Résumé

L'IETF a standardisé la transmission de paquets IPv6 au-dessus des réseaux personnels sans fil à faible puissance (LoWPAN) dans la RFC 6282 sous le nom de 6LoWPAN. Le standard 6LoWPAN vise à créer une couche d'adaptation permettant aux paquets IPv6 d'être pris en charge efficacement par des trames de petite taille comme celles définies dans le standard IEEE 802.15.4 particulièrement utilisé dans les réseaux de faible puissance et à pertes (Low Power and Lossy Networks ou LLN).

Dans ce projet, nous nous intéressons à la question du routage dans un contexte 6LoWPAN et plus particulièrement au mode « route over » avec le protocole RPL. Les objectifs du projet dans un premier temps sont de comprendre le contexte et les concepts clés liés au routage dans le 6LoWPAN ainsi que l'étude des travaux existants sur le protocole de routage RPL et son optimisation. Ensuite, nous allons proposer une approche pour améliorer le protocole RPL à travers l'un de ses composants majeurs qui est celui du Trickle Timer. Enfin, la dernière partie est une étude expérimentale où nous allons évaluer notre approche en utilisant le simulateur Cooja.

Mots-clés : 6LoWPAN, Réseaux LLN, IPv6, IEEE 802.15.4, RPL, Trickle Timer, Cooja.

Table des matières

LISTE DES FIGURES

LISTE DES FIGURES

LISTE DES ABREVIATIONS

6LoWPAN	ipv**6** **L**ow p**o**wer **W**ireless **P**ersonal **A**rea **N**etworks
AODV	**A**d hoc **O**n demand **D**istant **V**ector
AVG DEL	**AV**era**G**ed **DEL**ay
AWK	**A**lfred aho, peter **W**einberger, brian **K**ernighan
BATMAN	**B**etter **A**pproach **T**o **M**obile **A**d hoc **N**etworking
C	redundancy **C**ounter
CSMA/CA	**C**arrier **S**ense **M**ultiple **A**ccess with **C**ollision **A**voidance
DAG	**D**irected **A**cyclic **G**raph
DAO	**D**estination **A**dvertisement **O**bject
DAO-ACK	**D**estination **A**dvertisement **O**bject **ACK**nowledgement
DIO	**D**ODAG **I**nformation **O**bject
DIS	**D**ODAG **I**nformation **S**olicitation
DSR	**D**ynamic **S**ource **R**outing
DODAG	**D**estination-**O**riented **D**irected **A**cyclic **G**raph
EDC	**E**xpected **D**uty **C**ycled wakeups
ETX	**E**xpected **T**ransmission count
OLSR	**O**ptimized **L**ink **S**tate **R**outing
FFD	**F**ull **F**unction **D**evice
GTS	**G**uarantee **T**ime **S**lot
GUI	**G**raphical **U**ser **I**nterface
HC	**H**op **C**ount

LISTE DES ABREVIATIONS

IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IETF	**I**nternet **E**ngineering **T**ask **F**orce
Imax	**I**nterval **M**aximum
Imin	**I**nterval **M**inimum
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IoT	**I**nternet **O**f **T**hings
IP	**I**nternet **P**rotocol
IPv6	**I**nternet **P**rotocol **V**ersion **6**
K	redundancy constant
MAC	**M**edia **A**ccess **C**ontrol
MANET	**M**obile **A**d-hoc **NET**works
MRHOF	**M**inimum **R**ank with **H**ysteresis **O**bjective **F**unction
OCP	**O**bjective **C**ode **P**oint
OF	**O**bjective **F**unction
OF-FL	**O**bjective **F**unction **-F**uzzy **L**ogic
OF0	**O**bjective **F**unction **Z**ero
OLSR	**O**ptimized **L**ink **S**tate **R**outing
ORPL	**O**pportunistic **RPL**
PFI	**P**acket **F**orwarding **I**ndication
RFC	**R**equest **F**or **C**omments
RFD	**R**educed **F**unction **D**evice
RPL	**R**outing **P**rotocol for **L**ow-power and lossy networks
LAN	**L**ocal **A**rea **N**etwork

LISTE DES ABREVIATIONS

LoWPAN	**L**ow p**o**wer **W**ireless **P**ersonal **A**rea **N**etworks
LLN	**L**ow power and **L**ossy **N**etworks
ROLL	**R**outing **O**ver **L**ow-power and **L**ossy Links
TCP	**T**ransmission **C**ontrol **P**rotocol
UDGM	**U**nit **D**isk **G**raph Medium
UDP	**U**ser **D**atagram **P**rotocol
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WPAN	**W**ireless **P**ersonal **A**rea **N**etwor
WSN	**W**ireless **S**ensor **N**etworks

INTRODUCTION GENERALE

L'internet des objets c'est connecté divers objets qui sont généralement limités en termes de puissance de traitement, de batterie et de mémoire à travers des réseaux de faible puissance et à perte (LLN). Les réseaux formés par ces objets sont caractérisés par des liaisons instables avec des taux de perte élevés et des débits faibles [1], pour répondre à ces contraintes l'IETF[1] à proposer de nouveaux protocoles dont RPL un protocole de routage IPv6 pour les réseaux de faible puissance et à perte.

Dans la plupart des cas, les réseaux LLN seront utilisés sur des couches de liaison avec des tailles de trame restreintes comme celle définie dans le standard IEEE 802.15.4. L'IETF a standardisé une couche adaptative sous le nom de 6LoWPAN[2] qui désigne réseaux IPv6 personnels sans fil à faible puissance. L'intérêt d'un tel protocole est de permettre aux paquets IPv6 d'être pris en charge efficacement par des trames de petite taille [2].

L'objectif souhaité dans ce projet est de bien comprendre le contexte et les concepts clés liés au routage dans le standard 6LoWPAN particulièrement en mode « route over » avec le protocole de routage RPL ainsi que l'étude des travaux proposant des améliorations de RPL, ensuite nous allons proposer notre solution pour améliorer l'un des très important composent de protocole RPL qui est celui du Trickle Timer et enfin nous allons évaluer notre approche proposée en utilisant le simulateur Cooja.

Notre projet est structuré en cinq chapitres.

Dans le premier chapitre intitulé « 6LoWPAN et les réseaux LLN », nous allons présenter les réseaux LLN avec leurs limites suivies de leur relation avec la couche d'adaptation 6LoWPAN en décrivant les deux solutions d'adaptation proposées, la fragmentation et la compression d'en-tête ainsi que les deux modes de routage dans le contexte 6LoWPAN.

Le chapitre 2, intitulé « Routage dans les réseaux LLN », définit le routage et ses contraintes dans les réseaux LLN suivie de la présentation de différents protocoles de routage développés par les groupes de travail de l'IETF.

Dans le chapitre 3, intitulé « Le protocole de routage RPL », nous abordons le fonctionnement du protocole RPL ainsi qu'une description des travaux antérieurs proposés pour améliorer certains aspects de son fonctionnement.

.

IETF[1] est un groupe informel, international, ouvert à tout individu, qui participe à l'élaboration des standards Internet

6LoWPAN[2] est un groupe de travail crée par l'IETF afin de définir une couche d'adaptation sous le nom de 6LoWPAN.

Dans le chapitre 4, intitulé « Méthodologie d'optimisation proposée » nous allons décrire la solution que nous proposons qui se focalise sur un algorithme très important dans le protocole RPL qui est celui du « trickle timer » ensuite, nous allons identifier par expérimentation comment modifier cet algorithme afin d'assurer le bon fonctionnement de RPL selon différents scénarios.

Enfin, dans le chapitre 5 intitulé « Etude expérimentale », nous allons aborder la partie mise en œuvre de notre solution en utilisant le simulateur Cooja comme environnement de développement pour implémenter et observer les causes et les effets de l'approche proposée sur les performances de RPL.

CHAPITRE I

6LoWPAN et les réseaux LLN

I.1 Introduction

L'Internet des objets est une technologie qui permet de connecter n'importe quel ensemble d'objets entre eux [3]. Ces objets sont souvent très contraints en énergie et forment les réseaux de faible puissance et à pertes (LLN). Ces réseaux se caractérisent par une faible consommation d'énergie afin d'assurer à ces objets une longue durée de vie, les réseaux LLN possèdent les mêmes caractéristiques que les réseaux de capteurs sans fil (ou WSN).

Dans ce chapitre, nous allons introduire le contexte de ce travail à savoir l'internet des objets mais surtout les réseaux de faible puissance formés des objets contraints. Nous allons également nous intéressons à un nouveau protocole le 6LoWPAN qui représente une couche d'adaptation qui s'occupe de la normalisation des données circulant entre la couche réseau et la couche liaison des données (MAC) [2] définis par le support de communication utilisé souvent dans les réseaux LLN tel que la norme IEEE 802.15.4.

I.2 L'internet des objets

L'internet des objets (ou IoT) est une technologie qui permet de connecter n'importe quels ensemble d'objets du monde physique entre eux à travers l'internet et /ou des réseaux locaux comme les réseaux de capteurs sans fil (WSN), pas seulement des dispositifs électroniques, mais consiste à intégrer et embarquer des capteurs et systèmes intelligents dans les divers produits [3], pour récupérer les informations et les données (sur leur identité, leur caractéristiques et leur environnement …).

