

## Performance des mécanismes de sécurité du framework 6TiSCH

Mémoire de fin d'études  
par  
Rémy DECOCQ

Directeur : M<sup>r</sup> Bruno QUOITIN  
Service des Réseaux et Télécommunications

Année Académique 2019-2020



## **Remerciements**

Au terme de la rédaction de ce travail de fin d'études, je souhaite exprimer toute ma gratitude au personnel encadrant et à mes proches qui m'ont soutenu, contribuant de près ou de loin à ce mémoire.

Ma plus profonde reconnaissance à mon professeur et encadrant principal M<sup>r</sup> Bruno Quoitin, dont le suivi actif et les relectures attentives ont permis de mener ce travail à bien. Mes remerciements à David Hauweele dont l'implication volontaire a permis de concrétiser l'obtention de résultats.

Enfin, mes remerciements aux rapporteurs et au jury pour l'attention portée à l'évaluation de ce travail.



# Performance des mécanismes de sécurité du framework 6TiSCH

Rémy DECOCQ

## RÉSUMÉ

Les réseaux de capteurs sans-fils sont au cœur des applications industrielles de l’Internet des Objets (*Industrial IoT*). Ces réseaux présentent la particularité d’être composés de noeuds restreints en ressources et d’être déployés dans des environnements où les communications radios sont rendues difficiles. Dans de telles conditions, l’utilisation d’une pile réseau classique telle que *TCP/IP* n’est pas envisageable. Depuis 2013, le groupe de travail 6TiSCH à l’IETF travaille à la standardisation d’une pile adaptée à de tels réseaux. Cette pile 6TiSCH, bien que différente de la pile TCP/IP, permet des communications IP et donc une interconnectivité du réseau de capteur avec Internet.

Ce travail établit la structure typique d’un réseau dans lequel peut être déployée la pile. Une analyse de la pile 6TiSCH dans sa globalité est menée, dépeignant chaque couche qui la compose et les concepts qui y sont définis, ainsi que l’interaction entre ces couches. Ayant introduit ces différentes couches essentielles à sa compréhension, une attention particulière est portée au mécanisme qui permet à un nouveau noeud de rejoindre un réseau 6TiSCH actif. Cette *joining phase* est analysée en détails du point de vue de la sécurité. Effectivement, la joining phase est un procédé critique en terme de performances et sécurité. Après avoir introduit les solutions proposées par les standards, un nouveau schéma désigné par “méthode NPEB” est proposé pour améliorer les performances de la joining phase. Une série d’expérimentations en simulateur sont alors menées pour d’une part évaluer la joining phase standard avec et sans mécanisme de sécurité et d’autre part mesurer les gains de performances la méthode NPEB proposée.

**Mot-clés :** WSN, Industrial IoT, TSCH, pile 6TiSCH, joining phase



# Table des matières

# Introduction

Les réseaux de capteurs sont devenus monnaie courante dans l'industrie, on estime en 2020 que plusieurs dizaines de milliers de réseaux ont été déployés, comptabilisant plus de 18 milliards d'heures de fonctionnement. Originellement, ces réseaux fonctionnaient sur des standards propriétaires tels que WirelessHART et ISA100.1a [?]. Bien que capables de performances acceptables en pratique grâce à l'emploi du principe TSCH (*Time Slotted Channel Hopping*) [?], ces piles réseaux n'offrent pas un connectivité IP et ne s'inscrivent donc pas dans la tendance générale de l'IoT, dont l'interconnectivité avec Internet est une ligne directrice [?]. Le groupe de travail 6TiSCH est créé à l'IETF en 2013 avec pour double objectif de standardiser l'adoption de TSCH dans les réseaux de capteurs 802.15.4 et d'élaborer un pile réseau supportant les communications IP pour ces réseaux restreints. L'ensemble de spécifications produites constitue ce que l'on appelle la pile 6TiSCH.

Ce travail caractérise ces réseaux sans-fil restreints, leur architecture typique et les contraintes qui justifient l'élaboration d'une pile réseau adaptée. L'intégralité des différentes couches qui composent la pile 6TiSCH sont passées en revue, ainsi que la façon dont elles interagissent entre elles pour constituer une pile déployable et efficace dans un réseau sans-fil restreint. Une attention particulière est portée à la *joining phase 6TiSCH*, qui dénote la façon dont un nouveau noeud rejoint un réseau 6TiSCH déjà actif. Plus particulièrement, la sécurisation de cette procédure, qui est une phase critique, est analysée en détail. Des expérimentations sont menées en simulateur pour illustrer l'impact de la sécurité sur la joining phase, et une méthode est proposée pour améliorer les performances de la joining phase.

Ce document est divisée en deux parties principales. La première partie constitue un état de l'art de la pile 6TiSCH dans sa globalité et des caractéristiques des réseaux dans lesquels elle est destinée à être déployée. Le Chapitre 1 introduit les réseaux sans-fil restreints, l'évolution des standards et groupes de travail y contribuant et la technologie TSCH qui est le fondement de la pile 6TiSCH. Le Chapitre 2 précise l'architecture typique des réseaux considérés et établit une analyse couche par couche de la pile 6TiSCH et des principes qui lui sont propres. Finalement, le Chapitre 3 se concentre sur la joining phase qui fait intervenir les différentes couches présentées précédemment, en mettant l'accent sur les mécanismes de sécurité qui la protègent. La seconde partie fait l'état d'un travail expérimental sur la joining phase. Le Chapitre 4 présente la méthode NPEB, qui est une proposition de nouveau schéma pour l'amélioration des performances de la joining phase. Le Chapter ?? concentre les expérimentations effectuées sur la joining phase en simulateur : des métriques pertinentes sont sélectionnées pour d'une part illustrer l'impact de la sécurité sur celle-ci et d'autre part démontrer les gains de performance apportés par la méthode NPEB.

## **Chapitre 1**

# **Présentation du standard IEEE802.15.4 et TSCH**

## 1.1 L'IoT, les réseaux sans fils et leurs contraintes

L'Internet de Objets (*Internet of Things - IoT*) couvre de nombreux domaines d'applications tels que les *smart-homes/buildings/cities*, les soins de santé et les transports entre autres. Les équipements que l'on regroupe généralement derrière cette dénomination sont de nature variée : capteurs et actionneurs, périphériques de communication/multimédias, domotique, etc. mais s'accordent sur un point : ils sont connectés et tendent à l'être avec le réseau Internet. Également, comme ce sont généralement des équipements embarqués, ils sont restreints en ressources telles que l'énergie (alimentation par batterie), la puissance d'émission radio, la capacité mémoire et les performances CPU. Ces limitations ont conduit à la conception et la standardisation de nombreux protocoles qui y sont adaptés afin de rendre possible et efficace la communication en réseau de ces objets.

Cela est particulièrement vrai pour l'IoT (*Industrial IoT*) qui regroupe des secteurs tels que l'industrie agricole, l'automatisation des usines, la gestion des stocks, etc. Effectivement, l'optimisation apportée par ces protocoles adaptés est un facteur clé pour augmenter la rentabilité du déploiement de tels réseaux. Ils permettent également de déployer de façon plus efficace des réseaux de capteurs sans fils, dits WSNs (*Wireless Sensors Networks*), qui sont a priori peu adaptés aux contraintes des systèmes embarqués, notamment au niveau de la consommation énergétique imputée aux transmissions radios qui sont coûteuses. Les protocoles de communication qui orchestrent les WSNs sont donc conçus dans une optique de compromis entre la minimisation de la consommation énergétique des nœuds qui les composent et la fiabilité désirée de ces communications (délais, bande passante et déterminisme). La pile réseau 6TiSCH étudiée dans ce document est un ensemble de protocoles et principes qui apportent une réponse concrète à ces besoins.

Les WSNs étant des systèmes de communication restreints, ils font partie de ce qu'on appelle les LLNs (*Low-power and Lossy Networks*). L'implémentation d'une pile réseau pour les LLNs doit tenir compte de leurs caractéristiques qui sont principalement les suivantes :

- Présence du phénomène de *multipath fading* lié à la réflexion d'ondes surtout en environnement clos et métallique (usine) illustré par la Figure 1.1 et interférences extérieures/d'autres nœuds proches
- Forte densité de nœuds entraînant des interférences et possiblement déployés de façon imprécise à des endroits non déterminés
- Nœuds limités en ressources énergétiques, pour lesquels il est donc très coûteux de laisser leur radio allumée
- Radios peu puissantes, taux de transmission et portée faibles entraînant parfois le besoin de transmissions *multi-hop*
- Nœuds limités en ressources CPU, mémoire et stockage
- Changements réguliers dans la topologie, mobilité des nœuds et décalages entre leur horloges respectives

À titre d'exemple, un nœud TelosB (illustré par la Figure 1.2) typiquement utilisable dans un WSN mais considéré comme peu performant, présente les caractéristiques suivantes : microcontrôleur 16-bit et processeur cadencé à 8 MHz, disposant de 10 kB de RAM et 48 kB de mémoire flash. Il est couplable avec une radio CC2420 supportant le standard 802.15.4, opérant dans la bande de fréquence 2.4-2.485 GHz. Cette radio draine 25.8 mA en transmettant et 18.8 mA en recevant. Si la source d'alimentation du nœud est comparable à celle d'une paire de piles AA (3000 mAh de charge), et que la radio reste allumée en écoutant le canal sans arrêt alors le nœud vivra  $\frac{3000 \text{ mAh}}{18.8 \text{ mA}} \approx 159 \text{ h}$ . Cette hypothèse revient à considérer un *duty cycle* (portion de temps pendant laquelle la radio est allumée) de 100%. Selon [?], une pile de communication efficace devrait aboutir à un duty cycle bien en dessous de 1%. On porterait alors la durée de vie du nœud à  $\frac{3000 \text{ mAh}}{18.8 \text{ mA} \times 1\%} \approx 160000 \text{ h} \approx 22 \text{ mois}$ .

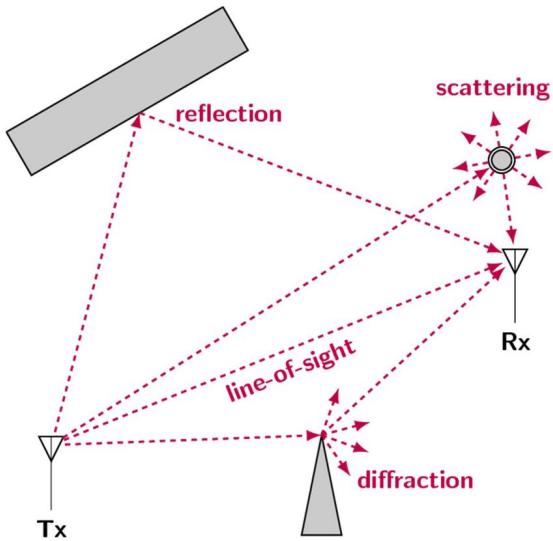


FIGURE 1.1 – Phénomène de multipath fading[?]

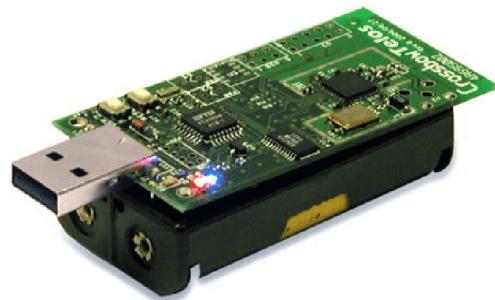


FIGURE 1.2 – Micro-controleur TelosB, Texas Instruments [?]

La technologie TSCH (détaillée dans la section 1.3) apporte une réponse efficace aux problématiques des WSNs induites par leurs caractéristiques évoquées plus haut. Les optimisations apportées par TSCH aux WSNs permettent d'envisager des grandeurs de durée de vie du réseau de l'ordre de la décennie [?], se traduisant donc par des gains conséquents pour les industries. Ainsi, de nombreux standards et piles réseaux (propriétaires ou non) ont été développés en se basant sur la pierre d'angle TSCH. 6TiSCH en fait partie intégrante. De façon plus généralisée, un pile réseau qui est destinée à faire communiquer des équipements restreints dans la vision actuelle de l'(I)IoT devrait présenter les qualités suivantes [?] :

- Une consommation minimale de l'énergie et des communications limitées au strict nécessaire
- Une fiabilité maximale des transmissions par rapport aux besoins et à la qualité de service attendue
- Une transparence des communications vers les réseaux extérieurs, sous-entendu l'Internet [?]

## 1.2 L'évolution des standards - amendement 802.15.4e et 6TiSCH

En 2013, l'IEEE Standards Association sort un standard de communication définissant des couches physique et lien (PHY et MAC) adaptées aux communications radios dans les LLNs. Le standard IEEE802.15.4 devient alors le standard *de facto* utilisé dans ces environnements restreints [?], prodiguant une base solide par dessus laquelle de nombreuses piles réseaux complètes (majoritairement propriétaires) se développent. Ainsi, en 2004 la ZigBee Alliance distribue la pile Zigbee 1.0, qui est l'implémentation d'une spécification propriétaire calquant les plus bas niveaux de sa pile sur le standard IEEE802.15.4 PHY/MAC. Elle se base sur une topologie en étoile capable de *multi-hop*, utilisant les notions de coordination entre PANs (Personal Area Network) et de différenciation des types de nœuds en fonction de leur (in)capacité à servir de coordinateur entre PANs ou entre nœuds [?].

Cependant, cette conception liée au standard 802.15.4 souffre de certaines limitations parmi lesquelles :

- le canal de transmission ne varie pas dans le temps, exposant les communications à des interférences sans pouvoir y réagir si elles restent constantes dans ce canal
- la synchronisation temporelle partielle : il est possible à un nœud de réservé un moment dans le temps où il pourra émettre sans risque de collision, mais dans tous les cas il devra faire usage de CSMA/CA, ce qui est coûteux en terme d'énergie
- les nœuds faisant office de coordinateur sont actifs tout du long de la période d'activité prévue pour qu'un nœud normal entame une communication
- les communications avec un réseau IP (donc Internet) ne sont pas gérées

Pour passer au dessus de certaines de ces limitations, un nouveau standard baptisé TSMP (*Time Synchronized Mesh Protocol*) est commercialisé en 2004 par la société Dust Networks. Ce dernier apporte une révolution dans la conception des paradigmes de communication propres aux LLNs : la combinaison des principes de TDMA (*Time Division Multiple Access*) et FDMA (*Frequence Division Multiple Access*). Cela permet de combattre les phénomènes de multipath fading et les interférences tout en synchronisant les nœuds sur une base temporelle commune, maximisant l'utilisation de leurs périodes de réveil. Le concept désigné par la combinaison de ces deux principes est appelé TSCH (*Time Slotted Channel Hopping*) et est discuté dans la section 1.3.

Plusieurs piles réseaux ont alors vu le jour, se reposant sur la fiabilité que TSCH apporte aux communications dans les LLNs. En 2007, le standard WirelessHART est publié, suivi par ISA100.11a en 2008. Ils deviennent des standards utilisés *de facto* dans l'IoT, malgré le fait qu'ils soient propriétaires [?]. Cependant, en 2008 le groupe de travail (WG - *Working Group*) IEEE802.15.4e est fondé avec pour mission de redéfinir la couche MAC de la spécification 802.15.4 pour y intégrer les principes de TSCH, de façon transparente du point de vue de la couche PHY (aucun changement hardware requis). Ainsi, l'amendement 802.15.4e est publié en 2011.

Bien que l'amendement 802.15.4e définit les concepts de TSCH et décrive la façon dont ils peuvent coexister avec le standard IEEE802.15.4 classique, il ne donne pas toutes les réponses nécessaires pour déployer une pile réseau opérationnelle qui réponde aux 3 exigences dégagées en fin de la Section 1.1. Par exemple, aucun mécanisme n'est décrit pour satisfaire des communications avec un réseau extérieur non restreint (Internet), la façon dont les nœuds apprennent leurs périodes de réveil n'est pas spécifiée non plus. Plusieurs groupes de travail ont vu le jour à ces fins, résultant de la collaboration entre l'IEEE et l'IETF pour aboutir à une pile réseau s'inscrivant dans la vision idéale de l'IoT décrite plus haut.

Le WG 6LoWPAN (*IETF IPv6 over Low power WPAN*) a commencé son travail en 2007 (donc en se basant sur IEEE802.15.4), définissant une couche d'adaptation d'IPv6 aux réseaux restreints, afin d'établir un pont entre le réseau Internet et les LLNs. Le but étant de permettre des communications IPv6 transparentes au-dessus de la couche réseau entre un nœud restreint d'un WSN et une machine quelconque connectée à Internet. Cela revient entre autres à définir (voir la Section 2.2.5) :

- un schéma de fragmentation et compression des paquets IPv6 trop lourds et contenant des informations redondantes, déductibles ou inutiles
- l'autoconfiguration des adresses IPv6 des interfaces du nœud
- des mécanismes de sécurité adaptés

Au niveau du routage, le groupe ROLL (*Routing Over Low power and Lossy Networks*) fondé en 2008 est chargé de conceptualiser un modèle de routage adapté aux LLNs adressés en IPv6 (se reposant sur 6LoWPAN). Le protocole de routage standardisé qui met en application ces principes s'appelle RPL. Il s'agit de définir entre autres (voir la section 2.2.6) :

- une topologie multihop capable de s'adapter rapidement aux changements (liens non fiables, nœuds mobiles ou défaillants) selon différentes métriques
- différents types de trafic envisageables dans l'architecture typique d'un WSN (Section 2.1) : MP2P (Multi-Point to Point), P2P et P2MP
- l'utilisation des facilités apportées par 6LoWPAN (messages de signalement portés par ICMPv6)

Le groupe de travail CORE (*Constrained RESTful Environments*) établi en 2010 a la charge des couches supérieures : dans l'optique de communications avec Internet où les applications tournent sans les contraintes d'équipements restreints, il est nécessaire d'adapter celles-ci pour permettre l'interopérabilité. Il en va de même pour les protocoles de transport utilisés traditionnellement : là où TCP permet d'assurer une fiabilité des transmissions, un contrôle de flux et de congestion, il est aussi coûteux en ressources. Dès lors, CORE travaille à l'élaboration du protocole CoAP (*Constrained Application Protocol*), destiné à supplanter HTTP pour les équipements restreints, tout en restant interopérable avec et se reposant sur un transport par UDP. La section 2.2.7 détaille le fonctionnement de CoAP.

Finalement, le groupe 6TiSCH (*IPv6 over the TSCH mode of IEEE802.15.4e*) est fondé en 2013 avec pour objectif de rassembler toutes les pièces du puzzle, standardisant les mécanismes manquant à la pile issue de l'agrégation du travail des différents groupes. Ainsi, la pile 6TiSCH est standardisée avec les objectifs suivants :

- intégrer IEEE802.15.4e en mode TSCH au framework IPv6 dans les LLNs, constitué par RPL au dessus de 6LoWPAN
- apporter une réponse au besoin de dissémination de l'information relative aux périodes de réveil des noeuds (distribution de la synchronisation temporelle) et la consistance de cette information entre les nœuds du réseau souhaitant communiquer
- spécifier comment construire de façon efficace l'ordonnancement et l'agencement de ces périodes de réveil pour chaque nœud du réseau, de façon centralisée ou distribuée
- décrire les différentes étapes qu'un nouveau nœud doit suivre pour rejoindre un réseau 6TiSCH déjà opérationnel et y planifier ses périodes de réveil
- encadrer la sécurité dans le réseau 6TiSCH à différents niveaux :
  1. distinguer quelles sont les clés nécessaires pour chiffrer/authentifier le trafic et les façons dont un nœud peut rejoindre un réseau en toute sécurité, vis-à-vis de lui-même et du réseau

2. sécuriser les communications relatives au trafic applicatif traversant le WSN
3. sécuriser les communications propres aux mécanismes mis en place dans 6TiSCH d'un nœud à l'autre du WSN

Le groupe 6TiSCH est encore actif et continue de publier des documents pour standardiser les notions propres à la pile réseau 6TiSCH. Une grande différence avec les piles propriétaires WirelessHART et ISA100.11a réside dans la façon dont est construit l'ordonnancement des périodes de réveil [?]. Là où ces derniers ont fait le choix de construire de façon centralisée par une entité dédiée, 6TiSCH met en place des mécanismes pour le construire de façon distribuée (définition d'une nouvelle couche intermédiaire appelée 6top, détaillée à la Section 2.3.3). L'avantage de la première solution réside dans son approche plus déterministe permettant un meilleur contrôle sur la qualité de service et l'aspect déterministe des transmissions, mais nécessite un NME (Network Management Entity) capable de gérer ce travail pour tout le réseau [?]. L'avantage de l'approche distribuée réside dans l'autonomie du déploiement et la capacité du réseau à s'adapter dynamiquement, mais peut impacter la fiabilité des communications si l'ordonnancement local est mal géré. À noter que 6TiSCH n'exclut pas la possibilité d'utiliser l'approche centralisée, et définit déjà des concepts permettant de correspondre au modèle d'architecture DetNet [?] qui se repose sur le concept de *Software Defined Networking*. Cependant, la standardisation n'étant pas encore complétée, il ne sera question que d'approche distribuée dans le reste de ce document.

### 1.3 Les principes fondamentaux de TSCH

Le concept de TSCH (*Time Slotted Channel Hopping*) se repose sur la combinaison de deux technologies qui étaient déjà utilisées indépendamment :

- **TDMA** (*Time Division Multiple Access*) : permet de transmettre plusieurs flux de trafic sur un même support de transmission en établissant une division temporelle de son utilisation. Une synchronisation temporelle entre les parties communicantes doit donc être partagée afin que chacune puisse émettre dans les périodes de temps qui lui sont assignées. Cela permet donc de [?]:
  1. synchroniser les communications entre nœuds et donc optimiser les dépenses d'énergie liées aux réveils
  2. envisager des propriétés déterministes pour les communications dans le réseau
- **FDMA** (*Frequency Division Multiple Access*) : utilisation d'un découpage en bandes de fréquences afin de multiplexer l'usage du spectre pour plusieurs flux de trafic simultanés. L'information de quelle bande est attribuée à quel flux à un moment donné doit être partagée entre les parties communiquant pour ce flux. Cela permet donc de :
  1. augmenter la capacité du réseau par des transmissions simultanées sur des fréquences distinctes
  2. augmenter la fiabilité du lien en minimisant les interférences et le multi-path fading [?]

En pratique, dans TSCH, les nœuds se synchronisent temporellement sur une structure de *slotframe* répétée continuellement qui est composée d'unités de temps de même durée appelées (*time*)*slots*. Une slotframe est caractérisée par sa taille, un identificateur unique et une valeur de priorité, plusieurs slotframes pouvant co-exister pour un même nœud (pour gérer plusieurs types de trafic par exemple). Pour un moment donné dans le temps (un timeslot), une slotframe ne référence qu'une seule action parmi les trois suivantes :

1. Dormir (*Sleep*) c'est à dire ne rien faire et laisser la radio éteinte pour la durée du slot
2. Transmettre (*Tx*) par radio une frame s'il y en a une valide en attente
3. Recevoir (*Rx*) par radio une frame prévue pour être transmise durant le slot par un autre nœud

Pour savoir quoi faire à chaque slot de temps venu, un nœud suit le *schedule* calculé à partir des actions référencées pour ce slot parmi ses slotframes actives, les règles de priorité permettant la sélection d'une seule action s'il y a un conflit. Ainsi, le schedule définit l'agenda de réveil du nœud et l'ordonnancement des transmissions. En plus du timeslot courant  $t$  relatif à chaque slotframe, le schedule dispose également un temps "absolu" relatif au début de son exécution appelé **ASN** (*Absolute Slot Number*) exprimé en nombre de timeslots écoulés (incrémenté de 1 à chaque timeslot). L'ASN sert de base temporelle pour la synchronisation des nœuds d'un même réseau, il a une valeur identique pour tous les nœuds à un moment donné. Une slotframe  $S$  active étant répétée continuellement, on note  $k$  le nombre de cycles qu'elle a effectué. Si  $\text{len}(S)$  dénote la longueur de  $S$  (le nombre de timeslots qu'elle contient) et que  $S$  est active depuis l'initialisation du réseau, alors on a :

$$\text{ASN} = k \times \text{len}(S) + t \quad \text{où } 0 \leq k, 1 \leq \text{len}(S), 0 \leq t < \text{len}(S) \quad (1.1)$$

Une transmission dans un slot donné doit se faire à une fréquence donnée dont les deux parties communiquant doivent avoir la connaissance avant le début du slot. Cette information est maintenue en utilisant la **matrice CDU** (*Channel/Distribution Usage matrix*) connue par chaque nœud et faisant état de la distribution du spectre de fréquences entre les nœuds pour chaque timeslot considérable. Un élément de cette matrice est désigné par le terme **cell** et y est indexé en utilisant un couple (*slotOffset*, *channelOffset*). Une cell représente donc une unité de ressource de transmission à allouer à un nœud du réseau, qui pourra alors planifier une transmission au temps et à la fréquence correspondants. Cette transmission est installée par le nœud dans une de ses slotframes, où il pourra la paramétriser en fonction des besoins car c'est lui qui dispose de la ressource, en choisissant par exemple d'en faire une cell de transmission (Rx) vers un nœud voisin connu. Si chaque cell de la matrice CDU n'est possédée que par au plus un nœud à travers le réseau, alors on observe qu'à aucun moment (aucun timeslot), la même bande de fréquence n'est utilisée par deux nœuds pour deux transmissions différentes. À moins que les nœuds ne soient désynchronisés, il n'y aura donc pas de collision.

La Figure 1.3 représente un exemple minimaliste de matrice CDU où le spectre de fréquences a été divisé en 4 : on ne considère pas les valeurs des fréquences obtenues directement, mais plutôt des indices dans une liste de taille 4 (ajout d'un niveau d'abstraction). Le temps est divisé en 3 slots, également indicés depuis 0. Certaines cellules dans cette matrice sont possédées par des nœuds du réseau représenté par la Figure 1.4. L'utilisation qu'ils en font est propre à chaque nœud, mais ici dans la Figure 1.5 on considère que chaque nœud maintient une slotframe de taille 3 dans laquelle il installe toutes ses cellules pour émettre en broadcast ou vers un voisin. Par exemple, le nœud A assigne sa cellule donnée par  $(\text{slotOffset}, \text{channelOffset}) = (1, 0)$  à une transmission (Tx) vers le nœud B, qui lui devra installer la cellule Rx correspondante, aux mêmes coordonnées.

Une cellule active peut ainsi être paramétrée en fonction des besoins, notamment le type de transmission (Tx/Rx) et le fait qu'elle soit partagée (*shared*). À noter que par un mécanisme extérieur, si une cellule Tx est installée pour transmettre vers un voisin, ce voisin devrait installer une cellule Rx aux mêmes coordonnées (et vice-versa). Si l'option *shared* est activée, l'accès au média est obtenu à l'aide du mécanisme CSMA/CA car cela indique qu'il est possible que d'autres nœuds possèdent la cellule et l'utilisent, entraînant un risque de collision. Dans la Figure 1.3, la cellule (2, 1) sera considérée comme *shared* par A et D.

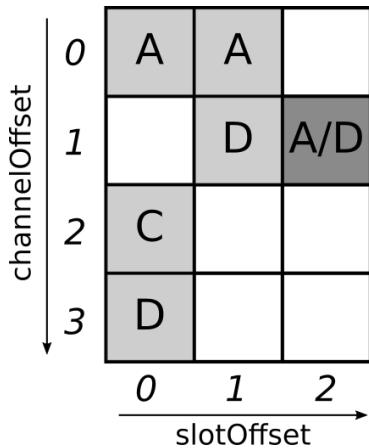


FIGURE 1.3 – Exemple de matrice CDU

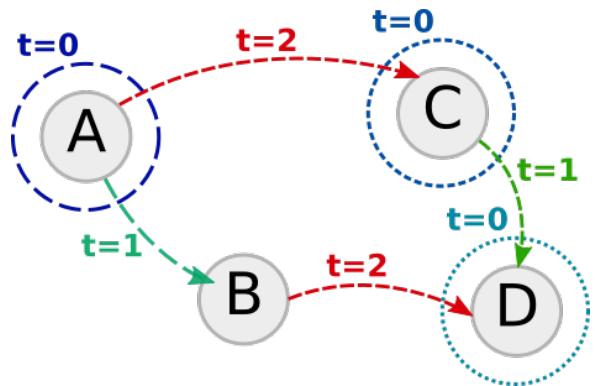


FIGURE 1.4 – Topologie d'exemple avec 4 nœuds et leurs transmissions en un cycle de slotframe, détaillées par la Figure 1.5

cycle k	slotframe 0			slotframe 1			slotframe 2			slotframe 3			slotframe 4		
	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
channels	A→brdcst	A→B		C→D	B→D A→C		C→brdcst			D→brdcst			A→brdcst	A→B	
0															
1				C→D	B→D A→C		D→brdcst			A→brdcst	A→B			C→D	B→D A→C
2	C→brdcst						A→brdcst	A→B					C→brdcst		
3	D→brdcst			A→brdcst	A→B			C→D	B→D A→C	C→brdcst			D→brdcst		
Timeslot t	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
ASN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Schedule B:	S	Rx	TxS	S	Rx	TxS	S	Rx	TxS	S	Rx	TxS	S	Rx	TxS
@channel f:	0	1			3	0		2	3		1	2		0	1
src/dst:	s:A	d:D		s:A	d:D		s:A	d:D		s:A	d:D		s:A	d:D	

FIGURE 1.5 – Hopping sequences des transmissions et schedule effectif résultant pour B (slotframes de taille 3)

Pour exploiter au maximum FDMA et combattre les effets d'interférence/multi-path fading, TSCH utilise un mécanisme de *channel hopping* [?]. Cela permet d'éviter le fait qu'à chaque cycle d'une slotframe S, une cell qui y est installée (et donc se répète) ne résulte en une transmission ayant lieu à une fréquence identique. Considérant une cell dans le schedule (donc dont on connaît la valeur de channelOffset  $chOffset$ ) et le timeslot courant, cela peut être obtenu simplement avec l'équation suivante :

$$f = F[(ASN + chOffset) \bmod n_{ch}] \quad (1.2)$$

$$= F[((k \times len(S) + t) + chOffset) \bmod n_{ch}] \quad \text{par l'équation 1.1} \quad (1.3)$$

Où  $n_{ch}$  est le nombre de fréquences considérées pour les transmissions (donc  $0 \leq chOffset < n_{ch}$ ) et  $F$  est une fonction bijective qui fait correspondre à un indice de fréquence une valeur réelle parmi celles disponibles (ce qui correspond à utiliser une table de conversion à  $n_{ch}$  entrées). En utilisant cette technique, on induit de la variation dans l'usage des fréquences pour une même cell pour des cycles successifs d'une slotframe, et ce malgré le fait que la cell soit statique et dédiée à une transmission de même type. Cependant, le nombre de fréquences disponibles étant borné, ces variations formeront un motif sur plusieurs cycles qui finira par se répéter. Une rotation complète de la *hopping sequence* désigne une itération de ce motif. En analysant l'expression 1.3, on peut facilement déduire que pour maximiser la longueur de ce motif et donc la diversité des fréquences utilisées, on doit avoir que  $len(S)$  et  $n_{ch}$  sont premiers relatifs [?] [?]. On peut alors considérer qu'il suffit d'utiliser des slotframes de longueur première, le nombre de fréquences  $n_{ch}$  n'étant pas un paramètre facilement manipulable. On aura alors une rotation complète après  $n_{ch}$  cycles de la slotframe S et la hopping sequence utilisant chacune des  $n_{ch}$  fréquences.

La Figure 1.5 illustre les hopping sequences des cells installées par chacun des noeuds, c'est-à-dire pour chacune la fréquence à laquelle la transmission aura lieu dans chaque cycle consécutif de la slotframe du noeud. À noter que pour rester concis, on considère ici que  $F$  ne renvoie pas des valeurs de fréquence, mais simplement l'identité. On a  $len(S) = 3$  qui est bien premier relatif avec  $n_{ch} = 4$ , et ce pour la slotframe de chacun des noeuds. Dès lors, on observe bien le phénomène de rotation complétée au cycle 4 qui reprend les même transmissions que le cycle 0. Effectivement, pour chaque cell on observe une hopping sequence couvrant les 4 différentes fréquences disponibles en 4 cycles de slotframe. Le noeud B est choisi arbitrairement pour exhiber un exemple de schedule : durant les 3 timeslots consécutifs de sa slotframe, B va :

1. Sleep : laisser sa radio éteinte durant la durée du slot
2. Rx : allumer sa radio (sur la fréquence f) et attendre de recevoir une frame de A, et s'il en recoit une valide renvoyer un ACK avant la fin du slot
3. TxS : s'il a une frame destinée à D, allumer sa radio (sur la fréquence f) et utiliser le mécanisme de CSMA/CA pour détecter si le lien est occupé. S'il est libre, il transmet la frame et attend un ACK de D avant la fin du slot

La consommation d'énergie dans un réseau TSCH peut varier en fonction de plusieurs facteurs. La taille des slotframes est le fruit d'un compromis : la diminuer implique qu'un timeslot est davantage répété et donc augmente la bande passante de la transmission associée à la cell qui y est installée, mais entraîne une augmentation de la consommation d'énergie. Un schedule n'est pas forcément statique au cours du temps, des cells peuvent être (dés)installées des slotframes dynamiquement pour correspondre aux besoins d'une application. Cela résulte en une variation de la "densité" du schedule calculé à partir de ces slotframes : moins il est dense et moins une application dispose de bande passante pour son trafic, mais cela fait également diminuer la consommation énergétique.

TSCH est une collection de concepts permettant des communications efficaces, mais il laisse en suspens plusieurs problèmes que l'implémentation d'une pile réseau basée sur ces concepts devra gérer, notamment :

- la construction, distribution et le maintient (peut-être dynamique) cohérent du schedule des nœuds à travers le réseau pour éviter les inconsistances (voir Section 2.3.3)
- la synchronisation temporelle : les nœuds doivent avoir une notion du temps identique pour commencer leur timeslot au même moment, et ce malgré le phénomène de *clock drifting*. L'ASN maintenu localement par chaque nœud doit également être identique à tout instant pour tous les nœuds d'un même réseau (voir Section 2.3.1)
- la formation du réseau : un nouveau nœud le rejoignant n'a aucune connaissance des schedules des autres nœuds, mais doit pouvoir obtenir des cells à hauteur de ses besoins et *in fine* avoir établi son propre schedule et servir à son tour de point d'attache au réseau pour un autre nouveau nœud (voir Section 2.3.2)
- la sécurité à différents niveaux : les hopping sequences (déterministes) restent sujettes aux attaques dites de jamming [?] [?], les opérations d'authentification/chiffrement prennent un temps non négligeable mais doivent tenir intégralement dans un timeslot [?], la distribution de clés est un problème dans les réseaux restreints [?] [?], etc.

