INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Implémentation des couches Physique et MAC pour un jardin intelligent

---

Florent Miranville, Tom Lassalle , Sandra Bejaoui, Louis Rousset, João Gabriel, Badr Diani,

Amine Benchekroun

# *Sommaire*

# 1. Introduction

L'essor de l'Internet des objets (IoT) multiplie les besoins en communication Machine-to-Machine (M2M). Face à la diversité des protocoles, l'approche matérielle classique devient coûteuse et rigide. La Radio Logicielle (SDR) offre une alternative flexible en déplaçant le traitement du signal vers le logiciel, permettant d'adapter les communications via une architecture générique.

Ce projet vise à concevoir et implémenter une chaîne de transmission complète (couches PHY et MAC) sur des plateformes USRP avec GNU Radio. L'objectif est de maîtriser les contraintes réelles d'un système de télécommunications, de la modulation à la gestion de l'accès au canal.

Pour ce bureau d'étude, nous développons un réseau de capteurs sans fil (WSN) pour un jardin intelligent. Ce système doit permettre à des capteurs autonomes de remonter des données environnementales (humidité, luminosité) vers une station centrale. Les principaux défis sont d'assurer une transmission fiable sur une centaine de mètres tout en minimisant la consommation énergétique pour préserver la batterie des nœuds.

Ce rapport détaille notre démarche : définition des besoins, conception d'une couche physique robuste et implémentation d'une couche MAC de type ALOHA adaptée au trafic sporadique des capteurs.

# 2. Vue d'ensemble du système

Le déploiement d'un réseau de capteurs sans fil dans un environnement de type jardin intelligent impose des contraintes spécifiques liées à la nature du terrain et aux objectifs de l'application. En premier lieu, la mobilité des nœuds est considérée comme fixe. Le système ne gère pas le déplacement des capteurs une fois installés, ce qui simplifie la gestion de la topologie réseau en évitant les mécanismes complexes de transfert intercellulaire. L'environnement extérieur nécessite une transmission capable de franchir une distance de l'ordre de la centaine de mètres malgré les obstacles naturels potentiels.

Concernant la nature du trafic et la Qualité de Service (QoS), le système se caractérise par une fréquence de communication faible. La priorité n'est pas la vitesse de transmission ni le débit, mais bien la fiabilité de la liaison pour garantir l'intégrité des données remontées. Les contraintes de temps réel sont modérées pour la majorité des capteurs, une latence de quelques secondes étant acceptable pour des mesures environnementales lentes comme l'humidité ou la température.

Enfin, la contrainte majeure de ce projet réside dans l'efficacité énergétique. Les capteurs étant alimentés par batterie, le protocole doit minimiser la consommation électrique pour maximiser la durée de vie du réseau. Bien que l'implémentation soit réalisée sur du matériel de radio logicielle (USRP) qui n'est pas optimisé pour la basse consommation, la conception des couches PHY et MAC doit simuler un comportement économe, notamment en limitant les phases d'écoute active.

# 3. System overview

L'architecture globale du système repose sur un réseau de capteurs sans fil organisé en topologie étoile, où l'ensemble des communications converge vers une passerelle centrale (Gateway). Cette passerelle assure l'interface entre le réseau de capteurs et l'application métier gérant le jardin. Le trafic est majoritairement montant (uplink), constitué de courts messages de données provenant des différents nœuds, bien que des commandes descendantes soient nécessaires pour le pilotage des actionneurs.

Le système intègre trois catégories distinctes de périphériques pour assurer la surveillance et l'automatisation du jardin. La première catégorie concerne les capteurs périodiques, qui constituent la majorité du trafic. Elle comprend un capteur d'humidité du sol pour mesurer le niveau d'eau dans le substrat, un capteur combinant température et humidité de l'air pour surveiller les conditions climatiques, ainsi qu'un capteur de luminosité (PAR/Lux) pour évaluer l'intensité lumineuse nécessaire aux plantes. Ces capteurs sont programmés pour effectuer et transmettre leurs mesures environ une fois par heure.

En complément de la surveillance périodique, le système dispose de capteurs événementiels et d'actionneurs critiques. Un capteur de débit ou de pression peut être utilisé pour vérifier en temps réel l'exécution d'une commande, par exemple pour confirmer l'activation de l'arrosage. Les actionneurs comprennent une électrovanne d'irrigation, qui contrôle l'arrivée d'eau sur demande, et un contrôleur d'éclairage LED capable d'ajuster l'intensité ou le spectre lumineux pour favoriser la croissance. Cette diversité de nœuds justifie le besoin d'un protocole MAC capable de gérer à la fois un trafic régulier et des envois sporadiques.

# 4. Design de la couche physique

## 4.1. Contexte et choix de modulation

Dans le cadre du projet de jardin intelligent, le système de communication doit assurer une transmission fiable des données sur une distance de l'ordre du centaine de mètres avec un faible débit, dans un environnement bruité et avec des capteurs relativement simples et peu énergivores. Notre priorité n'étant donc pas la vitesse mais la fiabilité de notre transmission.

Notre choix s'est donc naturellement porté sur le BPSK parmi les différentes modulations numériques envisageables (ASK/OOK, FSK, PSK, QAM) qui était la plus adaptée à nos contraintes avec sa très bonne résistance au bruit dûe à son encodage sur seulement deux états de phase bien séparés.

Les modulations de type ASK ou OOK sont basées sur la variation de l'amplitude du signal et sont particulièrement sensibles au bruit et aux atténuations. Elles seraient donc peu adaptées à des communications longue distance en extérieur.

La modulation FSK offre une bonne robustesse car portée par la fréquence, mais elle a besoin d'une bande passante plus large pour un même débit, ce qui rend son efficacité spectrale plus faible. De plus, sa mise en œuvre est plus complexe que la BPSK électroniquement.

Les modulations à plus forte efficacité spectrale, telles que QPSK ou QAM, permettent de transmettre plusieurs bits par symbole et d'augmenter significativement le débit. En revanche, elles sont plus sensibles au bruit, car les états de la constellation sont plus rapprochés. Ces modulations sont donc adaptées à des systèmes à haut débit et à fort rapport signal/bruit, ce qui n'est pas le cas pour notre application de jardin intelligent.