Gartner[1] prévoit en effet que 26 milliards d'objets seront installées en 2020. [4]

Figure I.1 : L'internet des objets. [5]

Gartner[1] entreprise de conseil et de recherche dans le domaine des techniques avancées.

I.2.1 Les applications les plus intéressantes de l'IoT

Des nombreux secteurs ont bénéficié de cette technologie tels que la santé, le transport, l'agriculture, et l'industrie. Nous présentons dans cette section quelques exemples d'application afin d'illustrer l'intérêt de cette technologie :

Smart Home

L'internet des objets est déjà une réalité dans la domotique et la gestion de l'habitat. Cela permet par exemple de piloter la température ou l'éclairage de son domicile en fonction des usages, dans l'optique d'économiser la consommation d'énergie.

Smart City

L'internet des objets peut résoudre les problèmes de congestion routière et réduire le bruit, la criminalité, la pollution, trouver des emplacements de stationnement disponibles gratuitement à travers la ville. De plus, les capteurs peuvent détecter les problèmes de falsification du compteur, les dysfonctionnements généraux et tout problème d'installation dans le système électrique.

Wearables

Les montres ne sont plus seulement pour dire le temps, l'Apple Watch et d'autres Montres intelligentes ou d'autres dispositifs sur le marché tels que Fitbit et Jawbone ont aidé à révolutionner le monde du fitness en donnant aux gens plus de données sur leurs séances d'entraînement.

I.3 Les réseaux de capteurs sans fil

Les Réseaux de capteurs sans fil (WSN) sont un ensemble des nœuds ou capteurs communicants dans une zone de captage, l'objectif est de capter et transmettre des données du monde physique vers un ou plusieurs points de collecte [6] afin de mesurer des phénomènes bien précis. Les émissions et réceptions des données sont effectués sur un médium « sans-fil », il peut être de type optique ou de type Radio-fréquence tel que Wi-fi ou encore la norme IEEE 802.15.4 sur laquelle s'est basée notre travail.

Les Réseaux de capteurs sans fil sont composés de grand nombre de nœuds capteurs autonomes déployés pour mesurer des paramètres physiques de l'environnement, ses principaux éléments sont : [7]

- **un ou plusieurs capteurs de grandeurs physiques** : (température, humidité, niveau sonore, etc), c'est pour capter et acquérir des données.
- **une radio :** permettant aux nœuds de communiquer, elle effectue toutes les émissions et les réceptions des données.
- **une batterie :** qui est souvent la seule source d'énergie.
- **un microcontrôleur :** la partie logicielle qui relie les capteurs et la radio elle est composée d'un système d'exploitation sur lequel s'exécute l'application de mesure et d'une pile protocolaire qui exécute les protocoles réseau.

I.3.1 Description de la norme IEEE 802.15.4

L'institut des ingénieurs électriciens et électroniciens (ou l'IEEE) soutient de nombreux groupes de travail pour développer et maintenir des normes de communications sans fil et filaires. Par exemple, 802.3 pour les câbles ethernet et 802.11 pour les LAN sans fil (WLAN). Le groupe de normes 802.15 spécifie une variété de réseaux personnels sans fil (WPAN) tel que la catégorie IEEE 802.15.4 qui est très utilisé dans les protocoles de communication des réseaux de capteurs sans fil, caractérisé par des débits de données trop faible de 250 kb/s, 40 kb/s et 20 kb/s, de basse consommation énergétique (inférieure à 0,01 mA en mode veille), et de faible portée jusqu'à 100 m a été développée pour les applications de surveillance et de contrôle à faible débit de données et pour les utilisations à faible consommation d'énergie à longue durée de vie. [8], en raison de sa caractéristique ils ont choisi comme l'un des supports de communication utiliser pour les réseaux LLN.

I.3.1.1 Topologie et fonctionnements du réseaux IEEE 802.15.4

L'IEEE a défini deux types de dispositifs pouvant participer à un réseau :

1) Le dispositif ayant toutes les fonctions possibles (Full Function Device FFD), peut assurer trois rôles dans un réseau : coordinateur PAN, routeur ou dispositif relié à un capteur.

2) Le dispositif ayant des fonctions limitées (Reduced Function Device RFD), utilisé pour des applications simples, un simple capteur avec un module de transmission.

Le FFD peut dialoguer avec des RFD et des FFD, tandis que le RFD dialogue avec un FFD uniquement. Pour communiquer sur un même réseau, il faut au moins un FFD. [6]

IEEE 802.15.4 fonctionne selon l'une des deux topologies : la topologie étoile ou la topologie Pair à Pair comme l'illustre **la figure I.2.**

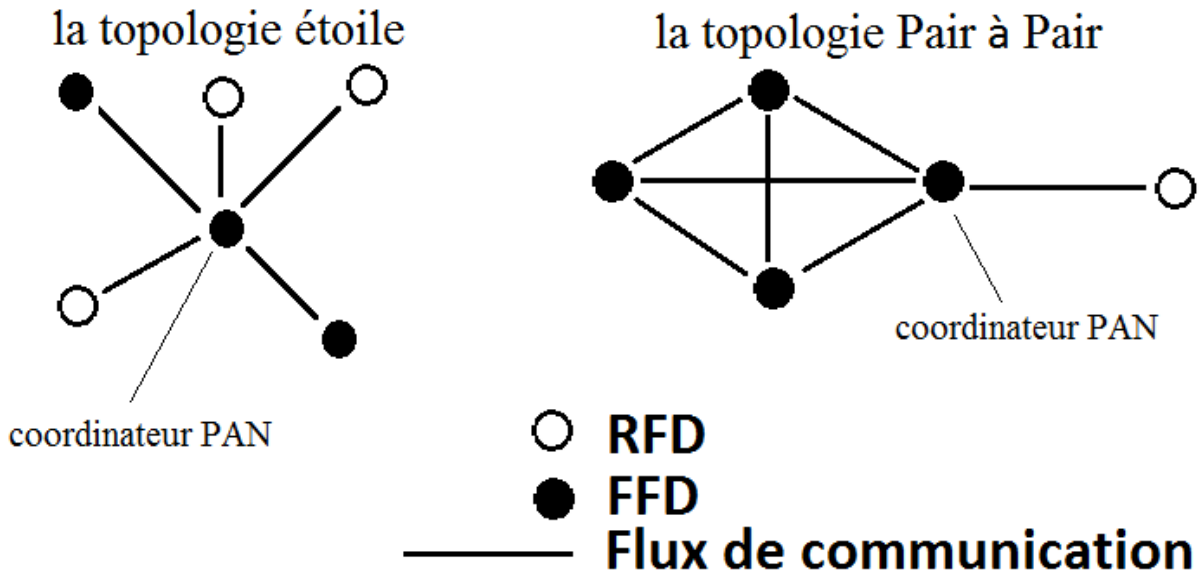

Figure I.2 : Exemples de topologie étoile et Pair à Pair.

I.3.1.2 L'architecture IEEE802.15.4

L'architecture IEEE802.15.4 est implémente sur deux couches : la couche physique (PHY) et la couche contrôle d'accès au support (MAC), chaque couche est responsable d'une partie de la norme et offre des services aux autres couches.

La couche physique contient l'émetteur/récepteur radio (RF), permet de l'échange de bits de données avec le support de transmission physique, ainsi que l'échange de bits de données avec la couche ci-dessus (sous-couche MAC). La couche physique peut aussi activer et désactiver l'émetteur-récepteur RF, détecter l'énergie et indiquer la qualité du signal [9] [10].

La couche MAC permet de générer des balises du réseau, si le dispositif est un coordinateur elle permet de soutenir l'association et la dissociation de PAN et soutenir la sécurité des périphériques.

Il existe cependant deux modes de fonctionnement de la couche MAC selon le type de topologie utilisé et le besoin en débit garanti, à savoir :

Le mode non-beacon utilisant CSMA/CA et qui est un mode asynchrone.

Le mode beacon ou mode synchrone avec l'envoi à période régulière d'une balise pour synchroniser les dispositifs, garantissant un débit au capteur ayant un GTS.

I.4 Low-Power and Lossy Networks

Les réseaux de faible puissance et à pertes (LLN) sont ceux dans lesquels les nœuds et leurs interconnexions sont fortement contraints par les ressources. Les nœuds sont généralement limités en termes de puissance de traitement, de batterie et de mémoire, et leurs interconnexions sont caractérisées par des liaisons instables avec des taux de perte élevés et des débits faibles. Les modèles de trafic sont également variés, et peuvent comprendre du point à point (P2P), du point à multipoint (P2MP) ou du multipoint à point (MP2P) [1]. Ils peuvent potentiellement comprendre des milliers de nœuds, ils supportent IPv6 et peuvent s'interconnecter par une variété de technologies de communication, tels que IEEE 802.15.4 ou Wi-Fi à faible consommation. Dans la plupart des cas, les LLN seront utilisés sur des couches de liaison avec des tailles de trame restreintes [11].