La pile 6TiSCH basée sur le mode TSCH de IEEE802.15.4e a pour objectif de standardiser des mécanismes pour apporter une réponse à ces problématiques, la spécification IEEE802.15.4 apportant déjà un canevas pour certaines d'entre elles. Par exemple, le mécanisme de *beacons* annoncés en broadcast pour qu'un nœud puisse rejoindre le réseau en utilisant les informations qu'ils transportent.

## Chapitre 2

# La pile réseau 6TiSCH

## 2.1 L'architecture type des WSNs

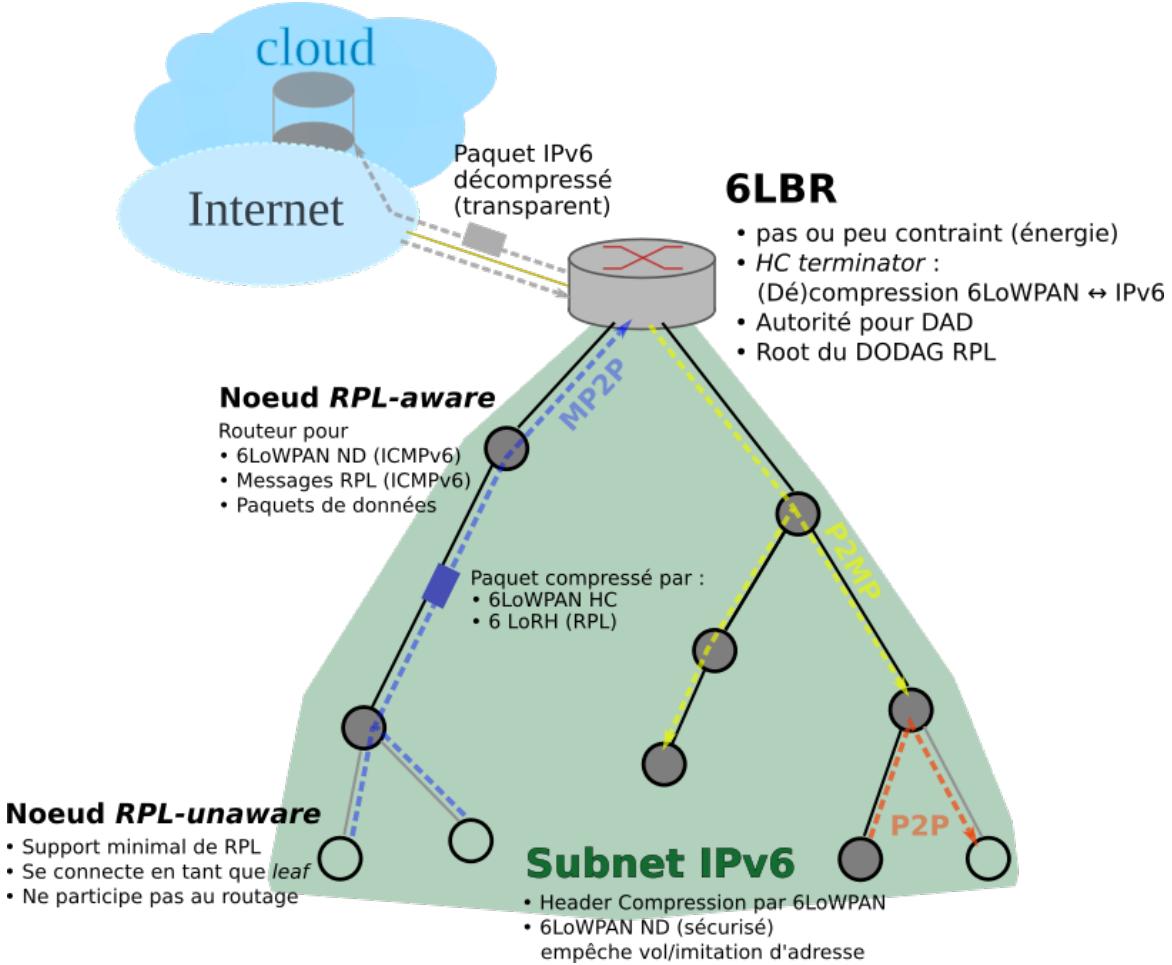


FIGURE 2.1 – Architecture d'un WSN 6TiSCH, calqué sur la configuration basique du standard [?]

La Figure 2.1 représente un déploiement minimalisté de 6TiSCH dans un seul WSN, désigné par le terme “sous réseau NBMA” (*Non-Broadcast Multi-Access*) dans la spécification définissant les architectures réseaux 6TiSCH envisageables [?]. Il en existe des plus complexes supportant la co-existence de plusieurs LLNs le long d’un *backbone* commun, permettant des communications de l’un à l’autre. Les variations dans la conception de tels réseaux reposent notamment sur quels services IPv6 sont mis en place et sur quels équipements. De façon générale, on peut distinguer deux catégories d’équipements dans ces architectures : les noeuds qui font partie intégrante du WSN tels que décrits dans la Section 1.1 (donc restreints en ressources) et les équipements qui gèrent le(s) réseau(x) au dessus de la couche lien, avec des services IPv6 ou de routage/scheduling centralisé (NME/PCE (*Path Computing Element*)) [?].

Un noeud rejoignant le WSN possède généralement une seule interface radio 802.15.4 à laquelle il doit assigner plusieurs adresses IPv6. Il utilise le mécanisme de *Neighbor Discovery* adapté au réseaux restreints défini par 6LoWPAN ND [?] [?]. Le traitement de cette requête de ND diffère en fonction du déploiement des services IPv6, mais elle peut être traitée soit localement par le routeur en bordure du WSN, soit se faire proxy par celui-ci qui la répercutera vers l’ensemble du backbone avec lequel il fait la liaison. Ce routeur porte alors le rôle de 6BBR (*6LoWPAN Backbone Router*) [?]. Grâce au mécanisme de 6LoWPAN ND, l’acquisition d’adresse peut se faire de façon sécurisée et garantir l’unicité dans le réseau.

L'équipement désigné par 6LBR (*6LoWPAN Border Router*) sert de passerelle entre le réseau 6TiSCH et les réseaux extérieurs non restreints, effectuant la traduction entre IPv6 et la couche d'adaptation d'IPv6 aux LLNs, 6LoWPAN (détailée dans la Section 2.2.5). Du côté du WSN, le 6LBR est le point d'ancrage de la topologie créée, à savoir le noeud racine du DODAG (*Destination-Oriented Acyclic Graph*) qui est une topologie arborescente maintenue par le protocole RPL (voir Section 2.2.6). Cette structure est donc directement liée à l'utilisation du multi-hop, chaque noeud pouvant communiquer directement (au niveau de la couche lien) avec son parent ou un de ses enfants. Le modèle ainsi défini par RPL permet d'envisager trois types de trafic qui ont des applications typiques [?] [?] :

1. MP2P (*Multi-Point to Point*) : un ensemble de noeuds participe à un flux convergeant vers un unique noeud. Par exemple, plusieurs senseurs du WSN effectuent une mesure et transmettent la donnée vers une unité centrale de stockage/traitement située à la bordure ou à l'extérieur du WSN, dans un cloud par exemple (point de collecte généralement désigné par le terme *sink*). Il s'agit d'un flux ascendant vers la racine du DODAG.
2. P2MP (*Point to Multi-Point*) : un noeud génère un flux vers plusieurs autres nœuds. Par exemple, cela peut correspondre à une commande d'action émise depuis la racine du DODAG et destinée à plusieurs actionneurs dans le WSN. Il s'agit d'un flux descendant.
3. P2P (*Point to Point*) : deux nœuds du WSN communiquent point à point. Par exemple, un senseur envoie une mesure à un actionneur qui agit en fonction. Le flux peut emprunter des routes ascendantes et descendantes, mais passera toujours par le plus proche ancêtre commun des deux nœuds.

## 2.2 Technologies et protocoles de la pile 6TiSCH

### 2.2.1 La pile dans son ensemble

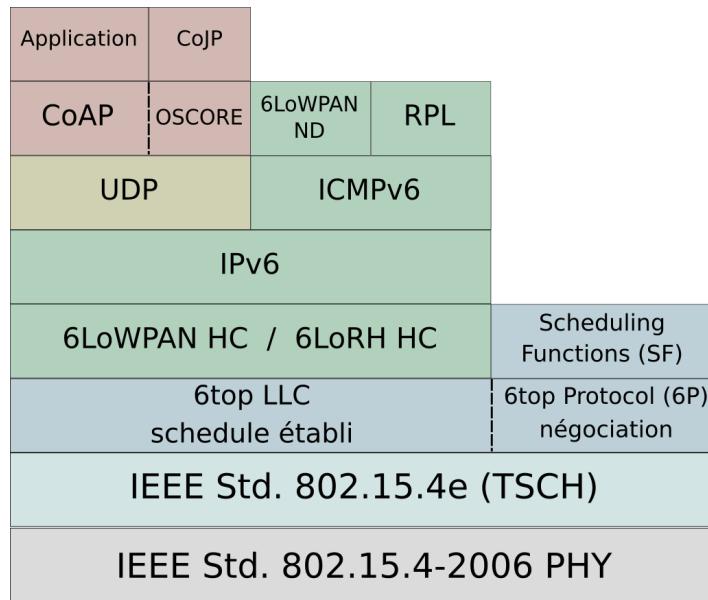


FIGURE 2.2 – Pile réseau 6TiSCH proposée par le standard [?]

La pile illustrée par la Figure 2.2 reprend les protocoles/concepts considérés comme standards pour la pile 6TiSCH, décrits dans les spécifications écrites par le WG 6TiSCH. Ils sont groupés par correspondance aux couches en utilisant un code couleur, depuis le bas de la pile on a :

- La couche physique (PHY) définie par le standard IEEE802.15.4 réutilisée telle quelle, aucune adaptation n'étant nécessaire pour déployer TSCH par dessus. Voir Section 2.2.2.
- La couche lien (MAC) suivant les spécifications du mode TSCH du standard IEEE802.15.4e, qui définit comment **exécuter** un schedule donné mais pas comment le **construire**, ni le **distribuer**. Voir Section 2.2.3.
- La couche d'abstraction de lien et de scheduling qui permet de palier à la problématique de distribution et construction du schedule, tout en définissant une utilisation transparente de celui-ci par la couche réseau (partie du LLC - *Logical Link Control*). Voir Section 2.2.4.
- La couche réseau basée sur IPv6 adapté aux réseaux restreints par la couche d'adaptation 6LoWPAN, où RPL est utilisé pour construire la topologie et la maintenir en tenant compte de la nature du réseau. Voir Section 2.2.5 et Section 2.2.6.
- La couche transport, UDP étant utilisé *de facto* préférablement à TCP qui est trop lourd pour des réseaux restreints. Voir Section 2.2.7.
- La couche application où CoAP est considéré comme un standard adapté à l'interopérabilité avec les réseaux non restreints et servant de base à CoJP (*Constrained Join Protocol*) qui gère le raccord d'un nouveau nœud au WSN, sécurisé à l'aide d'OSCORE (*Object Security for Constrained RESTful Environments*). Voir Section 2.2.7.

## 2.2.2 Couche physique - IEEE802.15.4 PHY

La couche PHY sur laquelle 6TiSCH base ses transmissions radios reste inchangée par rapport à celle proposée par la révision du standard IEEE802.15.4 en 2006. Il s'agit d'une technologie radio qui a fait ses preuves, apportant un bon compromis entre la consommation énergétique, la portée et les taux de transmissions possibles à l'échelle de réseaux couvrant la taille d'un immeuble [?]. Le standard définit plusieurs PHY, celle adoptée dans le cadre de 6TiSCH opère dans la bande de fréquences 2,4 - 2,485 GHz qui est non licenciée. Cette bande est découpée par le standard en 16 canaux (*channels*) larges de 2 MHz chacun, laissant 5 MHz libres entre deux. Les canaux choisis sont donc orthogonaux, c'est-à-dire ne se recouvrent pas et peuvent donc être utilisés simultanément pour des transmissions différentes. Pour une radio, le changement de canal prend un temps estimé à 192  $\mu$ s.

La technologie de DSSS(*Direct Sequence Spread Spectrum*) est utilisée pour étaler le signal (technique de *spread spectrum*), convertissant un bit de donnée en une séquence de 8 *chips*. Cela permet de réduire les risques d'interférences et d'apporter de la confidentialité si la séquence à la base de l'encodage n'est pas divulguée. Le taux de transmission physique du média étant de 2 Mbps, après également le fait de transmettre à 2 Mcps revient à transmettre les données initiales à un débit effectif de 250 kbps. La technique de modulation employée est le O-QPSK (*Offset Quadrature Phase-Shift Keying*), qui utilise des décalages de phases pour encoder l'information.

Une radio éteinte ne consomme pas de courant, contrairement à une radio allumée qui attend de recevoir, reçoit ou transmet. La consommation est identique, que le noeud soit en attente de réception ou activement en train de recevoir un signal. Comparativement, transmettre coûte généralement un peu plus cher considérant une puissance de 0 dBm (1 mW)[?], mais reste du même ordre de grandeur. Une mesure importante dans l'évaluation de l'efficacité avec laquelle une pile réseau utilise la radio est le *duty cycle*, calculé comme la proportion de temps pendant laquelle la radio reste allumée (Rx et Tx confondus) sur une durée totale. On s'attend à observer un duty cycle inférieur à 1% dans un WSN déployé avec une pile efficace [?].

Le standard IEEE802.15.4 définit un schéma de transmission qui est le suivant [?] [?] :

- Du côté du noeud qui s'apprête à transmettre une frame (voir Figure 2.3) :
  1. il émet un préambule de 4 bytes durant 128  $\mu$ s permettant à la radio du receveur de se verrouiller sur le signal de la transmission
  2. le préambule est suivi du SFD (*Start of Frame Delimiter*) de 1 byte indiquant que le début du payload physique vient juste après. Le premier byte de ce payload en indique la taille totale, qui est donc **limitée à 128-1=127 bytes**.
- Du côté du noeud qui est maintient sa radio en écoute :
  1. il démodule du “bruit blanc” jusqu'à percevoir le préambule, le reconnaître et se verrouiller dessus
  2. il attend la fin du SFD et considère le byte suivant comme indiquant la longueur du payload physique qui va suivre, allouant donc un buffer de taille suffisante pour y stocker le reste de la frame
  3. une fois le buffer rempli, il indique la réception d'une frame au micro-controleur en déclenchant une interruption

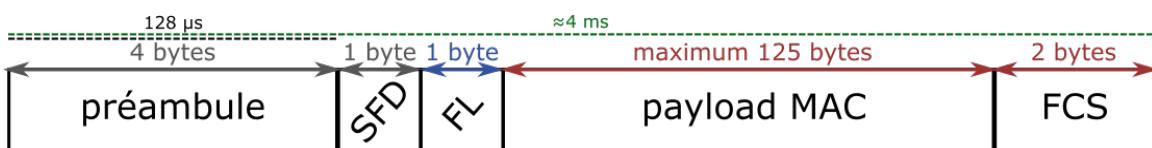


FIGURE 2.3 – Composants d'une transmission IEEE802.15.4 à la couche PHY. FL pour *Frame Length* et FCS pour *Frame Check Sequence* (somme de contrôle utilisant CRC)

## 2.2.3 Couche lien - IEEE802.15.4e en mode TSCH

Les concepts discutés dans cette section sont décrits formellement dans l'amendement du standard IEEE802.15.4e de 2011, et maintenus dans sa version plus récente utilisée comme référence par la suite dans ce document : l'IEEE Std 802.15.4-2015 [?]. Ce standard ne définit pas [?] : comment opérer des communications multi-hop, construire et maintenir un schedule, réagir aux changements dans la topologie, gérer les clés de sécurité (génération, (re)distribution), de quelle façon dont un nœud peut rejoindre un réseau en toute sécurité. Ce sont entre autres des fonctionnalités gérées par les couches supérieures, présentées dans le reste de ce chapitre.

### 2.2.3.1 Structure de PAN et addressage

On considère le réseau comme un LR WPAN (*Low Rate Wireless Personal Area Network*), équivalent à un WSN et où le standard distingue deux types de nœuds : les FFDs (*Full Function Devices*) et les RFDs (*Reduced Function Devices*). Les premiers sont des nœuds peu ou pas contraints en ressources, capables d'endosser les rôles de *PAN coordinator*, *simple coordinator* ou encore de simple nœud. Les seconds sont de simples nœuds limités en ressources, ils ne peuvent pas assumer les tâches d'un FFD qui sont entre autres la coordination entre PANs (pour un PAN coordinator), l'allocation d'adresse, la gestion du canal et l'émission régulière de *beacons* avertisissant la présence du réseau pour de nouveaux nœuds. Les topologies formées par ces nœuds peuvent être adaptées en fonction des besoins, celles utilisées communément sont illustrées par la Figure 2.4 (TSCH reste applicable dans tous les cas [?]) :

- Topologie en étoile autour d'un coordinator, toute communication ayant lieu avec lui
- Topologie *peer-to-peer* où les communications ont lieu entre RFD et coordinators, ces derniers pouvant communiquer entre eux et avec le PAN coordinator faisant le lien avec d'autres PANs
- Topologie *ad-hoc* entre RFDs, sans coordinator pour gérer les communications

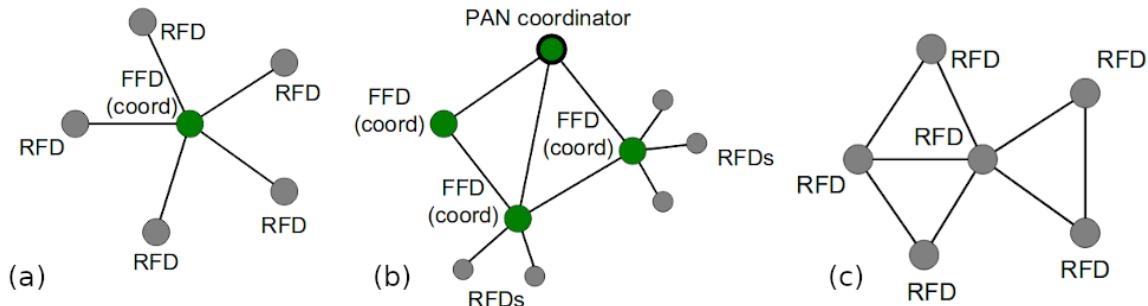


FIGURE 2.4 – Différentes structures envisageables et combinables pour un PAN IEEE802.15.4

À un PAN créé correspond un PAN ID supposé unique parmi les PANs à portée du nœud qui l'initialise. Chaque nœud dans ce PAN peut utiliser une adresse de 64 bits (dérivée par EUI-64 [?]) dite *extended*, notamment avant de l'avoir totalement rejoint. Une fois accepté par un coordinator, un nœud se voit allouer par celui-ci une *short address* de 16 bits dont le champ d'action est limité au PAN (si elle est utilisée seule). Un nœud dans un autre PAN peut être atteint en utilisant une combinaison de sa short address et de l'ID de ce PAN. Une frame 802.15.4 peut donc être adressée en utilisant une combinaison de modes d'adresses (champs destination et source décrits chacun par [PAN ID+short address] ou [extended address]). Un schéma récapitulant le contenu et le format d'une frame 802.15.4e, considérant également les éléments détaillés dans sections suivantes, est donné par la Figure 2.7.

### 2.2.3.2 IEs et *Enhanced Beacons*

L'amendement IEEE802.15.4e a amené le concept d'IE (*Information Element*) dans les champs facultatifs des frames. Un IE est un conteneur composé des champs *Type-Length-Value*, placé à la fin du header MAC classique. Un bit dans le header MAC indique si une frame contient des IEs. Ceux-ci sont groupés en deux types : les *Headers IEs*, consommés à la couche MAC et invisibles pour les couches supérieures et les *Payload IEs*, passés tels quels aux couches supérieures. Les Headers IEs sont placés à la fin du header MAC (et en font partie), les Payload IEs sont placés juste après, avant le payload MAC (et en font partie). Les Payload IEs servent donc à échanger des informations structurées entre nœuds voisins, permettant une extensibilité du standard. TSCH utilise des nouveaux IEs pour étendre le standard 802.15.4 *legacy*.

Comme détaillé dans la Section 2.3.2, la formation du réseau s'opère par le broadcast de frames spéciales de signalement, émises par les nœuds déjà membres du réseau. Ces frames, appelées *beacons*, ont fait l'objet d'adaptations au mode TSCH afin d'annoncer les informations qu'il requiert sous la forme d'IEs. Les frames améliorées en résultant sont appelées EBs (*Enhanced Beacon*), elles présentent au minimum les caractéristiques suivantes selon le standard [?]:

- adresse destination mise à 0xFFFF (short broadcast) et adresse source à la short address du nœud annonceur, ou extended si cette dernière n'est pas définie
- non-ACKable, aucun ACK ne doit être renvoyé après réception d'un EB
- IE *TSCH Synchronization* : informations concernant l'ASN courant et la *Join Metric*
- IE *TSCH Timeslot* : identifiant du template de timeslot qui organise les timings des actions internes effectuées durant un timeslot (Section 2.2.3.4)
- IE *Channel Hopping* : identifiant de la séquence de fréquences utilisée pour la conversion d'un channel offset vers un numéro de canal (ie. fréquence réelle utilisée)
- IE *TSCH Slotframe and Link* : renseigne au nœud rejoignant le réseau un schedule initial minimal pour y communiquer, [?] recommande une seule cell, typée RxTxS (voir point suivant)

### 2.2.3.3 Types de transmissions et retransmissions

Une transmission à la couche lien peut être broadcast (EBs principalement, à l'adresse dédiée 0xFFFF) ou unicast (paquets de données, à l'adresse short préférentiellement à l'extended). Elle peut demander le renvoi d'un ACK à la bonne réception de la frame ou un NACK si elle a bien été reçue mais que son traitement a résulté en une erreur (sécurité notamment), ou ne rien attendre en retour. Le mode TSCH associe chaque transmission à une cell, la cell étant paramétrée pour correspondre à l'action voulue.

Une cell peut être déclarée *dedicated* ou *shared* (S). Étant **dedicated**, un nœud considère que le lien derrière la cell n'est utilisé que par lui et n'emploie pas le CCA (*Clear Channel Assessment*) pour s'assurer que le canal est libre avant de transmettre. Étant **shared**, une cell est susceptible de servir pour des transmissions simultanées, CCA est donc employé et la transmission est lancée uniquement si le canal est libre, sinon elle est différée à une prochaine cell valide pour l'envoi de la frame. Une cell active (non considérée comme *Sleep*) est associée à un rôle dans la communication : réception d'une frame (**Rx**), envoi d'une frame (**Tx**) ou les deux en même temps (**RxTx**). Cela correspond aux comportements suivants :

- Rx : le nœud passe sa radio en écoute après un temps déterminé depuis le début du slot et attend de capter le préambule d'une frame pendant une courte période. Si rien n'est reçu, la radio est éteinte. Si une frame arrive durant cette période, qu'elle n'est pas jetée par le nœud (dst correspondante, sécurité, etc.) et qu'un ACK est demandé, alors celui-ci est construit et renvoyé avant la fin du slot.
- Tx : le nœud regarde s'il a une frame dans sa queue correspondant au destinataire de la cell, sinon il laisse sa radio éteinte pour le reste du slot. S'il en a une, il la transmet et passe ensuite sa radio en écoute pour le reste du slot si elle demande un ACK.
- RxTx : prend le comportement de Tx s'il y a une frame correspondante dans la queue, sinon prend le comportement de Rx. Si la cell est également Shared (S), le comportement résultant est similaire à du Slotted-Aloha [?].

Les transmissions Tx peuvent utiliser le mécanisme de CCA pour éviter les collisions qui se produisent même dans TSCH (proximité d'autre réseau, schedule inconsistent), ou l'ignorer et transmettre d'office la frame, en fonction d'un paramètre booléen *TschCca* [?]. Le fait de différer l'envoi d'une frame suite à la détection de l'occupation du canal par CCA n'est pas considéré comme une retransmission. En revanche, lorsqu'une frame requérant un ACK est effectivement transmise et que ce dernier n'est jamais reçu en retour, une retransmission devra avoir lieu dans une cell future qui soit valide pour cette frame.

Pour éviter les phénomènes de collisions successives répétées, la sélection de cette cell est basée sur l'algorithme de *random backoff* CSMA-CA. Il choisira aléatoirement une cell dans d'une certaine fenêtre, celle-ci grandissant au fur et à mesure des échecs de retransmissions sur un même lien. Il s'agit donc d'un *backoff* "discret", en terme de timeslots à écouler. Le paramètre BE (*backoff exponent*) régule cette taille de fenêtre et n'est incrémenté/reinitialisé que quand la (re)transmission a lieu dans une cell shared correspondant au même lien (vers le même voisin). Il est cependant réinitialisé dans le cas où une retransmission est réussie sur le même lien depuis une cell dedicated et que la queue est vide. La Figure 2.5 schématise le système de (re)transmission quand on considère que CCA est utilisé avant toute (re)transmission. Passés un certain nombre d'essais, 3 retransmissions étant la valeur conseillée par le standard [?], la frame est jetée et les couches supérieures en sont notifiées.

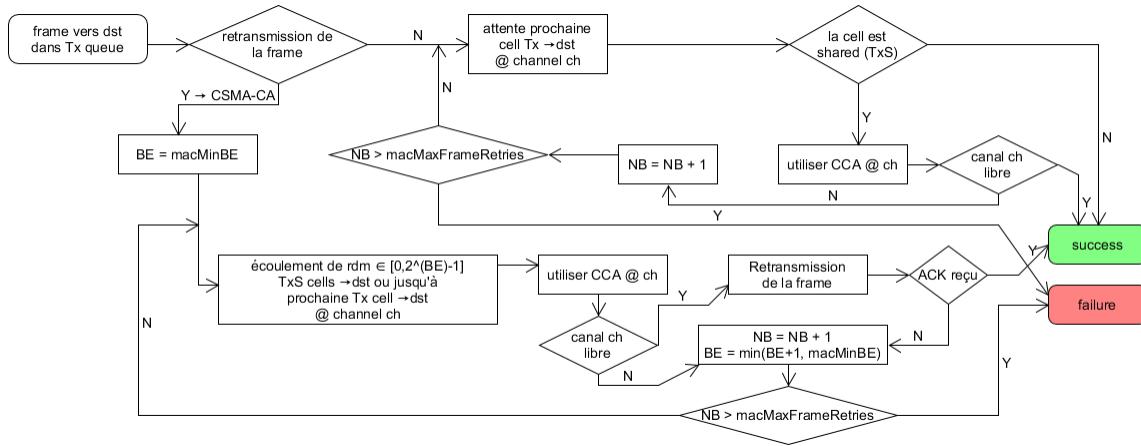


FIGURE 2.5 – Déroulement de la (re)transmission d'une frame destinée au nœud dst considérant *TschCca*=ON

#### 2.2.3.4 Organisation d'une transmission dans un timeslot

Une cell active se traduit en un timeslot dans lequel un nœud aura le rôle de Transmitter (Tx) et l'autre le rôle de Receiver (Rx). En un seul timeslot, si le Transmitter a une frame à destination du Receiver, elle doit être transmise intégralement et l'ACK correspondant, si demandé, doit être renvoyé avant la fin du timeslot. Selon [?] et [?], on a au maximum 127+6 bytes pour les parties PHY et MAC de la frame envoyée, ce qui prend approximativement 4ms et l'ACK en retour prend de l'ordre de 1ms. Bien que les nœuds soient synchronisés en terme de timeslots (même ASN), ils peuvent avoir une perception légèrement différente du temps courant à cause du phénomène de *clock drifting*, dû aux imperfections des cristaux et aux différences de températures [?]. Par conséquent, ils ne commencent pas leur timeslot exactement au même moment et une marge d'erreur doit être considérée. L'idée est de planifier la transmission à un instant précis du timeslot connu des deux nœuds, mais le Receiver allumera sa radio dans une fenêtre de temps autour de cet instant. Il anticipe ainsi le fait que sa perception du temps le place comme en avance ou en retard par rapport au Transmitter. La différence de perception peut être corrigée régulièrement par des mécanismes de resynchronisation décrits à la Section 2.3.1.

La démarche adoptée est la séquentiation d'un timeslot en périodes de temps discrètes, de longueur déterminée en fonction du rôle joué dans la communication. Ces périodes ont chacune un rôle particulier, définies par un nom et une valeur en  $\mu s$ . Un ensemble formé par ces périodes avec des valeurs données est appelé *timeslot template*, auquel est associé un ID unique. Tous les nœuds d'un réseau TSCH doivent être alignés sur un même template, celui-ci étant communiqué aux nœuds dans l'IE *TSCH Timeslot* des EBs. Le standard [?] définit un template par défaut associé à l'ID 0x00, que chaque nœud devrait connaître (éitant la transmission des 25 bytes d'IE pour un template personnalisé). Il considère une longueur totale de slot *macTsTimeslotLength* de 10 ms. La Figure 2.6 en schématisse l'exécution :

- Du côté du Transmitter :

1. Il débute le slot en appliquant CCA et, s'il ne doit pas différer la transmission, il passe sa radio en mode émission, commençant à transmettre la frame après exactement *macTsTxOffset*  $\mu s$  par rapport au début du slot.
2. Une fois la frame entièrement transmise, il attend *macTsRxAckDelay*  $\mu s$  (temps dédié au traitement de la frame et construction de l'ACK par le Receiver) et passe sa radio en mode écoute.
3. Il attend l'ACK *macTsAckWait*  $\mu s$ , s'il n'arrive pas dans ce délai, la transmission est considérée comme un échec et le mécanisme de retransmission peut être utilisé.

- Du côté du Receiver :

1. Il allume sa radio  $macTsRxOffset = macTsTxOffset - \frac{macTsRxWait}{2} \mu s$  après le début du slot, laissant alors une fenêtre de temps de *macTsRxWait*  $\mu s$  pour commencer à recevoir la frame. Si rien n'est reçu dans ce délai, il éteint sa radio pour le reste du slot.
2. Une fois la frame reçue dans son intégralité, il laisse s'écouler un délai de *macTsTxAckDelay*  $\mu s$  durant lequel il traite la frame, construit l'ACK et passe sa radio en mode émission.
3. Il transmet l'ACK résultant et éteint sa radio.

La valeur de *macTsRxOffset* définit la fenêtre de temps considérée pour palier à la désynchronisation des nœuds. Le fait de la fixer à  $macTsTxOffset - \frac{macTsRxWait}{2} \mu s$  permet d'absorber une différence de perception entre les horloges jusqu'à  $\frac{macTsRxWait}{2} \mu s$  [?]. Si les horloges étaient parfaitement synchronisées, le Receiver pourrait s'attendre à recevoir la frame exactement  $macTsRxOffset + \frac{macTsRxWait}{2} \mu s$  après le début du timeslot. Le standard [?] donne pour le template 0x00 les valeurs suivantes : *macTsTxOffset* à 2120  $\mu s$  et *macTsRxWait* à 2200  $\mu s$ . Les horloges ne peuvent donc pas être désynchronisées de plus de 1100  $\mu s$ , sinon la transmission échouera.

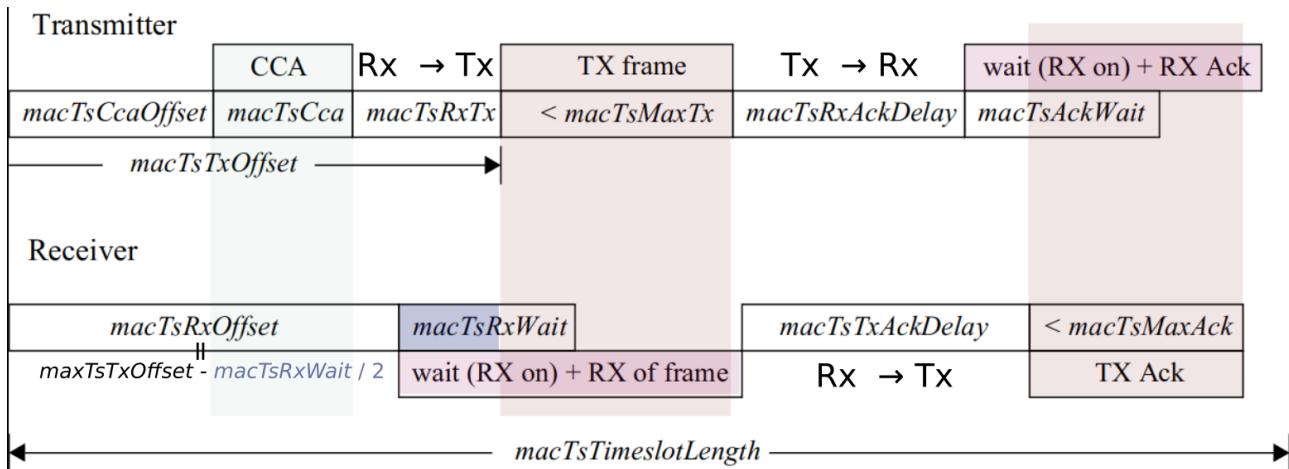


FIGURE 2.6 – Utilisation des différentes périodes délimitées dans un timeslot lors d'une transmission

### 2.2.3.5 Services de sécurité

Des mécanismes de sécurisation des communications à la couche 2 ont été prévus dans le standard pour assurer la confidentialité, l'authentification et l'intégrité des transmissions. L'algorithme considéré est AES, généralement avec une clé de 128 bits, car il est un bon compromis entre la taille du code, les performances en temps et la solidité apportée, en plus de bénéficier d'un port hardware [?]. Il s'agit donc d'un chiffrement symétrique pour lequel les nœuds doivent posséder la même clé afin de se comprendre, ce qui implique une notion de distribution du matériel de *keying* non décrite dans le standard [?].