Au final, la simplicité du BPSK permet une excellente séparation des symboles dans la constellation, d'où une très bonne résistance au bruit, et un faible taux d'erreur pour un rapport signal/bruit donné.

## 4.2. Principe de la modulation Binary Phase Shift Keying (BPSK)

Binary Phase Shift Keying (BPSK) est une technique de modulation numérique utilisée dans les communications numériques pour transmettre des données binaires sur un canal de communication. Dans BPSK, chaque symbole de données est représenté par une phase différente de la porteuse.

Les données binaires à transmettre sont codées en symboles. Dans BPSK, chaque symbole représente un bit. Par exemple, un bit "0" peut être représenté par une phase de 0 degré et un bit "1" peut être représenté par une phase de 180 degrés. Une porteuse sinusoïdale est utilisée pour transporter les données comme peut être observé dans la figure 1 et 2. Pour chaque symbole de données, la phase de la porteuse est modulée selon le bit correspondant. Si le bit est "0", la phase de la porteuse reste inchangée. Si le bit est "1", la phase de la porteuse est inversée de 180 degrés. Ou inversement selon l'interprétation initialement choisie.

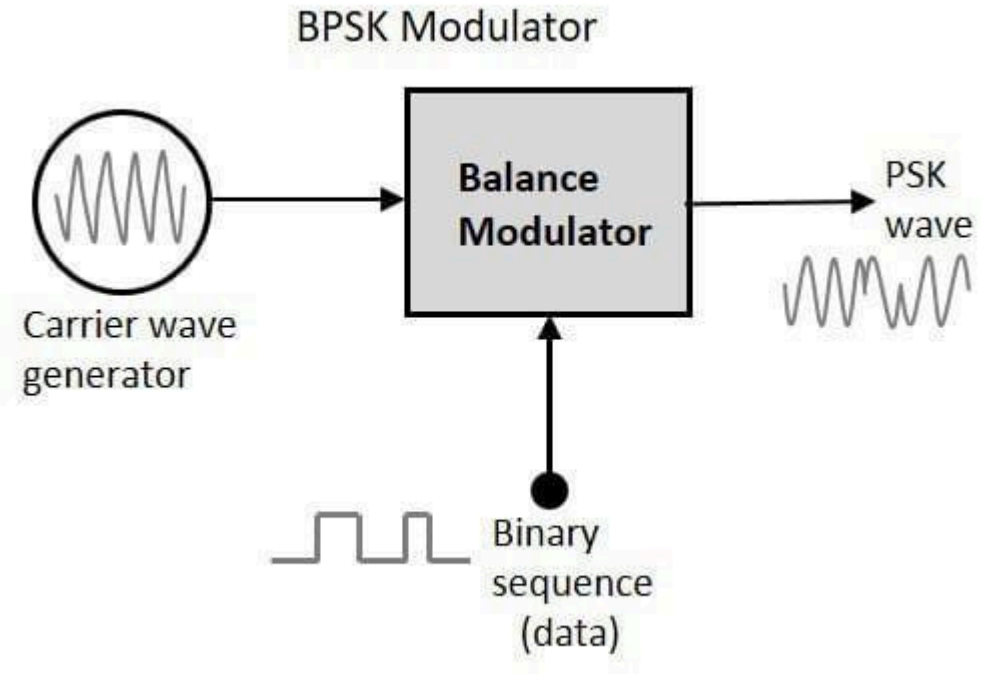



*Figure 1 : Schema of the modulation process of a BPSK modulator*

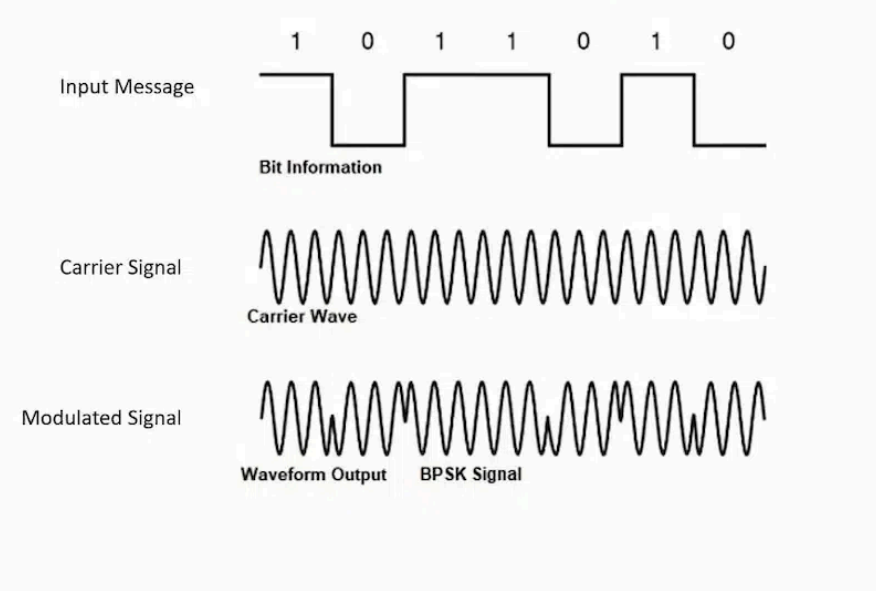



*Figure 2 : Applied example of a BPSK modulation*

## 4.3. Transmission des paquets

La Figure 3 présente la chaîne d'émission (TX) réalisée sous GNU Radio pour transmettre des trames radio structurées (packet-based), plutôt qu'un simple flux continu d'octets. L'objectif est de préserver explicitement les frontières de paquets, afin de faciliter l'interprétation côté récepteur et d'assurer une chaîne cohérente de bout en bout. Pour cela, nous nous appuyons sur une architecture tagged stream, où chaque paquet est associé à un tag de longueur packet_len. Ce tag pilote les blocs de mise en trame (framing) et garantit que toutes les opérations restent alignées au niveau paquet.