I.4.1 LLN versus WSN

Les réseaux LLN tout comme les réseaux de capteurs sans fil, l'une des principales différences entre eux c'est que les WSN ne relie pas avec l'internet et ne supportent pas IPv6, l'internet c'est pour contrôler les dispositifs à distance,

L'utilisation de IPv6 permet de garantir un espace d'adressage plus large que celui couvert par IPv4, pour la souplesse fournie qui permet l'amélioration des options et pour l'étiquetage des paquets appartenant à des « flux » particuliers.

I.4.2 IPV6

Au début des années 1990, il est devenu clair que le développement d'internet allait aboutir à l'épuisement des adresses disponibles fournie par IPV4, ce qui a conduit à l'émergence de IPv6 après des travaux menés au sein de l'IETF au cours des années 1990.

Les changements d'IPv4 à IPv6 tombent principalement dans les catégories :

Capacités d'adressage étendues :

IPv6 augmente la taille de l'adresse IP de 32 bits à 128 bits, donc l'augmentation de l'espace d'adressage.

Simplification du format d'en-tête :

Certains champs d'en-tête IPv4 ont été supprimés ou rendus facultatifs pour réduire les coûts et pour limiter le coût de la bande passante de l'en-tête IPv6.

Prise en charge de l'amélioration des extensions et des options :

Les modifications apportées à la manière dont les options d'en-tête IP ont encodé plus efficacement, des limites moins strictes sur la longueur des options et une plus grande souplesse pour introduire des nouvelles options à l'avenir.

Capacité d'étiquetage du débit :

Une nouvelle capacité est ajoutée pour permettre l'étiquetage des paquets appartenant à des « flux » particuliers de trafic pour lesquels l'expéditeur requiert un traitement spécial, comme la qualité non service ou « temps réel ».

Fonctions d'authentification et de confidentialité :

Des extensions pour prendre en charge l'authentification, l'intégrité est la confidentialité des données.

I.5 6LoWPAN

Les réseaux IPv6 personnels sans fil de faible puissance (6LoWPAN) : est un groupe de travail crée par l'IETF afin de permettre l'adoption de IPv6 par les réseaux LLN.

En effet, puisque l'adoption de l'IPv6 par les réseaux LLN a fait très vite l'unanimité à cause des milliards d'objets potentiels à connecter à internet, un autre problème s'est posé, celui de la taille des trames très petites définis par les protocoles de communications des couches sous-jacentes. Si on prend pour exemple la norme IEEE802.15.4, la taille des trames est uniquement de 127 octets alors que les paquets IPv6 ont une taille minimale de 1280 octet. Il est donc nécessaire de définir une couche d'adaptation afin de permettre au paquet IPv6 de transiter à travers des trames de taille beaucoup plus réduites.

C'est l'IETF qui a défini la couche adaptative 6LoWPAN afin de résoudre cette incompatibilité [2]. 6LoWPAN se base principalement sur deux mécanismes afin de réduire la taille des datagrammes IPv6 à savoir la fragmentation et la compression des entêtes afin de permettre aux paquets IPv6 d'être envoyés ou reçus via les réseaux LLN.

I.5.1 Architecture 6LoWPAN

Le réseau 6LoWPAN est composé par des nœuds partagent le même préfixe IPv6 et de routeur de bord qui s'occupe de la gestion de la compression et la fragmentation d'en-têtes IPv6.

On [6] distingue trois familles d'architecture dans un réseau 6LoWPAN, réseau **6LoWPAN simple** avec un seul routeur de bord (Edge router) connecte à d'autre réseau IP, réseau **6LoWPAN étendu** avec plusieurs routeurs de bord relient par une dorsale et réseau **ad hoc 6LoWPAN** non connecté.

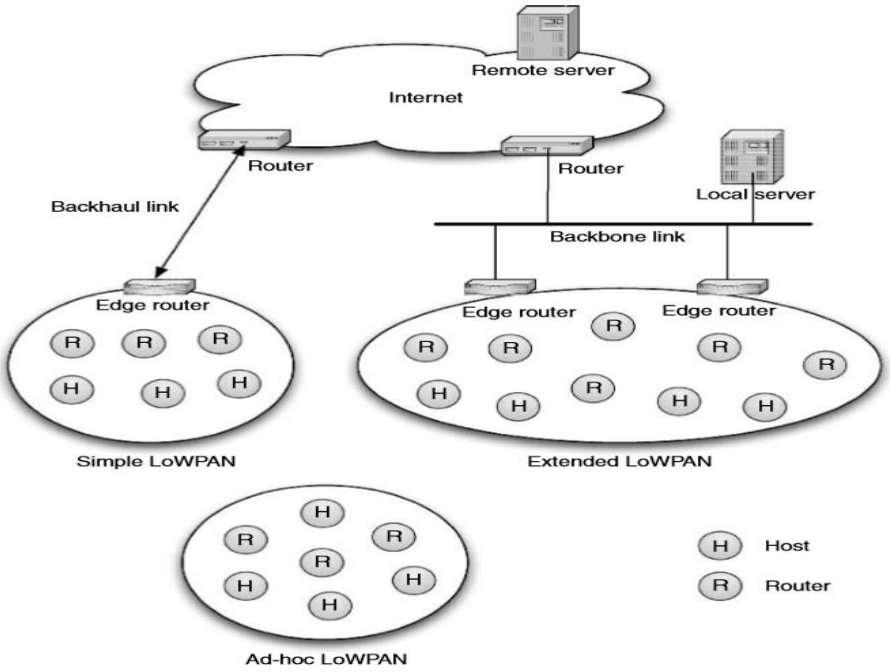

Figure I.3 : Architecture 6LoWPAN. [6]

I.5.2 Fonctionnement de 6LoWPAN

6LoWAPN c'est une couche d'adaptation situé entre la couche réseau et la couche liaison de données charge de réduire la taille des paquets IPv6 en utilisant la fragmentation et la compression.

1) Compression d'en-tête

La couche d'adaptation 6LoWPAN se charge de la compression des en-tête IPv6 dont la taille est de 40 octets et des en-têtes UDP de 8 octets venus de la couche réseau et les envoie vers la couche de liaison.

On peut distinguer trois scénarios de communication possibles afin d'illustrer l'intérêt d'un tel mécanisme [2] :

Premier scénario : La communication entre deux périphériques à l'intérieur du même réseau 6LoWPAN, en utilisant des adresses locales de liaison, l'en-tête IPv6 peut être compressée à seulement 2 octets. C'est le scénario du meilleur cas.

Second scénario : La communication destinée à un périphérique en dehors du réseau 6LoWPAN et le préfixe pour le réseau externe est connu, où l'en-tête IPv6 peut être compressé à 12 octets.

Troisième scénario : Similaire au second scénario, mais cette fois sans connaître le préfixe du périphérique externe, ce qui donne un en-tête IPv6 de 20 octets scénario du pire cas.

On constate que même dans le pire cas, on arrive quand même à réduire la taille de l'en-tête IPv6 de moitié. Si malgré ce mécanisme la taille du datagramme reste supérieur à celui des trames un autre mécanisme est proposé celui de la fragmentation.

2) Fragmentation et réassemblage

Afin de réduire la langueur des datagramme IPv6, la couche 6LoWPAN fragmente les paquets venus de la couche réseau et les envoie sous forme des trames de petites tailles (trames 802.15.4 dans le cas du standard IEEE802.15.4) à son équivalent sur l'équipement distant qui se charge de les réassembler. Chaque fragment est précédé d'un en-tête de fragmentation.

Bien que nécessaire, un impact plus ou moins important selon le mode de routage utilisé.

I.5.3 Le routage dans le 6LoWPAN

Le routage dans le 6LoWPAN est réalisé selon deux modes : mesh-under et route-over.

1) Mesh-under

Dans le Mesh-under, la couche réseau n'effectue aucun routage IP à l'intérieur d'un LoWPAN. La couche d'adaptation exécute le relayage de nœud en nœud sans solliciter la couche IP et achemine les paquets vers la destination sur plusieurs sauts radio donc la décision de routage se fait au niveau de la couche d'adaptation. [12]. Si un paquet IP est fragmenté par la couche d'adaptation les fragments sont livrés au saut suivant par routage maillé. Différents fragments d'un paquet IP peuvent passer par différents chemins et ils sont rassemblés à la destination. Dans ce cas, le paquet IPv6 n'est reconstitué que sur l'équipement destinataire. Si tous les fragments sont atteints avec succès, alors la couche d'adaptation du nœud de destination rassemble tous les fragments et crée un paquet IP. Dans le cas où un fragment quelconque manque, tous les fragments pour ce paquet IP sont retransmis vers la destination.