Le schéma considéré est CCM\* qui est un *wrapper* autour des primitives AES pour sécuriser une frame avec une clé donnée (Figure 2.8). En arrière-plan, l'authentification d'une frame entière se fait par le mode CBC-MAC (*Cipher Block Chaining-Message Authentication Code*) et le chiffrement par le mode CTR (*CounTeR*). Le premier va générer un hash sur la frame entière (aussi appelé MIC - *Message Integrity Code*) qui est tronqué et ajouté à la fin de la frame, recalculable par le Receiver s'il a la clé et la frame. Le second permet de chiffrer le contenu du paquet (Payload IEs et payload de la frame) à partir d'une clé et d'un compteur commun entre les parties communiquantes dit *nonce*, qui est *macAsn* dans le cas de TSCH.

Huit niveaux de sécurité sont distingués par le standard, en fonction de la longueur désirée du MIC et de l'usage du chiffrement. Chaque nœud maintient une table *macKeyTable* dont les entrées sont des conteneurs *KeyDescriptors* contenant entre autres la valeur de la clé, quels autres nœuds peuvent l'utiliser, sa source *KeySource*, son index *KeyIndex*. À chaque frame sécurisée, le nœud rajoute un ASH (*Auxiliary Security Header*) juste après le header MAC (voir Figure 2.7), indiquant quel est le niveau de sécurité à considérer et quelle clé employer en fournissant le couple (*KeySource*, *KeyIndex*). La taille de l'ASH varie entre 5 et 14 bytes, bien que celui-ci puisse omettre le *Frame Counter* en mode TSCH (l'ASN fait en partie office de nonce).

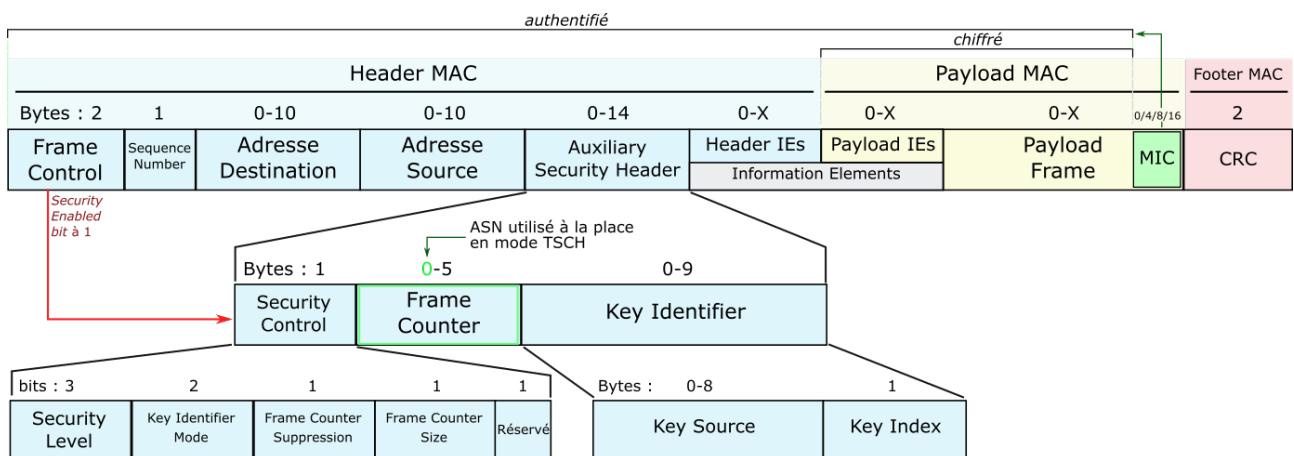


FIGURE 2.7 – Structure d'une frame 802.15.4e selon le standard [?]

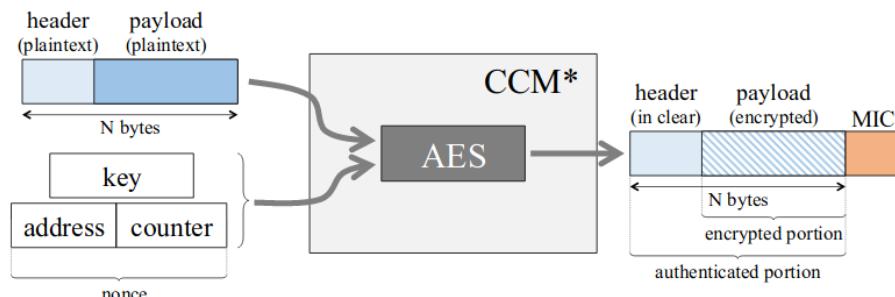


FIGURE 2.8 – CCM\* permet de chiffrer/authentifier des frames à partir d'une clé [?]

### 2.2.3.6 Structure de slotframe

Le standard IEEE802.15.4-2015 [?] reprend les notions théoriques de TSCH présentées à la Section 1.3. Un nœud maintient une table *macSlotframeTable* dont les entrées sont composées d'un identificateur de slotframe *macSlotframeHandle* et de la taille *macSlotframeSize* de cette dernière. Il maintient également une table *macLinkTable* qui correspond à l'ensemble des cells utilisées par le nœud (terme interchangeable avec *link*), chaque entrée possédant des attributs tels que l'identificateur de la slotframe où elle est installée, son type (Rx, RxS, etc.), l'adresse du nœud à l'autre extrémité du lien et les coordonnées de la cell *macTimeslot* et *macChannelOffset*. Un compteur global *macAsn* est incrémenté toutes les *macTsTimeslotLength*  $\mu$ s.

La co-existence de plusieurs slotframes est donc envisagée par le standard. Comme le nœud n'a qu'une radio, il ne peut considérer qu'une seule cell par timeslot. Des règles de précédences sont donc établies pour pouvoir départager des cells concurrentes :

1. une transmission (Tx) est prioritaire sur une réception (Rx)
2. une slotframe avec une valeur de *macSlotframeHandle* plus petite est prioritaire (et par extension les cells qui y sont installées)

### 2.2.3.7 Table des voisins

Chaque nœud maintient une liste *secDeviceList* des voisins avec qui il peut communiquer de manière sécurisée, sous la forme de *secDeviceDescriptors*, chacun stockant les attributs suivants :

- *secPanId* : l'identificateur du PAN auquel appartient le nœud
- *secShortAddress* : la short address du nœud, ou la valeur 0xFFFF s'il utilise uniquement son extended (0xFFFF pour inconnue)
- *secExtAddress* : l'extended address du nœud
- *secExempt* : booléen indiquant si le nœud peut ignorer la valeur du paramètre de niveau de sécurité minimum, c'est-à-dire que les frames originaires de ce nœud peuvent être acceptées quelque soit le niveau de sécurité considéré pour les sécuriser

## 2.2.4 Couche d'abstraction de lien - 6top

### 2.2.4.1 Composants de la couche 6top

6top (*6TiSCH Operation sublayer*, [?]) est une couche intermédiaire introduite par le WG 6TiSCH pour apporter des services requis par le déploiement d'un réseau TSCH basé sur la couche MAC 802.15.4. 6top peut être considéré comme une sous-couche dans les services qui devraient être offerts par la couche intermédiaire de LLC (*Link Layer Control*). La spécification [?] fait l'état de tous les services que l'entité fonctionnelle LLC devrait supporter dans un réseau TSCH, parmi ceux-ci 6top contribue aux points :

- 3.5. *Resource Management* : les cells TSCH constituent des ressources unitaires à distribuer entre les noeuds du réseau, deux voisins souhaitant communiquer le font par le biais de la même cell (installée comme Tx pour l'un et Rx pour l'autre). 6top permet la négociation distribuée et autonome des cells entre voisins à travers le protocole 6P (*6top Protocol*). 6top peut également servir d'interface à une entité centralisée (PCE, ...) pour prendre la main dans la configuration des schedules.
- 3.8 *Scheduling Mechanisms* : les variations de trafic entre noeuds sont courantes, aussi un schedule efficace est un schedule capable de s'adapter dynamiquement. Un mécanisme de surveillance de l'état des communications et de régulation des schedules par (dés)allocation subséquente de cells est mis en place au travers du concept de SF (*Scheduling Function*). À chaque application du réseau peut correspondre une SF, 6top gère leur exécution simultanée qui est prise en compte de façon transparente par 6P.

La couche 6top permet d'établir une abstraction des couches plus basses, de sorte que les couches supérieures ne doivent pas opérer avec 6top explicitement en terme de cell (couples (slotOffset, channelOffset)). À la place, les requêtes vers 6top devraient s'effectuer en terme de bande passante et QoS demandées pour un *bundle* de cells, un bundle étant un ensemble de cells d'une même nature (type et autre extrémité du lien) et donc interchangeables. Une "API" assure l'interfaçage avec ce niveau d'abstraction, la transformation en terme de cell et la négociation de leur (dés)allocation avec le voisin concerné étant invisible pour la couche supérieure qui présente sa requête [?]. Pour promouvoir l'adaptation dynamique du schedule, 6top maintient une *abstract neighbor table* dont chaque entrée correspond à un voisin, stockant des mesures relatives aux communications avec ce dernier : RSSI<sup>1</sup>/LQI<sup>2</sup> de chaque cell, timestamp du dernier paquet, nombre de paquets envoyés et reçus, etc. [?]. Ces informations peuvent également être accédées en passant par l'API de 6top, utilisées par exemple par l'*Objective Function* de RPL pour calculer un rang (voir Section 2.2.6).

6top permet donc la co-existence de plusieurs entités agissant sur le schedule : dispositif centralisé de calcul des routes (PCE), SFs adaptant dynamiquement le schedule en fonction des applications et donc potentiellement aussi nombreuses et variées que ces dernières. Afin d'éviter les conflits entre ces sources, 6top différencie les *hard/soft* cells :

- Une hard cell est une cell "hard-codée" ou installée par une entité extérieure (PCE) pour ses besoins (*forwarding* déterministe par exemple), sans passer par la couche d'abstraction. Ces cells ne devraient pas être manipulées par 6top, elles sont donc en lecture seule de son point de vue. Le standard [?] préconise l'usage d'une slotframe dédiée à ces cells, d'identificateur minimal par rapport aux autres slotframes de sorte à ce qu'elles aient toujours la priorité.
- Une soft cell est gérée localement par 6top. Une couche supérieure passant par l'API pour faire sa requête ne donne pas de cell concrète, c'est 6top qui se chargera de la sélection et manipulation de soft cells précises. Une SF qui détecte un besoin de changement dans le schedule pour s'adapter au trafic utilisera également des soft cells dans ses négociations par le protocole 6P.

Malgré ces dispositions, des collisions de schedule restent encore possibles [?]. Effectivement, le fait que 6top considère une négociation distribuée de voisin à voisin ne garantit pas une consistance globale de l'allocation des soft cells.

---

1. [https://fr.wikipedia.org/wiki/Received\\_Signal\\_Strength\\_Indication](https://fr.wikipedia.org/wiki/Received_Signal_Strength_Indication)

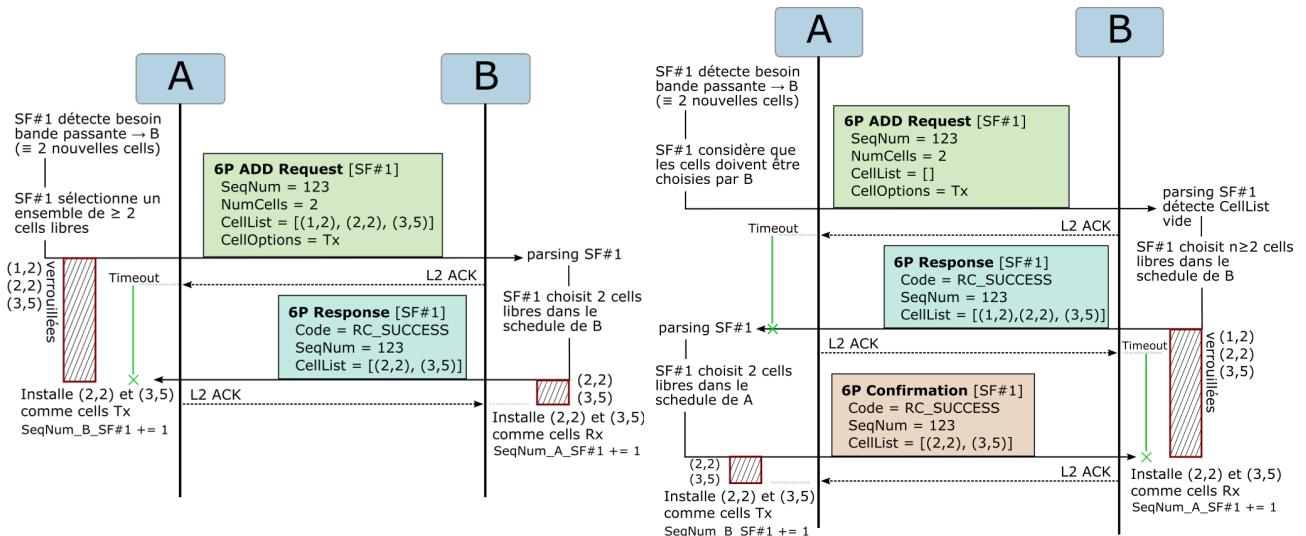
2. <https://www.sciencedirect.com/topics/engineering/link-quality>

#### 2.2.4.2 Le protocole 6P et les scheduling functions

6P permet à un noeud de communiquer avec son voisin pour négocier l'ajout/la suppression/le déplacement de cell dans leur schedule de façon consistante. Il s'agit donc d'une négociation distribuée sous la forme de *Transaction* entre nœuds voisins. Une Transaction est initiée par un message formaté de type *Request*, sous impulsion d'une SF qui détecte un besoin d'adaptation du schedule au trafic. Une Transaction s'opère dans une optique de négociation, aussi on a toujours un schéma calqué sur une proposition d'un noeud suivie d'une confirmation de ce qui convient à l'autre noeud qui la reçoit. Un échange 6P se fait préférentiellement en employant une cell dedicated entre deux voisins (*single hop*) et est totalement encapsulé dans un Payload IE (IE 6top de format générique gérant tous les messages).

Deux types de Transactions sont envisagés par 6P en fonction de quel noeud sera celui qui fait la proposition : on différencie des Transactions *2-steps* et *3-steps*. Trois types de messages sont échangés lors des Transactions : *REQUEST*, *RESPONSE* et *CONFIRMATION*. Par Transaction, un unique type de cell *CellOptions* est considéré, s'appliquant en fonction de la sémantique de la commande associée (ajout/suppression/déplacement de cell entre autres). Par exemple, une commande ADD accompagnée de CellOptions indiquant “Rx” va résulter en l'installation de cell(s) dans les schedules des deux nœuds, de type Rx pour le nœud à l'origine de la Request et Tx pour l'autre. Les principales commandes sont ADD, DELETE, RELOCATE, COUNT et LIST. À tout message 6top sont associés un SFID (*SF Identifier*) et un numéro de séquence *SeqNum*. Le premier permet d'identifier quelle SF doit traiter la Transaction, plusieurs SFs pouvant co-exister. Le second est utilisé pour faire correspondre les messages REQUEST/RESPONSE/CONFIRMATION et maintenir un état consistant entre les schedules impactés. Les Figures 2.9 et 2.10 illustrent respectivement une commande ADD orchestrée en 2 et 3-steps, avec les périodes durant lesquelles des cells sont “verrouillées” afin de maintenir un état consistant jusqu'à la fin de la Transaction qui valide définitivement leur installation.

Pour pouvoir négocier, deux nœuds doivent faire tourner la même SF. Le traitement des messages 6P reçus est redirigé vers la bonne SF en fonction du SFID contenu dans le message. Pour permettre les communications arbitraires entre même SFs d'un nœud à l'autre, 6top inclut un champ *Metadata* dans certaines commandes (16 bits) et une commande dédiée SIGNAL. Par exemple, le champ Metadata d'une commande ADD peut indiquer la slotframe dans laquelle installer les cells. La valeur est passée telle quelle à la SF, comme c'est également le cas pour les champs CellOptions (bitmap de 8 bits) et CellList (liste de cells représentées sur 4 bytes, 2/2 pour slotOffset/channelOffset). Ces derniers sont des paquets de bits opaques formatés par la SF, les contenus indiqués ici étant recommandés par le standard [?]. Une SF doit également gérer la détection et le traitement d'inconsistances, les erreurs renvoyées lors d'une Transaction et les valeurs de timeouts.



## 2.2.5 Couche d'adaptation à IPv6 - 6LoWPAN

Les WSNs se reposant sur IEEE802.15.4, et plus généralement les LLNs ont des caractéristiques qui rendent difficile l'adoption directe d'IPv6 comme couche réseau de ces réseaux. Notamment, le MTU minimal par défaut de IPv6 qui est de 1280 bytes n'est pas adapté pour une frame 802.15.4 limitée à 127 bytes. Le header IPv6 de 40 bytes réduit fortement la charge utile du paquet : considérant une frame sécurisée par 802.15.4, la frame ne transporte plus que 33 bytes de données. D'autres problématiques viennent également s'ajouter : adaptation du mécanisme de SLAAC (StateLess Address Auto-Configuration), support du broadcast dans tout le sous-réseau, donc du multihop pour atteindre les nœuds qui ne sont pas à portée directe, mécanismes de sécurité (obtention des adresses, génération d'adresses), etc [?]. La couche d'adaptation 6LoWPAN met en place des mécanismes rendant possibles les communications IPv6 avec et dans un WSN. Les 3 services principaux que 6LoWPAN et ses extensions assurent sont les compressions d'headers, la fragmentation de paquet et l'auto-configuration d'adresse.

Pour offrir une couche d'adaptation générique pouvant utiliser un service indépendamment de l'autre, 6LoWPAN définit des sous-headers de différents types :

1. *No 6LoWPAN* : le paquet contenu n'est pas compatible avec les mécanismes de 6LoWPAN
2. *Mesh Addressing Header* : permet le forwarding des frames à la couche lien, et donc le multihop géré à la couche d'adaptation
3. *Dispatch Header* : compression de headers ciblés par plusieurs schémas (IPv6, UDP, RPL, etc.)
4. *Fragmentation Header* : gestion de la fragmentation et du ré-assemblage du contenu du paquet

### 2.2.5.1 Mesh Addressing Header : forwarding

Le standard 802.15.4 ne définit aucune forme de routage et se repose donc sur les couches supérieures. On distingue deux schémas de routage impactant également la fragmentation :

1. *mesh-under* : routage par 6LoWPAN utilisant le mesh addressing header pour maintenir les adresses *originator* et *destination* (short ou extended dans le PAN) des deux extrémités de la communication, les nœuds intermédiaires forwardant les fragments du paquet sur base de ces adresses. On a donc une fragmentation effectuée par le nœud originator et un ré-assemblage une fois tous les fragments arrivé au nœud destination.
2. *route-over* : routage délégué à la couche IP par un protocole qui devra donc procéder au ré-assemblage du paquet à chaque nœud sur le chemin et maintenir une table de routage. Le protocole utilisé peut être RPL.

Utiliser le mesh-under permet d'avoir un délai de transmission plus court, au détriment de la fiabilité car un fragment perdu implique une retransmission complète de tous les fragments [?].

### 2.2.5.2 Dispatch header : compression d'headers

Plusieurs schémas d'encodage ont été définis, ceux-ci tirent parti du fait que certaines valeurs de champs peuvent être inférées à partir d'informations déjà contenues dans la frame, réduites à un ensemble de valeurs communes ou encore dérivée d'un contexte partagé. *LOWPAN\_IPHC* est utilisé pour laisser tomber les champs IPv6 *Length* et *Hop-Limit* et compresser les adresses source et destination selon leur nature. Dans le meilleur des cas, c'est-à-dire considérant l'utilisation d'adresses link-local et un unique saut, le header IPv6 peut être réduit à 2 bytes. Dans une communication multihop, il peut être compressé jusqu'à 7 bytes. Le schéma *LOWPAN\_NHC* gère les Next Headers IPv6 et les champs UDP tels que l'indicateur de longueur, la checksum et les ports. Dans le cas où les ports utilisés sont courants, le header UDP peut être compressé en 2 bytes. Dans le cas où le route-over est utilisé, 6LoRH [?] peut être utilisé pour compresser les informations que RPL rajoute dans les paquets (par exemple l'identificateur de l'instance RPL).

### 2.2.5.3 Fragmentation header : fragmentation

Le header de fragmentation fait 4 ou 5 bytes et permet d'embrasser de distinguer le premier fragment des autres, celui-ci contenant donc le header IPv6 (compressé ou non). Les valeurs reprises dans le header de fragmentation sont :

- *dispatch* : séquence de bits indiquant s'il s'agit du premier fragment
- *datagram\_size* : la taille totale du paquet avant sa fragmentation
- *datagram\_tag* : identifie de façon unique le paquet original duquel provient le fragment
- *datagram\_offset* : la position du fragment dans le paquet (pour les fragments suivant le premier)

Le problème inhérent à cette façon basique de procéder est que la perte d'un fragment empêche le réassemblage du paquet complet, condamnant les nœuds à recommencer la transmission entière. Des mécanismes d'acquittement sélectif ont été mis en avant pour palier à cela [?] [?]. Une fois un paquet IPv6 totalement ré-assemblé par la couche 6LoWPAN, il est transmis à la couche IP supérieure qui opère la décision de routage.

### 2.2.5.4 L'auto-configuration des interfaces en IPv6

Dans un réseau traditionnel, le mécanisme de ND (*Neighbor Discovery*) peut être employé par un nœud pour assigner de façon autonome des adresses IPv6 à ses interfaces, tout en garantissant l'unicité de celles-ci. Cependant, le mécanisme de découverte fait l'hypothèse que le broadcast est effectif sur tout le domaine, ce qui n'est pas le cas pour un nœud de WSN. Des mécanismes de multicast/broadcast sont spécifiés par 6LoWPAN au travers du dispatch header *Broadcast Header*. Cependant, le ND induit un trafic multicast trop important pour un WSN [?], incitant à une reconception du mécanisme qui standardisée par 6LoWPAN ND [?].

Pour éviter le trafic multicast généré par les messages de DAD (*Duplicated Address Detection*), les nœuds enregistrent leur adresse auprès d'un routeur qui maintient en cache toutes celles utilisées dans le réseau 6LoWPAN, associée à un temps d'expiration donné. Les requêtes d'enregistrement sont routées vers le 6LBR, dans des messages ARO (*Address Registration Option*) entre un nœud et un routeur (nœuds capable de router vers la racine RPL) et dans des messages ABRO (*Authoritative Border Router Option*) entre ces routeurs et le 6LBR (racine RPL).

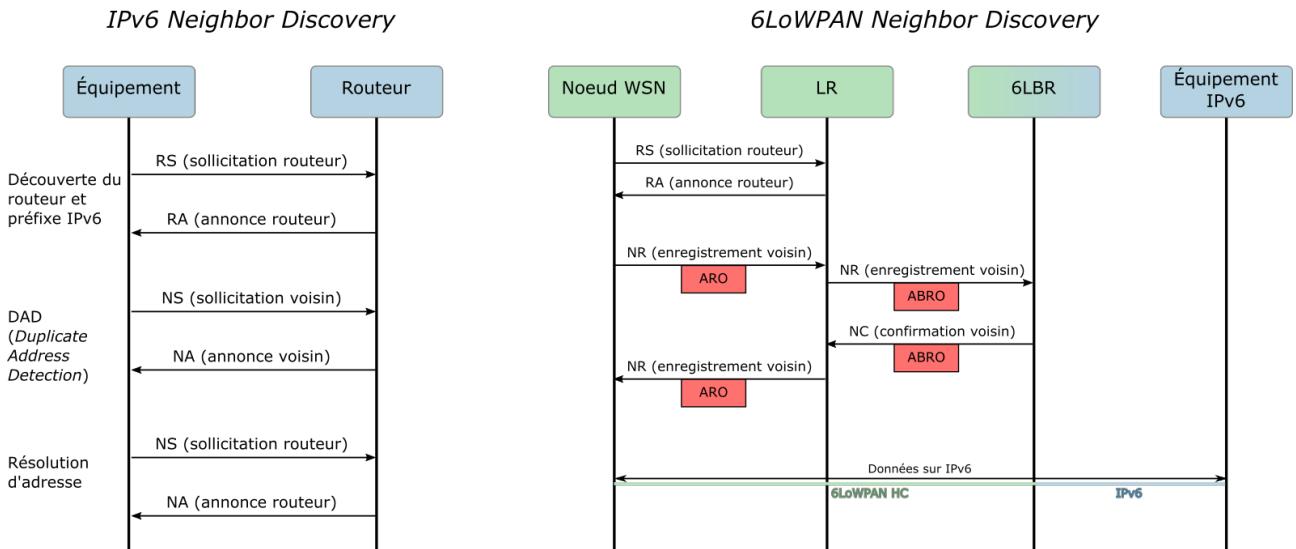


FIGURE 2.11 – Différences entre IPv6 ND traditionnel et 6LoWPAN ND

## 2.2.6 Couche réseau et routage - RPL

RPL (*Routing Protocol for LLNs*, [?]) est un protocole de routage à vecteurs de distance proactif, adapté aux communications MP2P qui sont communes dans un WSN. Typiquement, on a dans ces réseaux des communications multihops convergeant vers un point de collecte non restreint. Les topologies que RPL établit sont composées de DODAG(s) (*Destination Oriented Directed Acyclic Graph*) similaires à des arbres, dont la racine est typiquement le 6LBR (voir Section 2.1). Chaque nœud du réseau doit donc avoir au minimum un parent (excepté la racine *Root*) et peut servir de parent à plusieurs autres nœuds, à condition de supporter des capacités de routage. Un nœud *host* peut rejoindre le réseau en tant que feuille (*leaf*) en ne supportant qu'en partie RPL, ne routant que les paquets qu'il génère lui-même. Ces topologies sont construites par les nœuds sur base de différentes métriques et attributs arbitraires, qui sont exploités par une OF (*Objective Function*) pour adapter de façon optimale la topologie à des critères donnés. Des métriques classiques sont par exemple le nombre de sauts, le débit du lien, la fiabilité tandis ce que les attributs sont relatifs à l'état du nœud : sa source d'énergie, ses limitations de calcul/stockage, etc. Plusieurs applications peuvent être effectives conjointement dans un même WSN, aussi chacune est liée à une *Instance RPL* s'appuyant sur un DAG (ensemble de DODAGs) et une OF.

À chaque nœud d'un DODAG est associé un rang représentant sa "distance" avec la racine, c'est-à-dire le coût à considérer pour joindre la racine en passant par ce nœud, selon l'OF. RPL suit donc une approche "en gradient", remonter vers la racine revenant à emprunter un chemin dont les rangs sont strictement dégressifs. Pour sélectionner un parent parmi ses voisins atteignables dans le réseau, un nœud compare leurs rangs respectifs en tenant compte du coût nécessaire pour communiquer avec chacun. L'OF définit ce qui compose ces deux éléments (quelles métriques) et le calcul à appliquer, retournant une valeur correspondant au rang potentiellement obtenu pour chaque. Le parent préféré sera alors celui qui minimise le rang retourné. Une fois associé avec, il peut continuer à évaluer ses voisins de la sorte pour adapter la topologie de façon dynamique, mais n'opérera un changement que si la différence de rang est significative (hystérèse). La Figure 2.12 illustre la sélection d'un parent préféré  $P$  parmi  $P_N$  candidats, certains étant d'office éliminés car un de leur attribut ne respecte pas une contrainte préliminaire (imposée également par l'OF). L'OF est ici additive, mais peut prendre une forme arbitraire.

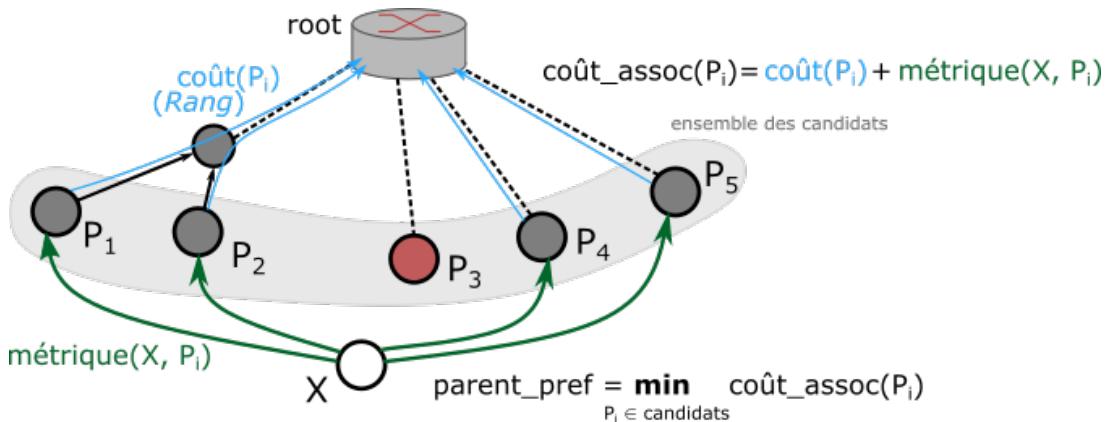


FIGURE 2.12 – Sélection d'un parent préféré par un nœud  $X$  utilisant une OF arbitraire

Un exemple de métrique communément utilisée par une OF additive est l'ETX (*Expected Transmission Count*). Elle représente le nombre moyen de (re)transmissions requises pour transmettre avec succès un paquet à un voisin (l'ETX d'un chemin multihop est l'addition des ETX de chaque saut). Sa valeur peut être estimée sur  $numTx$  transmissions en considérant le nombre d'ACK reçus en retour :

$$ETX = \frac{numTx}{numTxAcked} \geq 1 \quad (2.1)$$

Ces valeurs peuvent être obtenues en pratique depuis l'abstract neighbor table maintenue par la couche intermédiaire 6top [?].

## /—— A faire : DAGRank(rank) ——\

RPL utilise ICMPv6 pour transporter l'information utile à la construction et au maintient de la topologie, à travers plusieurs types de messages de signalisation :

- DIO (*DODAG Information Object*) : multicast par les noeuds déjà dans la topologie (*all-RPL-nodes address FF02::1A*) à intervalle de temps variable contrôlé par l'algorithme *Trickle*. Contient la version du DODAG et l'ID de l'instance dans laquelle le noeud émetteur évolue, ainsi que son rang, les métriques, les contraintes et l'OCP (*OF Code Point*) identifiant l'OF à utiliser. Un noeud souhaitant intégrer la topologie se sert de ces informations, émises par différents noeuds en faisant partie, pour sélectionner le parent qui l'y raccrochera. Un DIO permet donc d'établir des routes ascendantes.
- DIS (*DODAG Information Solicitation*) : émis en broadcast par un noeud souhaitant recevoir un DIO (évite une attente passive).
- DAO (*DODAG Advertisement Object*) : émis par un noeud vers la racine, permettant de construire des routes descendantes. Le support de ces routes peut se faire en deux modes différents : *storing* et *non-storing*. En mode non-storing, seule la racine maintient le nécessaire pour router vers le bas, obligeant tout paquet à remonter jusqu'à elle pour insérer dans le paquet la liste des noeuds intermédiaires. En mode storing, des noeuds intermédiaires maintiennent des tables de routage et sont donc capable de rediriger d'eux-mêmes le paquet vers le bon noeud enfant.

La dissémination de l'information (par émission de DIOs) est régulée par l'algorithme dit du Trickle, paramétrable pour établir un compromis entre la faculté d'adaptation/expansion de la topologie et la surcharge de trafic dans le réseau. L'idée de l'algorithme est d'éviter de répandre trop de fois la même information si elle n'apporte rien, c'est-à-dire qu'elle est consistante avec l'état actuel, déjà connu. Ainsi, à la détection d'une inconsistance doit correspondre une augmentation de la fréquence d'émission de DIOs signalant le nouvel état consistant. La notion de consistance dépend du contexte, cela peut par exemple correspondre au numéro de version du DODAG courant *DODAGID*, renouvelé suite à une reconstruction de la topologie. L'algorithme est paramétré par  $I_{min}$  et  $I_{max}$  les bornes extrémales de l'intervalle de temps entre deux transmissions et  $k$  un entier qui sert de seuil de redondance à partir duquel une information ne sera pas relayée. Les variables suivantes sont utilisées :  $c = 0$  un compteur d'information reçue considérée comme consistante et donc redondante,  $I = I_{min}$  l'intervalle courant et  $t$  le timer. Le comportement est alors le suivant :

1.  $c$  est mis à 0 et  $t$  à une valeur aléatoire tirée dans  $[\frac{I}{2}, I]$
2. quand une inconsistance est détectée dans la suite,  $I \leftarrow I_{min}$  et retour en 1.
3. tant que  $t$  n'a pas expiré, si un message consistant est reçu,  $c \leftarrow c + 1$
4. quand  $t$  expire, si  $c < k$  alors un DIO est émis
5. arrivé à la fin de l'intervalle  $I$ ,  $I \leftarrow \min(2I, I_{max})$  et retour en 1.

## 2.2.7 Couches transport et application - CoAP sur UDP

### 2.2.7.1 Transport par UDP

Les services que TCP assure traditionnellement dans les communications, tels que la fiabilité et le contrôle de congestion, se révèlent inappropriés dans le cadre de LLNs. Ils introduisent une surcharge trop importante de trafic dans un réseau restreint, de par le nombre de communications et les headers nécessaires. Les pertes récurrentes de paquets dans un LLN, dûes à sa nature, sont également mal interprétées par TCP qui y voit les effets d'une congestion et réagit en conséquence à tort. Des adaptations des mécanismes et allégements de TCP ont été proposés [?], mais le fait que les deux hôtes communiquant (ou un équipement intermédiaire de traduction) doivent en supporter l'implémentation reste un frein important. De plus, cela ne correspond pas à l'idée d'interopérabilité transparente entre les réseaux restreints de l'IoT et le reste d'Internet.

UDP a donc été choisi pour assurer le transport dans la pile 6TiSCH, déléguant l'implémentation de mécanismes de contrôle à la couche applicative. UDP est léger, surchargeant peu les paquets et ne maintenant aucun état relatif aux messages envoyés. En contrepartie, il ne garantit pas de fiabilité, de mécanismes de détection de messages dupliqués et de contrôle de flux. Afin de réduire encore davantage la charge liée au transport par UDP, 6LoWPAN assure une compression efficace, réduisant les headers UDP jusqu'à seulement 2 bytes.