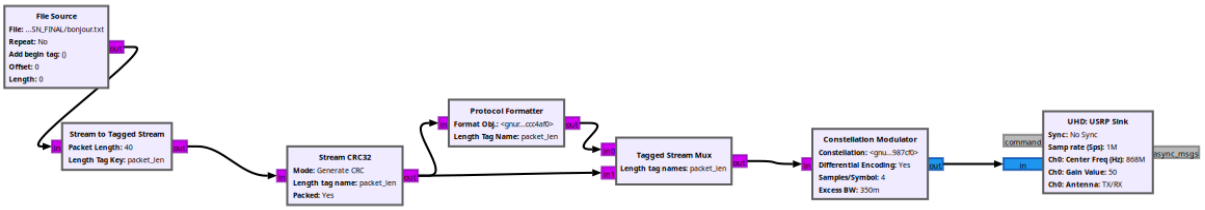



***Figure 3 – Chaîne de transmission de paquets (TX) sous GNU Radio.***

Blocs de la chaîne TX et rôle de chacun

- **File Source :** lit les données à transmettre depuis un fichier et produit un flux continu d'octets. Comme ce flux ne contient aucune information de découpage, il doit être segmenté avant la mise en trame.

- **Stream to Tagged Stream :** transforme le flux continu en une suite de paquets en ajoutant le tag packet_len.
  Dans notre cas, la taille est fixée à 40 octets (packet_len = 40). Ce choix correspond à un compromis : des paquets trop longs augmentent le risque qu'une erreur binaire rende une trame inutilisable, tandis que des paquets trop courts augmentent l'overhead (en-tête + CRC) par rapport à la charge utile.

- **Stream CRC32 (Generate CRC) :** ajoute un CRC32 à chaque paquet afin de permettre la détection d'erreurs dues au canal radio. Lors de la réception, le CRC est recalculé et comparé ; un paquet corrompu est alors rejeté. Le CRC est ajouté avant la mise en trame finale, de sorte que l'intégrité soit vérifiée sur les données effectivement transportées.

- **Protocol Formatter :** génère l'en-tête (header) au format par défaut GNU Radio. Cet en-tête contient les informations nécessaires au décodage, notamment la longueur du payload, ce qui permet au récepteur d'extraire correctement la charge utile. La configuration est assurée par la frame-format key fixée à 0101010101010011, afin d'imposer la même structure de trame côté émission et côté réception.

- **Tagged Stream Mux :** concatène l'en-tête avec le payload (payload + CRC) pour former une trame complète :
  Header + Payload(+CRC). Cette organisation permet au récepteur de se synchroniser sur la structure via l'en-tête, puis de vérifier la validité des données à l'aide du CRC.

- **Constellation Modulator (BPSK) :** module la trame en BPSK, choisie pour sa simplicité et sa robustesse, particulièrement adaptée aux essais de transmission par paquets et aux scénarios de type WSN. Les paramètres utilisés sont :
    - sps = 4 (4 échantillons par symbole) : améliore la mise en forme et facilite la synchronisation côté réception ;
    - excess_bw = 0.35 (roll-off) : limite l'occupation spectrale et améliore le compromis bande passante / interférences.

- **UHD: USRP Sink :** transmet le signal modulé via l'USRP avec une configuration définie par :
    - samp_rate = 1e6 (1 MS/s) : ce taux d'échantillonnage fixe la cadence de traitement numérique entre GNU Radio et l'USRP. Une valeur de 1 MS/s représente un compromis adapté à une transmission BPSK avec sps = 4 : elle permet une génération et un envoi stables du signal, tout en conservant une charge de calcul raisonnable et une bande passante suffisante pour transporter la trame sans dégradation liée au traitement. En pratique, ce choix simplifie l'expérimentation et garantit une chaîne cohérente côté émission/réception, car le même paramètre structure aussi les filtres et la synchronisation.
    - gain = 50 dB : le gain RF contrôle la puissance de sortie effective de l'émetteur. Un gain relativement élevé (50 dB) a été retenu pour assurer un niveau de signal reçu suffisant dans un environnement de test (atténuation, distance, obstacles et bruit). L'objectif est de limiter les pertes de paquets dues à un SNR trop faible et d'obtenir une réception exploitable et répétable. Ce réglage reste toutefois dépendant du contexte expérimental : il doit être ajusté si l'on observe de la saturation (clipping) ou au contraire un signal trop faible.

- center_freq = 868 MHz : le choix de 868 MHz est cohérent avec un contexte Wireless Sensor Network, car cette fréquence appartient à la bande ISM européenne largement utilisée pour l'IoT et les communications capteurs. De plus, par comparaison à 2.4 GHz, une fréquence plus basse permet généralement une meilleure portée et une meilleure robustesse face à certaines pertes de propagation, ce qui est pertinent pour l'envoi fiable de paquets courts dans notre expérimentation.

## 4.4. Réception des paquets

Notre récepteur implémente une chaîne complète de démodulation BPSK pour une fréquence porteuse de 868 MHz, une bande passante de 1MHz et un débit symbole de 250 kbauds. Il s'agit d'une configuration typique pour des couches physiques robustes dans la bande ISM 868 MHz. Vous pouvez voir ci-dessous la chaîne implémentée :