2) Route-over

Dans le mode route-over, toutes les décisions de routage sont prises dans la couche réseau où chaque nœud agit en tant que routeur IP. Lorsqu'un paquet IP est fragmenté par la couche d'adaptation, les fragments sont envoyés au saut suivant sur la base des informations de la table de routage. Si tous les fragments sont reçus avec succès, la couche d'adaptation crée un paquet IP à partir de fragments et l'envoie à la couche réseau. [12]. Si le paquet est destiné à lui-même, la couche réseau envoie le paquet IP à la couche de transport, sinon envoie le paquet au saut suivant sur la base des informations de la table de routage. Dans ce mode de routage, les paquets sont reconstitué sur chaque équipement intermédiaire afin de prendre la décision de routage s'il y a un ou plusieurs fragments manquants, tous les fragments sont retransmis à une distance de houblon. Route-over est plus efficace dans des conditions dégradées (perte de paquets).

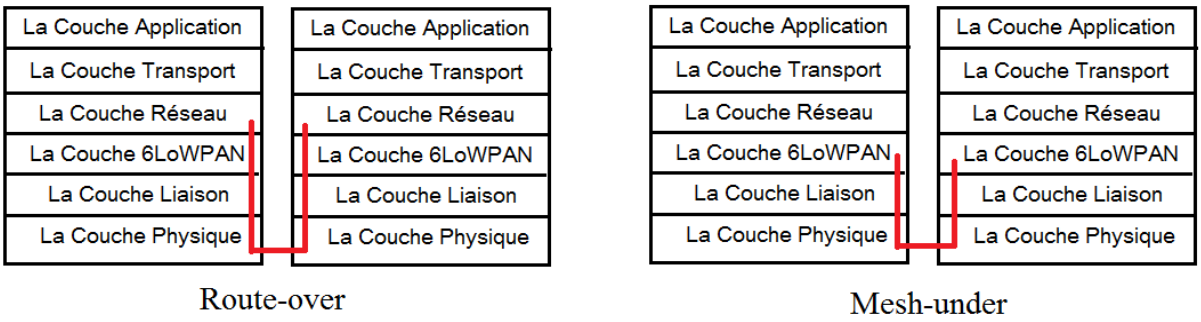

Figure I.4 : les deux modes de routage Route-over Mesh-under.

I.6 Conclusion

Dans ce premier chapitre nous avons présenté le contexte de ce travail et plus particulièrement les réseaux LLN qui connectent des objets bien particuliers qui ont de fortes contraintes en termes de ressources tels que les capteurs sans fil, les sondes etc. L'interconnections entre ces objets est également caractérise par de faible débit et des taux de pertes importants [1]. Dans la plupart des cas, les LLN utilisent des médiums de communications à faible consommation tel que la norme IEEE 802.15.4. Ces médiums disposent de trames avec tailles restreintes. Pour permettre aux paquets IPv6 qui a une taille de 1280 octets d'être envoyés ou reçus par ces médiums de communications [2] l'IETF crée la couche d'adaptations 6LoWPAN.

Dans le chapitre suivant, nous aborderons le côté routage dans les LLN, en citant la classification des différents protocoles de routage selon leurs fonctionnements, la problématique des contraintes de routage dans les LLN et les différents protocoles de routage proposés pour ce genre de réseau.

CHAPITRE II

Routage dans les réseaux LLN

II.1 Introduction

Les LLN sont des réseaux sensibles au vue de leurs caractéristiques et contraintes qui peuvent nuire au fonctionnement du routage.

Dans ce chapitre nous allons revenir sur la notion de routage pour mieux la définir, nous allons également décrire les principaux protocoles de routage qui existent. L'analyse de ces protocoles et des contraintes propres des réseaux LLN nous permettra de voir en quoi ils ne sont pas adaptés aux réseaux LLN.

II.2 Le routage

Le routage est un mécanisme pour l'acheminement des paquets de données d'un émetteur vers un ou plusieurs destinataires, à travers des routeurs et des protocoles de routage, les routeurs sont des machines qui relient deux ou plusieurs réseaux afin de commuter des paquets d'une interface vers une autre selon la destination des paquets, les protocoles de routage définissent la manière de fonctionnement des routeurs selon la propriété de protocole, les protocoles de routages sont classés en plusieurs grande classe selon des différent critères.

Les protocoles de routage servent à deux fonctions, la première est la construction avec la maintenance des routes pour certaines destinations, la deuxième consiste en l'acheminement des données sur ces routes.

Le plus grand défi des protocoles de routage est de trouver à un instant donné le meilleur chemin entre deux stations, c'est-à-dire la suite des nœuds pouvant acheminer les données le plus efficacement possible [13]. Cependant, les ressources du réseau peuvent changer à tout moment, la difficulté est donc de s'adapter à ces changements afin de maintenir la communication, d'une part, et de conserver un routage efficace d'autre part.

Il y'a deux type de routage, le routage statique consiste à faire les mises à jour des informations de routage de façon manuelle à chaque modification de la topologie réseau et le routage dynamique où les mises à jour des informations de routage sont faites de façon automatique à chaque modification de topologique réseau. [14]

II.3 Classification des algorithmes de routage

On peut classer les protocoles de routage selon de nombreux critères. Dans ce travail, nous mettons l'accent sur deux principaux critères :

1) A quel moment sont calculés les routes : on distingue alors les protocoles de routage proactifs et réactifs

2) Comment sont calculées les routes : on distingue alors les protocoles de routage à état de lien et les protocoles de routage à vecteur de distance.

1) **Protocoles proactifs versus protocoles réactifs :**

Selon l'instant où les routes de destination sont connues, on distingue deux familles de protocoles **proactifs et réactif**s. Cette classification a été proposée par le groupe de travail MANET de l'IETF. L'approche réactive vise à réduire la surcharge de trafic afin de minimiser

les ressources consommées, alors que l'approche proactive a pour but de minimiser le délai de mise en place des communications. Des approches hybrides sont aussi proposées.

Protocoles de routage proactifs :

Les protocoles de routage proactifs fonctionnent sur le principe que chaque nœud doit connaître une route vers n'importe quelle destination du réseau à tout instant. Un protocole de routage proactif désigne donc le fait de calculer les routes avant qu'il n'y en ait besoin. De cette façon, un nœud peut transmettre des données vers une destination après une simple consultation de sa table de routage et sans délai supplémentaire. Pour maintenir cette table de routage avec des informations valides [13], ces protocoles requièrent des mises à jour périodiques et permanentes, matérialisées par des messages envoyés par les nœuds du réseau. La fréquence d'émission doit être suffisamment importante pour prendre rapidement en compte les modifications de la topologie du réseau, mais suffisamment faible pour ne pas surcharger le réseau par ces messages de contrôle.

Parmi les protocoles proactifs les plus connus figurant OLSR (Optimized Link State Routing), BATMAN (Better Approach To Mobile Ad hoc Networking) et RPL (IPv6 Routing Protocol for Low power and Lossy Networks) qui fera l'objet d'une description détaillée au chapitre 3. [13]

Protocoles de routage réactifs

Les protocoles réactifs fonctionnent en construisant les routes à la demande et ne les maintiennent que si elles sont utilisées. L'avantage par rapport à une approche proactive est qu'il n'a pas besoin de maintenir de route lorsqu'il n'y a pas ou peu de trafic, cela permet d'être moins gourmand en ressources. Cependant, puisque la route n'existe pas avant d'être utilisée, un moment de latence est introduit avant que la route ne soit construite. [13]

Parmi les protocoles réactifs les plus connus AODV (Ad hoc On demand Distant Vector) et DSR (Dynamic Source Routing). [13]

2) **Protocoles de routage à état de lien versus protocoles de routage à vecteur de distance :**

Selon la méthode de routage, on distingue deux familles de protocoles à **états de lien / et à vecteur de distance**. Les algorithmes de routage basés sur ces deux méthodes utilisent la même technique du plus court chemin, et permet à un nœud donné, de trouver et d'atteindre la destination en utilisant le trajet le plus court existant dans le réseau.

Protocoles à États de lien

Dans les protocoles de routage à états de lien (Link State Protocols), chaque routeur décrit lui-même ses interfaces avec ses objets directement connectés ; ces objets peuvent être soit voisins routeurs adjacents, ou ils peuvent être directement rattachés à des réseaux. Cette information est passée inchangée d'un routeur à un autre, de sorte qu'en fin de compte, chaque routeur connaît tous les autres routeurs, ses interfaces et à quoi exactement ils se connectent essentiellement dans les protocoles de routage d'état de lien, chaque routeur connaît la topologie de réseau entière jusqu'à chaque routeur unique et chaque interconnexion simple.

Les protocoles de routage d'état de lien permettent à un routeur d'avoir une carte complète du réseau et d'utiliser des algorithmes spécifiques pour trouver les chemins les plus courts à chaque objet du réseau comme l'algorithme de Dijkstra [15].

Parmi les protocoles à états de lien : OLSR (Optimized Link State Routing) [13].

Protocoles à vecteur de distance

Les protocoles à vecteur de distance (distance vector protocols) sont basés sur des variations de l'algorithme de Bellman-Ford [16]. Les informations de routage ne sont échangées qu'entre des voisins directement connectés. Cela signifie qu'un routeur sait à partir de quel voisin un itinéraire a été appris, mais il ne sait pas où ce voisin a appris l'itinéraire ; Un routeur ne peut pas voire au-delà de ses propres voisins [15]. L'information de routage est mise à jour soit pro-activement soit ré activement en fonction de l'algorithme de routage. [16]

Parmi les protocoles à vecteur de distance les plus connus AODV (Ad hoc On demand Distant Vector) [13].

