtenir le mot de passe implique un peu plus de travail. Rendez-vous sur la console The Things Network et connectez-vous à votre application. Cliquez sur l'entrée de menu *API Keys* sur la gauche et commencez à ajouter une *API Key* avec *Add API key.* Un nouvel assistant (cf. **fig. 26**) apparaîtra. Choisissez les droits d'accès requis

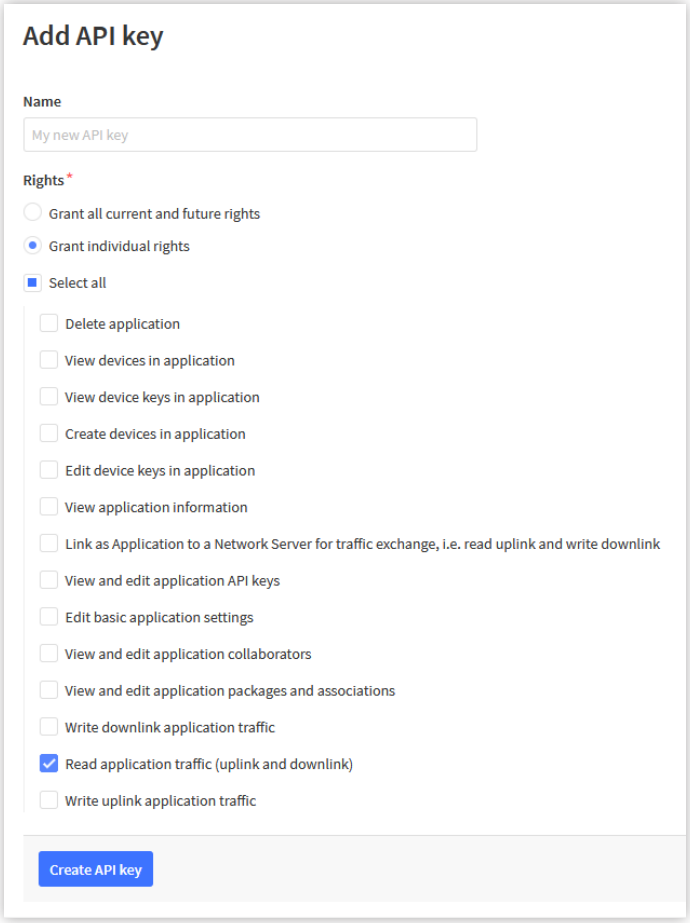


Figure 26. Assistant pour une nouvelle clé d'API.

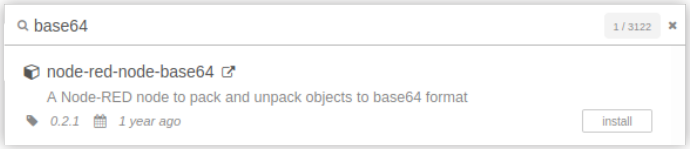


Figure 27. Paramètres du décodeur BASE64 dans Node-RED.

et terminez le dialogue avec *Create API Key*. L'étape suivante vous présentera une nouvelle clé API que vous pourrez utiliser. Veillez à la stocker dans un endroit sûr, car vous ne pourrez plus y accéder par la suite. Cette clé est le mot de passe que nous devons utiliser pour Node-RED.
Si tous les paramètres sont en place, les changements seront appliqués avec *Done*. Si maintenant vous utilisez ces paramètres, le nœud MQTT établira une connexion et de nouvelles données lui seront présentées. Seul inconvénient avec les données que nous recevons : la charge utile, les données que notre nœud envoie, est codée avec BASE64. Pour récupérer les octets bruts transmis, nous devons insérer un décodeur BASE64 qui fera la conversion. S'il est configuré comme dans la **figure 27**, les nœuds suivants pourront accéder aux données sous forme de tableau d'octets.
Pour plus de facilité, nous avons configuré un flux de démonstration comme celui de la **figure 28** pour afficher depuis un navigateur web les dernières données arrivées. La **figure 29** montre l'aspect final de la page web. Ici, il vous suffit d'entrer vos informations d'identification.