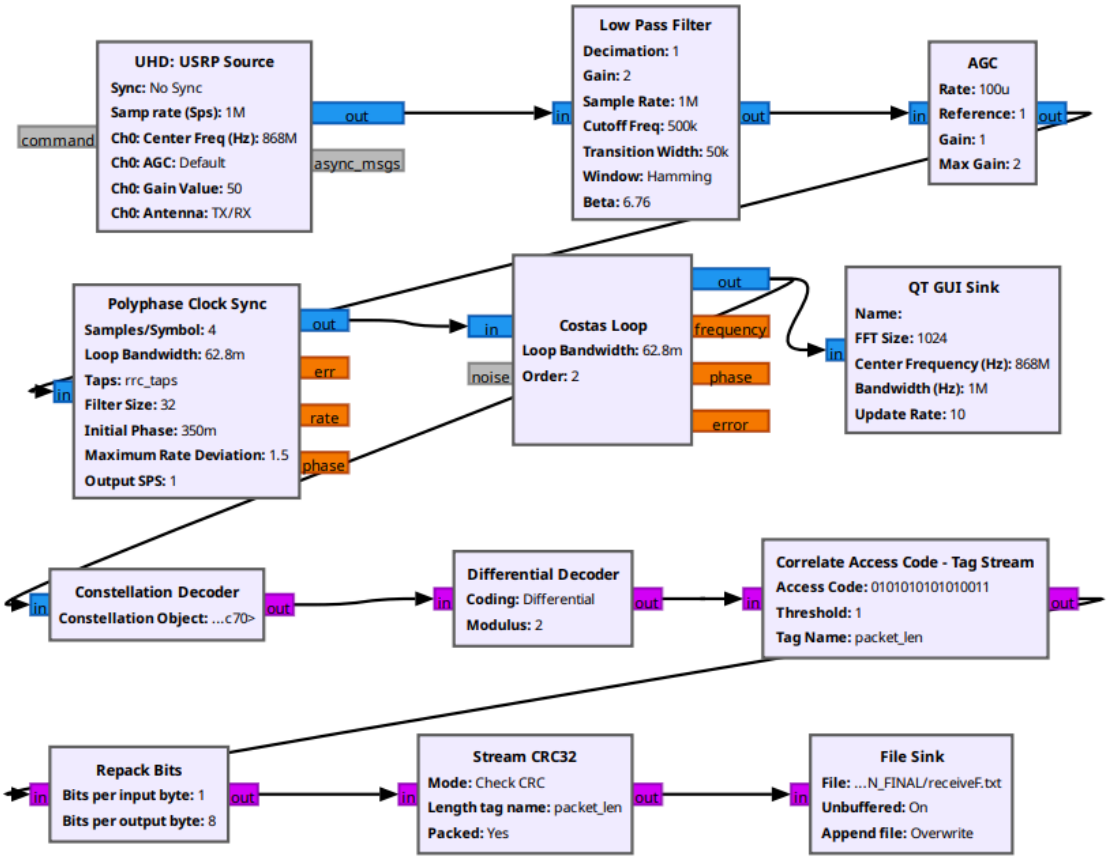



*Figure 3 : Packet Reception chain (RX) in GNU Radio*

Les blocs suivants forment une chaîne de traitement complète pour démoduler, synchroniser et récupérer des données numériques à partir d'un signal RF reçu par USRP :

- **USRP Source :** Le rôle de ce bloc est la réception du signal RF via l'USRP. On reçoit un signal dans la bande de 868 MHz avec une bande passante de 1 MHz, qui est assez typique pour des communications IoT.

- **Low Pass Filter** : Filtre passe-bas qui élimine les composantes haute fréquence indésirables du signal. Il permet d'isoler le signal d'intérêt en supprimant le bruit et les interférences hors bande. Les paramètres clés incluent la fréquence de coupure et la largeur de la bande de transition. Pour déterminer la fréquence de coupure, nous avons simplement utilisé le théorème de Shannon-Nyquist, à savoir une fréquence de coupure $F_{coupure}$ = samp_rate/2, soit 500 kHz. La largeur de la bande de transition dépend quant à elle de la bande utile du signal, or dans une modulation BPSK, la bande utile est proportionnelle au débit binaire, soit 250 kHz. Nous avons choisi de prendre 20% de la bande utile, soit 50 kbps.

- **AGC (anciennement Linear Equalizer) :** Le contrôle automatique de gain ajuste dynamiquement l'amplitude du signal pour maintenir un niveau constant en sortie, quelle que soit la variation du signal d'entrée. Cela compense les fluctuations dues aux conditions de propagation et normalise le signal avant la démodulation. L'AGC maintient l'amplitude du signal autour de 1, avec une adaptation rapide (100 µs) pour suivre les variations. Le gain ne peut pas dépasser 2 pour éviter l'amplification excessive du bruit.

- **Polyphase Clock Sync :** Ce bloc effectue la synchronisation d'horloge et le ré échantillonnage du signal. Le bloc reçoit un signal à 4 échantillons/symbole et le ré échantillonnage pour obtenir exactement 1 échantillon par symbole au moment optimal. Avec un débit symbole de 1M/4 = 250 kbauds, on transmet à 250 000 symboles/seconde.

- **Costas Loop :** Boucle de récupération de porteuse qui corrige les erreurs de fréquence et de phase du signal reçu.

- **Constellation Decoder :** Ce bloc prend les symboles complexes synchronisés et les convertit en bits en fonction de la constellation utilisée, dans notre cas BPSK. Il

associe chaque point du plan complexe à la combinaison de bits correspondante selon le schéma de modulation. Il est responsable de la démodulation.

- **Differential Decoder :** Le décodage différentiel inverse la transformation appliquée à l'émission. Chaque bit est calculé par XOR entre le bit reçu actuel et le précédent, éliminant l'ambiguïté de phase de 180°.

- **Correlate Access Code - Tag Stream :** Ce bloc recherche une séquence spécifique de bits (Access Code choisi : 010101010101010001 qui sert de marqueur ) dans le flux de données. Quand il détecte cette séquence, il insère un tag “packet_len” dans le flux pour marquer le début d'un paquet. Cela permet de synchroniser la trame et d'identifier où commencent les données utiles.

- **Repack Bits :** Ce bloc réorganise les bits en changeant la taille des mots. Il gère également l'ordre des bits (MSB/LSB first) et l'alignement, le but est de préparer les données pour les étapes suivantes du traitement.

- **CRC32 :** Le bloc calcule le CRC32 sur les données reçues entre les tags et le compare au CRC transmis. Si le CRC est correct, le paquet passe, sinon, il est éliminé.

- **File Sink :** Ce bloc permet de sauvegarder les paquets valides dans un fichier, chaque lancement du code écrase le fichier précédent avec les nouvelles données.

Ainsi, on reçoit le signal au niveau du bloc USRP Source, qui va être filtré et normalisé par les blocs Low Pass Filter et AGC . Le signal est ensuite synchronisé en temps et en phase par les blocs Polyphase Clock Sync et Costas Loop. Il est converti en bit et subit un décodage différentiel avec les blocs Constellation Decoder et Differential Decoder. Les bits obtenus sont analysés par le bloc Correlate Access Code qui va permettre la détection du début des trames envoyées et marque les paquets. Ces bits sont ensuite formatés en données exploitables avec le bloc Repack Bits, et le bloc CRC32 permet de vérifier l’intégrité des paquets reçus. On récupère ensuite les données à l’écrit dans un fichier texte grâce au bloc File Sink.