Chacune de ces classes de protocole a ses propres caractéristiques et contraintes. Le choix du protocole de routage dépend grandement du contexte dans lequel va s'effectuer ce routage et des contraintes à considérer afin d'atteindre les objectifs fixés au départ.

II.4 Contraintes de routage dans les réseaux LLN

Les réseaux LLN sont fortement contraints par les ressources, les nœuds sont généralement limités en termes de puissance de traitement, de batterie et de mémoire, [11] ces contraintes peuvent nuire le fonctionnement de routage il cause des interconnexions instables avec des taux de perte de paquets élevés et des débits faibles. Les protocoles de routage développé par le passé pour des réseaux ad hoc et les réseaux de capteurs sans fil ne s'adapte pas à ce dernier, donc pour réussir le routage dans les réseaux LLN il faut choisir ou développer des protocoles de routage spéciales prises en compte l'exigences d'une part et sans négliger la performance d'autre part.

II.5 Algorithmes de routage dans les réseaux LLN

L'IETF a formé plusieurs groupes de travail (WG) et leur a assigné la tâche de définir des protocoles de routage d'abord pour les réseaux ad hoc avec le groupe de travail MANET et puis pour les réseaux LLN avec les groupes de travail 6LoWPAN et ROLL.

Le groupe de travail 6LoWPAN : crée par L'IETF, dans l'objectif de créer une couche d'adaptation permettant aux paquets IPv6 d'être pris en charge efficacement par des trames de petite taille, afin d'obtenir un protocole plus léger qui maximise l'efficacité de la bande passante dans 6LoWPAN, un protocole de routage LOAD (6LoWPAN Ad Hoc On-Demand Distance Vector Routing) a été proposée par le groupe de travail 6LoWPAN, LOAD c'est un dérivé de l'AODV, mais adapté pour les adresses L2 et le routage sous maillage, et avec certaines simplifications sur AODV, après la création de ROOL le développement de LOAD a été suspendu par le groupe de travail 6LoWPAN, en attendant les résultats de ROLL et les expériences avec RPL. D'autres protocoles ont été développer par le groupe 6LoWPAN tel que DYMO-LOW, HI-LOW.

Le groupe de travail ROLL (Routing over Low-Power and Lossy Links), c'est un groupe de travail crée par L'IETF dans l'objectif d'élaborer un protocole de routage pour les LLN, basé sur IPv6. A partir de 2011, le groupe de travail ROLL travaille principalement sur RPL (protocole de routage pour les réseaux à faible puissance et à perte). Ce protocole sera largement décrit dans le chapitre qui suit.

II.6 Conclusion

Dans ce deuxième chapitre nous avons décrit le routage et ses contraintes dans les réseaux LLN avec ses différentes classes.

Avec la multiplication d'appareils sans fil, les groupes de travail tel que MANET ou ROLL commencent à élaborer de nouveaux protocoles de routage spécialement conçus pour répondu aux contraintes des environnements et de nombreux protocoles de routage tel que OLSR, AODV et RPL.

Dans le chapitre suivant nous décrivons le protocole de routage RPL spécialement conçu pour répondre aux contraintes des réseaux IPv6 à faible consommation et à perte (RPL) .

CHAPITRE III

Le protocole de routage RPL

III.1 Introduction

L'IETF a formé le groupe de travail ROLL dans l'objectif d'élaborer un protocole de routage pour les réseaux LLN, basé sur IPv6, à partir de 2008, le groupe de travail ROLL travaille principalement sur le protocole de routage pour les réseaux à faible puissance et à perte (RPL), Le protocole a été conçu pour être adapté aux contraintes des réseaux LLN [17]. Dans ce chapitre, nous allons aborder le fonctionnement du protocole RPL en citant la procédure de construction de la topologie RPL avec leur concept clés et en citant les deux modes d'opération « Non-Storing mode » et « Storing mode », les paradigmes de la communication, suivies de la classification des travaux antérieurs sur les améliorations de RPL.

III.2 RPL : le protocole de routage des réseaux LLN

RPL est sans doute l'un des protocoles de routage IPv6 les plus connus pour les réseaux à faible consommation et à perte (LLN) développé par ROLL dans la RFC 6550 pour répondre aux limites des réseaux LLN telles que la faible puissance de traitement, de batterie et de mémoire.

RPL vise principalement les réseaux de collecte, où les nœuds envoient périodiquement des mesures à un point de collecte [17]. Le protocole a été conçu pour être très adapté aux conditions du réseau et pour fournir des itinéraires de rechange, chaque fois que les itinéraires par défaut sont inaccessibles. RPL fournit un mécanisme pour diffuser l'information sur la nouvelle topologie de réseau formée dynamiquement.

III.3 Fonctionnement du protocole RPL

III.3.1 Les graphes DAG et DODAG

DAG (graphe orienté acyclique) est un graphe orienté qui ne possède pas de circuit. Il décrit les liens orientés entre les nœuds, se terminant à un ou plusieurs nœuds racines. [1]

RPL s'appuie sur la notion de DODAG (graphe acyclique orienté vers la destination), DODAG est un DAG a une seule destination à la racine c'est-à-dire à une seule racine DAG.

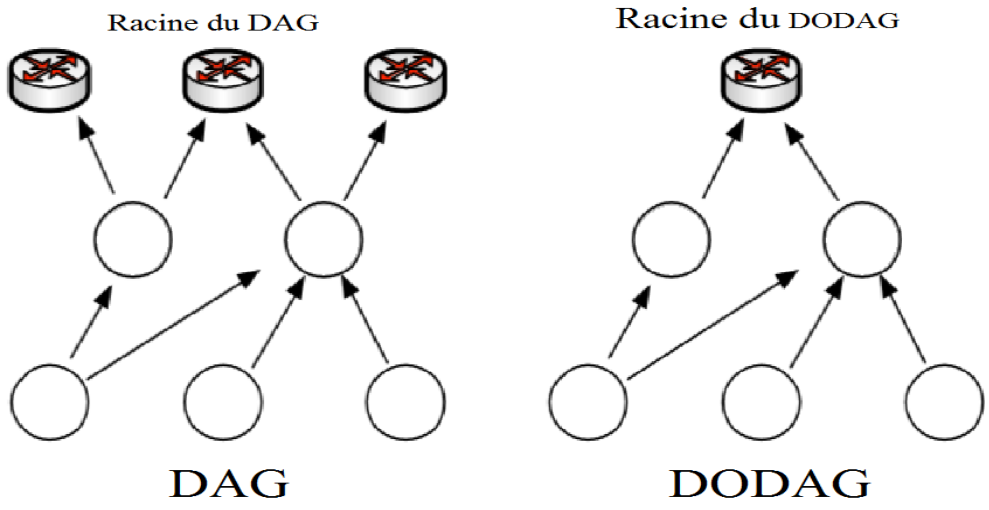

Figure III.1 : les graphes DAG et DODAG. [6]

III.3.2 Le modèle réseau :

RPL défini trois types de nœuds :

- **DODAG root** ou **Low power and lossy Border Router (LBR)** : Il se réfère à la racine d'un DODAG qui représente un point de collecte dans le réseau et a la capacité de construire un DAG. Le LBR agit également comme une passerelle (routeur edge) entre l'internet et le LLN, [18] il peut avoir des racines multiples configurent dans le réseau.
- **Nœud Routeur (ou statefull node) :** Il se réfère à un périphérique qui peut transmet et génère du trafic [18] des voisins. Un tel routeur n'a pas la capacité de créer un nouveau DAG.
- **Nœud host (ou stateless node) :** Il se réfère à un périphérique terminal capable de générer des données [18], mais qui n'est pas capable de router le trafic.