### 2.2.7.2 CoAP

CoAP (*Constrained Application Protocol*, [?]) apporte une réponse au besoin d'implémentation d'une adaptation de l'architecture RESTful aux LLNs. Une intégration directe du modèle client/serveur sur lequel HTTP se repose pour offrir des services Web dans Internet n'est pas envisageable. Effectivement, ce modèle fait l'hypothèse d'un transport fiable et d'une fragmentation gérée de façon transparente dans les couches plus basses. UDP ne permet pas d'assurer une fiabilité en matière de transport, et bien que 6LoWPAN gère la fragmentation, elle reste coûteuse et peut rendre le LLN inopérant [?]. CoAP définit un sous-ensemble des fonctionnalités d'une architecture RESTful, spécialisant son utilisation aux réseaux restreints. Une attention particulière est portée au maintien d'une interopérabilité maximum avec HTTP, permettant des interactions entre des équipements supportant CoAP et HTTP. CoAP se spécialise donc dans les communications M2M, se différenciant de HTTP sur plusieurs points :

- les équipements n'endosseront pas systématiquement le même rôle de serveur/client
- les échanges de messages sont asynchrones
- les mécanismes de fiabilité sont implémentés au niveau applicatif par CoAP lui-même
- la facilitation du déploiement des mécanismes de *caching/proxying*
- minimisation de l'*overhead* lié aux headers, limitation de l'utilisation d'ASCII

CoAP est composé de deux sous-couches logiques : la couche *Message* et la couche *Request/Response*. La couche Message est la première au dessus de UDP, apportant le contrôle des communications nécessaire pour pallier aux manquements de ce dernier. Elle considère 4 types de messages auxquels sont associés des rôles différents dans la communication asynchrone. Les mécanismes de retransmission et de détection de messages dupliqués s'appuient sur cette sous-couche et les attributs qu'elle définit. La sous-couche suivante (*Request/Response*) est relative à la sémantique des messages et leur contenu. Une *Request*, émise par le nœud alors considéré comme client, décrit une certaine méthode appliquée à une ressource du serveur. Par exemple, une méthode GET ciblant une ressource désignée par un URI. Une *Response* désigne le message émis en retour par le serveur, contenant le résultat du traitement de la requête, auquel est associé un code de retour. Dans l'exemple précédent, cela peut être un message d'erreur contenant la chaîne "Not Found" accompagnée d'un code 4.04.

Dans chaque header de message, on retrouve un champ *Message ID* de 16 bits (la Figure 2.14 illustre le format d'un message). Les quatre types de message transportent les Requests/Responses indépendamment de leur sémantique (la couche logique Message agissant en dessous). Le champs *Type* de 2 bits dans le header indique toujours un type parmi les quatre disponibles.

0. *Confirmable* (CON) : message à acquitter par le serveur, qui doit répondre par un message ACK pouvant potentiellement transporter (*piggybacking*) la réponse du serveur à la Request. Si la réponse à la Request n'est pas immédiatement disponible (temps d'acquisition d'une mesure par exemple), l'ACK peut être renvoyé directement, suivi par la réponse dans message CON une fois le traitement effectué (Figure 2.13). Le message ID permet de faire le lien entre le message initial et son ACK, l'ACK transportant le même ID. Si aucun ACK correspondant n'est reçu, un mécanisme d'*exponential backoff* gère les retransmissions.
1. *Non-Confirmable* (NON) : message ne nécessitant pas d'ACK ou de RST si son contenu n'a pas pu être traité. Cela ne signifie pas qu'à une Request envoyée dans un message NON, il n'y aura aucune Response de la part du serveur : ces deux messages ne seront juste pas acquittés et auront un Message ID différent. Dès lors, la corrélation entre ces Request et Response se fait en utilisant un mécanisme de *token* (Figure 2.15).
2. *Acknowledgement* (ACK) : confirme la bonne réception d'un message CON à son émetteur, transportant le même Message ID. Peut potentiellement porter en même temps la Response à une Request pour limiter le trafic sur le réseau (mécanisme de piggybacking).
3. *Reset* (RST) : message renvoyé par un noeud s'il n'est pas capable de traiter un message CON ou potentiellement NON (pas capable de produire un message d'erreur correspondant).

La détection des messages dupliqués se fait sur base des Message IDs indiqués par les messages reçus. Étant donné que le modèle de communication est asynchrone, un mécanisme de token est employé pour corrélérer une Response à sa Request, deux messages présentant un même token concernant donc une ressource identique. Un token est une séquence pseudo-aléatoire de 0 à 8 bytes que chaque client devrait générer et intégrer dans chacune de ses Requests. Dans le paquet, il est placé à la suite du header CoAP de taille fixe et suivi des potentielles *Options* CoAP et du Payload applicatif s'il y en a un (voir Figure 2.14). Les Options sont des conteneurs de la forme *Type-Length-Value* présents dans les Requests et Responses, permettant de transporter des informations complémentaires exploitables dans le contexte à la manière des en-têtes HTTP. Par exemple, l'URI d'une ressource ciblée est divisée en plusieurs Options *Uri-Host*, *Uri-Port*, etc. Le champs *Code* (8 bits) est divisé en une classe *c* (3 bits) et un détail *dd* (5 bits), définissant une correspondance directe avec les codes HTTP. Par exemple, *c=0* pour une Request permet d'indiquer une commande GET, POST, etc. en fonction de *dd*, *c = 4* pour une Response indique une erreur client (*dd=04* donne 4.04, erreur "Not found").

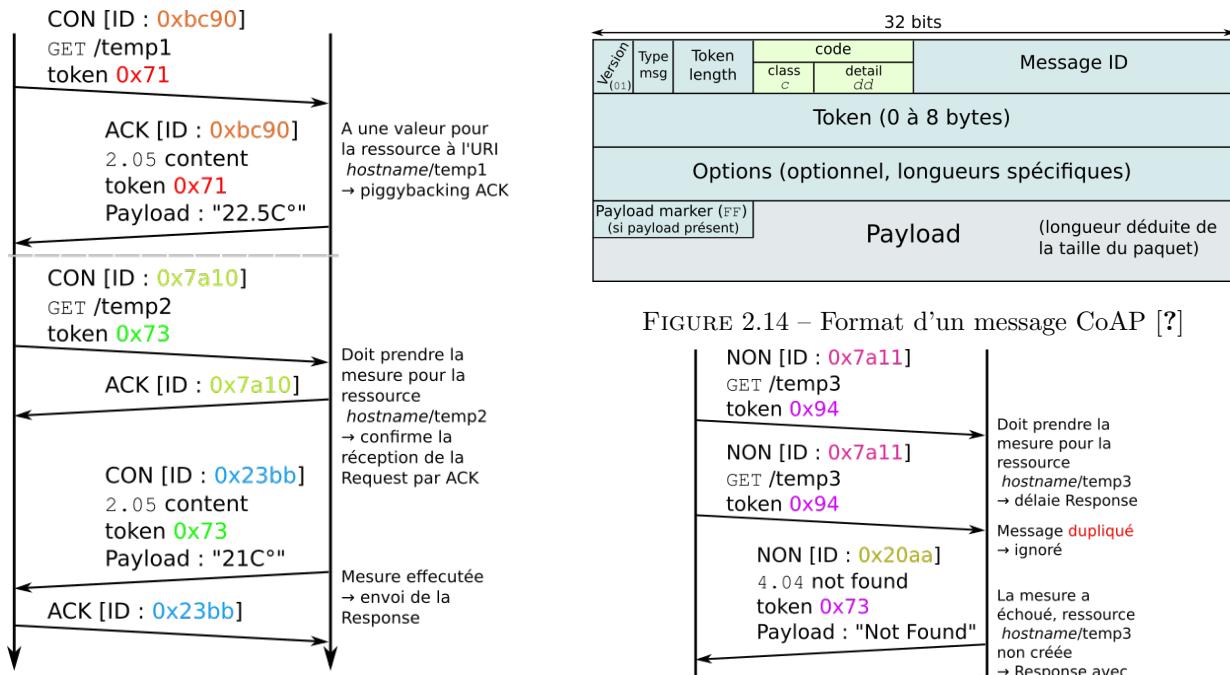
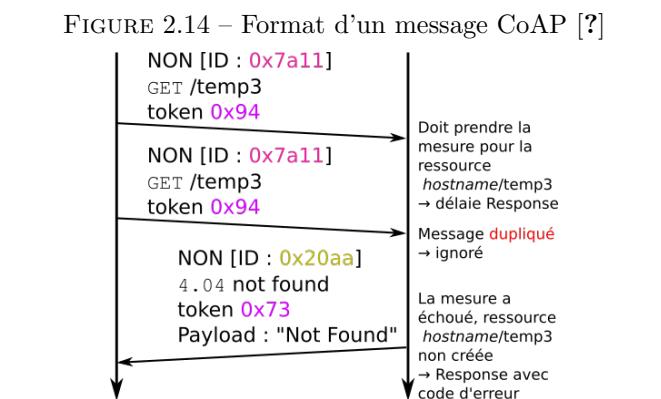


FIGURE 2.13 – Message CON et piggybacking de la Response dans l'ACK



CoAP utilise un schéma d'URI similaire à HTTP, identifiant la ressource sur un serveur dit *origin server* par un chemin pointant vers l'élément dans sa structure hiérarchique. L'origin server est identifié soit par un nom d'hôte à résoudre par un mécanisme similaire au DNS, soit par une adresse IP (CoAP supporte IPv4 et IPv6). Un serveur écoute sur un port UDP spécifié dans l'URI, ou le port 5683 par défaut. Une requête est paramétrable avec un nombre arbitraire d'arguments terminant l'URI. Tout comme HTTP avec son homologue HTTPS, CoAP supporte la sécurisation au niveau du transport par DTLS. Le schéma d'URI est alors différent (*coaps* et non *coap*), mais ne sera pas considéré ici. Le schéma est le suivant [?] :

$$\text{coap-URI} = \text{"coap :"} \text{ "/"} \text{ hostname} [\text{ ":" port}] \text{ path-abempty} [\text{ "?" query}]$$

Où *hostname* permet d'identifier le serveur, *path-abempty* est le chemin absolu de la ressource sur le serveur (peut être vide sous certaines conditions) et *query* est une chaîne paramétrisant la requête avec des arguments de la forme *key=value* séparés par "&". Un URI à transporter dans un message est fragmenté sous la forme d'Options *Uri-Host*, *Uri-Port*, *Uri-Path* et *Uri-Query* (ensemble désigné par le terme *Uri-\**). Les deux dernières peuvent être multiples, une Option *Uri-Path* correspondant à un segment du chemin (chaîne entre deux "/") et une Option *Uri-Query* correspondant à une paire (*key, value*). Une implémentation de CoAP doit supporter la découverte de serveur/service et de ressource dans le cadre de communications M2M. Cela est géré d'une part par un support de l'adresse multicast *all-CoAP-node* (FF0X::FD) et d'autre part en définissant un point d'entrée pour l'accès aux ressources, par défaut à l'URI /.well-known/core. Ce dernier permet de décrire les ressources disponibles sur le serveur.

Le mécanisme de *caching* permet de diminuer le trafic sur le réseau et de réduire les délais de réponse. CoAP se repose dessus pour optimiser les communications, permettant à un nœud intermédiaire entre le client et le serveur de réutiliser une Response antérieure stockée pour répondre à une Request ciblant la même ressource. À la différence d'HTTP, CoAP définit un message comme *cachable* selon le code de la Response, celle-ci étant accompagnée d'une Option *Max-Age* indiquant le temps de validité de l'information stockée le cas échéant.

Un proxy est un *CoAP endpoint* (supportant les services du protocole CoAP) qui peut être délégué par un client pour effectuer des Requests "à sa place". Il peut également faire du caching. Il apporte une augmentation des performances du réseau, permet d'accéder aux ressources protégées d'un serveur ou délayer la Request si le nœud serveur est endormi et ne peut répondre. Il peut faire la traduction entre CoAP et d'autres protocoles applicatifs, HTTP notamment. Les Requests vers un proxy peuvent indifféremment être des messages CON et NON. La spécification [?] distingue deux types de proxy :

- *forward-proxy* : ce proxy est explicitement désigné par le client qui passe l'URI de la ressource ciblée en conscience de cause dans des Options significatives pour le proxy. Soit sous la forme d'une chaîne contenant l'URI complet dans l'Option *Proxy-Uri*, soit en fournissant l'Option *Proxy-Scheme* indiquant le schéma cible (*coap*, *http*, ...) et les Options *Uri-\**. Ces Options vont alors être consommées par le proxy pour relayer la Request vers l'origin server qu'elles indiquent, supplantes par les Options *Uri-\** telles que celles qu'un client utiliserait pour une Request directe vers l'origin server (rôle alors de fait endossé par le proxy). Cela implique un parsing de l'URI et la fragmentation en Options *Uri-\** dans le cas de l'usage de l'Option *Proxy-Uri* et la suppression de l'Option *Proxy-Scheme* dans le second cas. Le proxy effectue la traduction vers le protocole indiqué par le schéma dans l'URI ou l'Option *Proxy-Scheme*.
- *reverse-proxy* : ce proxy est renseigné comme l'origin server dans l'URI que le client intègre dans sa Request, construite en considérant le proxy comme jouant directement le rôle du serveur. Le proxy agit alors de manière arbitraire, le comportement résultant se base sur le contenu de la Request et la configuration du proxy. Vraisemblablement, il va rediriger la requête vers un serveur qu'il sait en possession de la ressource, agissant lui-même comme un client en utilisant le protocole qu'il juge adapté pour communiquer avec ce serveur.

Un proxy qui fait de la traduction de requêtes entre protocoles (CoAP ↔ HTTP notamment) est désigné par le terme *cross-proxy* dans la spécification [?]. CoAP supporte un sous-ensemble des fonctionnalités de HTTP, mais la correspondance entre les deux peut être faite dans les deux sens par un cross-proxy. Ainsi, on a en fonction du sens de la requête :

- le cross-proxy CoAP-HTTP qui permet à un client CoAP n'implémentant pas HTTP d'accéder à une ressource détenue par un serveur HTTP. En fonction du type de proxy, le client CoAP construit sa requête différemment :
  - ▷ vers un forward-proxy, soit en intégrant l'URI commençant par le schéma `http://...` dans l'Option Proxy-Uri, soit en instanciant les Options Uri-\* accompagnées de l'Option Proxy-Scheme renseignant la valeur “`http`”
  - ▷ vers un reverse-proxy, en lui adressant une Request comme si c'était le serveur à contacter, que le proxy devra traiter de manière à répondre comme s'il était le serveur
- le cross-proxy HTTP-CoAP permet à un client HTTP de contacter un serveur CoAP, construisant sa requête en fonction du proxy considéré :
  - ▷ vers un forward-proxy, en introduisant un header HTTP *Request-Line* avec la valeur associée “`coap`”
  - ▷ vers un reverse-proxy, en forgeant sa requête comme s'il s'adressait à un serveur HTTP, que le proxy devra traiter de manière à répondre comme s'il était le serveur

Dans le premier cas, la traduction peut être effectuée assez facilement étant donné que CoAP ne définit pas de fonctionnalités qui n'ont pas leur équivalent pour HTTP. En revanche, le second cas peut poser problème et le proxy l'implémentant devra définir des mécanismes de filtrage pour ce qui n'a pas de correspondance dans CoAP (méthodes HTTP, options et codes).

#### 2.2.7.3 CoJP et OSCORE

Le protocole CoJP (*Constrained Join Protocol*, [?]) a été élaboré par le groupe 6TiSCH pour gérer la phase sensible qu'est le raccord d'un nouveau noeud au WSN. Il fait partie d'un framework minimal [?] écrit pour organiser l'échange de messages nécessaire au raccord utilisant un “canal sécurisé”, tout en limitant au maximum le nombre de messages échangés. CoJP définit la sémantique des messages, encodés par des structures de données CBOR (*Concise Binary Object Representation*, [?]) qui seront transportées comme des payloads de messages CoAP (voir Figure 2.16). Les Responses et Requests ainsi échangées sont protégées par OSCORE (*Object Security for Constrained RESTful Environments*, [?]) qui, en établissant un canal sécurisé, permet :

- une confidentialité des transmissions point à point
- l'authentification et la garantie d'intégrité des données
- une protection contre les attaques de *replay*
- une corrélation sécurisée entre Request et Response se correspondant

Les éléments liés à la couche logique Message de CoAP ne sont pas chiffrés par OSCORE car ceux-ci doivent être visibles par les équipements intermédiaires, notamment les proxys. Le Chapitre 3 analyse en détail CoJP et tous les mécanismes de la pile 6TiSCH relatifs à la procédure de raccord d'un nouveau noeud au WSN.

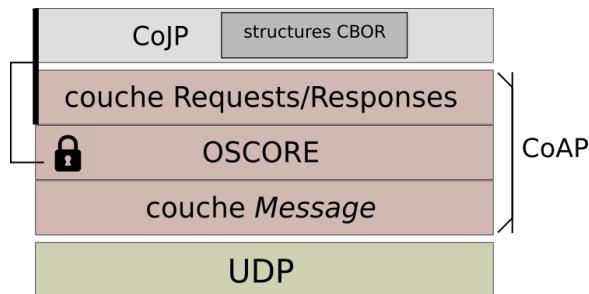


FIGURE 2.16 – CoJP et OSCORE positionnés par rapport aux couches logiques CoAP

## 2.3 Principes inhérents au déploiement de la pile

### 2.3.1 Synchronisation temporelle

Dans un réseau TDMA, les noeuds doivent restés synchronisés temporellement pour assurer les communications, ce qui revient dans le cas de TSCH à faire en sorte que tous les timeslots soient alignés et commencent au même instant pour chaque noeud. Le phénomène de clock drifting décalant les horloges des noeuds relativement les une aux autres, il est quantifié en *ppm* (*parts per million*). Si deux noeuds ont leur horloges dérivant de façon opposée à 10 *ppm* (l'une “ralentissant” et l'autre “accélérant”), une seconde après qu'elles aient été synchronisées elles présenteront un décalage relatif de 20  $\mu s$ . Pour absorber ce décalage, la Section 2.2.3.4 présentait l'usage d'une période tampon de *macTsRxWait*  $\mu s$  désignée par *guardtime* dans la littérature. Plusieurs méthodes sont utilisées pour resynchroniser les noeuds entre eux, [?] exprime la durée maximum entre les resynchronisations par la formule suivante :

$$MaxResyncPeriod = \frac{guardtime}{clockdrift} \quad (2.2)$$

Passé ce délai, les noeuds seront désynchronisés et ne pourront plus s'entendre l'un l'autre, ce qui potentiellement exclut un noeud du réseau qu'il devra rejoindre à nouveau. Typiquement, on observe un clockdrift de 10 *ppm*, en se plaçant de le pire cas où il a lieu de façon opposée pour deux noeuds et considérant un guardtime de 1 ms, on a

$$MaxResyncPeriod = \frac{1 \text{ ms}}{20 \times 10^{-6}} = 0,5 \times 10^5 \text{ ms} = 50 \text{ s}$$

#### 2.3.1.1 Resynchronisation entre noeuds voisins

La resynchronisation peut se faire de façon implicite en utilisant les échanges de messages qui font partie du trafic régulier dans le WSN (paquets de données et ACKs). S'il n'y a aucun trafic (noeud endormi), un message *keepalive* expressément destiné à la resynchronisation doit être échangé dans un délai inférieur à la valeur de *MaxResyncPeriod* s'appliquant aux noeuds. De façon approximative, [?] estime le duty cycle engrangé par ces messages de maintient à 0,02% (cette valeur augmentant linéairement avec le clockdrift de l'horloge). Dans une communication entre voisins, le standard IEEE802.15.4e [?] considère deux méthodes pour resynchroniser les noeuds (supposant qu'un des deux est la time source de l'autre). Dans les deux cas, le receveur de la transmission calcule la différence de temps  $\Delta$  entre l'instant attendu d'arrivée de la frame (fixe par rapport au début du timeslot) et le moment où le premier symbole de la frame est réellement reçu.

- *ACK-based* : permet à l'émetteur de resynchroniser son horloge sur celle du receveur
  1. L'émetteur envoie le premier symbole de la frame *macTsTxOffset*  $\mu s$  après le début du timeslot selon son horloge, ce qui devrait se produire *macTsRxOffset* +  $\frac{guardtime}{2}$   $\mu s$  après le début du timeslot selon la perception du temps du receveur si les deux horloges étaient parfaitement synchronisées
  2. Le receveur marque le *timestamp* du moment où le premier symbole est reçu et calcule la correction  $\Delta = macTsRxOffset + \frac{guardtime}{2} - timestamp \mu s$ . On a alors  $\Delta < 0$  si le receveur est “en retard” par rapport à l'émetteur en terme de perception du temps (et vice-versa si  $\Delta > 0$ ).
  3. Le receveur renseigne cette valeur de  $\Delta$  dans l'IE *Time Correction* de la frame ACK qu'il renvoie à l'émetteur (à la suite dans le même timeslot).
  4. L'émetteur reçoit l'ACK et, si le receveur est renseigné comme sa time source, se resynchronise dessus en ajustant son horloge de  $\Delta \mu s$  (valeur signée).
- *Frame-based* : permet au receveur de resynchroniser son horloge sur celle de l'émetteur
  1. Déroulement identiques aux étapes 1 → 2 de la méthode ACK-based.
  2. Si le receveur a l'émetteur renseigné comme sa time source, se resynchronise dessus en ajustant son horloge de  $\Delta \mu s$  (valeur signée), un ACK est renvoyé mais il ne servira pas à la resynchronisation.

### 2.3.1.2 Synchronisation globale du réseau

La notion de temps globale, commune à tous les nœuds d'un même réseau 6TiSCH, est apportée par l'ASN. Celui-ci a été initialisé à 0 à la création du réseau et est incrémenté d'une unité à chaque timeslot expiré, ce qui implique que la durée d'un timeslot est fixe pour tous les nœuds du réseau. Encodé sur 5 bytes, considérant des timeslots de 10 ms, ce compteur est suffisant pour une utilisation sur plus d'une centaine d'années [?]. Il est annoncé explicitement dans les EBs émis par les nœuds déjà membre du réseau (voir Section 2.3.2), mais est toujours implicite dans les communications internes. Un nœud qui a une mauvaise valeur d'ASN calcule des sauts de fréquences erronés et perd toute connectivité avec les autres nœuds. Outre les calculs de fréquence, l'ASN est utilisé dans la composition du nonce pour les opérations de sécurité, de par sa nature de compteur différent à chaque communication successive dans le temps.

À n'importe quel moment, un nœud membre du réseau doit avoir un parent time source sur lequel il réajuste son horloge par les procédés ACK/frame-based. Les boucles de synchronisation temporelle doivent être évitée, aussi les relations entre time sources doivent être structurées. [?] soutient le fait que cette structure devrait profiter de la topologie déjà formée par RPL (DODAG), un parent time source d'un nœud étant choisi comme le parent RPL dans le DODAG. La spécification [?] va plus loin en recommandant une Instance RPL globale dédiée à la synchronisation : la TSGI (*Time Synchronization Global Instance*). La configuration dépend alors de l'architecture et de la présence d'une source temporelle commune entre plusieurs WSN. Un nœud, quand il annonce le réseau avec des EBs, communique une valeur *Join Priority* qui indique à quel point il serait un bon point de raccord au réseau pour un nouveau nœud. Si un nœud n'est pas encore dans le DAG associé à la TSGI, il annonce une Join Priority de 0xFF indiquant qu'il ne peut pas être time source et ne souhaite pas servir de point de raccord. Une fois inclus à la TSGI, il annonce une Joint Priority basée sur son rang dans le DODAG qu'il a rejoint. Une faible valeur indiquant un meilleur point de raccord, cela est cohérent avec l'approche en gradient de RPL : au plus il est proche de la source temporelle du DODAG au mieux c'est.

Différentes politiques existent pour contrôler le “trafic de resynchronisation” dans le réseau, l'objectif étant de devoir émettre un minimum de messages keepalives qui n'ont que cette utilité. D'autres trafics de contrôle réguliers peuvent conjointement servir d'appoint pour la synchronisation, notamment les messages de signalement RPL et l'émission d'EBs. [?] propose que la couche 6top puisse influer sur le timer Trickle de RPL afin de réguler l'envoi de messages de signalement DIO à une cadence qui permette aussi d'assurer le maintien de la synchronisation en parallèle. Il n'y a pas de standardisation claire de ces politiques.

### 2.3.2 Formation du réseau

Un réseau 6TiSCH se forme progressivement à partir du 6LBR auquel les nœuds restreints à portée se raccordent. Les autres nœuds qui ne sont pas à portée directe utiliseront des nœuds déjà raccordés comme intermédiaires, formant un réseau multihop. Le processus de raccord, désigné dans ce document par *joining phase* est composé de deux étapes distinctes :

1. L'étape de *scanning/advertising* : un nœud désireux de rejoindre un réseau, appelé *pledge*, écoute dans un des canaux possibles afin d'intercepter un EB émis par un nœud déjà membre du réseau. Cet EB contient toutes les informations nécessaires au pledge pour procéder au raccord en passant à l'étape suivante (synchronisation temporelle, cells à utiliser pour le reste du join, etc.).
2. L'étape d'intégration au réseau : CoJP est utilisé par le pledge pour communiquer avec l'autorité du réseau qui validera son raccord en lui renvoyant les paramètres nécessaires à son intégration complète (notamment les clés actives dans le réseau au niveau lien). Cet échange se fait en utilisant un nœud déjà raccordé comme intermédiaire et n'est composé que d'un message de requête et sa réponse par l'autorité. Le WG 6TiSCH en définit le format des messages et le déroulement intégral dans la spécification [?].

Le WG 6TiSCH envisage deux types de déploiement en fonction de la façon dont les paramètres pour procéder à l'intégration sont injectés dans les nœuds, avant qu'ils ne rejoignent le réseau. Leur application dépend directement des besoins pratiques liés au déploiement. Ces deux options sont [?] :

- le mécanisme *One-Touch* : le pledge est supposé partager une clé unique avec l'autorité du réseau sans qu'ils n'aient pour autant communiqué : elle doit leur être fournie manuellement par un moyen quelconque.
- le mécanisme *Zero-Touch* : la configuration a lieu en usine durant la fabrication de l'équipement. Le fabricant inclut un certificat digital et maintient un service de validation en ligne que l'autorité du réseau 6TiSCH pourra contacter afin de valider un certificat présenté par un pledge.

Dans le cadre de ce travail, seul le mécanisme de One-Touch est considéré car c'est celui qui permet d'avoir le plus d'emprise sur la phase de déploiement. La phase de joining One-Touch et son aspect sécurité sont étudiés dans le Chapitre 3.

Spécifiquement conçu pour 6TiSCH, le protocole CoJP a été pensé de façon à limiter au maximum les ressources consommées par les équipements durant l'étape d'intégration, du point de vue du pledge mais également du réseau en place. En revanche, l'étape de scanning/advertising est beaucoup plus délicate, des deux côtés. Le standard IEEE802.15.4 [?] ne définit ni le contenu des EBs, ni la stratégie d'émission à adopter pour les annoncer aux pledges. Les caractéristiques du contenu d'un EB (voir Section 2.2.3.2), ainsi qu'une *minimal slotframe* utilisable pour leur émission sont décrits dans la spécification [?]. La procédure de scanning pour capter un EB est très coûteuse pour un pledge [?], celui-ci n'ayant à priori aucune connaissance des moments auxquels ils sont émis par les membres du réseau et dans quel canal physique.

Les principaux travaux qui s'y intéressent dans la littérature ciblent la stratégie d'annonce des EBs, permettant par des techniques variées d'ajuster le compromis temps de raccord au réseau/énergie consommée. [?] étudie l'utilisation du timer Trickle de RPL pour coupler les émissions d'EBs à celles des DIOs, les désavantages de cette technique employée en pratique dans ContikiOS, et en proposent finalement un dérivé. [?] considère la minimal slotframe (donc statique) qui induit un comportement similaire à slotted ALOHA, mais évite les collisions avec une approche probabiliste pour l'émission d'EB dans ces cells. [?] étudie la possibilité d'émettre plus qu'un seul EB dans un même timeslot d'annonce, mais à des fréquences différentes pour accélérer le processus de scanning par le pledge.

### 2.3.3 Ordonnancement des transmissions - *minimal scheduling* avec MSF

Le WG 6TiSCH a défini une SF basique mais polyvalente, en adéquation avec la configuration minimum qui a été décrite dans la spécification [?]. La MSF (*Minimum Scheduling Function*) fait l'objet d'un draft [?] décrivant son comportement et ses interactions avec la couche 6top. Elle gère d'une part l'établissement d'un schedule minimal utilisable pour la phase de join et d'autre part l'adaptation dynamique de cells négociées entre un noeud et son parent RPL. La MSF est prévue pour un trafic régulier vers la racine. Elle utilise la minimal cell décrite dans [?] pour les émission broadcast d'EBs et de DIOs, distinguant trois slotframes :

0. réservée au trafic de *bootstrapping*, minimal cell en broadcast permettant de lancer la procédure de join
1. cells autonomes installées automatiquement pour établir une première communication directe
2. cells additionnelles négociées par le biais de transactions 6P en fonction du trafic observé

MSF ne considère des communications bidirectionnelles qu'avec un noeud *preferred parent*, qui n'est autre que le parent RPL une fois la topologie rejointe (ou le noeud servant d'intermédiaire lors du join). Afin de minimiser les transactions 6P, surtout si elles doivent passer par la minimal cell, MSF prévoit l'installation d'*autonomous cells* dont les coordonnées sont obtenues à partir d'un hash de l'EUI-64 (extension de l'adresse MAC). Ainsi, chaque noeud est censé maintenir une cell *AutoRxCell* dans la slotframe 1 dont les coordonnées (*slotOffset*, *channelOffset*) sont calculées à partir de l'EUI-64 de sa propre adresse (donc fixes). Le calcul tient compte du fait que la minimal cell est installée d'office et ne retourne donc jamais un *slotOffset* égal à 0. Pour envoyer une frame vers un voisin, il peut de façon similaire installer une cell *AutoTxCell* à partir de l'EUI-64 de ce voisin, ce dernier ayant alors déjà installé aux même coordonnées son *AutoRxCell*.

La MSF gère la façon dont les communications pour le processus de join s'organisent en terme de cells, principalement en installant temporairement des cells autonomes pour le 6LoWPAN ND et l'échange CoJP, ainsi que la minimal cell pour recevoir les DIOs par la suite. Une fois la topologie intégrée (rang RPL acquis), le noeud pourra négocier ses premières cells avec son parent (slotframe 2) et annoncer des EBs et DIOs (slotframe 0). À terme de la phase de join, le noeud sera synchronisé et authentifié dans le réseau, participera à la topologie activement (selon ses capacité de routage) et possédera des cells Tx négociées vers/depuis son parent.

La négociation de cells dans la slotframe 2 a lieu en réaction à trois événements :

- les ressources allouées au trafic (bande passante en terme de nombre de cells) ne correspondent pas au trafic observé. Si elles sont trop faibles, la MSF déclenche une transaction ADD pour ajouter des cells, si les cells déjà installées sont trop peu utilisées elle déclenche une transaction DELETE.
- suite à un changement de parent dans la topologie. La MSF tente de réinstaller un nombre équivalent de cells avec le nouveau parent (transactions ADD), puis lance une transaction CLEAR avec l'ancien parent pour éliminer les cells précédemment négociées.
- pour traiter la détection d'une collision entre schedules, suite à l'observation d'une différence de PDR ( $\text{Packet Delivery Ratio} = \frac{\text{numTxAck}}{\text{numTx}}$ ) entre deux cells négociées avec le parent. Cela peut arriver car la négociation est par nature un processus distribué. La MSF déclenche alors une transaction RELOCATE pour déplacer ces cells problématiques et les installer à des coordonnées qui n'induisent aucune collision.

Le draft MSF [?] définit encore d'autres mécanismes, tels que le maintient intelligent d'un *cell pool* dans lequel piocher les cells à ajouter ainsi que la gestion du timeout et des erreurs 6P.

Il existe d'autres SFs proposées, assurant des propriétés et garanties différentes en fonction des besoins intrinsèques au déploiement. On peut notamment citer le draft de la SF0 [?], LLSF pour *Low Latency SF* [?] qui améliore SF0 en agençant les timeslots en *daisy chain* et ReSF pour *Recurrent Low-Latency SF* [?] qui utilise un modèle pour optimiser le scheduling en présence de trafic récurrent.

## Chapitre 3

# Analyse détaillée de la sécurité pour la phase de join

## 3.1 Ressources et standards utilisés

### 3.1.1 CBOR et COSE

CBOR (*Concise Binary Object Representation* [?]) est un format de données spécialement conçu pour le traitement de messages par des systèmes restreints. Le code requis pour l'encodeur et le décodeur est minimal, les messages sont de taille réduite tout en permettant l'extensibilité. La spécification se base principalement sur celles définissant le format JSON, adaptant les besoins à ceux de communications entre systèmes restreints. Les données au niveau applicatif échangées lors de la phase de join sont formatées en utilisant CBOR.

COSE (*CBOR Object Signing and Encryption* [?]) est une spécification définissant les services de sécurité associables au format CBOR. Elle décrit comment créer et traiter les signatures, les codes d'authentification de message (MAC) et le chiffrement des données sérialisées avec CBOR. Le transport de clés assisté par CBOR y est également discuté.

### 3.1.2 OSCORE

#### 3.1.2.1 Principes fondamentaux d'OSCORE

OSCORE [?] et sa place dans la pile 6TiSCH sont brièvement introduits par la Section 2.2.7.3. OSCORE décrit les mécanismes et concepts nécessaires à l'établissement d'un canal sécurisé au niveau applicatif entre deux noeuds comprenant CoAP, mais qui souhaitent protéger un maximum d'informations dans les messages échangés. Effectivement, certains champs CoAP (des Options notamment) doivent rester visibles par les proxys opérant comme intermédiaire au niveau applicatif entre les deux noeuds. D'autres, tels que le payload et le code du message doivent être protégés des proxys, qui pourraient sinon les consulter et les manipuler. En d'autres termes, OSCORE agit uniquement sur la couche logique Request/Response de CoAP, du client au serveur et vice-versa. Techniquement, la sécurité du canal entre ces deux points est assurée par COSE qui opère le chiffrement et les vérifications d'intégrité des champs à protéger, la protection contre les attaques replay et la liaison entre Request et Response.