# 5. Design de la couche MAC

## 5.1. Contexte

La couche MAC (Medium Access Control) gère l'accès au support de transmission et arbitre les transmissions. Elle s'interpose entre la couche physique responsable notamment de la modulation, du codage canal, et de la transmission radio et les couches supérieures (liaison/applicative) qui génèrent les données à transmettre. Dans ce projet, nous implémentons une couche MAC de type ALOHA, entièrement en Python, intégrée dans GNU Radio via un bloc Python. Elle prend en charge l'encapsulation en trames, la gestion des collisions (par temporisation et retransmissions), et l'interface avec la couche physique.

## 5.2. Contexte du système

Dans le cadre de notre application, nous effectuons de la remontée d'informations en provenance de différents capteurs (luminosité, humidité, température, etc.). Notre système correspond donc à un réseau de capteurs dans lequel plusieurs nœuds capteurs transmettent des données vers un nœud collecteur (gateway), uniquement de manière montante (uplink). Les messages échangés sont courts et émis à une faible fréquence.

Ce type de trafic est donc asynchrone, irrégulier et non continu. Par ailleurs, les capteurs étant alimentés sur batterie, le réseau est soumis à une contrainte forte de faible consommation énergétique.

### 5.2.1. Objectifs de cette couche MAC

Le rôle primordial de la couche MAC est d'arbitrer l'accès au support de transmission partagé entre les multiples nœuds du réseau. Dans notre architecture en étoile où le trafic est majoritairement montant (uplink), elle doit permettre aux différents capteurs de transmettre leurs données vers la passerelle de manière autonome, sans nécessiter de synchronisation temporelle complexe ni de planification centrale rigide, inadaptées à la simplicité des nœuds.

Au-delà de l'accès, cette couche doit assurer la fiabilité des échanges dans un environnement non fiable. Elle a pour objectif de détecter et de gérer les pertes de paquets, qu'elles soient dues à des erreurs de transmission ou à des collisions entre plusieurs émetteurs simultanés. Pour ce faire, elle doit implémenter un mécanisme de contrôle reposant sur des acquittements (ACK) et des retransmissions automatiques, garantissant ainsi que les données critiques parviennent bien à destination.

Enfin, la conception de la couche MAC doit répondre impérativement à la contrainte d'efficacité énergétique. Le protocole choisi doit minimiser la complexité de traitement et la surcharge (overhead) des trames pour réduire le temps d'occupation du canal et l'énergie consommée par bit utile transmis. Il doit également éviter les phases d'écoute inutile du canal, permettant aux capteurs de rester le plus longtemps possible dans un état de veille pour préserver leur batterie.

## 5.3. Choix de conception

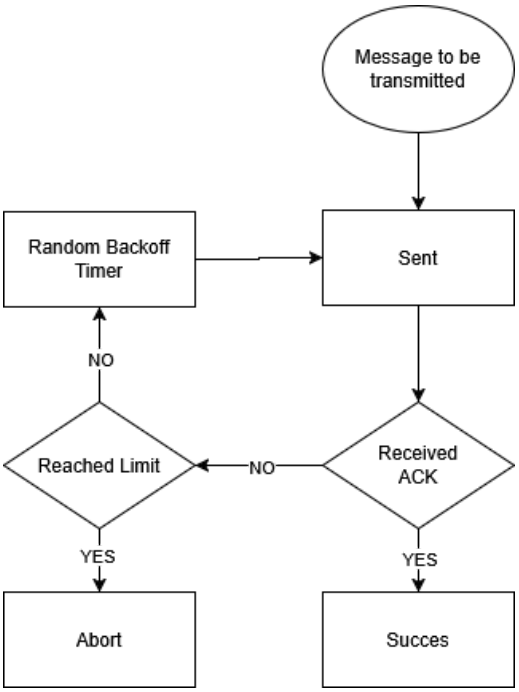
### 5.3.1. Pourquoi ALOHA ?

Le protocole ALOHA est bien adapté à ce contexte. En effet, les émissions des capteurs sont peu fréquentes : la température, l'humidité ou la luminosité sont des grandeurs à variation lente qui ne nécessitent pas de mesures en temps réel. Il existe donc une faible probabilité de collisions entre les transmissions des différents nœuds. Or, ALOHA est particulièrement efficace dans les réseaux à faible charge, ce qui le rend pertinent pour ce cas d'usage.

De plus, le trafic étant essentiellement unidirectionnel (uplink), l'utilisation d'ALOHA est pleinement envisageable, sans nécessiter de mécanismes complexes de gestion des échanges descendants.

### 5.3.2. Principe de fonctionnement de l'ALOHA MAC

Le protocole ALOHA est un protocole extrêmement simple. Chacun émetteur envoie ses paquets de données quand il le souhaite. Si un paquet de données est perdu à cause d'une collision avec un autre paquet, l'émetteur initial le retransmet après un temps aléatoire.



**Discussion énergétique**

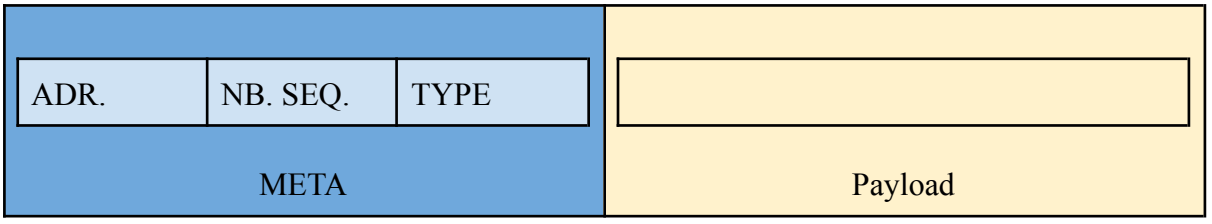
ALOHA ne nécessite pas d'écoute préalable du canal avant l'émission (contrairement aux protocoles de type CSMA comme la B-MAC). Cela permet de réaliser une économie d'énergie au niveau des capteurs, car les phases d'écoute radio sont réduites, or celles-ci sont fortement consommatrices d'énergie.
En revanche, ce fonctionnement impose une écoute quasi permanente de la gateway, ce qui entraîne une consommation énergétique plus élevée pour cette dernière. Toutefois, cette contrainte est acceptable, la gateway étant considérée comme alimentée sur secteur.