III.3.3 Messages de contrôle dans RPL

Il existe principalement quatre types de message de contrôle décrits comme suit :

1. DIO (Objet d'Information DODAG) : contient des informations permettant à un nœud de découvrir une instance RPL apprendre ses paramètres de configuration de calculé son rang et de choisir des parents qui minimisent le coût sur la route vers la racine du DODAG. [1]
2. DIS (Sollicitation Information DODAG) : pour solliciter un DIO de leurs voisins.
3. DAO (Destination Annonce Objet) : pour propager les informations de destination en remontant le DODAG le message DAO est unicast de l'enfant au parent. [1]
4. DAO-ACK : Le message DAO- ACK est envoyé en tant que paquet unicast par un destinataire DAO en réponse à un message DAO unicast. [1]

III.3.4 Topologie et Procédure de construction du DODAG

Fonction Objectif (OF) : définit comment les nœuds RPL sélectionnent et optimisent les itinéraires au sein d'une instance RPL. [1]

Rank : c'est le rang d'un nœud définit la position individuelle du nœud à d'autres nœuds à l'égard d'une racine DODAG. Classement strictement augmente dans la direction vers le bas et diminue strictement vers le haut. La façon exacte de calculer le rang se dépend de la fonction objectif (OF). Le rang peut suivre de façon analogue une distance topologique simple, peut être calculée comme une fonction de métriques de lien, et peut considérer d'autres propriétés telles que contraintes. [1]

RPLInstanceID : Les DODAGs ayant le même RPLInstanceID partagent la même fonction Objectif. [1]

Instance RPL : Une instance RPL est un ensemble d'un ou plusieurs DODAG qui partagent un RPLInstanceID. [1]

Une même instance peut inclure plusieurs DODAG : par exemple si l'objectif est de trouver les routes passant par les nœuds ayant la plus grande énergie résiduelle, plusieurs DODAG peuvent être associer. [6]

Plusieurs instances peuvent être déployés sur un même réseau physique. [6]

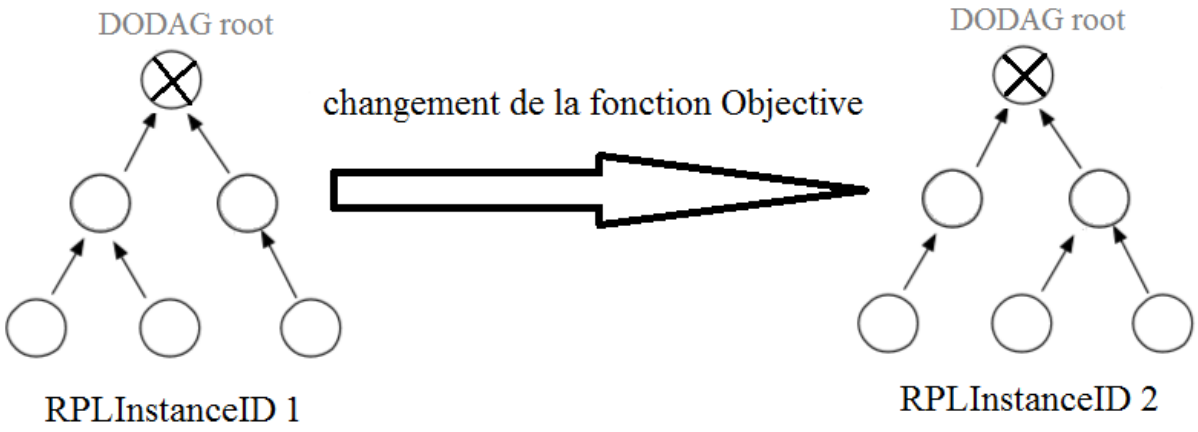

Figure III.2 : deux instances sur un seul DODAG physique.

Les nœuds envoient périodiquement des DIO en multicast sur les liens, en partant de la racine.

Les nœuds écoutent les DIO et utilisent leurs informations pour rejoindre un nouveau DODAG ou maintenir un DODAG existant.

Après échange de DIO, chaque nœud a un ensemble de nœuds parents.

Après la construction du DODAG avec des DIO et des DIS, un nœud n'a pas connaissance de ses enfants, seules les routes montantes sont connues [17].

Les nœuds informent les parents de leur présence et de leurs liens vers leurs enfants avec des messages DAO.

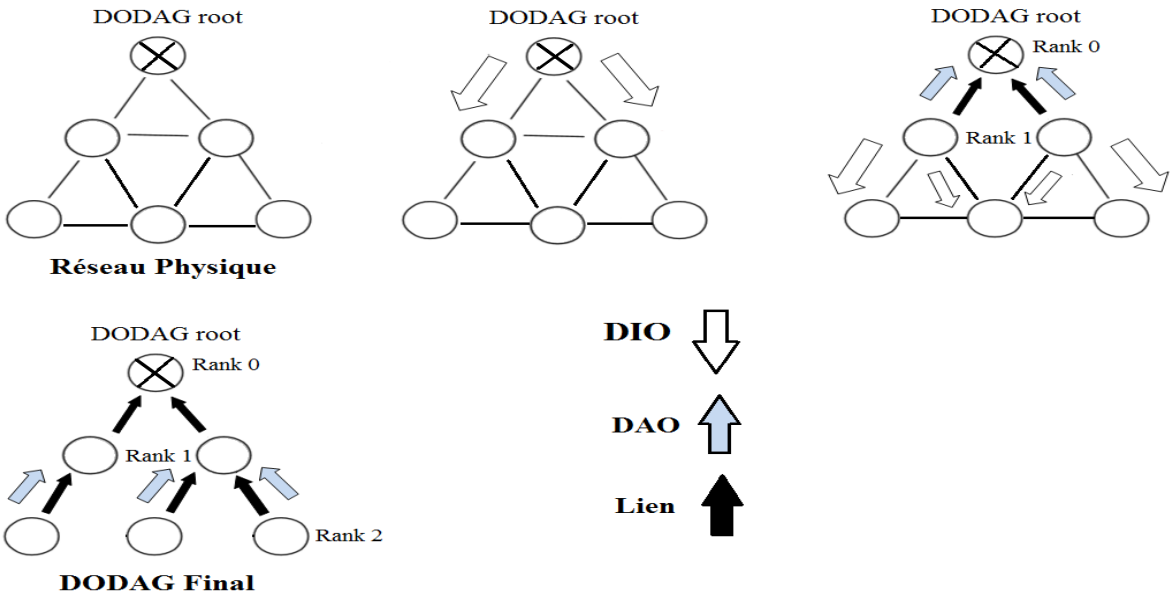

Figure III.3 : L'envoie des messages DIO et DAO et la construction de DODAG

La métrique est la valeur d'une route en comparaison avec d'autres routes, chaque métrique dispose de sa méthode de valorisation, la valeur représente le coût, la métrique peut inclure un ou plusieurs combinaisons de :

-Des propriétés sur les liens (fiabilité, latence, lien cryptées).

-Des propriétés sur les nœuds (énergie résiduelle, puissance de l'émetteur, sensibilité de récepteur). [17]

Dans l'exemple **Figure III.4**, les parents sont choisis suivant les coûts sur les liens. [17]

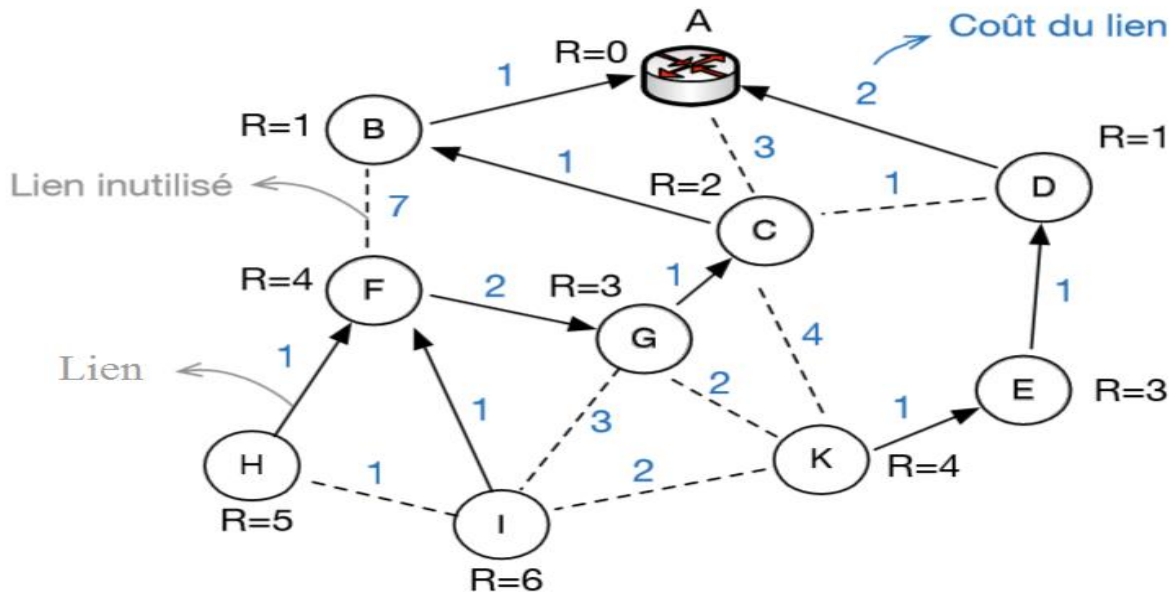

Figure III.4 : Exemple d'un DODAG construit selon les coûts sur les liens. [6]

III.3.5 Maintenance de la topologie

En cas de rupture d'un lien, le DODAG peut être réparé de deux façons ;

Réparation globale :

La racine initie une reconstruction complète du DODAG avec des messages DIO, des numéros de séquence sont utilisés pour différentier les DODAG ancien et nouveau, réparation couteuse en trafic.

Réparation locale :

Recherche d'un nouveau parent par le nœud affecté, le DOGAG n'est plus optimal, seule une réparation globale permettra une nouvelle optimisation [17].