L'emploi d'OSCORE dans le cadre de la phase de join est justifié par le fait qu'un pledge utilise un proxy pour échanger avec le réseau tout du long de la phase de join. Plus précisément, afin d'obtenir les informations nécessaires à son intégration dans un réseau 6TiSCH (les clés actuellement en usage notamment), le pledge va se servir du noeud déjà intégré au réseau comme d'un proxy intermédiaire (CoAP). Ce proxy va accepter les frames émanant du pledge même si elles ne sont pas sécurisées, mais relayer le payload applicatif en l'encapsulant dans des frames sécurisées puisque, lui, possède déjà les clés du réseau. Il les relaie vers l'autorité du réseau qui validera le pledge et répondra avec le matériel nécessaire pour que ce dernier puisse faire partie intégrante du réseau et échanger des frames sécurisées avec les noeuds. Le noeud qui faisait office de proxy devra alors forward cette réponse vers le pledge en attente. Les informations ainsi échangées entre le pledge et l'autorité du réseau doivent rester confidentielles, et même le proxy ne devrait théoriquement pas y avoir accès. Cependant, pour effectuer son travail de proxy CoAP, il a besoin d'accéder à certaines parties applicatives du message, d'où l'emploi d'OSCORE qui permet les deux en même temps.

L'emploi de DTLS à la place d'OSCORE n'est pas une solution viable, car un proxy intermédiaire entre le pledge et l'autorité aurait alors accès à l'entièreté du contenu applicatif. Gündoğan *et al.* [?] mettent en avant d'autres problèmes dans leur travail, tels que la baisse de performance que l'établissement de sessions DTLS occasionne. Aucun chiffrement de ces échanges l'est encore moins, les frames échangées entre le pledge et le proxy n'étant pas non plus protégées au niveau de la couche lien. Une description détaillée des échanges spécifiques à la phase de join se reposant sur OSCORE est établie dans la section Section 3.1.3. La suite de cette section décrit les mécanismes propres à OSCORE, sans les relier à leur utilisation dans la phase de join. Elle fait état de la façon dont un canal fortement sécurisé au niveau applicatif peut être établi entre deux hôtes, en tenant compte de la présence de proxy entre eux. OSCORE utilise de nombreux concepts, considère que certaines informations sont partagées entre hôtes, se repose sur des technologies de chiffrement etc., aussi certains aspects sont ignorés ici, tout en fournissant le bagage nécessaire à la compréhension de son emploi dans la procédure de join 6TiSCH.

### 3.1.2.2 Processus d'encapsulation OSCORE

OSCORE peut être utilisé pour protéger des échanges CoAP, mais également HTTP ou même une hybridation des deux (proxys cross-protocol). Afin d'établir un canal sécurisé pour des communications, le client et le serveur établissent un *contexte de sécurité* partagé qui permettra de dériver le matériel nécessaire (clés de chiffrement, etc., voir plus loin) pour la sécurisation par un algorithme AEAD (*Authenticated Encryption with Authenticated Data* [?]). Un algorithme AEAD permet le chiffrement et l'authentification de données, assurant la confidentialité et l'authenticité de façon combinée. De plus, certaines parties des données peuvent être uniquement authentifiées car elles sont destinées à rester lisibles. COSE décrit l'encadrement autour de l'utilisation de cet algorithme et en place la sortie dans un objet CBOR structuré *COSE\_Encrypt0* [?]. OSCORE récupère dans les champs de cette structure les pièces pour constituer le contenu d'un paquet générique, appelé *message OSCORE* et qui encapsule les champs protégés du paquet original. L'encapsulation peut se faire dans un message HTTP ou CoAP, en fonction de ce que le proxy destination suivant comprend (cross-proxy ou simple proxy CoAP). Le processus complet est le suivant (voir Figure 3.1) :

1. Transformation du message original en message CoAP intermédiaire dont les champs sont divisés en deux catégories : les *outer fields* et les *inner fields*. La première catégorie désigne les champs qui seront visibles par un proxy, la seconde désigne ceux qui seront chiffrés. Les Options CoAP sont également liées à un type de classe prédefini : U pour *Unprotected*, I pour *Integrity only* et E pour *Encrypted*.
2. Les différents champs à protéger sont groupés en fonction de comment l'algorithme AEAD devra les traiter, le *plaintext* à chiffrer et authentifier est détaillé par la Figure 3.3, les autres composants le sont plus loin. L'algorithme AEAD est appliqué en utilisant la clé du contexte de sécurité, produisant en sortie un objet structuré *COSE\_encrypt0* qu'on désigne par *COSE\_obj*.
3. À partir des composants du message CoAP intermédiaire et des champs calculés du *COSE\_obj* obtenus après application de l'algorithme AEAD, un message OSCORE encapsulant est construit. L'option OSCORE y est rajoutée, illustrée par la Figure 3.2. Il peut être HTTP ou CoAP et signale qu'il s'agit d'un message OSCORE par soit un header HTTP, soit directement l'Option OSCORE pour CoAP.

La procédure inverse est effectuée à la réception du paquet par l'hôte destination, celui-ci se servant du même matériel partagé issu contexte de sécurité qu'il est censé connaître. L'Option OSCORE sert entre autres à ce qu'il puisse récupérer le bon contexte parmi ceux qu'il connaît.

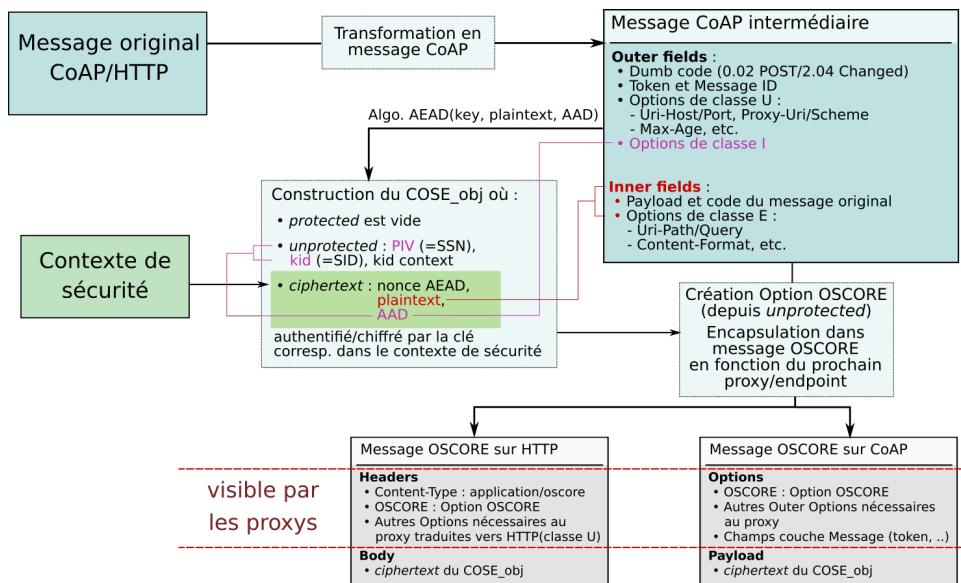


FIGURE 3.1 – Schéma de la création d'un message OSCORE sécurisant un message original

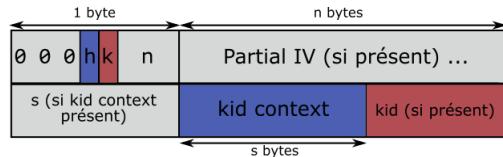


FIGURE 3.2 – Contenu de l’Option OSCORE

FIGURE 3.3 – *plaintext* inclus dans le COSE\_obj

### 3.1.2.3 Contextes de sécurité et échanges sur le canal OSCORE

Un contexte de sécurité regroupe tout le matériel nécessaire à la sécurisation des échanges avec un hôte distant, pour une ressource donnée. OSCORE fait l’hypothèse que les deux hôtes partagent des paramètres pré-établis qui constituent le *common context* (CC) : une clé dite *Master Secret* (et optionnellement un *Master Salt*), un vecteur d’initialisation *Common IV* et quels sont les algorithmes précis à utiliser pour le procédé de dérivation de clés/chiffrement du COSE\_obj. De ce CC, sont “dérivés” deux contextes distincts pour chaque hôte : le *sender context* (SC) qui est utilisé pour protéger les messages à envoyer et le *recipient context* (RC) utilisé pour déchiffrer les messages reçus. Ces contextes sont littéralement dérivés dans le sens où une fonction de dérivation de clés HKDF (*HMAC-based Extract-and-Expand Key Derivation Function*, [?]) construit, à partir du CC et d’un ID donné, une nouvelle clé spécifique au contexte en question. Dans le cas du SC, une *sender key* (SK) et dans le cas du RC, une *recipient key* (RK). L’ID en question est un nom (chaîne de caractères) qui désigne un hôte et est appelé *sender ID* (SID) en association avec le SC et *recipient ID* (RID) avec le RC. Il a une valeur arbitraire et est établi par un moyen non spécifié, mais un hôte doit avoir un SID pour envoyer un message (propre à lui-même) et un RID pour valider un message reçu (propre à l’émetteur du message).

La Figure 3.3 schématisé l’utilisation des contextes dans un échange : il faut noter que pour un envoi de message le SC utilisé par l’émetteur contient le même matériel que le RC utilisé par le récepteur. Effectivement, ils doivent utiliser une clé identique pour la protection et la vérification de la ressource. Le terme “réécupération de contexte” désigne le fait que soit le contexte existe (déjà dérivé) et est récupérable depuis les éléments en substance, soit il est dérivé à la volée depuis le CC en utilisant les éléments en substance. Un élément non repris dans la Figure 3.3 est le *context ID* : inclus optionnellement dans l’option OSCORE, il permet de distinguer les contextes de sécurité et préciser lequel utiliser (à partir que quel CC dériver). Il permet l’utilisation d’un même SID interprétable dans des contextes distincts et est employé durant la procédure de join 6TiSCH.

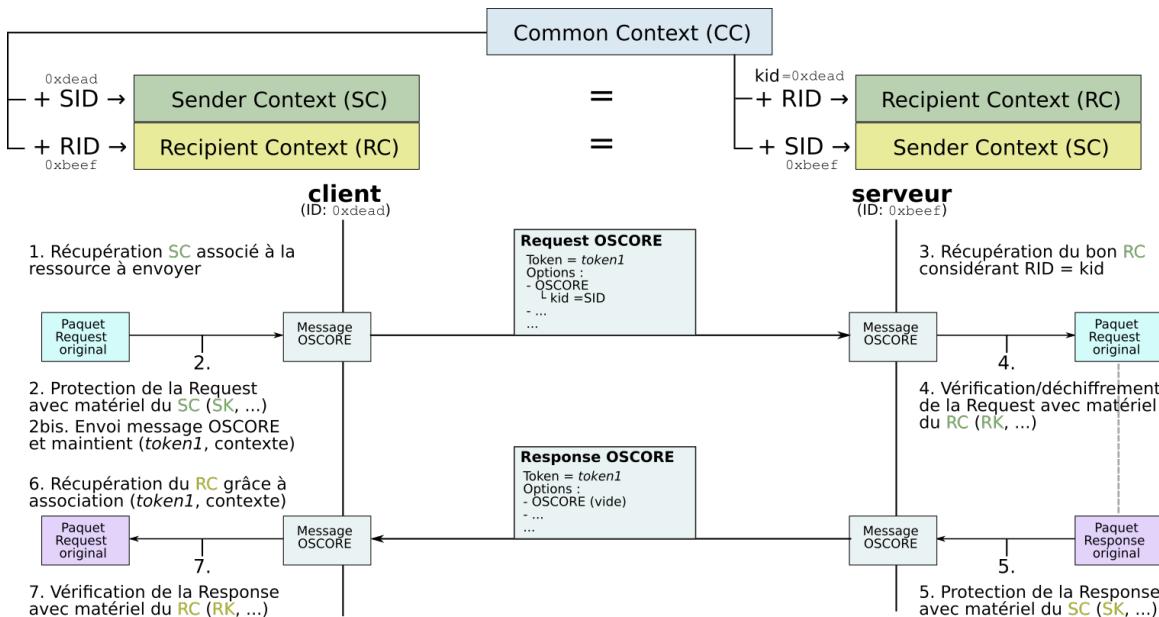


FIGURE 3.4 – Schéma fonctionnel d’un échange OSCORE et des contextes utilisés pour sécuriser le “canal”

La Figure 3.1 établit que le champ *ciphertext* du COSE\_obj contient, outre le plaintext, lAAD (*Additional Authenticated Data*) et le nonce AEAD. LAAD désigne les données qui doivent être authentifiées mais rester lisibles, et donc répliquées dans le champ *unprotected* du COSE\_obj. LAAD est passé indépendamment du plaintext à l'algorithme AEAD, et est calculé comme un bytestream d'une structure *Enc\_structure* [?] dont le contenu est illustré par la Figure 3.5. On y retrouve logiquement les Options catégorisées en classe I (vérification d'intégrité uniquement).

```
AAD = Enc_structure = ["Encrypt0", "", external_aad]
                                bytestream de
                                aad_array = [
                                    oscore_version, (=1)
                                    algorithms,      ([algo. AEAD util.])
                                    request_kid,    (=kid=SID)
                                    request_piv,    (=partial IV=SSN)
                                    options,        (=Options classe I)
                                ]
```

FIGURE 3.5 – Détail de quels champs du paquet original et valeurs sont utilisés pour calculer lAAD

Le nonce AEAD est inclus dans toute Request émise par un client. Il permet de vérifier le partial IV qui a la valeur du Sender Sequence Number (CoAP) du client et qui est comparé avec la Replay Window du serveur. La Replay Window sert de parade aux attaques de types replay. La construction du nonce AEAD (Figure 3.6) est indirectement liée à la clé utilisée pour chiffrer le COSE\_obj, car le Common IV a été calculé en utilisant cette dernière. Il dépend également du SID du client et du partial IV. Ces éléments sont connus par le serveur à la réception de la Request, le Common IV étant dans le common context et l'Option OSCORE contenant le SID (valeur de *kid*) et le PIV (voir Figure 3.2). Le serveur peut donc calculer le nonce AEAD et le comparer à celui contenu dans le *ciphertext* de la Request reçue.

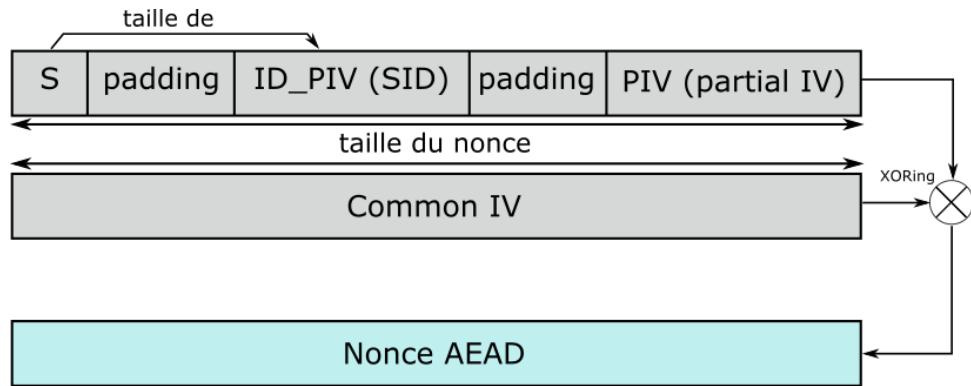


FIGURE 3.6 – Procédure de calcul du nonce AEAD à partir du PIV (=Sender Sequence Number)

La Figure 3.7 résume un échange Request/Response protégé dans un canal OSCORE dans sa globalité : le dérivation des contextes, la formation et le chiffrement des messages OSCORE et la place du proxy dans cet échange. La spécification OSCORE [?] est beaucoup plus complète, notamment au niveau des conditions/garanties de sécurité, mais n'offre pas une telle vue d'ensemble de tout le processus.

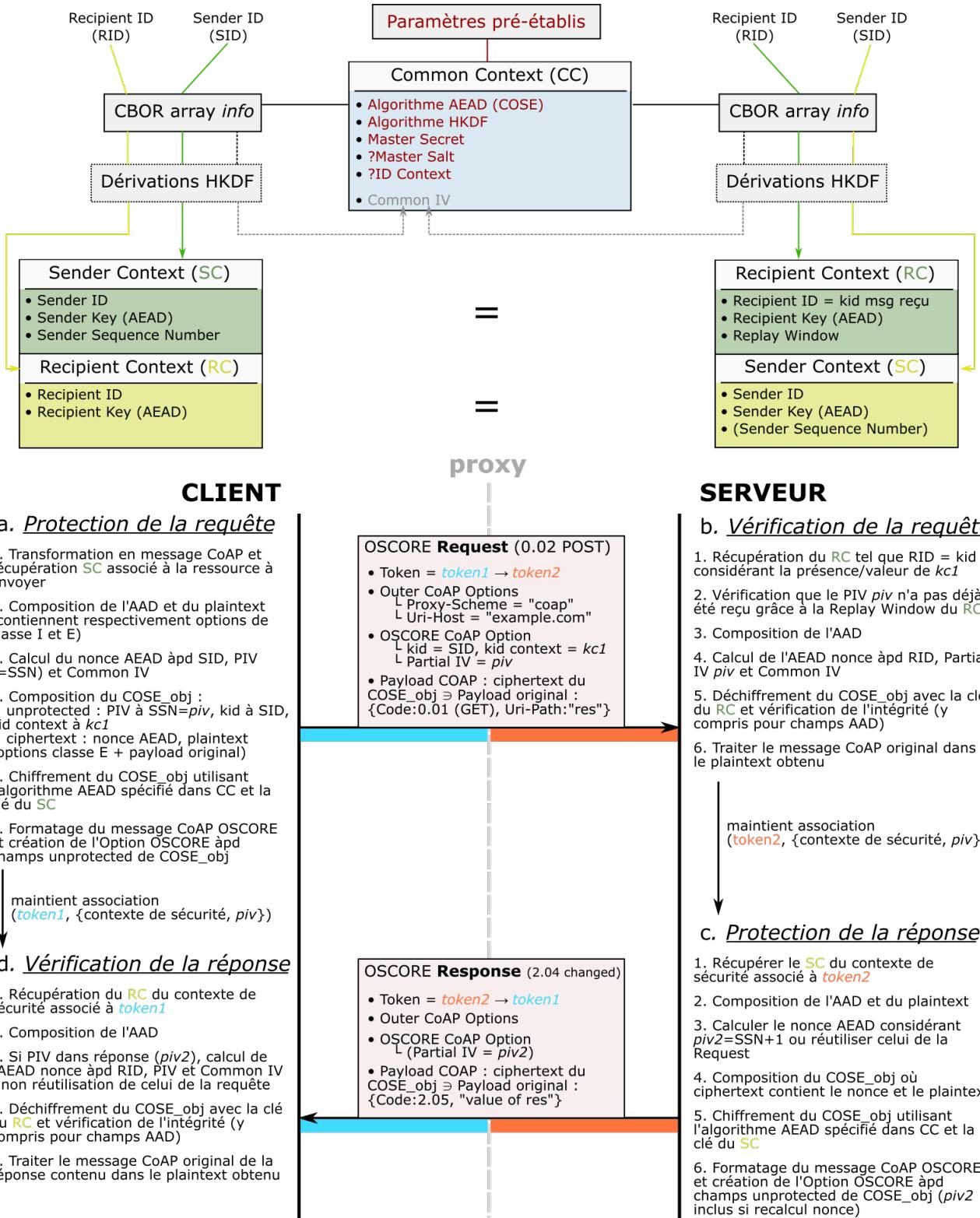


FIGURE 3.7 – Vue d'ensemble d'un échange client-serveur protégé par OSCORE : dérivation des contextes, formation et procédure de chiffrement COSE des messages et place du proxy dans l'échange

### 3.1.3 CoJP

Le protocole qui encadre la phase de join 6TiSCH est intégralement décrit par le WG 6TiSCH dans la spécification “Minimal Security Framework for 6TiSCH” [?] qui est toujours en phase de draft. Il s’agit de CoJP (*Constrained Join Protocol*) qui, comme introduit dans la Section 2.2.7.3, se repose sur CoAP, OSCORE et CBOR pour les échanges d’informations relatives au processus de join. Afin de répondre à des questions de mise en pratique des standards 6TiSCH et particulièrement de la phase de join et de sa configuration, le standard “Minimal IPv6 over the TSCH Mode of IEEE802.15.4e (6TiSCH) Configuration” [?] est utilisé. Dans le reste de ce chapitre, les éléments de ce standard sont considérés et employés de façon transparente quand la spécification [?] ne les précise pas directement, celle-ci faisant certaines abstractions vis-à-vis de la mise en pratique.

La phase de join permet le join sécurisé d’un nouveau noeud à un réseau 6TiSCH déjà établi. Le terme de “join sécurisé” réfère à un accès authentifié au réseau et la distribution des paramètres de configuration nécessaires au noeud pour intégrer le réseau (clés, identifiants, etc.). Dans le cadre de la joining phase, les acteurs sont les suivants (illustrés dans la Figure 3.8) :

- le **pledge** : un noeud 802.15.4 qui souhaite rejoindre le réseau 6TiSCH, dont l’EUI-64 est désigné ici par *pledgeID*
- le **JRC** (*Join Registrar/Coordinator*) : une entité centrale qui a autorité sur le réseau 6TiSCH, mais pouvant être située à son extérieur dans un backbone.
- le **JP** (*Join Proxy*) : le pledge n’ayant pas un accès direct au JRC, il a besoin d’un noeud intermédiaire déjà membre du réseau pour acheminer ses messages vers ce dernier. Un join proxy effectue ce travail, agissant comme un proxy au niveau applicatif (CoAP).
- le **joined node** : terme désignant un pledge qui vient de finaliser avec succès son join sécurisé.
- le **6LBR** (*6LoWPAN Border Router*) : noeud faisant la jonction entre le réseau WSN 6TiSCH et le backbone (illustré dans la Section 2.1), supposé comme étant la racine de l’arborescence RPL

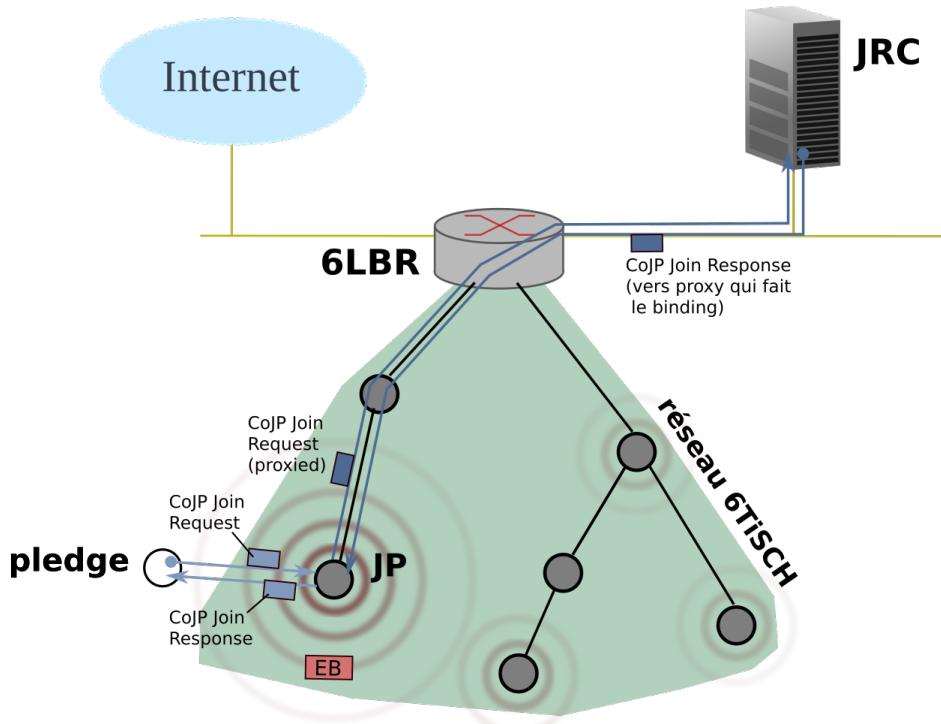


FIGURE 3.8 – Place dans le réseau des différents acteurs de la phase de join

### 3.1.3.1 Problématiques liées à la joining phase

La joining phase est critique dans la création et le maintient de la stabilité d'un réseau 6TiSCH. À noter que ce réseau reste un réseau 802.15.4 et est donc vu au niveau lien comme une structure PAN. La joining phase encadre notamment les problématiques suivantes :

1. Un pledge n'a aucune connaissance du réseau, si ce n'est certains paramètres pré-établis. Il doit obtenir des éléments essentiels tels que les clés courantes utilisées pour protéger le trafic au niveau lien, un identifiant dans le réseau (sous-entendu une short address), l'adresse IPv6 du JRC et un parent RPL.
2. Les paramètres fournis à un joined node peuvent changer au cours du temps. Par exemple, en cas de rekeying du réseau suite à une détection d'attaque. Il convient de redistribuer les nouveaux paramètres de façon consistante.
3. Le pledge doit pouvoir s'authentifier auprès du JRC et communiquer de façon sécurisée avec ce dernier. Des paramètres sont pré-établis à ces fins (pour communiquer sur un canal OSCORE) mais la façon dont ceux-ci sont distribués au pledge et JRC de manière concordante n'est pas précisée.
4. Le mode TSCH implique qu'un pledge n'a à priori aucune idée de quand et sur quel canal écouter à la recherche d'EB qui lui permette d'initier le processus de join. Il s'agit donc d'un scan passif qui coûte cher au pledge énergétiquement parlant.
5. Les microcontrôleurs constituant les nœuds de WSNs ont un espace mémoire limité, le processus de join ne devrait pas faire intervenir de nouvelles technologies ou protocoles qui ne sont pas déjà utilisés de façon générique dans 6TiSCH.

### 3.1.3.2 Hypothèses faites par CoJP (paramètres pré-établis)

La spécification [?] fait l'hypothèse que certains paramètres sont pré-établis, pour le pledge et le JRC. Cela peut se faire physiquement par une interface JTAG, une console, *over-the-air* dans une cage de Faraday, etc. ou en utilisant par exemple un protocole d'échange de clés. Ces **paramètres pré-établis** sont les suivants (ceux qui sont optionnels sont notés entre [...] ) :

- identifiant du pledge (*pledgeID*) : doit être unique parmi l'ensemble des nœuds sur lequel le JRC a autorité. Typiquement mis à l'EUI-64 qui est globalement unique et attribué d'office à une interface 802.15.4 (dérivé de l'adresse MAC). Peut aussi être issu d'un autre procédé de génération qui respecte l'unicité.
- une clé partagée (*PSK, Pre-Shared Key*) : une clé secrète partagée entre le pledge et le JRC, ce dernier doit de ce fait stocker quelle clé est assignée à chaque *pledgeID*. Ces clés doivent être cryptographiquement fortes (minimum 128 bits, générées aléatoirement). Un travail portant sur l'échange préliminaire sécurisé des clés utilisables par OSCORE est en cours de standardisation, désigné par LAKE (*Lightweight Authenticated Key Exchange protocol*, [?]).
- [ un identifiant de réseau (*netwID*) ] : identifie le réseau 6TiSCH, annoncé dans les EBs émis par les nœuds qui en font déjà partie. Il s'agit typiquement du PAN ID. S'il n'est pas fourni à un pledge, celui-ci tentera de rejoindre n'importe quel réseau sans distinction.
- [ les algorithmes différent de ceux par défaut ] : si non spécifiés, la spécification [?] définit quel algorithme utiliser pour quelle action par défaut (actions relatives à la sécurité).
- Pour un pledge qui aspire à servir de 6LBR, l'adresse IPv6 GUA du JRC : permet au pledge de contacter directement le JRC sans utiliser de nœud intermédiaire (par son interface vers le backbone). Un pledge normal l'apprend dynamiquement au cours du processus de join.

### 3.1.3.3 Composants du protocole CoJP et leur sécurisation

Dans cette sous-section, les mécanismes et éléments propres au protocole CoJP sont décrits. Le déroulement de la phase de join dans sa globalité et l'utilisation de CoJP durant celle-ci sont détaillés dans la Section 3.2. CoJP définit deux échanges de messages distincts (illustrés par la Figure 3.9) :

1. *Join Exchange* : celui inclus dans la phase de join à proprement parler. C'est dans cet échange que sont transportés
  - (a) la demande de join émise par le pledge vers le JRC (proxied par le JP qui est le seul noeud que peut atteindre le pledge)
  - (b) la réponse du JRC en fonction de la demande du pledge, du JRC vers le pledge (vers le JP en fait, qui relaie en sa qualité de proxy vers le pledge)
2. *Parameter Update Exchange* : ne fait pas partie du join en lui-même, mais est initié par le JRC vers un joined node pour mettre à jour des paramètres qui lui avaient été transmis lors du join exchange (par exemple en cas de rekeying du réseau). Le joined node répond avec une réponse vide qui permet au JRC de s'assurer de la bonne réception de l'update. Aucun proxy applicatif ne doit intervenir ici car le noeud a déjà rejoint le réseau.

Dans ces échanges, l'information est transportée comme du payload applicatif CoAP, sous la forme d'objets CBOR. La spécification [?] en définit deux : les objets *Join\_Request* et *Configuration*, dont le contenu essentiel est brièvement énuméré dans les messages de la Figure 3.9. L'objet *Join\_Request* est destiné à recevoir les paramètres que le pledge veut communiquer au JRC pour accompagner sa demande d'intégration au réseau. L'objet *Configuration* est destiné à recevoir les paramètres de configuration actuellement en vigueur dans le réseau, donnés par le JRC qui doit les fournir à un pledge pour permettre son intégration. La sémantique des champs que contiennent ces objets est décrite ci-dessous (ceux qui sont facultatifs sont entourés de [...]).

*Join\_Request* (pledge → JRC) :

- [ role ] : rôle désiré par le pledge dans le réseau (6TiSCH node ou 6LBR)
- [ netwID ] : identifiant du réseau 6TiSCH que le pledge désire rejoindre (un JRC peut faire autorité sur plusieurs réseaux 6TiSCH en même temps)
- [ unsupported configuration ] : utilisé dans la gestion de conflits de configuration, non détaillée dans ce document

*Configuration* (JRC → pledge/joined node) :

- link-layer key set : tableau d'objets *Link\_Layer\_Key* où chacun décrit une clé utilisée au niveau lien et son usage (non détaillé ici)
- [ short identifier ] : objet *Short\_Identifier* qui reprend l'identifiant attribué par le JRC (unique) et le temps de validité de ce dernier
- [ JRC address ] : l'adresse IPv6 GUA du JRC, de sorte que le pledge, une fois joined node, puisse s'adresser directement à ce dernier (et servir de JP)
- [ blacklist ] : tableau de short identifiers dont le JRC ne souhaite pas l'acceptance des frames par le pledge (une fois qu'il deviendra joined node), permet par exemple d'isoler des noeuds corrompus ou non désirés par l'autorité que constitue le JRC
- [ join rate ] : débit maximum de trafic de join en *bytes/s* que le pledge, une fois joined node et endossant le rôle de JP, peut forward vers l'intérieur du réseau (drop des paquets excédants)

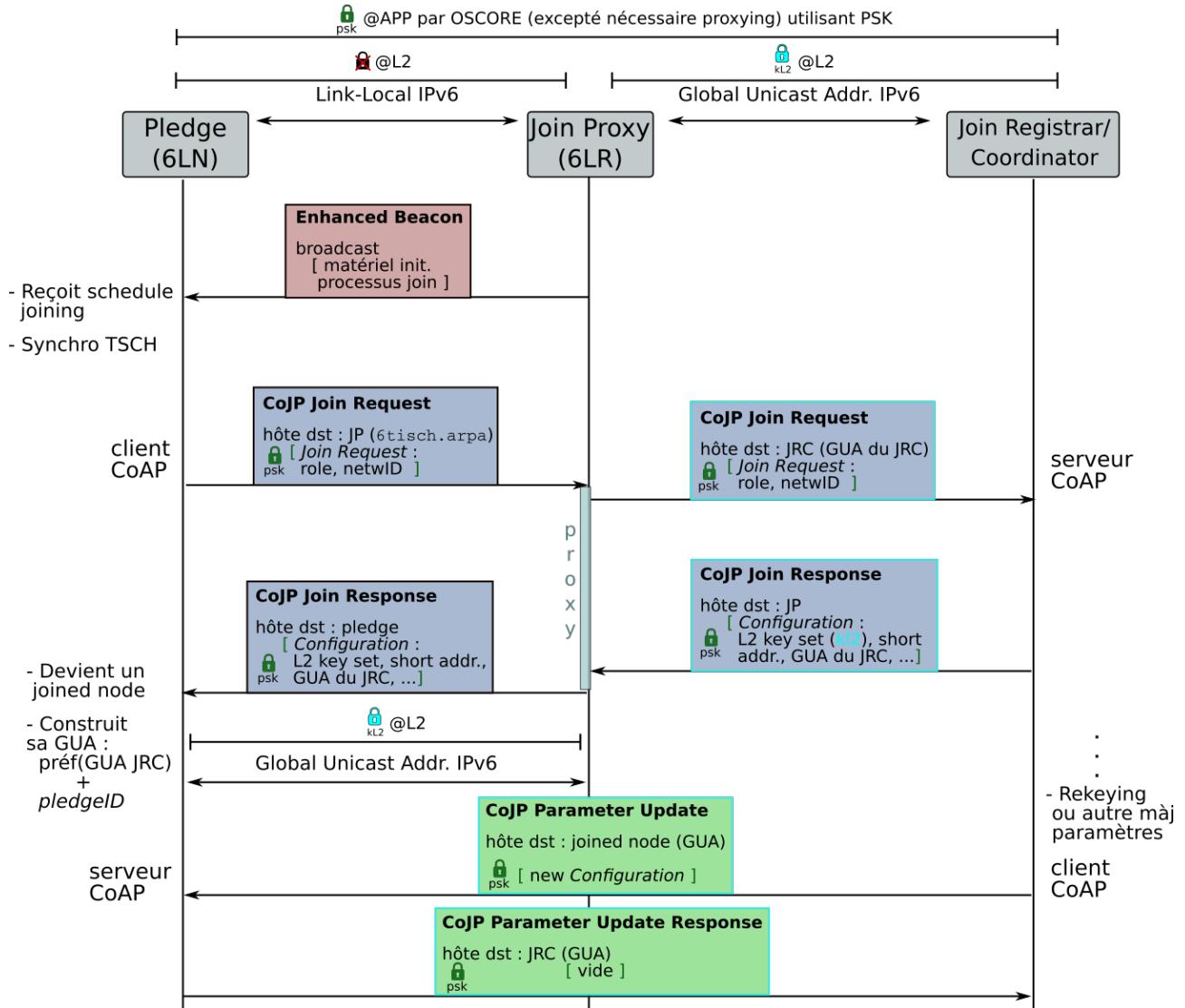


FIGURE 3.9 – Vue d'ensemble de la joining phase et de la mise à jour des paramètres distribués par CoJP

La sécurité du contenu des échanges CoJP (niveau applicatif donc) s'appuie intégralement sur OSCORE. C'est pourquoi CoJP fait l'hypothèse qu'une PSK est connue initialement des deux côtés de l'échange : le pledge et le JRC doivent pouvoir dériver des contextes de sécurité qui se correspondent. Les détails de quelles valeurs sont utilisées pour les différents paramètres durant la dérivation sont explicités plus loin. Le canal sécurisé établi par OSCORE permet d'assurer, par les propriétés de ce dernier, la confidentialité, l'authenticité des données et une protection contre les rejeux. Il s'agit bien d'une protection au niveau applicatif, tout en permettant le proxying par le JP détaillé dans la section suivante. Au niveau lien, le pledge n'ayant aucune connaissance des clés utilisées dans le réseau, il n'y a aucune protection appliquée pour les échanges entre le pledge et le JP. En revanche, le JP étant un joined node, il a connaissance des clés et toutes les frames échangées entre le JP et le JRC (multi-hop possible, voir Figure 3.8) sont sécurisées au niveau lien avec ces clés. La Figure 3.9 illustre schématiquement les échanges CoJP et à quel niveau s'appliquent ces chiffrements/authentifications. À noter qu'une fois un pledge devenu joined node, les échanges qu'il peut avoir par la suite avec le JRC utiliseront toujours le même contexte de sécurité OSCORE, mais ne feront plus intervenir de JP en plus d'être intégralement protégés au niveau lien.