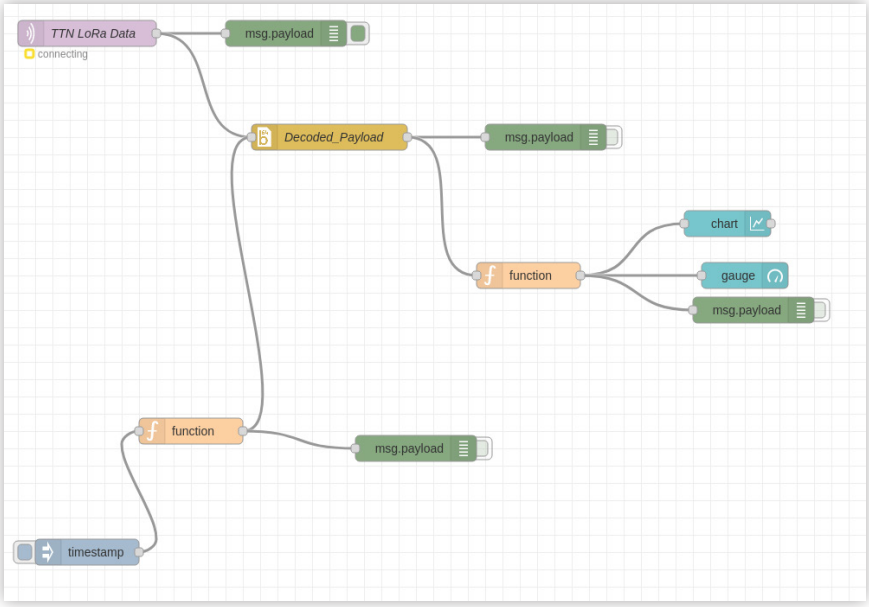


Figure 28. Exemple de flux de données dans Node-RED.

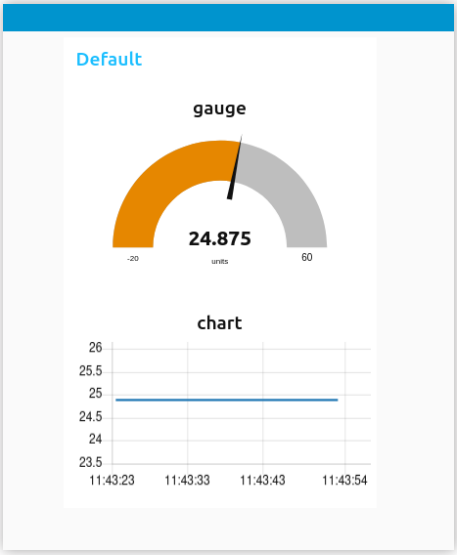


Figure 29. Données présentées sous forme de page web.

Simple et adapté aux débutants

Bien que ce ne soit pas le projet le plus complexe que vous puissiez réaliser, il vous permettra de commencer à bricoler avec le RPi Pico et LoRaWAN. L'utilisation de MicroPython peut être, surtout pour les débutants, une transition en douceur vers le monde des systèmes embarqués. Le temps nécessaire à la construction et à l'exécution de ce projet le rend adapté aux salles de classe (virtuelles) et à l'enseignement. Mais nous vous avions promis un circuit imprimé. Si vous ne touchez pas aux composants, vous pouvez vous arrêter là. Si vous jouez du fer à souder, continuez !