III.3.6 Le fonctionnement de l'algorithme Trickle Timer

RPL utilise le Trickle Timer pour réduire la surcharge des messages de contrôle en ne transmettant les mises à jour que lorsque des incohérences sont détectées dans le réseau. Si un nœud entend des mises à jour DIO de ses voisins qui sont cohérentes avec sa propre compréhension de la topologie de réseau, un compteur de redondance est incrémenté [17]. Si le nombre de mises à jour cohérentes entendues dans un intervalle de temps particulier dépasse

le nombre de redondances, le nœud ne transmet aucune mise à jour et la période d'écoute est doublée. Toutefois, si une mise à jour incohérente est entendue, Trickle Timer est réinitialisé et une mise à jour est rapidement propagée.

III.3.7 Les modes d'opération du protocole RPL

En fonction de la capacité des nœuds en terme de mémoire et de la taille éventuelle du réseau, le protocole RPL offre deux modes de fonctionnement, le fonctionnement en « storing mode » et celui en « Non-Storing mode ».

Le fonctionnement en « Non-Storing mode » Dans ce mode, seul la racine est en mesure de stocker des informations de routage. Les autres nœuds du réseau conservent uniquement les adresses de leur parent direct. Toutes les informations sur la structuration du DODAG sont transmises à la racine dans les messages DAO. En cas de besoin de router des données vers une destination quelconque, les nœuds transmettent ces données à la racine en passant par leur parent. La racine effectuera un routage à la source vers la bonne destination.

Le fonctionnement en « Storing mode » Dans ce mode, les nœuds intermédiaires sont en mesure de garder en mémoire des informations de routage puis de rediriger les données reçues vers la bonne destination en consultant les informations du routage. Contrairement au « Non-Storing mode », dans le fonctionnement en « Storing mode », les messages DAO ne sont pas tous transmis à la racine. Chaque nœud transmet son message à son parent direct (parent à un saut) qui maintient une table de routage à son niveau.

III.3.8 Les paradigmes de communication

RPL prend en charge trois paradigmes de communication (MP2P), (P2MP) et (P2P). Dans ce qui suit, nous détaillons le fonctionnement de ces modèles de communication.

Multipoint à point (MP2P) : RPL a été conçu principalement pour optimiser le type de flux de trafic multipoint à point (MP2P) [17], cette communication MP2P a été fourni par la constructions des routes à partir de chaque nœud vers la racine DODAG à l'aide de DIO du parent préféré d'un nœud il s'agit de « Routes ascendantes » (Upward Routes) [17] Les destinations des flux MP2P sont des nœuds désignés qui ont une certaine importance pour l'application, tels que la fourniture de connectivité à l'Internet ou au réseau IP privé principal. [1]

Point à multipoint (P2PM) : Il s'agit de Routes descendantes (Downward Routes) [1], RPL prend en charge le trafic P2MP il utilise un mécanisme de publicité de destination qui prévoit des itinéraires descendants de la racine vers d'autre nœuds (préfixes, adresses ou groupes de multidiffusion) [17]. Par exemple les messages DIO, P2PM est le modèle de trafic requis par plusieurs applications LLN ([RFC5867], [RFC5826], [RFC5673] et [RFC5548]). [1]

Point à point (P2P) : Pour le trafic P2P, la construction des routes ça dépend de mode de fonctionnement du protocole RPL. Si le cas de mode **Non-Storing mode** le paquet dirige vers une racine, ensuite la racine effectuera le routage vers la destination, si le cas de mode **Storing mode,** le paquet s'écoule vers la racine jusqu'à ce qu'il atteint un ancêtre qui a une route connue vers la destination. Cet ancêtre commun peut être la racine DODAG. Dans d'autres cas, il peut s'agir d'un nœud plus proche de la source ou de la destination. [1] [19]

III.3.9 Fonction objectif

La fonction Objectif (OF) définit comment les nœuds RPL sélectionnent et optimisent les itinéraires dans d'une instance RPL. L'OF est identifié par un point de code objectif (OCP) dans l'option de configuration DIO. Un OF définit comment les nœuds traduisent une ou plusieurs métriques et contraintes, pour les buts de, calculer le rang de chaque nœud dans le DODAG, définit également comment les nœuds sélectionnent les parents.

Le groupe de travail ROLL a défini deux fonctions objectives dans le RPL originale cette séparation destinée à permettre à la RPL d'être adaptée pour répondre aux critères d'optimisation différents, d'applications et de conceptions de réseaux : [17] [1]

• OF0 : Objective Function Zero Ici, la métrique d'acheminement adoptée est le nombre de sauts. OF0 est conçu pour permettre l'interopérabilité entre les implémentations diffèrent de RPL. [17]

• MRHOF (Minimal Rank with hystérésis Objective Function) : La métrique utilisée par MRHOF est déterminée dans le conteneur de métrique du DIO. Le plus souvent c'est le nombre de retransmission attendue (ETX) qui a utilisé avec une hystérésis pour éviter les différences minimes de rang. Cette métrique permet à RPL de trouver les chemins stables à partir des nœuds vers une racine. En l'absence d'une métrique dans le conteneur métrique DIO, MRHOF utilise par défaut ETX [20].

III.4 Travaux d'optimisation de RPL

Même s'il est proposé pour répondre aux contraintes de routage dans les réseaux LLN. Plusieurs travaux ont montré que RPL souffre de certains défauts.

Dans cette partie, nous allons examiner une partie de ces travaux d'amélioration de RPL que nous avons classé en plusieurs catégories : (A) optimisation de la fonction objective, (B) optimisation de la topologie logique le nombre de chemins choisis et (C) Optimisation de la fonction Trickle Timer.

A) Optimisation de la fonction objectif

1) Choix des métriques

Selon une seule métrique

Par défaut, une seule métrique est utilisée dans les fonctions objectifs définies par ROLL que cela soit le nombre de saut ou l'ETX. D'autres travaux proposent néanmoins d'utiliser de nouvelles métriques visant à améliorer un aspect de façon plus spécifique. Les auteurs dans [21] proposent de s'appuyer sur le délai de transmission moyen ou (Averaged Delay - AVG DEL) qui vise à réduire le délai entre les nœuds, tout en supposant que les nœuds fonctionnent sur des cycles de sommeil et des cycles de réveil différents et ont des cycles de travail très faibles (moins de 1%). L'inconvénient avec cette approche c'est qu'elle rajoute plus de données à un DIO déjà large, ce qui augmente le risque de fragmentation.

Selon plusieurs métriques

Dans cette catégorie d'approche plusieurs métriques sont utilisées en même temps pour calculer le rang des nœuds. Le problème qui se pose est donc de trouver comment combiner ces métriques. Plusieurs solutions ont été proposées dans la littérature.

Les auteurs dans [22] s'appuient sur une approche de combinaison additive et pondérée. Dans ce cas les métriques sont normalisées et des nombres réels positifs sont utilisés comme facteurs de multiplication pour ajuster les poids relatifs des métriques de routage en fonction de l'application. Ainsi, ils ont proposé de combines deux métriques : le nombre de saut (Hop Count ou HC) et une métrique d'indicateur de confiance appelée Packet Forwarding Indication ou PFI qui permet de tester la confiance qu'un nœud peut avoir en son parent en calculant la probabilité de transfert des paquets. Cette métrique suit le concept d'ETX mais capture la fiabilité du nœud plutôt que la fiabilité du lien. En utilisant PFI, un nœud peut identifier et exclure des nœuds malicieux et parer ainsi aux attaques du trou noir ou du trou gris (black- and grey-hole attacks)

2) Choix de la fonction objective

Une autre approche proposée dans [23] s'appuie sur les principes de la logique floue pour combiner plusieurs métriques. Une nouvelle fonction objective OF-FL a également été mise en place. Les métriques considérées dans cette approche sont le délai de bout-en-bout, le nombre de saut, la qualité du lien et l'énergie du nœud. La logique floue est utilisée pour évaluer le meilleur voisin comme étant le parent préféré. Les études de simulation sont réalisées sur une implémentation Contiki / Cooja de RPL.

B) Optimisation de la topologie logique

Cette stratégie s'appuie sur le constat que comme il s'agit de réseaux à faible puissance et à perte, il serait intéressant non pas de considérer un chemin unique vers la racine mais plusieurs.

De nombreux travaux de recherche ce sont intéressés à cette approche où une myriade de bénéfices a été trouvée tels que la fiabilité, la tolérance aux pannes ou encore l'évitement de la congestion par l'équilibrage de charge efficace.

Ainsi, les auteurs proposent dans [24] un RPL optimisé appelé ORPL qui diminue la latence de routage en utilisant une approche multipath. Ils utilisent la métrique du cycle de service attendu (EDC) [25], qui est revendiquée comme l'équivalent multi chemin de ETX, il utilise également un minuteur Trickle pour diffuser les mises à jour de routage, l'inconvénient de ce protocole et qu'il produit une surcharge significative de trafic de contrôle.