### 3.1.3.4 Join Exchange CoJP et Join Proxy

Au moment d'initier le Join Exchange CoJP, un pledge dispose d'informations qu'il a reçues du noeud qui va servir de Join Proxy (JP). Au niveau lien (TSCH), le pledge est synchronisé temporellement et a connaissance des cells qu'il peut utiliser pour les échanges durant la joining phase (information issue de l'EB reçu). Au niveau réseau, le pledge utilise son adresse Link-Local après s'être assuré qu'elle est unique. L'obtention de ces informations est décrite dans la Section 3.2.1.1. Il s'apprête donc à envoyer sa Join Request (CoJP) vers le JP.

Le pledge et le JRC protègent le contenu applicatif échangé avec le matériel issu du contexte de sécurité, dérivé comme décrit par la Section 3.1.2.3 et à partir des paramètres pré-établis donnés à la Section 3.1.3.2. Il en résulte un contexte de sécurité OSCORE où sont connus :

- *Master Secret* = PSK
- *Master Salt* = "" (chaîne vide)
- *ID Context* = identifiant unique du pledge que le JRC doit considérer pour décider de si le pledge est autorisé dans le réseau (*pledgeID*, vraisemblablement son EUI-64), transmis dans le champs *kid context*
- Le nom identifiant un pledge de façon générique dans la communication OSCORE est la chaîne vide "", il sera utilisé comme Sender ID dans l'envoi de message par le pledge (et donc dans la dérivation du Sender Context pour le pledge).
- Le nom identifiant le JRC dans la communication OSCORE est la chaîne "JRC" et il sera utilisé comme Recipient ID **par le pledge**, et donc dans la dérivation du Recipient Context utilisé par la suite quand une réponse sera reçue.
- L'Algorithme AEAD est supposé être AES-CCM-16-64-128, sauf si un autre algorithme a été pré-établi (voir algorithmes supportés par COSE, [?])
- La fonction de dérivation de clés (HKDF) est supposée être HKDF SHA-256, sauf si une autre fonction a été pré-établie (voir algorithmes de dérivation définis pour COSE, [?] et [?]).

#### pledge → JP (Join Request)

La Join Request émise par le pledge contient un objet Join\_Request indiquant quel réseau il souhaite rejoindre (*netwID*) et le rôle qu'il souhaite y endosser : 6LBR ou noeud régulier. Le cas 6LBR correspond au cas spécial où le pledge veut en fait être la racine d'un nouveau réseau (et donc ne passe pas par un JP), il n'est pas traité ici. Le détail de la Join Request est donné par la Figure 3.10. Elle est protégée par OSCORE, le pledge ayant utilisé son Sender Context pour cela, et donc la Sender Key qu'il contient. L'Option OSCORE est construite avec le champ *kid* prenant la valeur du SID, donc une chaîne vide puisque c'est le pledge qui envoie. Cependant, afin que le JRC puisse récupérer le contexte associé au pledge (contenant la PSK qu'il partage avec), il a besoin de savoir de quel pledge il s'agit. C'est pourquoi le champ *kid context* de l'Option OSCORE prend la valeur de l'ID Context, qui est l'identifiant du pledge. Le pledge maintient en mémoire l'association entre la valeur du token créé et le contexte de sécurité.

Le JP accepte la frame contenant la Join Request bien qu'elle ne soit pas sécurisée, en mettant un flag 802.15.4 "secExempt" associé au pledge dans la table des voisins (Section 2.2.3.7). Il a alors accès aux informations que OSCORE ne chiffre pas et va effectuer son rôle de proxy en consommant les Options qui s'y rapportent. Le proxy va opérer son rôle en mode *stateless*. Un proxy stateless CoAP [?] compresse l'information dont il a besoin pour relayer la future réponse, dans le token CoAP du message même qu'il relaie. Effectivement, la future réponse référençant la même valeur de token, le JP pourra donc en retirer ce dont il a besoin pour acheminer vers le pledge : adresse IPv6 du pledge et port UDP source. Cette technique utilise des tokens de taille étendue (spécification [?]) et permet de sauvegarder de la mémoire (état maintenu dans les messages eux-mêmes et non par le JP) et d'éviter par ce biais des attaques de déni de service.

Outre le fait d'agir comme un stateless proxy en créant un nouveau token dépendant de l'état à stocker, le JP va effectuer son rôle de proxy normal. Le pledge ayant renseigné le nom d'hôte générique pour désigner le JRC "6tisch.arpa", le proxy va le résoudre en l'adresse IPv6 GUA du JRC qu'il connaît, lui. Il va également mettre le message CoAP à *NON* à la place de *CON* (expliqué à la Section ??).

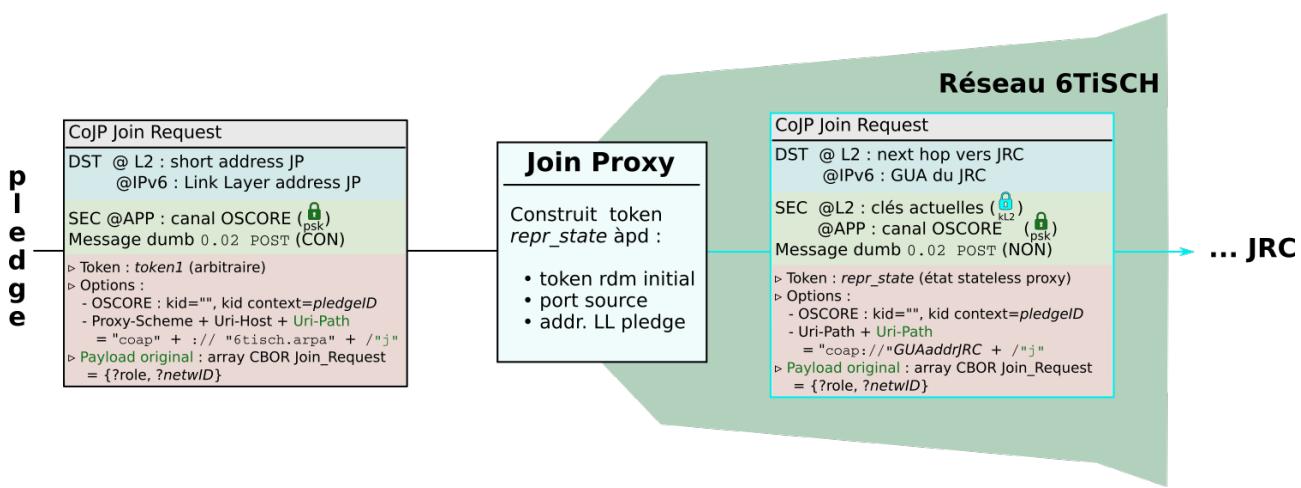


FIGURE 3.10 – Forwarding d'une Join Request par un Join Proxy

#### JP → JRC (Join Request)

La Join Request transite dans le réseau 6TiSCH, en plusieurs sauts vraisemblablement, vers le 6LBR qui lui-même la relaie vers le backbone où se trouve le JRC. Il est également possible que les fonctions de 6LBR et JRC soient co-situés et sont en fait le même hôte. Un fois la Join Request parvenue au JRC, ce dernier récupère le contexte de sécurité associé au pledge. Celui-ci contient la PSK partagée avec le pledge et le Master Salt entre autres (voir début de Section 3.1.3.4). Le Recipient Context est calculé considérant le RID prenant la valeur du nom désignant le JRC "JRC" (puisque c'est le JRC qui reçoit). La Recipient Key obtenue permet de déchiffrer et vérifier le payload, qui est l'objet CBOR Join\_Request.

Le JRC traite la demande d'intégration du pledge en tenant compte de quel réseau il veut rejoindre, et de si ce dernier y est autorisé (détails propres à au déploiement). Si c'est le cas, il va alors lui attribuer un identifiant *shortID* dans ce réseau, qu'il sait unique car il gère toutes les demandes d'intégration pour ce réseau. Il va également inclure dans sa réponse les clés au niveau lien qui y sont d'application à ce moment là, sa propre adresse IPv6 GUA, ainsi que d'autres paramètres (voir Section 3.1.3.3), le tout étant placé dans les champs adéquats de l'objet CBOR Configuration. Ce dernier sera le payload de la Join Response.

#### JRC → JP (Join Response)

Le JRC a protégé la Join Response avec le Sender Context, obtenu du contexte de sécurité dérivé à la réception de la Join Request. Il a été calculé en considérant un SID à la valeur de l'ID du JRC ("JRC") puisque c'est ce dernier qui envoie. Le paquet transite jusqu'au JP, toujours en étant protégé au niveau lien car on reste dans le réseau 6TiSCH.

Le JP reçoit la Join Response et reconstruit l'état à partir du token CoAP qu'elle contient (voir Figure 3.11). De cet état, il a les informations nécessaires pour acheminer le message vers le pledge.

#### JP → JRC (Join Response)

Au niveau applicatif, le seul changement concerne le type de message qui passe de *NON* à *CON*. Le JP relaie la Join Response dans une frame à destination du pledge. À noter que cette frame n'est alors pas sécurisée au niveau lien. De ce fait, si OSCORE n'était pas utilisé (sans être remplacé par une sécurité niveau transport), n'importe quel noeud pourrait voir en clair quelles sont les clés du réseau communiquées par le JRC.

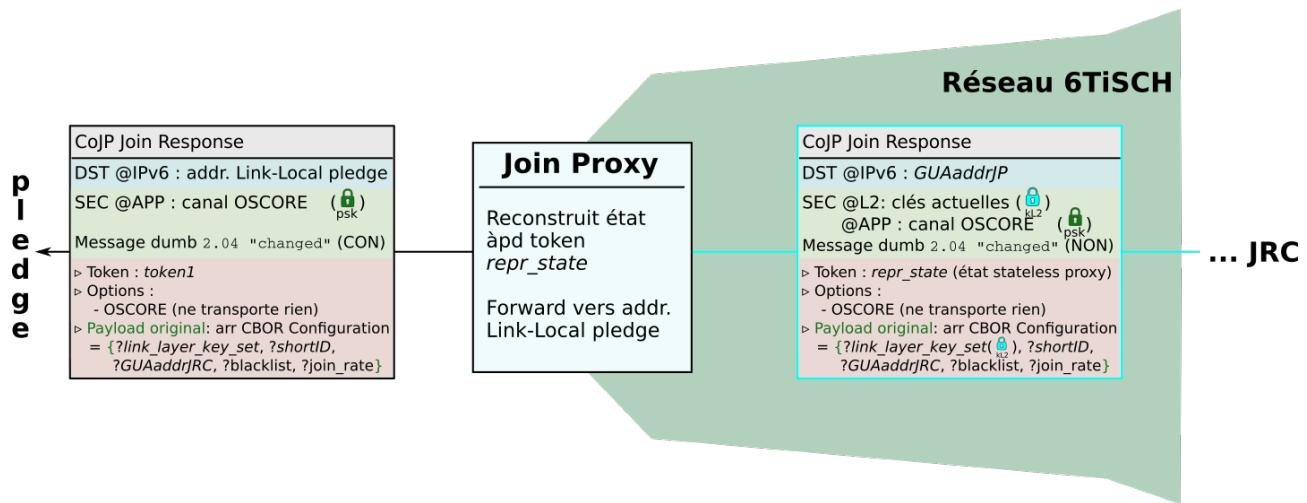


FIGURE 3.11 – Forwarding d'une Join Response par un Join Proxy

Se basant sur la valeur du token dans le message CoAP reçu (qui doit être identique à celui de la Request), le pledge récupère le contexte de sécurité qui avait été initialement dérivé. Il se sert du Recipient Context obtenu de ce dernier et de la Recipient Key référencée dedans pour vérifier/déchiffrer la Join Response. Une fois l'objet Configuration obtenu, le pledge installe ce qu'il contient pour intégrer le réseau. Alors en possession de la GUA du JRC et des clés courantes du réseau, il sera également désormais à même de servir de JP pour un autre nœud. À cette fin, il commencera à émettre des EBs annonçant le réseau qu'il a rejoint.

## 3.2 Analyse du déroulement de la phase de join

### 3.2.1 Procédure dans sa globalité

La phase de join comprend deux étapes distinctes. La première, l'étape de scanning/advertising, correspond à la recherche d'EB par le pledge, sa synchronisation temporelle au niveau lien (TSCH) et éventuellement un échange IPv6 Neighbor Discovery entre le pledge et le JP. La seconde est l'étape d'intégration au réseau proprement dite, où le Join Exchange CoJP est effectué. Si le pledge est accepté, il reçoit alors le matériel nécessaire au niveau lien pour intégrer le réseau 6TiSCH et y communiquer. Il devra alors lancer les procédures pour se placer dans la topologie RPL et pour obtenir une adresse globale IPv6.

On considère ici un déploiement tel que supposé dans la spécification [?]:

- Deux clés sont utilisées au niveau lien dans le réseau 6TiSCH :  $K1$  est utilisée pour authentifier les EBs,  $K2$  pour authentifier et chiffrer les frames normales et ACKs.
- Le minimal schedule utilisé pour les communications relatives au processus de join (dit *join schedule*) est composé de la seule cell  $(0, 0)$  en mode TxRxS (accueille tout le trafic broadcast de manière générale).
- Pledge et JRC disposent des paramètres pré-établis relatifs à CoJP sur OSCORE (voir Section 3.1.3.4)

#### 3.2.1.1 Étape de scanning/advertising (EBs) et obtention d'adresse IPv6

Le déroulement de cette étape décrit textuellement ci-dessous est synthétisé par la Figure 3.12.

Le pledge, une fois prêt à entamer son intégration au réseau, va chercher à capter des EBs émis par des joined nodes des potentiellement différents réseaux 6TiSCH à portée. Il n'y a pas de technique particulière énoncée dans les standards pour encadrer la façon dont le pledge doit procéder, que ce soit sur l'écoute de canal ou le nombre d'EBs dont il pourrait attendre la réception pour comparaison avant de lancer la suite du processus. La comparaison d'EBs peut se faire sur plusieurs critères. La Section 2.2.3.2 introduisait brièvement le contenu d'un EB pour TSCH, le draft [?] y apporte une structure supplémentaire contenant divers champs, dont certains sont utiles dans la discrimination entre EBs.

Le processus discriminatoire peut se baser sur les critères suivants ( $\triangleright$  pour ceux basés sur [?]):

- La Join Metric (de l'IE TSCH Synchronization), valeur dans  $[0x00, 0xFF]$  issue d'une normalisation du rang RPL du noeud émetteur (voir [?]). Une valeur plus faible indique une proximité avec la racine de l'arborescence et donc une préférence dans le fait de servir de parent.
- Le PAN ID du PAN auquel l'annonceur appartient ou l'adresse lien du noeud annonceur (source de l'EB).
- La force du signal avec lequel l'EB a été reçu.
- $\triangleright$  La Proxy Priority dont une valeur moins élevée dans l'intervalle  $[0x00, 0x7e]$  indique une bonne capacité à supporter du trafic non chiffré et une disponibilité des ressources (places dans la cache, etc.).
- $\triangleright$  La PAN Priority dont une valeur moins élevée indique la propension à recevoir de nouveaux noeuds, comparable d'un PAN à l'autre.
- $\triangleright$  Le Network ID tel que défini dans [?], plutôt relatif à un ensemble de PANs qu'à un seul en particulier : valeur arbitraire qui peut être calculée comme un hash du préfixe /64 du réseau.

Une fois un EB sélectionné, le pledge doit se synchroniser temporellement avec le noeud qui l'a émis et qui servira de JP. L'EB contient l'IE *TSCH Timeslot* qui donne l'ID du template de l'organisation des transmissions dans un timeslot (voir Section 2.2.3.4) duquel peut être déduit  $TsTxOffset$ , c'est-à-dire le temps écoulé depuis le début du timeslot à l'instant du début d'émission de la frame. Possédant le timestamp de la réception de l'EB, le pledge est alors capable de se calquer sur les timeslots utilisés par le réseau. L'IE *TSCH Synchronization* donne l'ASN, l'IE *Channel Hopping* identifie la séquence des fréquences utilisées et finalement l'IE *TSCH Slotframe and Link* donne l'initial schedule que le JP considère pour le trafic de join (le pledge doit donc le suivre). Cette slotframe est donc installée par le pledge qui sait désormais quand se réveiller pour communiquer.

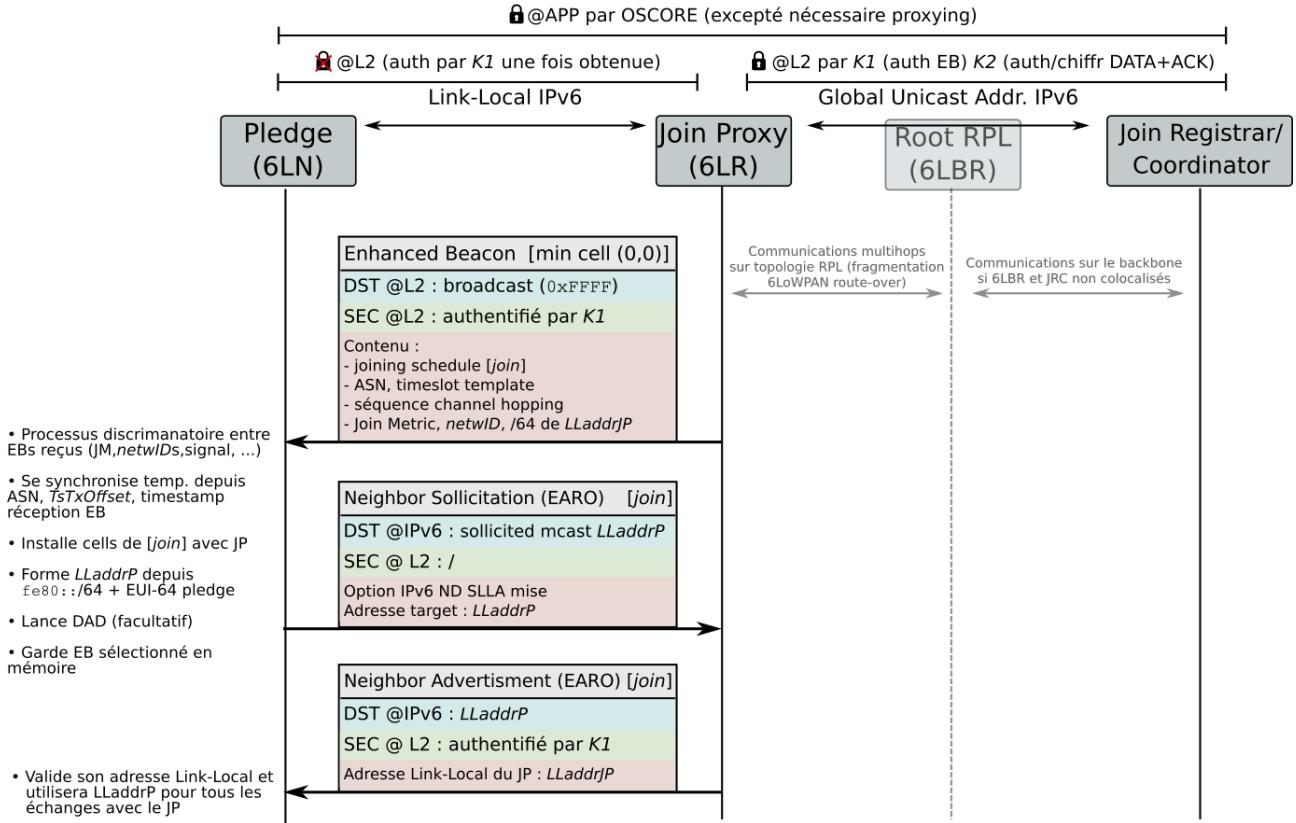


FIGURE 3.12 – Réception d’EB et validation de l’adresse IPv6 du pledge

Le pledge doit connaître l’adresse Link-Local du JP que ce dernier souhaite qu’il utilise pour le reste de la joining phase. À priori elle peut être dérivée par le pledge depuis la long address indiquée comme source de l’EB, mais elle peut aussi être spécifiée dans l’EB même (champ Join Proxy Interface ID de [?]).

Le pledge va former et utiliser son adresse Link-Local IPv6 pour le reste de la phase de join. Il le fait à partir de son *pledgeID*, qui est vraisemblablement son EUI-64 et du préfixe connu  $\text{fe80::}/64$ . Si c’est le cas, il peut se passer de l’échange suivant qui est le Neighbor Discovery IPv6 avec le JP, car l’EUI-64 est supposé unique. Sinon, il peut y avoir un échange de messages Neighbor Solicitation/Neighbor Advertisment afin de valider l’unicité de l’adresse Link-Local employée par le pledge pour le reste du processus de join.

Ces échanges ne sont couverts par aucune garantie en terme de sécurité, le pledge agit comme s’il pouvait avoir confiance en l’EB reçu du Join Proxy. Cependant, il est notable que dans la Figure 3.12, les messages émis par le JP sont quand même authentifiés par la clé K1. Ce sera utile au pledge pour vérifier plus tard, une fois le keying material du réseau reçu, que l’EB sur lequel il a basé sa synchronisation était conforme (développé dans la Section ??).

### 3.2.1.2 Étape d'intégration au réseau (échange CoJP) et opérabilité RPL

L'échange CoJP peut alors être initié par le pledge, schématisé par la Figure 3.13. La dérivation des contextes de sécurité OSCORE et le contenu des messages est tel que décrit par la Section 3.1.3.4. Les messages échangés entre le pledge et le JP le sont toujours en utilisant le join schedule, tandis ce que entre le JP et le JRC il s'agit de cells arbitraires qui peuvent être le résultat de négociations 6top.

Le JP accepte les frames en provenance du pledge et ce même si elles ne sont pas sécurisées. La spécification [?] précise que cela peut se faire en mettant un flag "secExempt" associé au pledge dans les tables utilisées par 802.15.4. En revanche, elle ne précise pas comment le JP détecte que le message issu du pledge est une Join Request. Les communications entre le JP et le JRC sont protégées au niveau lien par les clés  $K1$  et  $K2$ . Ce sont ces clés que le JRC écrit dans l'objet *link\_layer.key\_set* de l'objet Configuration renvoyé dans la Join Response.

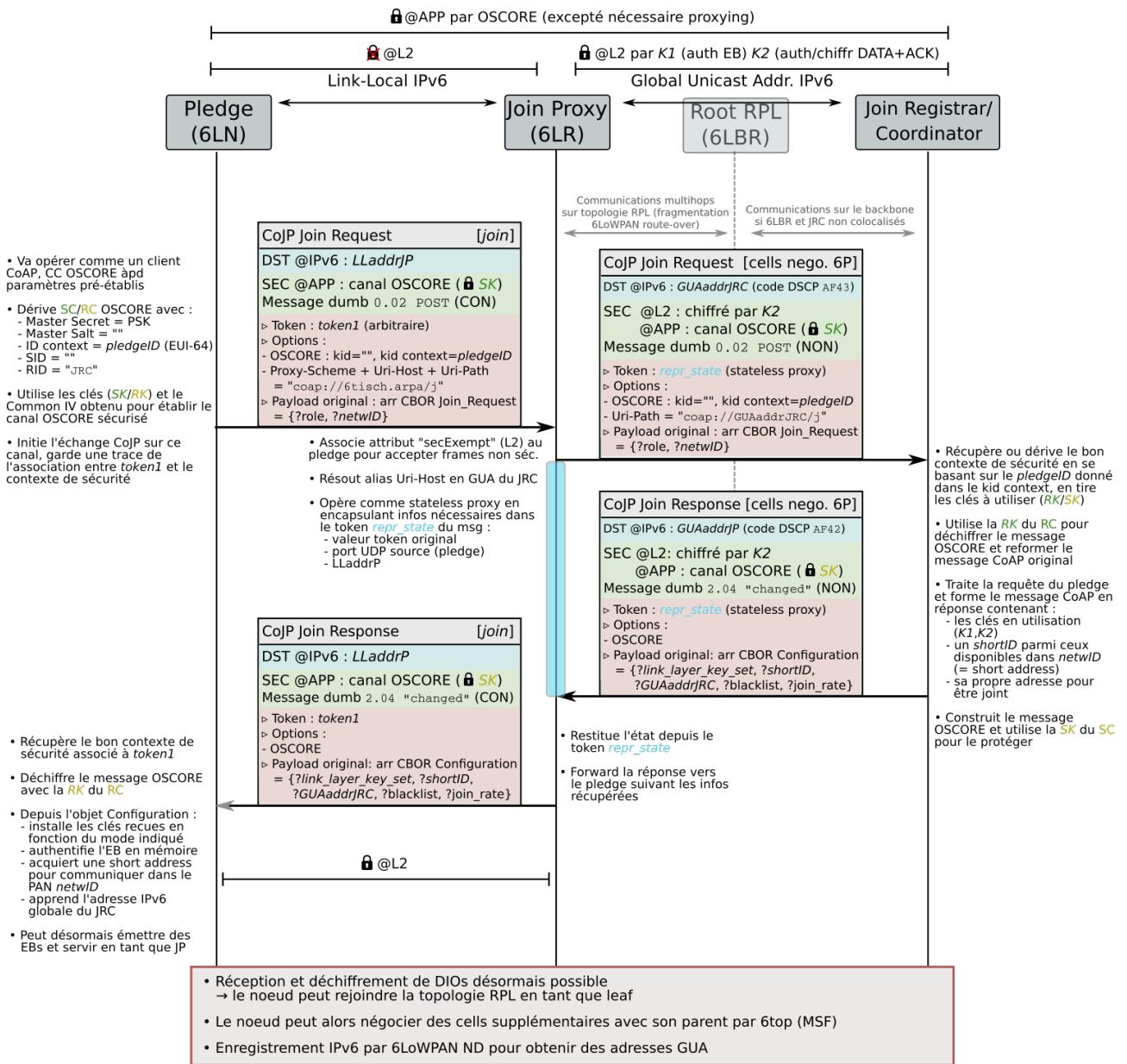


FIGURE 3.13 – Échange CoJP menant à l'intégration au réseau du pledge

Une fois le contenu de l'objet Configuration installé par le pledge, il est considéré comme intégré au réseau (joined node) dans le sens où il peut comprendre et vérifier les frames échangées, et en envoyer lui-même. Il peut également vérifier l'intégrité de l'EB qui lui a initialement permis de lancer le processus de join (voir Section ??). Une fois effectué, le désormais joined node devrait idéalement pouvoir servir de JP s'il en a les capacités. Pour cela, il doit encore se situer dans la topologie RPL et obtenir une adresse globale IPv6 à partir du préfixe attribué au réseau. De plus, comme le joined node peut désormais envisager du trafic unicast avec d'autres noeuds du réseau, il ne devrait plus utiliser le minimal schedule (sauf pour le trafic purement broadcast) mais installer des cells spécifiques à ses voisins. Les négociations de ces cells peuvent se faire par 6top et sont gérées par la Scheduling Function en activité pour le joined node. Considérant qu'il s'agit de la MSF, le noeud devra d'abord s'être positionné dans la topologie RPL car les cells installées en dépendent.

Le joined node est maintenant capable de déchiffrer les DIOs émis en broadcast, gouvernés par le Trickle Timer RPL aux autres noeuds de la topologie (voir Section 2.2.6). Il peut en avoir stocké s'il en a reçu pendant que le processus de join était en cours, mais également solliciter les noeuds à portée avec un DIS. Il en tire un ensemble de noeuds parents potentiels. Une fois qu'il a sélectionné un (sur base des critères de l'OF) il émet un DAO pour construire la route vers lui. À partir de ce moment, la MSF permet l'installation des autonomous cells basées sur l'EUI-64 du joined node et du parent sélectionné, ainsi que la négociation de cells additionnelles par le biais de 6top (voir Section 2.3.3).

Le préfixe IPv6 du réseau peut être obtenu de différentes façons. Si le JRC avait effectivement donné sa GUA dans l'objet Configuration, le joined node peut en déduire le préfixe. Sinon, un échange Router Solicitation/Router Advertisment peut également être initié par le joined node. Une fois sa GUA formée, le joined node doit l'enregistrer. Ce processus fait intervenir le mécanisme de 6LoWPAN ND ainsi que du proxying au niveau du 6LBR. Ce dernier maintient une table des adresses IPv6 utilisées dans le réseau 6TiSCH pour lequel il fait porte de sortie vers le backbone, et un compteur de fraîcheur pour chaque adresse. Ce processus est complexe et fait intervenir plusieurs spécifications, un aperçu des différents échanges est donné dans la section 4.2.2. Registration du draft [?].

## Chapitre 4

# Amélioration des performances de la joining phase avec la méthode NPEB

## 4.1 Les faiblesses de la joining phase

### 4.1.1 La *minimal cell*

Comme illustré par les figures détaillant la joining phase dans la Section 3.2, tous les échanges entre le pledge et le JP se font en utilisant le *joining schedule* décrit dans l'EB initialement reçu. La spécification [?] préconise l'utilisation d'une seule cell ((0,0)) comme joining schedule et par défaut pour tout trafic de type broadcast, dite *minimal cell*. Cette cell est également encore utilisée par le pledge une fois devenu joined node, principalement jusqu'à capter un DIO qui lui permettra de rejoindre la topologie RPL. Cependant, ce comportement est dépendant de la Scheduling Function active dans le réseau : par exemple, MSF installe des autonomous cells temporaires dédiées au trafic de join avant même que le pledge ne rejoigne la topologie (voir Section 2.3.3 et [?]). Qu'importe la Scheduling Function active, la spécification [?] stipule que ce n'est qu'une fois que son rang est obtenu dans la topologie qu'un noeud est autorisé à émettre à son tour des EBs. Ces EBs seront émis dans la minimal cell.

La minimal cell est donc utilisée à de nombreuses fins et par des noeuds qui ne sont même pas encore intégrés au réseau. Cela peut entraîner un phénomène important de congestion dans les réseaux denses [?], menant à un rallongement du temps de join et une perte d'énergie. Ni le standard IEEE802.15.4e [?], ni la spécification [?] ne proposent une stratégie formelle pour l'émission des EBs qui permettrait de pallier ce problème. La Section 2.3.2 introduisait les différentes solutions qui ont été proposées dans la littérature. L'approche du broadcast Bayésien étudiée par Vučinić *et al.* [?] est celle qui est communément retenue et implémentée dans les simulateurs [?].

De plus, l'usage de la minimal cell pose des questions de l'ordre de la sécurité : elle est de par sa nature très sensible aux attaques de type *selective jamming*. Il est prouvé dans [?] que n'importe quelle cell peut être ciblée par un attaquant en utilisant des propriétés inhérentes à TSCH. La minimal cell est de plus facilement distinguable des cells "régulière" car c'est la seule dans laquelle des EBs sont émis et ces derniers ne sont pas chiffrés. La technique proposée dans [?] pour lutter contre le selective jamming n'est pas applicable pour la minimal cell. Pour un attaquant qui souhaite neutraliser complètement le réseau, la minimal cell fait donc une cible de premier choix étant donné sa facilité d'accès et le tout le trafic important qui y transite.

### 4.1.2 Écoute active du pledge

Avant d'avoir entendu son premier EB, un pledge n'a aucune connaissance du réseau. Il doit donc écouter "à l'aveugle" jusqu'à, par chance, en capter un émis par un membre du réseau qu'il souhaite rejoindre. Étant donné que la minimal cell sera traduite à chaque cycle de slotframe en un channel différent (voir Section 1.3), il n'y a pas possibilité d'établir une stratégie spécifique. Le pledge écoute donc à chaque slot à une fréquence aléatoire ou continue à écouter sur la même.

La spécification [?] n'impose pas le comportement que le pledge doit adopter pour sélectionner le noeud auquel il va se synchroniser temporellement en utilisant un EB reçu. Cependant, il y est proposé la procédure suivante : après avoir reçu le premier EB, un pledge écoute encore activement MAX\_EB\_DELAY secondes ou jusqu'à recevoir un EB de NUM\_NEIGHBORS\_TO\_WAIT voisins distincts. Alors, il se synchronise avec le voisin dont l'EB reçu indiquait la meilleure valeur pour le champ *Join Metric* (présenté à la Section 3.2.1.1). La spécification [?] propose NUM\_NEIGHBORS\_TO\_WAIT à 180 secondes et NUM\_NEIGHBORS\_TO\_WAIT à 2.