De plus l'ALOHA ne demande pas de synchronisation ce qui permet d'éviter de nombreux échanges entre les différents noeuds

**Limite de cette méthode**
Le protocole ALOHA présente néanmoins des limites. Il n'est pas optimal lorsque le trafic devient élevé, notamment en présence d'un grand nombre de nœuds actifs simultanément ou lorsque les exigences en débit augmentent. Dans ces situations, le taux de collisions croît, ce qui dégrade les performances du réseau.

### 5.3.3. Construction d'une trame

Nous avons choisi de générer les trames de la façon suivante, chaque trame est composée de 2 parties, les métadonnées comportant divers informations le message envoyé, et la partie charge utile comportant les données que l'on souhaite envoyer.




La partie méta données est sous la forme d'une dictionnaire structuré de la manière suivante :

- **Adresse MAC**

Ce champ permet d'identifier l'émetteur du paquet et permet ainsi au récepteur de savoir à qui répondre afin d'envoyer l'acquittement au bon nœud.

- **Numéro de séquence**

Le numéro de séquence permet d'associer un acquittement au bon paquet. Et également de détecter les duplicatas (ex : dans le cas d'un ACK perdu l'émetteur va retransmettre le message ; le récepteur recevra ainsi deux fois la même donnée).

- **Type de message** ("Data" ou "ACK")

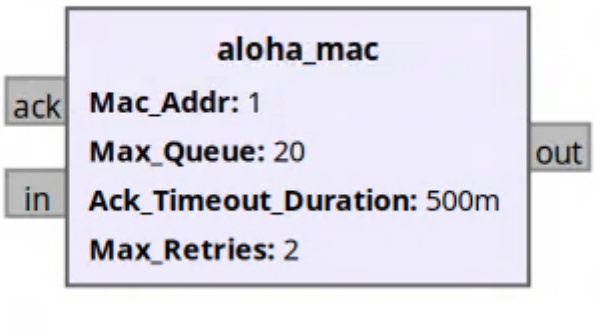
Qui permet de distinguer les PDUs d'acquittement et de données.

## 5.4. Fonctionnement de la fonction ALOHA MAC

Nous avons implémenté la couche MAC en utilisant un bloc python de GNU Radio, ce dernier permet d'émettre et recevoir des message au travers de ses ports. Les paquets utilisés dans ce bloc sont des PDUs (Protocol Data Unit). C'est un type de structure de données utilisé dans GNU Radio. un PDU est constitué de deux parties principales (meta, payload) :

- **Métadonnées :** est un dictionnaire PMT qui peut contenir diverses informations sur les données envoyées (l'heure d'émission, la fréquence, le numéro de séquence etc..)
- **Payload :** contient les données utiles (le message émis par l'application)

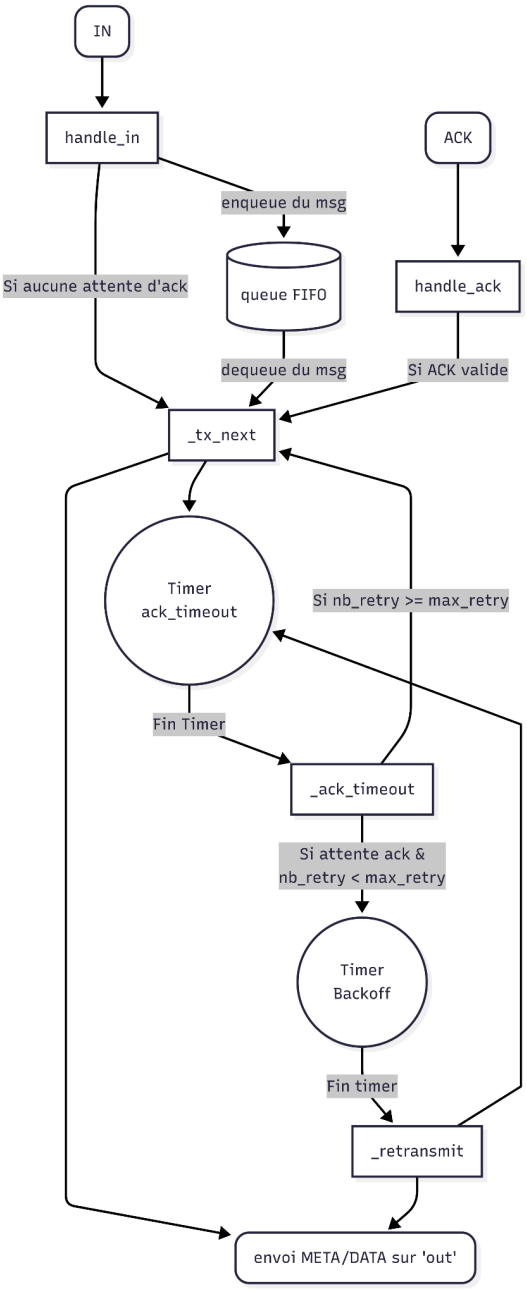
### 5.4.1. Structure générale du bloc




Il s'agit d'un bloc python GNU Radio comportant les ports messages suivants :

- **in :** PDU provenant de l'application (couche supérieur) dans notre cas un simple message
- **out :** PDU sortant vers la couche physique (couche inférieur)
- **ack** : PDU ACK entrant depuis la couche physique

### 5.4.2. Fonctionnement général




**Réception d'un paquet "DATA" : handle_in**

Cette fonction est déclenchée lorsqu'un message est réceptionné sur l'entrée "in" elle a pour rôle premier de stocker donnée reçu dans une FIFO qui permet de conserver un message qui serait reçu lors de l'attente d'un acquittement. Si un nouveau message arrive alors que la FIFO est pleine ce dernier est alors supprimé et l'utilisateur est averti. Cette fonction déclenche également la fonction de traitement du message à condition de ne pas être dans l'état d'attente d'un acquittement (waiting_ack).