Un mot d'avertissement

Habituellement, lorsque nous présentons des schémas et des circuits imprimés dans Elektor, ils ont été câblés et testés (au moins pour vérifier qu'ils ne risquent pas de prendre feu ou de tuer de jolis chatons). Pour ce circuit imprimé et ces schémas, les choses sont différentes. Le circuit imprimé, et donc le schéma, est un travail en cours. Bien qu'il devrait fonctionner, il n'a pas encore été testé, donc soyez conscient qu'il peut y avoir quelques bugs. Vous pouvez télécharger tous les fichiers KiCad de ce montage depuis la page Elektor ou le dépôt GitHub d'Elektor [12] pour le modifier, l'adapter à vos propres besoins. Si vous avez vu des défauts de conception ou des choix de conception stupides (ce qui ne veut pas dire quelque chose comme l'utilisation d'un RPi Pico), n'hésitez pas à nous faire part de vos suggestions. Cela ne se limite pas aux défauts. Si pour vous, il manque quelque chose, n'hésitez pas à suggérer des changements. Comme je l'ai déjà mentionné, il s'agit d'un travail en cours. Vos commentaires sont les bienvenus.

Schéma : partie 1

Les schémas sont divisés en deux parties, même si tous les composants sont sur une seule feuille KiCad. La première partie concerne la connexion du RFM95, notre émetteur-récepteur LoRa, et du capteur de température DS18B20 au RPi Pico. Pour le RFM95, nous avons besoin d'une liaison SPI, composée de *MISO*, *MOSI*, *SCK* et *nCS*. En outre, nous devons ajouter RESET et DIO0 pour un fonctionnement minimal. La **figure 5** montre le schéma de connexion du RFM95 au RPi Pico. Notez également que nous avons ajouté deux condensateurs, 100 nF (C2) et 4,7 µF (C1), à côté du RFM95. Les condensateurs doivent fournir les pics de courant si le RFM95 est en mode transmission ou réception, comme recommandé par la fiche technique.

Vous pouvez également voir R5, une résistance de 4,7 kΩ sur la ligne de réinitialisation. Normalement, elle n'est pas nécessaire, car le RFM95 offre une résistance de rappel interne. L'utilisation du module dans plusieurs projets a montré qu'une résistance de rappel externe évite les réinitialisations indésirables qui se produisent de temps à autre sans elle. Les éléments qui ne sont pas utilisés par le logiciel MicroPython, mais qui peuvent l'être ultérieurement par les bibliothèques C/C++ sont *DIO0*, *DIO1*, *DIO2* et *DIO3*. Alors que DIO0 est nécessaire pour le fonctionnement général de la plupart des bibliothèques LoRa, les autres DIO fournissent des fonctions optionnelles. D'autres bibliothèques peuvent utiliser ces fonctions.

Pour le DS18B20 (fig. 3), nous alimentons « volontairement » la broche VCC. Le protocole One Wire permet d'avoir une alimentation passive en utilisant seulement la broche de données et la masse (alimentation parasite). Certaines cartes non authentiques se comportent bizarrement lorsqu'elles ne sont pas alimentées via VCC. Par conséquent, les trois broches, y compris VCC, sont connectées, en cas d'installation involontaire d'un de ces capteurs contrefaits. Soyez donc prudent : si vous achetez par erreur des capteurs DS18B20 contrefaits, les performances peuvent en souffrir, ou les alimenter avec la seule broche de données ne fonctionnera pas du tout. Le protocole One Wire impose une résistance de rappel de 4,7 kΩ sur la ligne de données.

Le RPi Pico est un module, vous n'avez donc pas besoin de connecter de nombreux composants. Ce qui est différent, ce sont les étiquettes V_BUS et V_SYS attachées aux broches 40 et 39. Elles seront utilisées pour connecter une batterie rechargeable au dispositif, ce qui nous amène à la deuxième partie du schéma.

Schéma : partie 2

La deuxième partie du schéma (**fig. 30**) concerne la section pour une batterie rechargeable. Elle consiste en un chargeur au lithium MCP73871-1CC avec trajet de courant. Ainsi, si nous fournissons de l'énergie à son entrée, il la transmettra à sa sortie, et n'utilisera pas du tout la batterie. Si l'alimentation externe est déconnectée ou insuffisante, une commutation automatique vers la batterie se produira. Le chargeur est actuellement configuré pour charger des batteries de 1000 mA.