Que ce soit durant la première phase (avant la réception d'un premier EB) ou durant la seconde (en attente d'un potentiel meilleur voisin), le pledge est actif et maintient sa radio allumée pour recevoir des EBs. Cela est extrêmement coûteux en terme d'énergie pour le pledge, comme illustré dans la Section 1.1 qui estimait la durée de vie d'un noeud présentant un tel duty cycle de 100% à environ 160 heures. Il s'agit donc d'une phase délicate que l'on souhaite la plus courte possible, tout en garantissant la synchronisation avec le meilleur voisin disponible.

## 4.2 Présentation de la méthode de *Neighbors Propositions EB*

Afin de donner une réponse aux problèmes inhérents à la joining phase telle que définie par les standards 6TiSCH qui ont été décrits par la Section 4.1, la méthode NPEB (*Neighbors Propositions EB*) a été élaborée et évaluée dans ce travail. Le reste de ce chapitre en décrit les principes fondamentaux, les inconvénients et les pistes d'amélioration. Dans le chapitre suivant, la Section ?? fait état des gains de performance de la joining phase NPEB en comparaison avec la joining phase standard, par le biais d'expérimentations en simulateur.

### 4.2.1 Intuition de la méthode NPEB

La méthode NPEB a été élaborée en considérant un objectif double :

- accélérer et optimiser en terme d'énergie (du point de vue du pledge) le processus de join
- permettre au pledge de sélectionner le meilleur (défini plus loin) voisin possible avec lequel initier le processus de join

Un voisin est considéré “meilleur” si d'une part sa place dans la topologie RPL est avantageuse (proche du noeud Root) et d'autre part l'intensité du signal reçu est suffisante. Le premier concept se repose sur le champ *Join Metric* annoncé dans un EB par le noeud qui l'émet, et qui selon la spécification [?] doit être calculé sur une métrique de routage du noeud (et normalisé dans l'intervalle [0, 255]). Par défaut, RPL est considéré et la spécification préconise de mettre la valeur de la *Join Metric* à  $DAGRank(rank) - 1$  (voir Section 2.2.6). Le second principe est directement lié à la disposition physique du réseau et la proximité des noeuds. Un noeud captant une frame (un EB en l'occurrence) capture le signal avec une certaine intensité, mesurée comme étant le RSSI (*Received Signal Strength Indication*, unité dBm). Un RSSI plus élevé indique une meilleure réception et mène donc vraisemblablement à un meilleur PDR si des frames sont échangées (une table de conversion établie empiriquement est présentée dans [?]). Ces deux mesures numériques facilement accessible pour un pledge lui servent de base comparative pour déterminer un meilleur voisin.

Le principe de la méthode NPEB se repose sur une augmentation des EBs standards avec de nouvelles informations relatives aux voisins du noeud qui émet l'EB. Ces EB augmentés sont désignés par NPEB pour *Neighbors Propositions Enhanced Beacon*. L'idée est qu'un pledge capturant un tel NPEB obtienne des informations sur d'autres noeuds qui sont également vraisemblablement dans son voisinage. Ces informations permettent au pledge de comparer ces potentiels voisins pour déterminer quel serait le meilleur candidat avec lequel lancer le processus de join. L'extensibilité des EBs standards est rendue possible en pratique grâce au mécanisme d'Header IE standardisé dans 802.15.4e (présenté à la Section 2.2.3.2).

En plus d'augmenter les EBs standards, la méthode NPEB considère que la minimal cell n'est plus la seule envisageable pour émettre des (NP)EBs. Chaque noeud membre du réseau peu utiliser en plus des cells arbitraires de la slotframe de base dans laquelle est installée la minimal cell (on considère ici la slotframe 0 décrite dans [?]). L'augmentation du nombre de cells d'annonce dans un même slotframe permet d'une part d'augmenter les chances qu'un pledge en écoute active capte un NPEB, et d'autre part d'émettre de façon déterministe là où les annonces dans la minimal cell sont probabilistes si le broadcast Bayesien est employé. Cependant, un pledge n'a a priori pas connaissance de quelle(s) cell(s) arbitraire est(sont) choisie(s) par ses voisins, ce qui l'obligerait à rester en écoute active. Pour palier cela, dans les informations associées à chaque voisin annoncé dans un NPEB, on retrouve les coordonnées des cells additionnelles que ce dernier utilise pour émettre ses NPEBs.

À la réception d'un premier NPEB, un pledge obtient donc une liste de voisins potentiels et pour chacun de ces voisins une cell à laquelle il garantit qu'il émettra lui-même un NPEB. Ces voisins sont dans le même réseau que le noeud qui a émis le NPEB, si le pledge est intéressé à le rejoindre il peut se baser sur le contenu EB standard du NPEB pour temporairement se synchroniser temporellement (au niveau TSCH). Il peut alors soit continuer le processus de join avec le noeud dont il vient de recevoir le NPEB ou continuer une écoute active, soit décider qu'un des voisins annoncés dans le NPEB est intéressant pour lui (potentiellement meilleur). Si c'est le cas, il sait quand ce voisin va émettre un NPEB (cell annoncée) et peut alors passer en sommeil jusqu'à ce moment au lieu de rester en écoute active. À son réveil, il captera potentiellement le NPEB si le voisin était à portée raisonnable, et pourra alors recommencer le même processus de façon itérative.

Bien que l'utilisation de cells d'annonce arbitraires en plus de la minimal cell est dans le principe avantageux pour le pledge qui obtient des informations sans faire d'écoute active, cela a un coût pour les noeuds du réseau. Premièrement, utiliser davantage de cells implique un coût énergétique pour la transmission du NPEB. Deuxièmement, comme un noeud "promet" qu'il va émettre un NPEB dans cette cell, il doit prioriser ce trafic pour éviter que le pledge n'attendent et se réveille pour rien, même s'il était à portée. Finalement, si les cells arbitraires ne sont pas choisies intelligemment, il y a un risque de collision. Ces faits motivent l'adoption de la notion supplémentaire de cycle. Un cycle correspond à une itération de slotframe (voir Section 1.3). Ainsi, à chaque cell arbitraire est associé un nombre de cycles *nbr\_cycles* dénotant que la cell n'est effective que tous les *nbr\_cycles*. Le cycle courant *curr\_cycle* est un compteur diminué à chaque nouvelle itération de slotframe qui, arrivé à la valeur 0, indique que la cell est effective (installée) pour l'itération courante. Après cette itération, la cell est retirée et *curr\_cycle* est réinitialisé, on a donc que  $0 \leq curr\_cycle \leq nbr\_cycles$ . L'information (*curr\_cycle*, *nbr\_cycles*) est donnée pour chaque cell dans le NPEB. Le pledge est alors capable de déterminer quand il devra se réveiller pour entendre le NPEB (possiblement plusieurs cycles plus tard) avec des calculs simples.

Tout du long du processus itératif, le pledge maintient les informations récoltées à propos des voisins dans une table appelée *NPtable*. Elle est indexée par un identifiant unique à chaque voisin qui peut être en pratique l'adresse MAC (ou la short address 802.15.4 si on ne considère qu'un seul réseau ou un mécanisme pour les différencier). À chaque NPEB reçu, les voisins qui y sont proposés voient leur entrée mise à jour dans la NPtable (ou y sont ajoutés si absents). Pour un voisin proposé, les informations reçues sont son identifiant unique, sa Join Metric et le *schedule NPEB* constitué des coordonnées d'une cell (timeslot et channel offset), du cycle courant *curr\_cycle* et du nombre de cycles *nbr\_cycles*. Le noeud qui émet le NPEB injecte également son propre schedule NPEB, les autres informations citées précédemment étant accessibles dans les champs de l'EB standard. En plus de ces champs, on retrouve dans la NPtable la dernière mesure de l'intensité du signal (*RSSI*) reçu de chaque voisin. Il renseigne également le statut d'écoute avec chaque voisin : une valeur numérique indique qu'un NPEB a été entendu de ce voisin, une valeur nulle le fait qu'une tentative d'écoute a échoué (voisin vraisemblablement trop loin) et pas de valeur le fait qu'aucune tentative d'écoute n'a encore été faite.

Noeud voisin	Join Metric	Cell émission NPEB	Cycle courant	# de cycles	RSSI
80-97-DF-48-00-01	0	(1, 0)	0	2	None
57-5F-CC-B1-00-02	14	(1, 2)	5	5	0
18-14-DA-48-00-03	7	(2, 11)	3	7	-83 (dBm)

TABLE 4.1 – Exemple de NPtable avec différents statuts d'écoute possibles (None/0/valeur RSSI)

La Table 4.1 est un exemple de NPtable reprenant les différentes situations d'écoute possibles. La première entrée indique que ce voisin est le Root de la topologie car sa Join Metric mise à 0 (voir [?]). Le cycle courant *curr\_cycle* vaut 0, ce qui signifie que la cell (1,0) a été installée dans la slotframe courante par ce noeud et qu'il compte y émettre un NPEB. Le RSSI mis à **None** indique qu'aucune écoute n'a encore été tentée à une cell annoncée de la sorte par ce noeud. En revanche, la valeur 0 pour le second noeud indique qu'une écoute a été tentée à la cell (1,2) au cycle précédent et a échoué. Finalement, la valeur du RSSI pour le troisième noeud indique qu'un NPEB de sa part a été capturé et est vraisemblablement à l'origine des deux autres entrées de la NPtable (voisins proposés dans ce NPEB).

La NPtable est utilisée par le pledge pour gouverner son processus de join et celui-ci la remplit au fur et à mesure. Le moment et le voisin avec lequel il décide de lancer la suite de la joining phase (Join Exchange CoJP) dépend d'une procédure décisionnelle détaillée par la Section 4.2.3. Une fois qu'il a rejoint le réseau, il ne supprime pas la NPtable car celle-ci contient déjà des informations sur les noeuds environnants. La NPtable est maintenue et utilisée pour sélectionner les voisins à annoncer une fois que le noeud sera autorisé à émettre lui-même des NPEB (à l'obtention de son rang RPL). Cependant, les voisins annoncés de la sorte seront ceux qui lui ont été proposés par NPEBlors de la joining phase. Pour remplir davantage la NPtable de façon dynamique, il est possible d'inclure des Header IEs dans les frames échangées, ce qui est détaillé à la Section 4.2.2.

La suite de cette Section illustre avec un exemple le déroulement du processus itératif et de l'évolution des différentes composants de la méthode NPEB présentées précédemment. Il met en scène un pledge qui tente de rejoindre le réseau en n'ayant aucune connaissance préalable de celui-ci. Les noeuds du réseau sont supposés avoir déjà alimenté leur NPtable par les mécanismes décrits dans la Section 4.2.2.

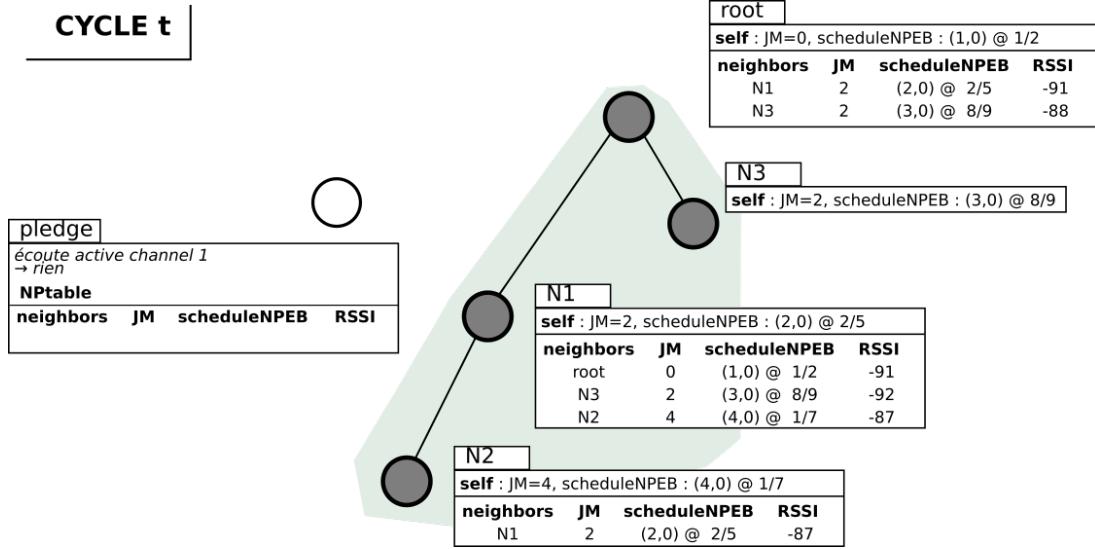


FIGURE 4.1 – [Cycle t] État initial du réseau où les NPtables des noeuds sont déjà alimentées

La Figure 4.1 illustre l'état initial du réseau considéré au cycle  $t$ , c'est-à-dire correspondant à la valeur d'ASN  $lenSF * t$  où  $lenSF$  est la taille de la slotframe. Les noeuds du réseau ont déjà des informations sur certains de leur voisin, maintenues dans leur NPtable. La notation pour le schedule NPEB correspond à  $cell @ curr\_cycle/nbr\_cycles$ . Durant ce cycle  $t$ , le pledge écoute activement sur le channel 1 (stratégie arbitraire), mais aucun NPEB n'est émis ( $curr\_cycle$  n'est à 0 pour aucun noeud).

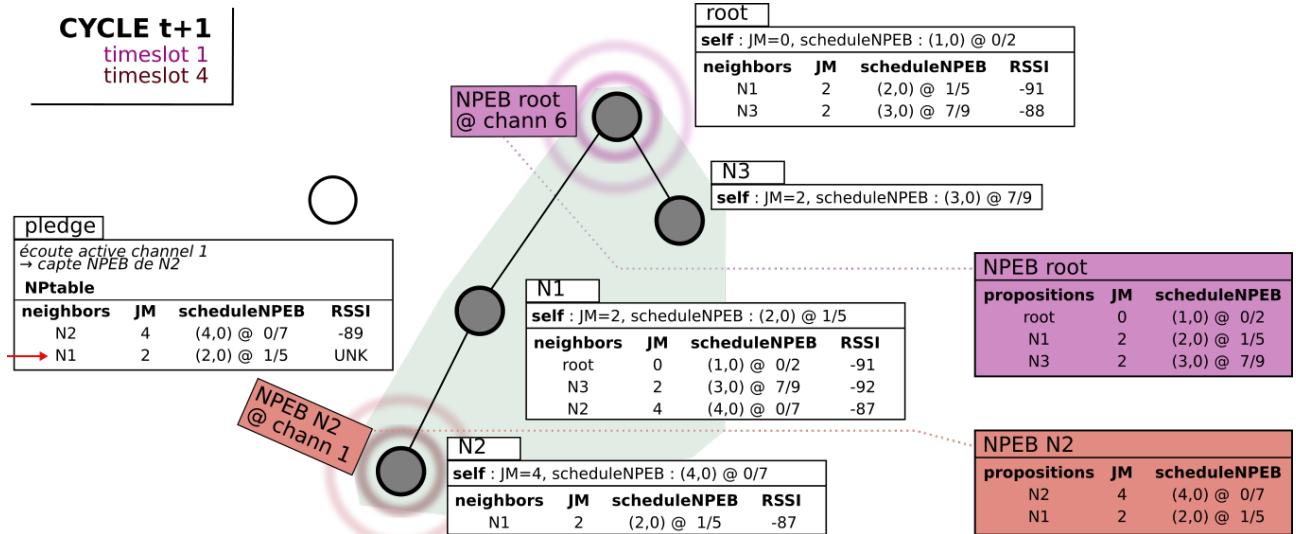


FIGURE 4.2 – [Cycle t+1] Une itération de slotframe écoulée, deux NPEBs programmés pour ce nouveau cycle

La Figure 4.2 représente ce qu'il se passe durant le cycle suivant le cycle  $t$ . Les compteurs *curr\_cycle* pour chaque voisin stocké dans les NPtables ont été décrémentés, ainsi que ceux propres à chaque noeud. Les compteurs de la cell (1,0) du noeud *root* et la cell (4,0) du noeud N2 sont tombés à 0, chacun a alors installé la cell correspondante dans sa slotframe. Au timeslot 1, le noeud *root* émet donc un NPEB à un channel offset de 0, qui correspond au channel 6 pour l'ASN courant (valeur pour l'exemple). Les voisins sélectionnés depuis sa NPtable pour être annoncés le sont par un processus décisionnel décrit dans la Section 4.2.3. Le pledge écoute toujours activement sur le channel 1 et n'entend donc pas cet NPEB. En revanche, le NPEB émis par N2 au timeslot 4 est entendu car le channel résultant est le channel 1 qu'il écoute. Il remplit sa propre NPtable en conséquence et utilise une procédure de décision décrite dans la Section 4.2.3 pour décider quelle action opérer. Ici, il s'avère que N1 semble un voisin intéressant et a priori meilleur que N2 en se basant sur la Join Metric. Donc, le noeud va passer en sommeil et se réveiller un cycle plus tard pour entendre le NPEB émis par N1 dans la cell (2,0).

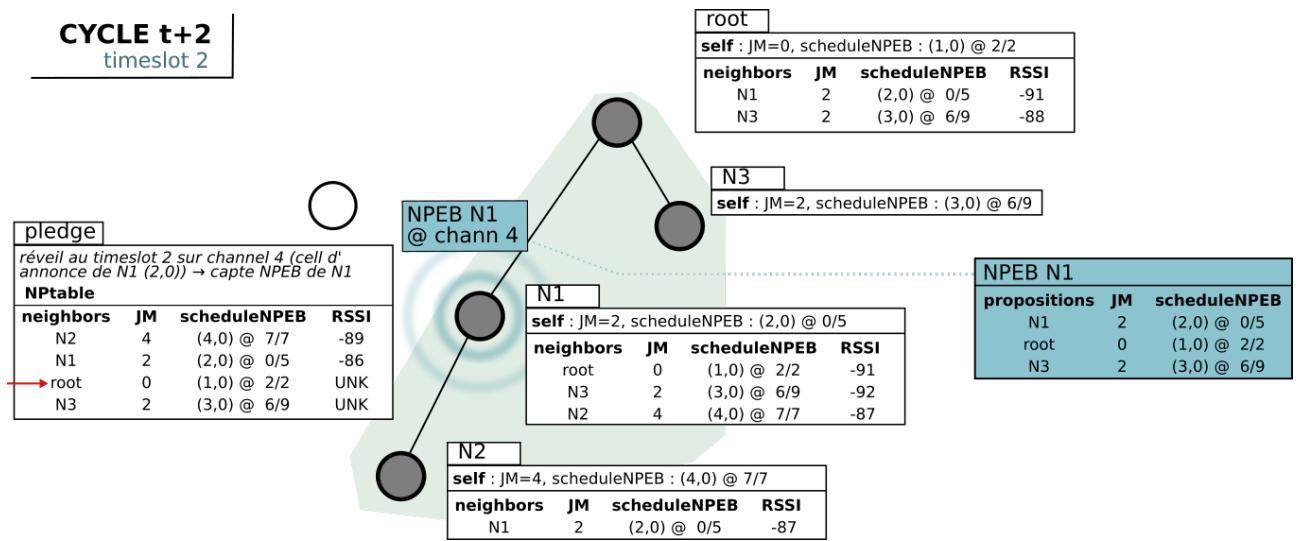


FIGURE 4.3 – [Cycle t+2] sommeil du pledge jusqu'à la cell d'annonce indiquée par N1

La Figure 4.3 illustre le pledge se réveillant au timeslot 2 du cycle suivant, pour écouter sur ce qu'il calcule comme étant le channel 4. Comme prévu, il entend un NPEB émis par N1 qui est à portée. Ici, la procédure de sélection des voisins à annoncer n'a pas retenu le noeud N2 sur base de sa Join Metric moins bonne que celle des candidats root et N3. Le pledge complète sa NPtable avec l'information sur ces nouvelles propositions de voisin et obtient une indication du RSSI avec le noeud N1 grâce à la frame qu'il vient de capturer. Obtenant les coordonnées d'une cell d'annonce de root, il décide de se rendormir les deux cycles suivants pour l'écouter. Effectivement, si root est à une portée raisonnable, il sera d'office le meilleur voisin (en terme de Join Metric) et lancer le processus de join avec ce dernier sera très avantageux (pas de noeud intermédiaire, accès le plus direct vers le JRC). De plus, une fois le processus de join terminé (devenu joined node) et root sélectionné comme preferred parent dans la topologie, cela permettra d'établir une route la plus courte possible vers le Root du DODAG. Ce sera également avantageux si le joined node vient à servir de noeud intermédiaire, par exemple en servant de Join Proxy quand il commence à émettre des (NP)EBs.

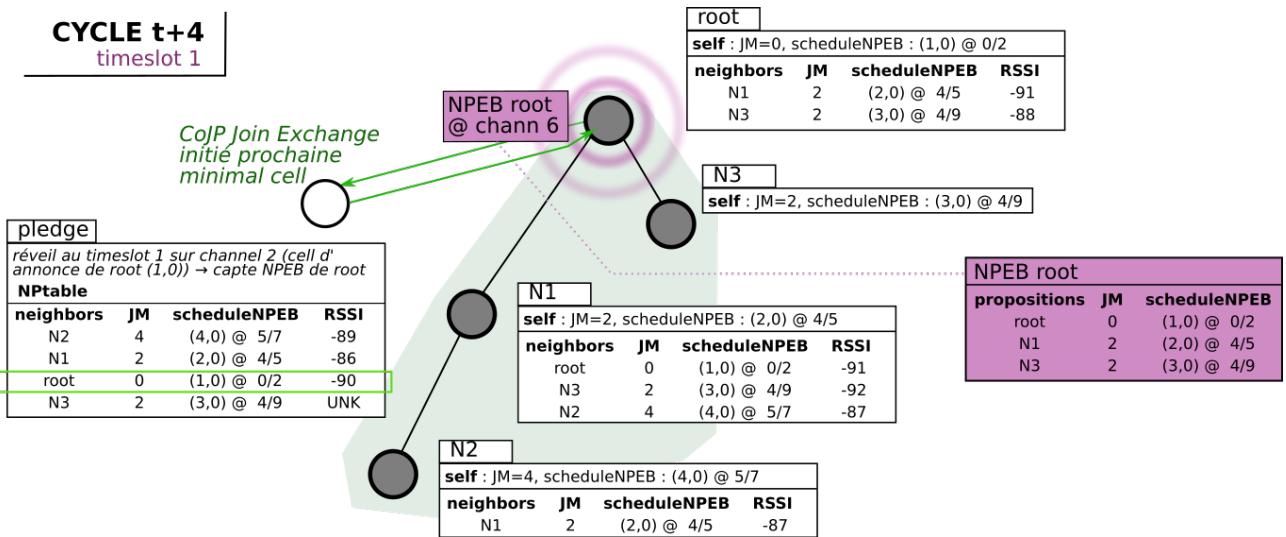


FIGURE 4.4 – [Cycle t+4] sommeil du pledge jusqu'à la cell d'annonce indiquée par root et lancement de la suite du processus de join avec celui-ci

La Figure 4.4 correspond à la dernière itération, quand le pledge se réveille deux cycles plus tard et capte le NPEB émis par root. Il a la confirmation que root est à portée et ne trouve pas d'autre voisin plus intéressant à écouter dans sa NPtable. Il va donc lancer la suite de la joining phase avec root. Il est synchronisé temporellement avec le réseau et suppose son adresse IPv6 link-local valide, il s'agit donc d'émettre sa Join Request CoJP comme décrit à la Section 3.2.1.2. Une fois qu'il aura obtenu le matériel de sécurité du réseau (clés 802.15.4 déployées) et un rang dans la topologie RPL, il sera à même d'émettre des NPEBs dans un cell qu'il sélectionne. Ces NPEBs pourront se base sur l'information déjà présente dans la NPtable, et les cells d'annonces également pour éviter les conflits avec des voisins connus.

Cet exemple illustre la façon dont la méthode NPEB tente d'atteindre le double objectif établi en début de cette Section. Le fait qu'une fois un premier NPEB entendu, un pledge ne soit plus obligé de continuer une écoute active pour obtenir des informations sur d'autres voisins constitue un gain en énergie pour lui. La rapidité du processus itératif dépend des procédures de décisions décrites dans la Section 4.2.3, mais supprime en partie la composante aléatoire de la recherche active d'EB. La méthode vise également à lancer la suite de la procédure de join avec le meilleur voisin possible, comme illustré ici où le pledge part d'une capture de NPEB du noeud N2 mais finit par s'adresser au Root du DODAG pour son intégration au réseau.

## 4.2.2 Méthodes de remplissage de la NPtable

### 4.2.2.1 NPEB-based

Cette méthode est la mécanique de base sur laquelle se repose le processus itératif d'écoute des NPEBs. Une sélection des voisins à proposer est effectuée (voir Section 4.2.3, idem pour les deux méthodes suivantes de cette section) à chaque envoi de NPEB. En pratique, l'information est encodée dans un Header IE 802.15.4 tel que brièvement décrit dans la Section 2.2.3.2 et standardisé dans [?]. Cet IE est rajouté à la fin du Header MAC et est transmis en clair, comme décrit dans la Section 2.2.3.5.

La spécification [?] stipule qu'un joined node devrait ignorer tout EB reçu mais peut toujours écouter à la recherche d'EBs émis par un autre réseau. Ici, comme les NPEBs émis par un noeud du même réseau peuvent transporter de l'information utile pour mettre à jour la NPtable, ils ne sont pas ignorés mais a priori cette réception est le fruit du hasard. Afin d'adapter dynamiquement la NPtable d'un joined node, les méthodes Data-based et ACK-based présentées dans la suite de cette Section sont utilisées.

#### 4.2.2.2 Data-based

Lors d'un échange de données quelconque, une frame est échangée entre deux voisins. De la même façon que pour les NPEB, des propositions de voisins peuvent être passées à travers un IE 802.15.4. Cependant, on peut envisager ici l'utilisation d'un Payload IE qui serait donc chiffré au niveau lien afin de divulger le moins possible les schedules NPEB des noeuds du réseau, même si cela restera de toute façon le cas avec les NPEBs. Il faut alors tenir compte de l'overhead du chiffrement, et de façon générale veiller à ne pas entraîner de fragmentation si le payload de la frame est conséquent.

Les voisins qui échangent des frames augmentées avec des propositions de voisins le font pour mettre à jour leur NPtable et la remplir avec de nouveaux noeuds. Le schéma qui est intuitivement le plus intéressant est du trafic downstream, venant d'un noeud plus haut dans la topologie. Les voisins qu'un tel noeud propose seront meilleurs en terme de Join Metric que pour du trafic upstream, et donc plus avantageux à annoncer dans les NPEBs destinés aux pledge.

#### 4.2.2.3 ACK-based

Le principe est identique aux voisins proposés avec la méthode Data-based, mais en ne considérant que les ACKs 802.15.4 qui supportent également les IEs. Cependant, utiliser des ACK pour communiquer des propositions de voisins est plus avantageux. Premièrement, comme expliqué pour la méthode Data-based, les voisins proposés devraient préférentiellement venir d'un noeud plus haut dans la topologie. Pour les ACKs, c'est donc le cas avec du trafic upstream (l'ACK est renvoyé par le parent). C'est ce type de trafic qui est majoritaire dans l'IoT [?]. Secondement, les ACKs ne transportent pas de payload des couches supérieures et sont donc extensibles avec plus d'information, tout en restant chiffrables [?].

### 4.2.3 Procédures décisionnelles et paramétrage

Cette Section décrit les différentes procédures pour les prises de décisions impliquées dans la méthode NPEB, à différents niveaux. Ces procédures sont adaptables en fonction du comportement escompté, les définitions qui en sont données ici sont basiques et destinées à servir de *baseline* effective dans un contexte général. Si des modifications sont apportées, elles devraient l'être en tenant compte du fait que ces procédures tournent sur des équipements restreints où des calculs complexes devraient être évités.

#### 4.2.3.1 Sélection des voisins à annoncer

Cette procédure intervient lorsqu'un noeud du réseau *node* souhaite proposer des voisins dans sa prochaine frame qui peut être un NPEB, du transport de payload des couches plus hautes ou un ACK (voir Section 4.2.2). *Node* a alors un certain nombre de voisins dans sa NPtable qu'il peut proposer, quelque soit le statut d'écoute : s'il est `None` ou mis à 0, rien n'a été reçu de ce voisin mais cela n'exclut pas qu'il envoie des NPEBs tel que annoncé selon le schedule NPEB associé. *Node* souhaite sélectionner parmi ces voisins quels sont les plus avantageux à annoncer, du point de vue d'un noeud qui recevrait ces propositions.

La Procédure 1 est une façon basique de déterminer les `NBR_MAX_NEIGH_ANNOUNCED` meilleurs voisins à annoncer, basée sur la Join Metric. Elle associe un score à chaque voisin, diminué par la Join Metric qui se situe dans  $[0, 255]$  où une faible valeur indique une bonne place dans la topologie RPL (seul le noeud Root du DODAG atteint la valeur 0). Afin de diriger un comportement convergent vers les noeuds plus hauts de la topologie (illustré par l'exemple de la Section 4.2), le preferred parent de *node* est favorisé et son score est augmenté. De même si le voisin considéré est Root du DODAG.

Cette procédure est très simple et a du sens, mais elle peut être affinée en tenant compte de, par exemple, les valeurs *curr\_cycle* de chaque voisin. L'intuition est de donner une préférence aux voisins qui vont bientôt émettre un NPEB (valeur de *curr\_cycle* faible, voire 0 si dans le cycle courant). Considérant *max\_cycle* la valeur la plus élevée parmi les *curr\_cycle* des voisins, une façon simple de procéder serait d'insérer après la ligne 4 une augmentation du score en dépendant ( $score \leftarrow score + maxcycle - NPtable[neighbor][curr\_cycle]$ ).

---

**Procédure 1** selectNeighborsToAnnounce : sélection des meilleurs voisins à annoncer par le noeud *node*

---

**Entrée:** *NPtable* de *node*, *NBR\_MAX\_NEIGH\_ANNOUNCED*

**Sortie:** Une liste de maximum *NBR\_MAX\_NEIGH\_ANNOUNCED* voisins connus de *node*

```

1: scoredNeighbors  $\leftarrow []$ 
2: pour neighbor  $\in NPtable$  faire
3:   score  $\leftarrow 255$ 
4:   score  $\leftarrow score - NPtable[neighbor][joinMetric]$ 
5:   si neighbor est Root de la topologie ou preferred parent de node alors
6:     score  $\leftarrow score * 2$ 
7:   fin si
8:   ajouter le couple (neighbor, score) à scoredNeighbors
9: fin pour
10: trier scoredNeighbors par ordre décroissant sur le score
11: retourne les 1ère composantes des NBR_MAX_NEIGH_ANNOUNCED premiers éléments de scoredNeighbors

```

---

La procédure pourrait également tenir compte de la valeur des RSSI des voisins pour lesquels elle est connue. Bien qu'elle n'ait pas de sens du point de vue du noeud qui reçoit les propositions, elle indique quels noeuds sont proches de *node*. Dans le cas d'une annonce NPEB, *node* peut proposer préférentiellement un voisin avec lequel il a un très bon RSSI, vraisemblablement proche. Un pledge qui capte le NPEB et considère l'écoute de ce voisin sera sûrement également capable d'entendre le NPEB dudit voisin, bien que cela ne soit pas garanti pour autant.

#### 4.2.3.2 Sélection d'un candidat voisin parmi ceux proposés au pledge

Comme décrit dans la Section 4.2, un pledge qui souhaite continuer le processus itératif pour entendre davantage de NPEBs doit sélectionner un voisin suivant à écouter. Ce voisin est choisi parmi les propositions apprises jusqu'ici, stockées dans la *NPtable* du pledge. La sélection n'est pas faite au hasard mais en se basant sur une procédure définie qui permet de dégager le voisin le plus intéressant pour le pledge selon plusieurs critères.

La Procédure 2 est une procédure basique pour sélectionner le meilleur voisin à écouter en se basant sur la Join Metric. Elle fonctionne sur le même principe que la Procédure 1. Cependant, dans ce cas-ci, les voisins qui ont déjà été entendus ou pour lesquels l'écoute du NPEB à la cell indiquée a été infructueuse sont écartés. Un seul voisin est retourné par la procédure.

Tout comme mentionné pour la Procédure 1, cette procédure pourrait être modifiée pour prendre en compte le cycle courant *curr\_cycle* associé à chaque cell de voisin proposé. Sélectionner un voisin qui programme d'émettre un NPEB dans les cycles à venir, ou même qui a installé une cell pour le faire dans l'itération de slotframe courante (*curr\_cycle* = 0), peut permettre d'accélérer le processus car le pledge dormira moins longtemps. En revanche, l'information du RSSI n'est ici pas exploitable puisqu'on ne s'intéresse qu'au voisins proposés encore non entendus (RSSI = None).

---

**Procédure 2** selectNeighborToListen : sélection du meilleur voisin à écouter parmi ceux proposés

---

**Entrée:** *NPtable* courante du pledge contenant des propositions de voisins (non vide)

**Sortie:** Un voisin considéré comme le meilleur à écouter ensuite pour capter son NPEB

```
1: scoredNeighbors ← []
2: pour neighbor ∈ NPtable faire
3:   si NPtable[neighbor][RSSI] ≠ None alors
4:     passer le reste de cette itération de boucle
5:   sinon
6:     score ← 255
7:     score ← score – NPtable[neighbor][joinMetric]
8:     si neighbor est Root de la topologie ou preferred parent de node alors
9:       score ← score * 2
10:    fin si
11:    ajouter le couple (neighbor, score) à scoredNeighbors
12:  fin si
13: fin pour
14: si scoredNeighbors est vide alors
15:   retourne None
16: fin si
17: trier scoredNeighbors par ordre décroissant sur le score
18: retourne la 1ère composante du premier élément de scoredNeighbors
```

---

#### 4.2.3.3 Processus itératif d'écoute de NPEBs

La Procédure ?? est la formalisation du processus itératif illustré dans la Section 4.2. Elle est appelée à la réception d'un NPEB par un pledge qui n'a pas encore décidé avec quel voisin il lancera son processus de join complet. Un pledge dans cette situation maintient une référence *bestNP* qui indique quel est selon ses connaissances le meilleur candidat courant. Également, il maintient un timer courant décrémenté au fur et à mesure des périodes actives. Il sert à trancher si le pledge n'a lancé la suite de son processus de join avec aucun voisin après un temps donné (assimilable à *MAX\_EB\_DELAY* de la spécification [?]). La situation de départ est que le pledge a sa *NPtable* vide, *bestNP* à None et le timer *remainingListenDelay* initialisé à 180 secondes. Il lance alors un écoute active sur un channel au hasard, jusqu'à ce qu'un NPEB soit capté et la Procédure ?? appelée (*remainingListenDelay* s'écoulera à partir de ce moment lors des prochaines phases d'écoute active).