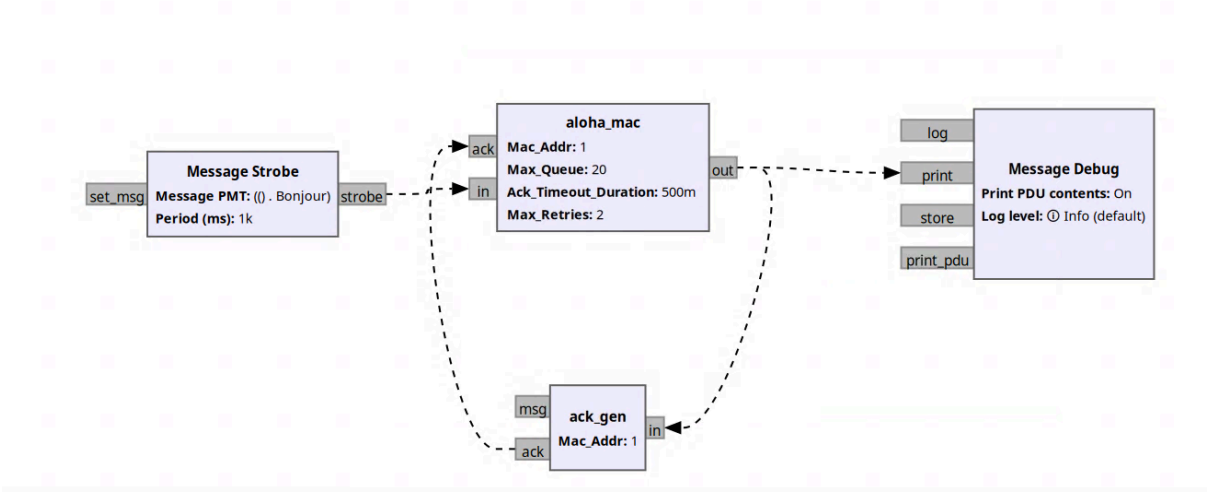
Le fonctionnement du protocole repose sur une machine à trois états principaux qui orchestre l'envoi et la fiabilité des données. Initialement, le système se trouve dans un état de repos (IDLE), où il est prêt à traiter

de nouvelles informations provenant de l'application. Lorsqu'une donnée est reçue, le système déclenche immédiatement son émission sur le canal radio et bascule dans un état d'attente (WAIT_ACK), activant simultanément un chronomètre pour surveiller la réception d'une confirmation. Si un acquittement (ACK) valide parvient au système avant la fin de ce délai, la transmission est considérée comme un succès et la machine retourne immédiatement à l'état de repos, prête pour le message suivant.

Cependant, si le délai d'attente expiré sans réception d'acquittement, le système doit gérer l'échec potentiel. Deux cas de figure se présentent alors selon le nombre de tentatives déjà effectuées. Si le nombre maximal d'essais n'est pas encore atteint, le système passe dans un état de pause temporaire (BACKOFF), où il attend durant un délai aléatoire afin d'éviter de nouvelles collisions immédiates. Une fois ce temps écoulé, il émet à nouveau le message et retourne à l'état d'attente. En revanche, si le compteur de tentatives a atteint la limite autorisée lors de l'expiration du délai, le système considère l'envoi comme un échec définitif, abandonne le paquet actuel et revient à l'état de repos pour traiter la suite de la file d'attente.

## 5.5. Test du bloc ALOHA MAC

Afin de valider le comportement logique de notre couche MAC avant son intégration avec la couche physique, nous avons réalisé une série de tests fonctionnels en simulation sous GNU Radio Companion. L'objectif était de vérifier la machine à états implémentée en Python, en particulier la bonne gestion des files d'attente, le déclenchement des timers et le mécanisme de retransmission en l'absence d'acquittement. Pour ce faire, nous avons conçu un flowgraph de test isolant la logique MAC des contraintes radio réelles.



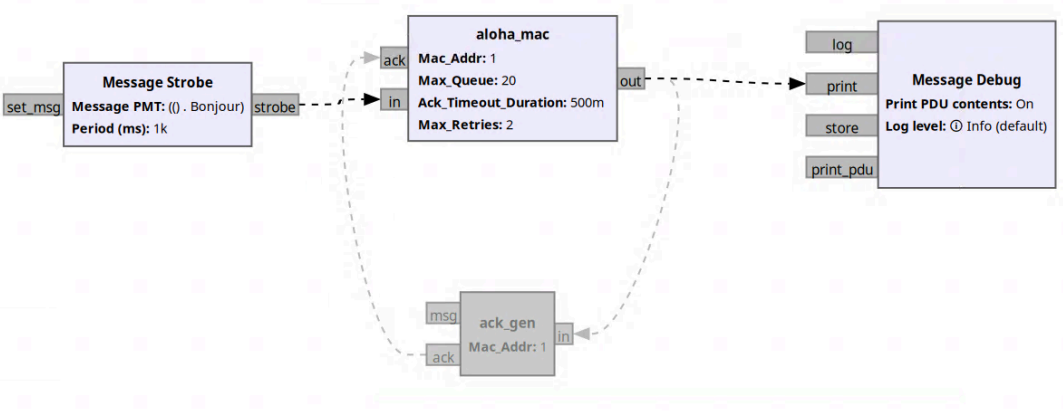

Dans cette configuration de test, un bloc "Message Strobe" simule la couche applicative en générant périodiquement (toutes les 1000 ms) un message contenant la chaîne de caractères "Bonjour". Ce bloc est connecté à l'entrée de notre bloc "aloha_mac". Pour boucler le système et simuler le récepteur distant, nous utilisons un bloc auxiliaire nommé "ack_gen" (ou une boucle de retour directe dans certains tests unitaires). Ce dernier a pour rôle de renvoyer un acquittement valide dès réception d'une trame de données. Les sorties sont connectées à un bloc "Message Debug" pour visualiser en temps réel dans la console le contenu des PDUs (Protocol Data Units) et leurs métadonnées.