Pour protéger la batterie attachée, un circuit XB8089D, déjà utilisé dans le kit du *superchargeur* [14], est inclus. Il empêchera la surcharge, la

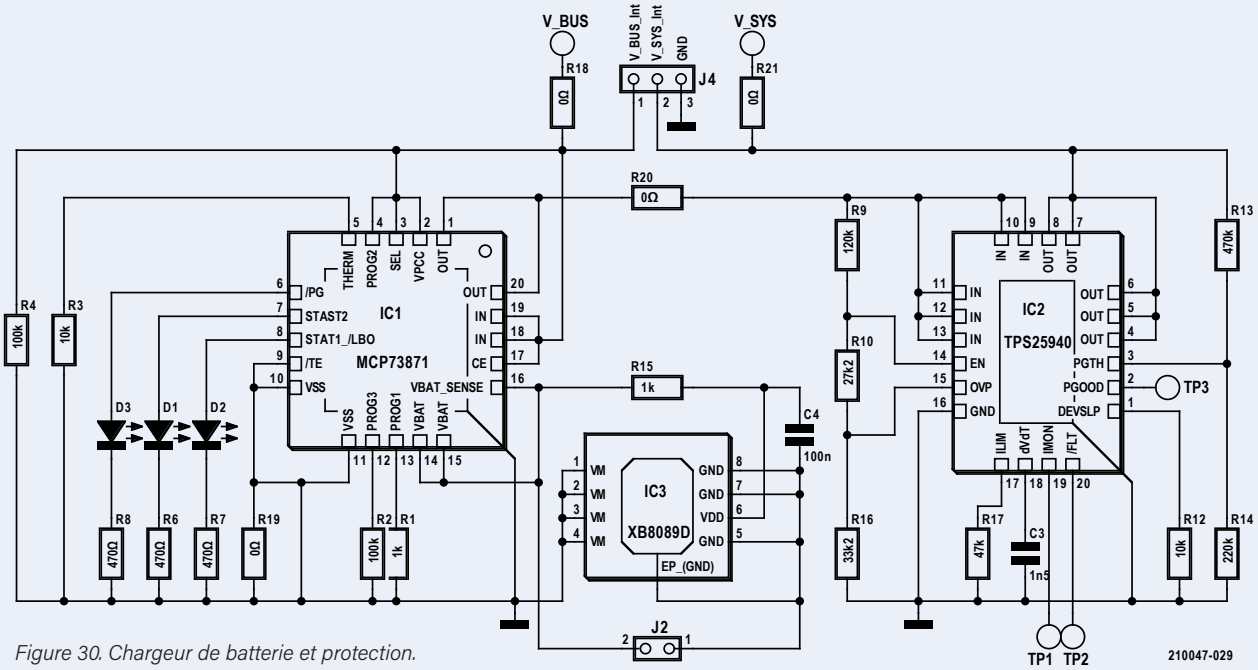


Figure 30. Chargeur de batterie et protection.

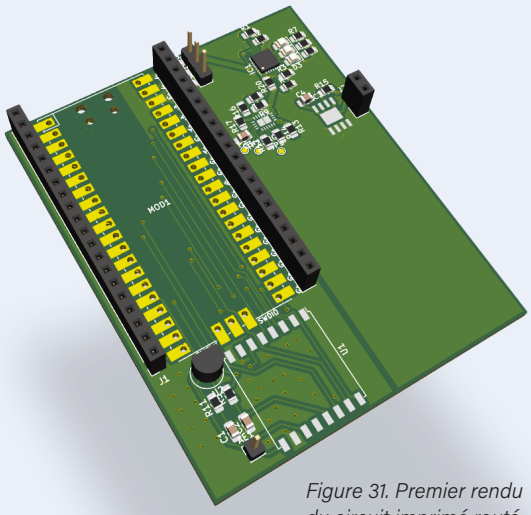


Figure 31. Premier rendu du circuit imprimé routé.