Une fois le NPEB reçu, il est parsé premièrement comme un EB standard et deuxièmement la partie Neighbors Propositions est traitée. La fonction *feedTable(.)* met à jour et remplit la *NPtable* du pledge avec les informations sur les voisins proposés. De plus, le voisin *neighbor* qui a émis le NPEB est considéré de la même façon (le pledge a en plus une valeur pour le RSSI associé). La fonction *isBetterNeighbor(n1, n2)* est appelée sur deux voisins dont un NPEB a été reçu et retourne vrai si *n1* a une meilleure Join Metric et un meilleur RSSI que *n2*.

La fonction *shouldUndergoSynchroWith(node)* permet de trancher quand arrêter le processus itératif et est donnée par la Procédure ?? (détailée plus loin). Elle utilise tous les éléments connus du pledge pour décider si un candidat devrait être celui avec lequel lancer la suite du processus de join. Le candidat est ici *bestNP*, le meilleur voisin dont un NPEB a été capté par le pledge jusqu'ici, en prenant en compte le NPEB *NPEB* qui vient d'être reçu. La fonction *carryOnJoiningWith(.)* correspond au lancement de la suite du processus de join (Join Exchange CoJP) non détaillée ici car indépendante de l'utilisation de la méthode NPEB. À noter que les échanges subséquents auront alors lieu dans la minimal cell tel que décrit par la spécification [?], ou gérés par la Scheduling Function (MSF utilisée par exemple les autonomous cells, voir Section 2.3.3).

---

**Procédure 3** receiveNPEB : traitement de l'évènement réception d'un NPEB en écoute active

---

**Entrée:** *NPtable* du pledge, *NPEB*, *bestNP=None*, *remainingListenDelay=180*

**Sortie:** Le pledge a pris une décision à partir de sa connaissance du réseau

```
1: feedTable(NPtable, NPEB)
2: neighbor ← NPEB[self]
3: si bestNP est None ou isBetterNeighbor(neighor, bestNP) alors
4:     bestNP ← neighor
5: fin si
6: si shouldUndergoSynchroWith(NPtable, bestNP) alors
7:     carryOnJoiningWith(bestNP)
8: sinon
9:     mostInterestingNP ← selectNeighborToListen(NPtable)
10:    si mostInterestingNP est None alors
11:        [lancer écoute active pour remainingListenDelay secondes
12:         avec fonction de callback carryOnJoiningWith(bestNP)]
13:    sinon
14:        scheduleIntNeigh ← NPtable[mostInterestingNP][scheduleNPEB]
15:        targetASN ← computeNPEBAnnouncementASN(scheduleIntNeigh)
16:        [passer en sommeil jusqu'à réveil à targetASN pour écouter NPEB de mostInterestingNP
17:         suivi d'une écoute active de remainingListenDelay secondes si rien reçu
18:         avec fonction de callback carryOnJoiningWith(bestNP)]
19:    fin si
20: fin si
```

---

La fonction *selectNeighborToListen()* est la Procédure 2 détaillée précédemment. Si elle ne retourne aucun voisin (ligne 10), alors le pledge retourne en écoute active pour les *remainingListenDelay* secondes restantes et décrémentera de fait ce timer. S'il arrive à expiration, le processus de join sera lancé avec *bestNP*. Si un NPEB est reçu en écoute active durant cet intervalle de temps, la procédure *receiveNPEB* est à nouveau appelée (nouvelle itération).

Si en revanche un voisin intéressant est retourné par *selectNeighborToListen()* (ligne 13), le pledge récupère le *schedule NPEB* associé à ce voisin depuis la *NPtable*. Il sait alors dans combien de cycles (*curr-cycle*) il va installer une cell d'annonce de NPEB. La fonction *computeNPEBAnnouncementASN()* opère des calculs simples pour calculer l'ASN à laquelle cette annonce aura lieu. Il va ensuite passer en sommeil jusqu'à ce moment, où il se réveille pour recevoir le NPEB. S'il est capté, la procédure *receiveNPEB* est à nouveau appelée pour une nouvelle itération du processus d'écoute. Sinon, le pledge repasse en écoute active comme détaillé précédemment dans le cas *mostInterestingNP = None*.

La fonction *shouldUndergoSynchroWith()* est détaillée par la Procédure **??**. Elle permet de décider si un voisin dont un NPEB a été reçu est un voisin qui présente des caractéristiques suffisantes que pour être utilisé dans la suite du processus de join. Elle retourne donc un booléen qui, s'il est **Vrai**, mettra fin au processus itératif d'écoute de voisin. La décision s'opère sur plusieurs critères, sous forme de seuils paramétrables. Dans un premier lieu, le RSSI doit être suffisamment élevé (ligne 2) que pour avoir des échanges fiables. Pour se faire, une conversion en terme de PDR est faite, basée sur une table établie empiriquement présentée dans **[?]**. Ensuite, si le noeud est le Root de la topologie (ligne 6), on sait que le PDR est correct par la proposition précédente et donc qu'il faut aller vers ce noeud. Si ce n'est pas le cas, la Join Metric annoncée doit être assez bonne que pour garantir une bonne place dans la topologie (ligne 9). À noter que le seuil **MAX\_JM\_UNDERGO\_SYNCHRO** à 0 revient à imposer qu'il soit Root et donc validé par la déclaration précédente, le paramétrage doit se faire en fonction de l'Objective Function utilisée par RPL. Finalement, si le voisin a passé les précédentes conditions, et que tous les voisins proposés ont déjà été écoutés (ligne 13), alors il est choisi pour continuer le processus de join.

---

**Procédure 4** shouldUndergoSynchroWith : décision d'arrêt pour le processus itératif validant le voisin candidat pour la suite du processus de join

---

**Entrée:**  $NPtable$  courante du pledge,  $neighbor$  voisin candidat,  $MIN\_PDR\_UNDERGO\_SYNCHRO=0.6$ ,  $MAX\_JM\_UNDERGO\_SYNCHRO=0$

**Sortie:** Vrai si le processus itératif doit être stoppé

- 1:  $pdr \leftarrow \text{rssiTnPdr}(NPtable[neighbor][RSSI])$
- 2: **si**  $pdr < MIN\_PDR\_UNDERGO\_SYNCHRO$  **alors**
- 3:     **retourne** Faux
- 4:     **fin si**
- 5:      $neighJM \leftarrow NPtable[neighbor][joinMetric]$
- 6:     **si**  $neighJM == 0$  **alors**
- 7:         **retourne** Vrai
- 8:         **fin si**
- 9:     **si**  $neighJM > MAX\_JM\_UNDERGO\_SYNCHRO$  **alors**
- 10:         **retourne** Faux
- 11:         **fin si**
- 12:     **si** plus de 2 voisins dans  $NPtable$  **alors**
- 13:         **si** il ne reste plus de voisin à écouter dans  $NPtable$  (ont tous RSSI  $\neq$  None) **alors**
- 14:             **retourne** Vrai
- 15:         **fin si**
- 16:         **fin si**
- 17:     **retourne** Faux

---

#### 4.2.4 Discussion des optimisations possibles

Cette Section fait l'état de plusieurs optimisations apportables à la méthode NPEB telle que présentée dans les Sections précédentes. L'intuition qui les justifie est donnée pour chacune. Lors des expérimentations, les optimisations considérées dans l'implémentation seront mentionnées avec les résultats.

##### 4.2.4.1 Placement des cells d'annonce

Les noeuds du réseau peuvent a priori choisir n'importe quelle cell arbitrairement, excepté celles du timeslot 0 puisque la minimal cell y est installée. Cependant, il est intéressant de pouvoir les disposer de sorte qu'un pledge puisse enchaîner les itérations du processus d'écoute avec le moins de temps mort passé en sommeil entre deux NPEBs. Si la minimal cell est utilisée pour émettre des NPEBs, il est donc avantageux que les timeslots suivants soient considérés en priorité pour y installer des cells d'annonce de NPEB. Ainsi, le pledge recevant un NPEB en tout début d'itération de slotframe se voit proposer des voisins dont les cells d'annonce s'enchaînent directement les timeslots suivants.

Les noeuds du réseau peuvent tirer parti de leur  $NPtable$  pour sélectionner leur(s) cell(s) d'annonce intelligemment. Effectivement, le schedule NPEB associé à chaque voisin renseigne les coordonnées des cells qui sont donc déjà utilisées. Un noeud sélectionne donc le premier timeslot de la slotframe qu'aucun voisin connu n'exploite déjà. Il sélectionne le channel offset de façon aléatoire. La Figure ?? illustre l'enchaînement des cells d'annonce au début de la slotframe. Le fait que certaines cells soient installées par plusieurs noeuds n'est pas gênant comme ils ne sont à priori pas voisins l'un pour l'autre (sinon un timeslot différent aurait été choisi).

Cependant, un pledge qui rate le début de la slotframe devra attendre sa prochaine itération. Pour éviter cela et avoir une répartition minimum, un noeud qui installe une deuxième cell d'annonce le fait suivant le même processus mais en considérant un timeslot minimum qui est la moitié de la slotframe. S'il vient à installer une troisième cell, il considérera un timeslot minimum au tiers de la slotframe, etc.

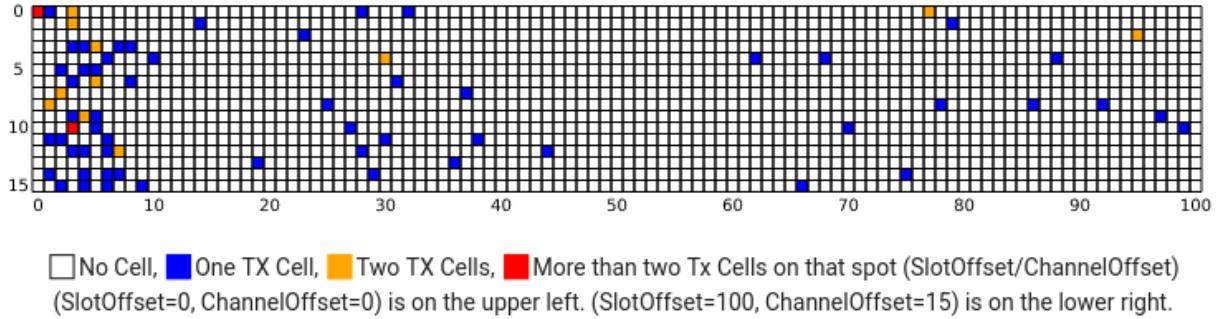


FIGURE 4.5 – Illustration de la sélection de cell tirée du simulateur 6TiSCH [?] modifié pour supporter la méthode NPEB

#### 4.2.4.2 Nombre de cycles

Une fois les coordonnées d'une cell d'annonce décidées, un noeud doit évaluer le nombre de cycles *nbr\_cycles* qui séparera chaque annonce. Intuitivement, un noeud qui a une mauvaise place dans la topologie devrait émettre moins de (NP)EBs que ses parents. Une façon de refléter cela dans le nombre de cycles est de le choisir proportionnellement à la valeur de la Join Metric du noeud. Plus la Join Metric est élevée (le noeud a une moins bonne place dans la topologie), plus *nbr\_cycles* le sera également.

Une stratégie faisant intervenir le nombre de voisins connus peut également être envisagée, à la manière du broadcast Bayesien [?]. Ici, il est envisageable d'augmenter *nbr\_cycles* pour espacer les émissions de NPEB au plus le noeud a de voisins connus.

#### 4.2.4.3 Cycles décalés entre voisins

Le fait que les annonces ne soient pas opérées à chaque itération de slotframe peut éventuellement mener à une situation de “famine”. L'exemple le plus flagrant est si l'on considère deux noeuds voisins dans la topologie qui ont la même Join Metric. Par la Section précédente, ils calculent pour une cell d'annonce un même *nbr\_cycle*. Si *curr\_cycle* est également identique, ils sont alignés et émettent toujours dans des itérations de slotframes simultanées. Un pledge dans le voisinage peut alors écouter activement des cycles durant sans entendre ni l'un ni l'autre.

Une façon simple et non coûteuse de contrer cela est de considérer le *curr\_cycle* connu des voisins quand une nouvelle cell d'annonce est créée. Si possible, le *curr\_cycle* initial sera mis à une valeur non utilisée par les voisins qui annoncent au même *nbr\_cycles*. Éventuellement, les multiples peuvent aussi être pris en compte pour être complet. Il en résulte un décalage de cycles entre les annonces et un étalement sur le temps évitant les temps d'écoute active trop longs pour les pledges.

#### 4.2.4.4 Cell RxS après une annonce NPEB

La méthode NPEB permet à un pledge de trouver efficacement un bon voisin, mais une fois celui-ci choisi il reste au pledge à effectuer la suite de son processus de join (Join Exchange CoJP). Cela se fait indépendamment de la méthode NPEB, en utilisant la minimal cell si la Scheduling Function en activité ne prévoit aucun mécanisme dédié. Ce n'est pas le cas de MSF qui installe des autonomous cells temporaires dédiées au trafic de join (voir Section 2.3.3), le raisonnement suivant n'est donc pas pertinent si MSF est utilisée. Les performances globales du join sont donc encore impactées par l'usage de celle-ci, tel que décrit dans la Section 4.1.1.

Pour éviter la congestion dans la minimal cell, il est envisageable que les noeuds du réseaux installent d'autres cells destinées à la réception de la Join Request CoJP émise par un noeud qui souhaite continuer son processus de join. Une manière d'intégrer cela à la méthode NPEB est de prévoir un nouveau champ *nbr\_rx\_cycles\_after* dans le schedule NPEB de chaque noeud. *nbr\_rx\_cycles\_after* indique le nombre de cycles suivants l'annonce d'un NPEB de la part du noeud où celui-ci installe une cell *RxS*, aux mêmes coordonnées (donc *nbr\_rx\_cycles\_after* < *nbr\_cycles*). Pour un pledge qui vient d'entendre un NPEB et de valider le voisin pour continuer son processus de join, cela lui permet d'envoyer sa Join Request vers le noeud au cycle suivant, évitant de le faire dans la minimal cell. Le noeud se mettra en écoute à ce moment et pourra traiter la Join Request CoJP comme si elle était reçue de façon standard dans la minimal cell. La suite du processus de join se déroule normalement, la Join Response CoJP sera relayée vers le pledge dans la minimal cell.

Le principal inconvénient de procéder de la sorte est la perte d'énergie liée à l'installation régulière d'une cell *RxS*, où le noeud peut écouter dans le vent si aucun pledge ne lance son processus de join avec lui. Idéalement, *nbr\_rx\_cycles\_after* est donc mis à 1. Ce qui signifie que le noeud installe la cell *RxS* quand *curr\_cycle* = *nbr\_cycles* mais pas pour les cycles suivants. Cette technique est intuitivement plus avantageuse durant la phase de bootstrapping du réseau où la minimal cell est fortement utilisée, tous les noeuds essayant de rejoindre presque simultanément.

## 4.3 Inconvénients de la méthode NPEB

### 4.3.1 Non conformité avec le standard MSF

La méthode NPEB ne respecte pas strictement la spécification de MSF [?]. Celle-ci fait l'hypothèse que la minimal cell est la seule utilisée pour l'émission d'EBs et de DIOs. Les autonomous cells que MSF installe sur base d'un hash de l'EUI-64 des noeuds communicants tiennent compte de cette contrainte et ne peuvent être installées au timeslot 0. En revanche, tous les autres timeslots sont valides, ce qui peut donc interférer avec les timeslots choisis pour les cells d'annonce de NPEB.

En pratique, cela n'est que modérément gênant car les cells d'annonce peuvent être déplacées, l'impact sera alors que la modification du schedule NPEB devra se répandre parmi les voisins qui proposent le noeud en question. Cela peut mener à une situation où un pledge planifie l'écoute d'une annonce NPEB qui n'arrivera jamais car la cell a été déplacée. Pour éviter cette situation, on peut tirer parti du fait que les coordonnées des autonomous cells que MSF installe sont fixées et connues à l'avance (dérivées de l'EUI-64 du noeud cible). Les timeslots réservés de la sorte à l'installation d'autonomous cell sont d'avance exclus de ceux considérés pour programmer les cells d'annonce. De plus, les negotiated cells ne sont installées par MSF qu'aux timeslots libres donc non pris par une cell d'annonce NPEB. Ces dernières sont négociées entre voisins, donc les annonces NPEB de chacune des parties sont prises en compte.

### 4.3.2 Overhead des messages et NPtable

Sont évalués ici l'overhead induit par les propositions de voisin dans des IEs 802.15.4 et le montant d'informations à stocker en mémoire pour maintenir la NPtable. On suppose ici que les noeuds sont identifiés de façon unique par leur adresse MAC.

#### 4.3.2.1 Proposition de voisin

Émettant un NPEB, un noeud renseigne ceux qui le reçoivent sur lui-même. Son adresse MAC et sa Join Metric sont des champs de l'EB standard. La réception de la frame en elle-même permet de dériver la valeur de RSSI associable à ce noeud par le receveur. Le seul champ additionnel à propos de lui-même ajouté par le NPEN est le schedule NPEB du noeud. Il contient une ou des cells d'annonces que le noeud souhaite propager dans le réseau. À chaque cell sont associées les valeurs de *curr\_cycle* et *nbr\_cycles*. Pour la négociation de cells, la spécification [?] encode une cell sur 4 bytes, ce que l'on considère également ici. On peut raisonnablement considérer un maximum de 15 cycles entre deux annonces de NPEB, et donc considérer 1 byte pour l'information relative aux cycles (4 bits pour *curr\_cycle* et *nbr\_cycles*). La valeur 0x00 est une valeur spéciale utilisée pour renseigner le fait que la cell n'est plus à considérer pour les annonces (le noeud n'émettra plus de NPEB dans celle-ci). Au total, un schedule NPEB renseignant une cell d'annonce est donc encodable sur 5 bytes.

Un NPEB contient une liste de propositions de voisins, chacun identifié par son adresse MAC, donc 6 bytes. À chaque voisin est associé la valeur de sa Join Metric (1 byte) et son schedule NPEB (idem que décrit précédemment). Pour faciliter le parsing, il est nécessaire de donner pour chaque schedule NPEB le nombre de cells qu'il renseigne (raisonnablement 4 bits suffisent). De même, le nombre de voisins proposés *n* doit être indiqué (raisonnablement 4 bits suffisent). Au total, proposer *n* voisins est encodable sur

$$4 + n(6 * 8 + 1 * 8 + 4 + \text{nbrcells}_i * 5 * 8) = 4 + n(60 + \text{nbrcells}_i * 40) \text{ bits}$$

À titre d'exemple, un NPEB proposant deux voisins qui renseignent chacun une cell introduit un overhead brut (sans compter les champs descriptifs de l'IE 802.15.4 *Type-Length*) de  $[5 * 8] + [4 + 2 * (60 + 40)] = 244$  bits  $\approx 30$  bytes. La taille d'un EB n'est pas fixée et peut varier, par exemple si l'extension d'EB [?] est utilisée. [?] indique que la taille minimum d'un EB est de 40 bytes. La taille maximum d'une frame 802.15.4 étant de 127 bytes (voir Section 2.2.2), cela laisse une marge raisonnablement suffisante pour annoncer plus de deux voisins.

#### 4.3.2.2 Stockage de la NPtable

Les noeuds du réseau ont un espace mémoire limité, aussi la NPtable doit être maintenue de sorte à ne pas stocker d'information futile. Une estimation de la place que les données brutes occupent peut être obtenue de la même façon qu'à la Section ?? . La structure utilisée pour les stocker est dépendante de l'implémentation et n'est pas détaillée ici.

La maintenance de la NPtable doit se faire de façon intelligente en considérant les informations qu'elle fournit. Un procédé de nettoyage de la NPtable doit être lancé régulièrement pour décider de quelles sont les propositions de voisins qui doivent être préservées. Intuitivement, les voisins dont l'écoute à la cell annoncée à échoué ( $RSSI = 0$ ) sont ceux à écarter en priorité. La Procédure 1 (selectNeighborsToAnnounce()) peut être réutilisée pour décider de quels sont les meilleurs voisins à maintenir dans la NPtable. Les détails de la procédure ne sont pas présentés ici.

## 4.4 Implémentation de la méthode NPEB

Comme détaillé dans le Chapitre suivant à la Section ?? , la méthode NPEB est implémentée comme une extension du simulateur 6TiSCH écrit par le WG 6TiSCH [?]. Le code existant est modifié afin de pouvoir supporter la méthode NPEB, et des structures telles que la NPtable sont ajoutées aux informations qu'un noeud maintient. Le simulateur modifié est accessible sur GitHub<sup>1</sup>. Comme l'intégration est faite dans le code même, les changements sont limités à la partie de l'implémentation représentant la couche 2 d'un noeud (802.15.4 en mode TSCH), qui est intégralement reprise dans le fichier `SimEngine/Mote/tsch.py`. D'autres changements mineurs sont effectués pour journaliser les nouveaux évènements et définir le nouveau type d'EB.

---

1. <https://github.com/RemDec/6tisch-simulator-NPEB>

## Chapitre 5

# Évaluation de la joining phase sécurisée et de la méthode NPEB en simulateur

## 5.1 Choix du simulateur

Une comparaison des simulateurs et émulateurs supportant la pile réseau 6TiSCH est établie dans [?]. Le simulateur 6TiSCH, présenté dans ce même article, a été sélectionné pour mener la phase expérimentative de ce travail. Il a été écrit par les membres du WG 6TiSCH et cible spécifiquement la pile étudiée, ce qui simplifie d'une part son utilisation et d'autre part son extensibilité. Il s'agit d'un simulateur dit *discrete event-driven*, c'est-à-dire qu'il n'émule pas la pile 6TiSCH au niveau binaire, mais plutôt du point de vue comportemental. Cependant, il parvient à simuler de façon fidèle le déploiement de la pile en situation réelle [?]. En addition au simulateur 6TiSCH, un framework de traitement de ses sorties développé au Service Réseaux et Télécommunications est utilisé. Il permet de manipuler les lancements du simulateur et les données journalisées, par le biais d'une surcouche écrite en Python.

## 5.2 Méthodologie

Les expérimentations sont divisées en deux parties. La première s'intéresse à la joining phase telle que décrite dans les spécifications 6TiSCH, que le simulateur implémente conformément. Cependant, comme le simulateur se restreint au niveau comportemental, l'échange CoJP est bien simulé mais les traitements cryptographiques liés à OSCORE (dérivation de contextes, chiffrement/déchiffrement) ne sont pas pris en compte. Ces expérimentations visent à évaluer quel est l'impact de l'échange CoJP sur la joining phase. Des métriques pertinentes sont choisies pour illustrer au mieux les différences en considérant ou non la joining phase. La seconde partie évalue la méthode NPEB présentée dans le Chapitre 4. D'autres métriques pertinentes pour observer les effets de la méthode sur la formation du réseau sont sélectionnées. Similairement, l'impact de la joining phase sécurisée est évalué quand la méthode NPEB est considérée. Chaque résultat obtenu est discuté et justifié.

Les variations étudiées ici se concentrent sur l'emploi ou non de la joining phase, croisé avec l'utilisation de la méthode NPEB ou non. Aussi, afin d'avoir une base commune d'interprétation pour toutes les expériences, on ne considère pas un réseau dont les caractéristiques varient. Les runs consécutives du simulateurs se font sur des seeds différents, mais il est possible d'imposer des contraintes sur la configuration du réseau. Ainsi, on considère des réseaux de 30 noeuds, similairement à ce qui est fait dans le cadre de tests réels sur des testbeds déployés [?]. La disposition de la topologie est aléatoire mais suit les contraintes suivantes : chaque noeud doit avoir au minimum 3 voisins avec lesquels il a un lien caractérisable par un PDR minimum de 0.5. À noter que le simulateur tente de respecter ces contraintes au mieux, mais ne le garantit pas systématiquement [?]. Un nombre raisonnable

## 5.3 Impact de la sécurité sur la formation du réseau

### 5.3.1 Sélection des métriques

### 5.3.2 Discussion des résultats

## 5.4 Amélioration des performances par la méthode NPEB

### 5.4.1 Sélection des métriques

### 5.4.2 Discussion des résultats

# **Conclusion et recommandations**

# Bibliographie

- [1] Roger Alexander, Anders Brandt, JP Vasseur, Jonathan Hui, Kris Pister, Pascal Thubert, P Levis, Rene Struik, Richard Kelsey, and Tim Winter. RPL : IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550, March 2012.
- [2] Carsten Bormann and Paul E. Hoffman. Concise Binary Object Representation (CBOR). RFC 7049, October 2013.
- [3] Carsten Bormann, Zach Shelby, Samita Chakrabarti, and Erik Nordmark. Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). RFC 6775, November 2012.
- [4] Tengfei Chang, Mališa Vučinić, Xavier Vilajosana, Simon Duquennoy, and Diego Dujovne. 6TiSCH Minimal Scheduling Function (MSF). Internet-Draft draft-ietf-6tisch-msf-10, Internet Engineering Task Force, December 2019. Work in Progress.
- [5] Tengfei Chang, Thomas Watteyne, Qin Wang, and Xavier Vilajosana. Llsf : Low latency scheduling function for 6tisch networks. 05 2016.
- [6] Glenn Daneels, Bart Spinnewyn, Steven Latré, and Jeroen Famaey. Resf : Recurrent low-latency scheduling in ieee 802.15.4e tsch networks. *Ad Hoc Networks*, 69, 11 2017.
- [7] Diego Dujovne, Luigi Alfredo Grieco, Maria Rita Palattella, and Nicola Accettura. 6TiSCH 6top Scheduling Function Zero (SF0). Internet-Draft draft-ietf-6tisch-6top-sf0-05, Internet Engineering Task Force, July 2017. Work in Progress.
- [8] Diego Dujovne and Michael Richardson. IEEE 802.15.4 Information Element encapsulation of 6TiSCH Join and Enrollment Information. Internet-Draft draft-ietf-6tisch-enrollment-enhanced-beacon-14, Internet Engineering Task Force, February 2020. Work in Progress.
- [9] Dionysis Efstatiou, Stelios Ioannou, Sotiris Nikoletseas, and Dimitra Patroumpa. Demo abstract : Emergency building evacuation guided by a wireless sensor network. pages 203–206, 10 2011.
- [10] Cenk Gündoğan, Christian Amsüss, Thomas Schmidt, and Matthias Wählisch. Iot content object security with oscore and ndn : A first experimental comparison. 01 2020.
- [11] Klaus Hartke. Extended Tokens and Stateless Clients in the Constrained Application Protocol (CoAP). Internet-Draft draft-ietf-core-stateless-05, Internet Engineering Task Force, March 2020. Work in Progress.
- [12] IEEE. Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)*, pages 1–709, April 2016.
- [13] A. Karalis. Atp : A fast joining technique for ieee802.15. 4-tsch networks. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pages 588–599, June 2018.
- [14] Dr. Hugo Krawczyk and Pasi Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869, May 2010.
- [15] David McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116, January 2008.
- [16] Esteban Municio, Glenn Daneels, Mališa Vučinić, Steven Latré, Jeroen Famaey, Yasuyuki Tanaka, Keoma Brun, Kazushi Muraoka, Xavier Vilajosana, and Thomas Watteyne. Simulating 6tisch networks. *Transactions on Emerging Telecommunications Technologies*, 09 2018.
- [17] M. R. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. A. Grieco, G. Boggia, and M. Dohler. Standardized protocol stack for the internet of (important) things. *IEEE Communications Surveys Tutorials*, 15(3) :1389–1406, Third 2013.

- [18] Michael Richardson. 6tisch Zero-Touch Secure Join protocol. Internet-Draft draft-ietf-6tisch-dtsecurity-zerotouch-join-04, Internet Engineering Task Force, July 2019. Work in Progress.
- [19] F. Righetti, C. Vallati, G. Anastasi, and S. Das. Performance evaluation the 6top protocol and analysis of its interplay with routing. In *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 1–6, May 2017.
- [20] Jim Schaad. CBOR Object Signing and Encryption (COSE). RFC 8152, July 2017.
- [21] Savio Sciancalepore, Giuseppe Piro, Gennaro Boggia, and Luigi Grieco. Application of ieee 802.15.4 security procedures in openwsn protocol stack. *IEEE Standards Education e-Magazine*, 4, 12 2014.
- [22] Savio Sciancalepore, Mališa Vučinić, Giuseppe Piro, Gennaro Boggia, and Thomas Watteyne. Link-layer security in tsch networks : Effect on slot duration. *Transactions on Emerging Telecommunications Technologies*, 07 2016.
- [23] Göran Selander, John Mattsson, Francesca Palombini, and Ludwig Seitz. Object Security for Constrained RESTful Environments (OSCORE). RFC 8613, July 2019.
- [24] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014.
- [25] Manoj Stanley, Yi Huang, Tian Hong Loh, Qian Xu, Hanyang Wang, and Hai Zhou. A high gain steerable millimeter-wave antenna array for 5g smartphone applications. 03 2017.
- [26] Pascal Thubert. An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4. Internet-Draft draft-ietf-6tisch-architecture-28, Internet Engineering Task Force, October 2019. Work in Progress.
- [27] Pascal Thubert, Carsten Bormann, Laurent Toutain, and Robert Cragie. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header. RFC 8138, April 2017.
- [28] Pascal Thubert, Erik Nordmark, Samita Chakrabarti, and Charles E. Perkins. Registration Extensions for IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Neighbor Discovery. RFC 8505, November 2018.
- [29] Pascal Thubert, Charles E. Perkins, and Eric Levy-Abegnoli. IPv6 Backbone Router. Internet-Draft draft-ietf-6lo-backbone-router-13, Internet Engineering Task Force, September 2019. Work in Progress.
- [30] Marco Tiloca, Simon Duquennoy, and Gianluca Dini. Robust Scheduling against Selective Jamming in 6TiSCH Networks. Internet-Draft draft-tiloca-6tisch-robust-scheduling-02, Internet Engineering Task Force, June 2019. Work in Progress.
- [31] Andrew Tinka, Thomas Watteyne, Kristofer Pister, and Alexandre Bayen. A decentralized scheduling algorithm for time synchronized channel hopping. *ICST Trans. Mobile Communications Applications*, 11 :e5, 09 2011.
- [32] Jose Vera-Pérez, David Todolí-Ferrandis, Salvador Santonja-Climent, Javier Silvestre-Blanes, and Víctor Sempere-Payá. A joining procedure and synchronization for tsch-rpl wireless sensor networks. *Sensors*, 18 :3556, 10 2018.
- [33] X. Vilajosana, T. Watteyne, M. Vučinić, T. Chang, and K. S. J. Pister. 6tisch : Industrial performance for ipv6 internet-of-things networks. *Proceedings of the IEEE*, 107(6) :1153–1165, June 2019.
- [34] Xavier Vilajosana, Kris Pister, and Thomas Watteyne. Minimal IPv6 over the TSCH Mode of IEEE 802.15.4e (6TiSCH) Configuration. RFC 8180, May 2017.
- [35] M. Vucinic, M. Pejanovic-Djurisic, and T. Watteyne. Soda : 6tisch open data action. In *2018 IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*, pages 42–46, April 2018.
- [36] Mališa Vučinić, Tengfei Chang, Bozidar Skrbic, Enis Kocan, M. Pejanovic-Djurisic, and Thomas Watteyne. Key performance indicators of the reference 6tisch implementation in internet-of-things scenarios. *IEEE Access*, PP :1–1, 04 2020.
- [37] Mališa Vučinić, Göran Selander, John Mattsson, and Dan Garcia-Carillo. Requirements for a Lightweight AKE for OSCORE. Internet-Draft draft-ietf-lake-reqs-03, Internet Engineering Task Force, April 2020. Work in Progress.
- [38] Mališa Vučinić, Jonathan Simon, Kris Pister, and Michael Richardson. Constrained Join Protocol (CoJP) for 6TiSCH. Internet-Draft draft-ietf-6tisch-minimal-security-15, Internet Engineering Task Force, December 2019. Work in Progress.

- [39] Mališa Vučinić, Thomas Watteyne, and Xavier Vilajosana. Broadcasting strategies in 6tisch networks. *Wiley Internet Technology Letters*, 1, 12 2017.
- [40] Qin Wang, Xavier Vilajosana, and Thomas Watteyne. 6TiSCH Operation Sublayer (6top) Protocol (6P). RFC 8480, November 2018.
- [41] Thomas Watteyne, Maria Rita Palattella, and Luigi Alfredo Grieco. Using IEEE 802.15.4e Time-Slotted Channel Hopping (TSCH) in the Internet of Things (IoT) : Problem Statement. RFC 7554, May 2015.
- [42] Thomas Watteyne, Xavier Vilajosana, Branko Kerkez, Fabien Chraim, Kevin Weekly, Qin Wang, Steven Glaser, and Kristofer Pister. Openwsn : A standards-based low-power wireless development environment. *Wiley Transactions on Emerging Telecommunications Technologies*, 23 :480–493, 08 2012.
- [43] Yang Wei, Yadong Wan, Jie He, and Yuanlong Cao. Security vulnerabilities and countermeasures for time synchronization in tsch networks. *Wireless Communications and Mobile Computing*, 2018 :1–14, 12 2018.
- [44] Wikipédia. 6lowpan — wikipédia, l'encyclopédie libre, 2020. Dernière lecture le 21/01/2020.
- [45] D. Zorbas, P. Kotzanikolaou, and C. Douligeris. R-tsch : Proactive jamming attack protection for ieee 802.15.4-tsch networks. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, pages 00766–00771, June 2018.