Le premier scénario de test valide le fonctionnement nominal, c'est-à-dire une communication sans perte où chaque envoi est immédiatement suivi d'un acquittement. Comme l'illustrent les traces d'exécution ci-dessous, le bloc MAC émet les messages les uns après les autres. Nous observons que le numéro de séquence ("seq") s'incrémente de manière linéaire (seq 1, seq 2, seq 3, etc.) à chaque nouvelle ligne de log. Cela confirme que le bloc reçoit correctement l'ACK, quitte l'état d'attente "WAIT_ACK" et passe au traitement du message suivant dans sa file d'attente sans déclencher de retransmission inutile.

```
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 1) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 2) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 3) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 4) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 5) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 6) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 7) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 8) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 9) (src_mac . 1)) . Bonjour)
**********************************
```

Le second scénario met à l'épreuve la robustesse du protocole face aux pertes de paquets. En simulant l'absence de réponse (par exemple en déconnectant le générateur d'ACK ou en ignorant les messages entrants), nous forçons le bloc MAC à entrer en mode de récupération d'erreur. Les logs ci-dessous montrent clairement ce comportement : une même trame, identifiée par un numéro de séquence identique (ex: "seq 1"), est émise à plusieurs reprises. Cela démontre que le timer "ack_timeout" expire comme prévu, déclenchant la fonction de retransmission après le délai de "backoff" aléatoire, jusqu'à ce que le nombre maximal d'essais soit atteint ou qu'un acquittement soit enfin reçu.

```
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 1) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 1) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 1) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 2) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 2) (src_mac . 1)) . Bonjour)
**********************************
******* MESSAGE DEBUG PRINT ********
(((type . DATA) (seq . 2) (src_mac . 1)) . Bonjour)
**********************************
```



# 6. Test des couches physique et MAC

## 6.1. Protocoles

Au niveau de la couche physique, les tests ont consisté en l'assurance d'une émission et réception de paquets de données dans un milieu rayonné, sous l'utilisation d'un bloc CRC. Ainsi, nous avons envoyé la chaîne de caractère suivante :

"Bonjour, je m'appelle Florent1Bonjour, je m'appelle Florent2Bonjour, je m'appelle Florent3"

Les paquets envoyés doivent être de taille "packet_len", qui doit avoir la même valeur que le nombre de caractères du paquet. La chaîne ci-dessus contient 90 caractères, et on veut envoyer des paquets de 30 caractères, afin de recevoir les phrases séparées par les nombres. Dans notre cas, au vu de l'utilisation des blocs CRC, nous avons dû rajouter 10 à la valeur du packet_len pour prendre en compte la taille du CRC. Le fonctionnement du bloc File Source fait que le premier paquet est toujours omis, donc lorsque l'on lance le récepteur puis l'émetteur, on reçoit logiquement "Bonjour, je m'appelle Florent2", ce qui est le cas.

## 6.2. Difficultées rencontrées

Nous avons été face à trois grands murs dans ce bureau d'étude, dont deux qui ont pu être franchis. Le premier a été au niveau de la couche physique, lors de l'envoi et de la réception correcte d'une chaîne de caractère contenu dans un fichier texte (File Source en mode repeat) dans un milieu rayonné. En effet nous avons décidé de nous lancer directement en rayonné en premier lieu car cela correspondait fortement à notre contexte, et aussi pour ne pas à avoir à adapter le code GNURadio et donc soulever la possibilité d'avoir de nouveaux problèmes après une éventuelle transition local-> rayonné. Le second mur traite également de la couche physique et de l'envoi/réception d'un fichier texte, mais cette fois-ci incluant un nouveau bloc CRC et le mode repeat désactivé, l'idée étant de ne recevoir que le trafic utile et aucun bruit. Comme GNURadio offre déjà une logique CRC, nous avons préféré l'utiliser directement au niveau de la couche physique et de l'omettre de la couche MAC. Enfin, le dernier mur que nous n'avons pas pu franchir est le plus important, la liaison entre couche MAC et couche physique, permettant au deux niveaux de fonctionner ensemble.

## 6.3. Améliorations envisageable

**Couche physique :**

La principale amélioration à apporter la couche physique serait l'affinage et l'optimisation des paramètres utilisés pour qu'il soit plus adapté au contexte de notre projet. En effet, afin de régler les problèmes rencontrés rapidement, nous avons pris des valeurs aux limites théoriques comme par exemple $F_{coupure}$ = 500kHz pour une samp_rate de 1MHz, alors que la bande utile du signal est de 250 kHz. De ce fait, nous laissons une grande partie bruitée du signal passer au-dessus du filtre, ce qui va à l'encontre d'une approche basse consommation.

# 7. Conclusion

Ce projet a permis de réaliser une chaîne de transmission complète pour un réseau de capteurs de jardin intelligent, en exploitant la flexibilité de la radio logicielle sur plateformes USRP. Cette approche pratique nous a confrontés aux contraintes réelles d'un système de télécommunications, nous permettant de maîtriser l'ensemble de la pile protocolaire, de la modulation du signal à la gestion de l'accès au canal.

Nos choix techniques ont privilégié la robustesse et la simplicité pour répondre aux exigences énergétiques. L'utilisation de la modulation BPSK assure une liaison fiable dans un environnement bruité, tandis que le protocole MAC ALOHA s'adapte parfaitement au trafic sporadique des capteurs. L'implémentation d'une machine à états spécifique via des blocs Python dans GNU Radio a validé avec succès la logique de contrôle, garantissant la fiabilité des échanges par un mécanisme d'acquittement.

Bien que fonctionnelle, cette architecture présente des limites, notamment une sensibilité accrue aux collisions si le nombre de nœuds augmente fortement. Si ce prototype valide les concepts fondamentaux, un déploiement à grande échelle bénéficierait d'améliorations telles que l'écoute du support (CSMA) ou l'ajout de codes correcteurs d'erreurs pour optimiser davantage la portée et la résilience du réseau.

# Sources

https://wiki.gnuradio.org/index.php/Message_Passing

https://wiki.gnuradio.org/index.php/Main_Page