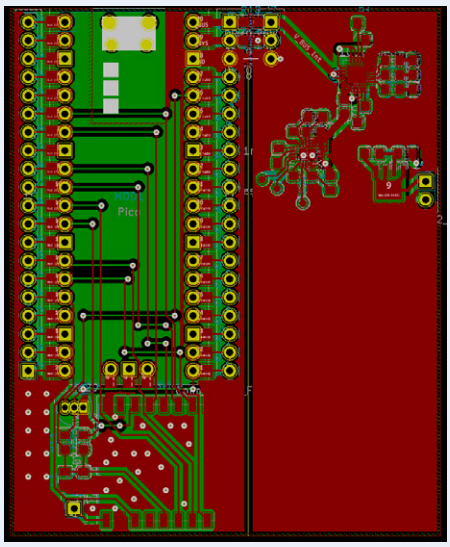


Figure 32. Aperçu de la disposition réalisée.

décharge profonde, la surintensité et l'inversion de polarité. Le dernier élément est un *eFuse* (fusible électronique) TPS25940 de Texas Instruments. Celui-ci agit comme une diode idéale, empêchant le retour du courant vers le chargeur de batterie au lithium. Il servira d'élément de protection contre les surintensités et les décharges profondes. La surtension et la sous-tension sont déterminées par R9, R10 et R16. La fiche technique fournit les formules nécessaires pour calculer les valeurs souhaitées. Nous utilisons 120 kΩ pour R9, 27,2 kΩ pour R10 et 33,2 kΩ pour R16 pour obtenir 5,37 V de seuil de surtension et 2,957 V de seuil de sous-tension. Cela devrait être dans la plage de sécurité, pour la batterie et le convertisseur DC/DC du RPi Pico. Nous utilisons R17 (47 kΩ) pour limiter le courant à 1,89 A, car c'est le maximum que le chargeur au lithium sera capable de fournir. Pourquoi est-ce une mauvaise idée d'utiliser un fusible réarmable (*polyfuse*) ? Lisez l'article d'Elektor [13]. L'utilisation d'une diode introduirait une chute de tension d'au moins 0,3 V à travers elle, ce qui signifie que nous convertissons de l'énergie en chaleur.

Quelque chose que l'on ne voit pas souvent dans les circuits imprimés sont R20, R21 et R18. Ce sont des résistances de 0 Ω qui vous permettront de déconnecter des parties du chargeur de batterie. Si certaines parties doivent être examinées de plus près ou ne fonctionnent pas du tout, elles peuvent être séparées et contournées sans couper les pistes sur le circuit imprimé. Il suffit de retirer quelques composants CMS.

Placement rapide sur un circuit imprimé

Le schéma étant terminé, tous les composants ont été placés sur un circuit imprimé et routés. Ce n'est pas très joli pour l'instant, mais pour un premier essai, cela fera l'affaire. La **figure 31** montre le premier rendu. La **figure 32** donne un aperçu du routage. Vous pouvez également voir que le circuit imprimé est conçu pour que la partie chargeur puisse être complètement retirée ou utilisée à d'autres fins.

Si elle ne fonctionne pas, vous pouvez également la retirer du circuit imprimé. C'est comme un deux-en-un.

Que faire ensuite ?

Les étapes suivantes dépendront des réactions de nos lecteurs, c'est-à-dire vous. Si vous aimez ce projet et souhaitez que nous le poursuivions, que nous y ajoutions des composants ou bien si vous avez des suggestions pour l'améliorer, n'hésitez pas à laisser un commentaire ou à nous envoyer un message. De même, si vous avez des propositions pour différentes parties, nous sommes intéressés. Vous pouvez également nous laisser un message sur d'autres projets potentiellement intéressants.

Pour finir

Utiliser MicroPython sur le RPi Pico et travailler sur un projet LoRaWAN est assez simple et peut être amusant même pour un débutant. Mais la stabilité du logiciel peut être un problème. Pour des installations plus fiables, il est préférable de programmer en C/C++. Si les scripts MicroPython fonctionnent, c'est un moyen rapide et pratique de se lancer. Ce que vous mesurez et transmettez dépend de vous, et le code utilisé est suffisamment simple pour être testé par des enfants. Pourquoi transmettre uniquement des mesures de température ? Vous pouvez faire un système d'alarme LoRaWAN avec quelques détecteurs PIR. Ou vous pouvez surveiller si vos plantes ont besoin d'eau. Laissez libre cours à votre imagination.
Bien que ce soit le premier projet avec un RPi Pico, ce ne sera pas le dernier. Nous en préparons d'autres, mais notez qu'ils ne sont peut-être pas terminés. Néanmoins, nous espérons qu'ils vous inspireront pour vos propres projets ou qu'ils vous donneront des informations qui pourront vous être utiles à l'avenir.

(210047-04)

PRODUITS

- **Carte à microcontrôleur Raspberry Pi Pico**
  www.elektor.fr/raspberry-pi-pico-microcontroller-board
- **Plaque d'essai (830 points)**
  www.elektor.fr/breadboard-830-tie-points
- **Mini plaque d'essai et fils de liaison**
  www.elektor.fr/mini-breadboards-jumper-wires
- **Module émetteur-récepteur LoRa ultra-long (868 MHz) RFM95 de SeeedStudio**
  www.elektor.fr/rfm95-lora
- **Livre en anglais « KiCad Like a Pro » (2e édition)**
  www.elektor.fr/kicad-like-a-pro
- **Passerelle intérieure LoRaWAN LPS8 de Dragino**
  www.elektor.fr/dragino-lps8-indoor-lorawan-gateway
- **Concentrateur GPS LoRaWAN pour Raspberry Pi (868 MHz) PG1301 de Dragino**
  www.elektor.fr/dragino-pg1301
- **Livre électronique en anglais « Programming with Node-RED »**
  www.elektor.fr/programming-with-node-red-e-book

Contributeurs
Conception et texte : Mathias Claußen
Rédaction : Jens Nickel
Mise en page : Harmen Heida
Traduction : Denis Lafourcade

Des questions, des commentaires ?
Envoyez un courriel à l'auteur (mathias.claussen@elektor.com) ou contactez Elektor (redaction@elektor.fr).

LIENS

[1] **« LoRaWAN : décollage facile », Elektor 03-04/2020** : www.elektormagazine.fr/191065-04
[2] **Fichiers du projet sur GitHub** : https://github.com/ElektorLabs/210047-LoRa-with-the-Raspberry-Pi-Pico
[3] **Passerelle intérieure LoRaWAN LPS8 de Dragino** : www.elektor.fr/dragino-lps8-indoor-lorawan-gateway
[4] **Concentrateur GPS LoRaWAN pour Raspberry Pi (868 MHz) PG1301 de Dragino** : www.elektor.fr/dragino-pg1301
[5] **Livre en anglais « Get Started with MicroPython on Raspberry Pi Pico »** : www.elektor.fr/get-started-with-micropython-on-raspberry-pi-pico
[6] **Document PDF en anglais, gratuit, « Get Started with MicroPython on Raspberry Pi Pico »** : https://hackspace.raspberrypi.org/books/micropython-pico/pdf/download
[7] **Dépôt GitHub de uLoRa** : https://github.com/fantasticdonkey/uLoRa
[8] **MicroPython sur Raspberry Pi Pico** : www.raspberrypi.org/documentation/pico/getting-started/
[9] **EDI Thonny** : https://thonny.org/
[10] **Pile The Things Network v3** : https://eu1.cloud.thethings.network/console/
[11] **Instructions d'installation de Node-Red** : https://nodered.org/docs/getting-started/raspberrypi
[12] **GitHub d'Elektor** : https://github.com/ElektorLabs
[13] « **superchargeur LiPo DIY », Elektor 05-06/2021** : www.elektormagazine.fr/191188-B-03
[14] **« superchargeur LiPo en kit », Elektor 01-02/2021** : www.elektormagazine.fr/191188-04
[15] **Fichiers Gerber de la carte du RFM95** : https://github.com/ElektorLabs/191069-RFM95-BOB/