C) Optimisation de Trickle Timer

Trickle timer qui est une sorte de minuterie visant à propager les messages de contrôle tout en essayant de les réduire est l'un des composants phare du protocole RPL. L'une des catégories d'approches pour optimiser RPL s'intéresse justement à Tricke timer. En effet, alors que la RFC 6206 qui décrit le principe de Trickle et son fonctionnement avertit les chercheurs/programmeurs sur la faible probabilité de d'améliorer Trickle, de nombreux travaux d'optimisation de RPL se sont intéressés à cet algorithme. Parmi ces travaux on peut citer E-tricke pour Enhanced-Trickle décrit dans [26] où les auteurs proposent un nouvel algorithme

appelé E-Trickle qui n'a pas de période d'écoute. Au lieu de réinitialiser c, le compteur de cohérence, au début de l'intervalle, il réinitialise c à un moment choisi de manière aléatoire pour éliminer l'effet cumulatif du problème d'écoute courte. Les auteurs dans [37] ont également proposé une version optimisée de Trickle (appelée opt-Trickle) qui, lorsqu'un nouvel intervalle commence, choisit les valeurs de t en fonction de l'état actuel. Si l'intervalle est réinitialisé, il choisit t de 0 à Imin, et s'il a été configuré ou démarré à partir d'un intervalle expiré, il choisit t de la moitié de l'intervalle à l'intervalle entier [I/2, I].

III.5 Conclusion

RPL est un protocole de routage développé par le groupe de travail ROLL pour répondre aux contraintes très spécifiques des réseaux LLN. Le protocole a été conçu pour être très flexible aux différentes variations des ressources du réseau. Sa fonction de construire des routes s'appuie sur la notion de DODAG. Chaque nœud calcule son rang et choisit des parents qui minimisent le coût sur la route vers la racine. Les coûts sont calculés selon une ou plusieurs métriques, le choix de ces métriques dépend de l'application et de ces objectifs/contraintes [17]. Le RPL original tel que défini dans la RFC 6550 ne peut pas répondre à tous les besoins spécifiques des applications. RPL doit donc être optimisé afin d'atteindre certains objectifs particuliers. Plusieurs travaux se sont penchés sur l'optimisation de RPL comme l'indique l'état de l'art. dans le chapitre qui suit, nous allons décrire notre méthodologie d'optimisation de RPL.

CHAPITRE IV

Méthodologie d'optimisation proposée

IV.1 Introduction

Dans les chapitres précédents nous avons présenté les concepts clés liés au routage dans un contexte 6LoWPAN, plus particulièrement en mode « route over » avec le protocole de routage RPL, ainsi que l'étude des travaux proposant des améliorations de RPL.

Dans ce chapitre, nous allons décrire la solution que nous proposons dans le cadre de notre objectif qui est l'optimisation du protocole RPL. Notre solution se focalise sur un algorithme très important dans le protocole RPL qui est celui du trickle timer responsable de l'envoi des messages de contrôle dans RPL.

Trickle est un mécanisme de propagation initialement développé pour la reprogrammation des algorithmes pour diffuser efficacement des mises à jour de code via un réseau de capteurs sans fil.
Cependant, les chercheurs ont trouvé que Trickle était une technique robuste utilisable sur une large gamme d'applications, y compris la découverte du service, la gestion la propagation du trafic de contrôle et la diffusion multicast [27]. En raison de sa fiabilité, de son évolutivité et de son faible coût d'entretien, l'algorithme Trickle est très populaire et il fait l'objet de nombreux travaux de recherche récents sur Internet of Things. Néanmoins, certains chercheurs affirment que parce que l'algorithme Trickle est déjà concis et efficace, ça ne vaut pas le coût de le modifier. Dans ce travail, nous avons choisi de traiter cette question en passant par analyse expérimentale de l'impact d'un changement de Trickle sur les performances de RPL selon différents scénarios.

Notre approche vise dans un premier temps à identifier dans quelle mesure le trickle timer influe sur les performances de RPL en analysant les paramètres de la fonction « Trickle Timer ». Ensuite, nous allons identifier par expérimentation comment modifier l'algorithme du trickle timer afin de réduire la surcharge des messages de contrôle tout en assurant le bon fonctionnement de RPL selon différents scénarios.

IV.2 Notre méthodologie pour l'optimisation de RPL

IV.2.1 Fonctionnement de l'algorithme Trickle timer

Le protocole RPL utilise la fonction « Trickle Timer » pour réduire la surcharge des messages de contrôle et utiliser les ressources limitées de manière plus optimale.

Il existe trois paramètres configurables dans la minuterie « Trickle Timer » : Imin, Imax et la constante de redondance k et expliquées ci-dessous.

Constante de redondance (k)

C'est un nombre naturel supérieur à 0 et est utilisé pour supprimer la transmission DIO si le compteur de redondance « C » dépasse la valeur de ce paramètre.

Imax

Lorsque la période d'écoute maximal est atteinte, les DIO sont transmis au taux égal à la période

d'écoute maximal.

Imin

Ce paramètre donne le minimum de temps entre deux DIO. La valeur de démarrage de « Trickle Timer » commence à partir de cette paramètre, Si un nœud entend des mises à jour DIO de ses voisins qui sont cohérentes avec sa propre compréhension de la topologie de réseau, le compteur de redondance « c » incrémente. Si le nombre de mises à jour cohérentes entendues dans une période d'écoute particulier dépasse le nombre de redondances « K », le nœud ne transmet aucune mise à jour et la période d'écoute est doublée jusqu'à ce qu'il atteigne une période d'écoute maximal (I max), Toutefois, si une mise à jour incohérente est entendue, « Trickle Timer » est réinitialisé ces paramètres et une mise à jour est rapidement propagée. [27]

IV.2.2 Description de l'algorithme « Trickle Timer »

Le fonctionnement de l'algorithme Trickle timer peut être décrit comme suit :

étape 1 : **« Trickle Timer »** commence son premier intervalle en ajustant I à une valeur de la plage [I min, I max], généralement il définit le premier intervalle à une valeur de I min.
étape 2 : Lorsqu'un intervalle commence, **« Trickle Timer »** réinitialise le compteur c à 0, Si la taille du nouvel intervalle dépasse la longueur d'intervalle maximum, I max. **« Trickle Timer »** définit l'intervalle I à I max, et attribue une valeur sélectionnée au hasard dans l'intervalle à une variable nommé t, choisie dans la plage [I / 2, I).
étape 3 : Lors de la réception d'un message cohérent, **« Trickle Timer »** augmente son compteur d'une valeur de 1.
étape 4 : Au moment choisis au hasard t, si le compteur c est supérieur ou égal à la constante de redondance k, **« Trickle Timer »** supprime son message planifié. Sinon, le message est transmis.
étape 5 : Lorsque l'intervalle I expire, **« Trickle Timer »** double la taille de l'intervalle et revenue à l'étape 2.

étape 6 : Si **« Trickle Timer »** détecte un message incohérent, **« Trickle Timer »** définit I sur I min, et commence à l'étape 1. [27]

1. Initialisation
I ← I min
2. Démarrer un nouvel intervalle
c ← 0
if I max ≤ I then
I ← I max
end if
t ← random [I/2, I]
3. Transmission cohérent reçue
c ← c +1
4. La variable aléatoire t expire
if c < k then
transmission de DIO

else

suppression de DIO

end if
5.L'intervalle I expire
I ← I ×2
6. Transmission incohérente reçue
I ← I min
Recommencer depuis l'étape 2

De façon générale, **« Trickle Timer »** permet au protocole RPL de réduire le trafic de contrôle lorsque le réseau est consistant et stable puis d'envoyer plus de messages en cas d'inconsistance.

IV.2.3 Description de l'approche proposée

L'objectif global de notre projet est de proposer une méthode d'optimisation de RPL en visant les paramètres de la fonction **« Trickle Timer »** afin d'assurer le bon fonctionnement de RPL selon différents scénarios. Les deux paramètres I min et I max ont un impact sur l'intervalle de temps entre les DIO successifs envoyés par chaque nœud. Si on réduit cet intervalle, le nombre des messages de contrôle augmente donc le DODAG va se construire rapidement mais la consommation des ressources en énergie des nœuds augmente également. D'un autre côté, si on augmente l'intervalle, le nombre des messages de contrôle diminue, donc les ressources des nœuds sont préservées mais cela peut conduire à un mauvais fonctionnement de protocole RPL. Donc il faut choisir des valeurs confortables pour trouver un bon compromis entre assurer le bon fonctionnement e RPL tout en préservant les ressources des nœuds.

Notre approche est divisée en deux phases :

(1) La première phase consiste à analyser l'impact de la variation du « I min » sur RPL en fixant le paramètre « I max » et d'autres paramètres du protocole RPL et du simulateur Cooja.

(2) La deuxième phase consiste à étudier l'impact de la variation de « I max » en fixant la variable « I min » et d'autres paramètres du protocole RPL et du simulateur Cooja, ces paramètres seront présentés dans la section **6** du chapitre **V**.

Les variations du Imin et du Imax seront étudiées selon plusieurs scénarios d'évaluation comme cela sera présentés dans le chapitre V. Pour cela plusieurs critères d'évaluation sont nécessaires pour connaître l'impact de ces variations sur le fonctionnement de RPL.

IV.3 Critères d'évaluation proposés

Pour évaluer l'approche proposée, nous devons observer les effets de la variation de Imin et de la variation de Imax sur les performances de RPL. Pour cela, nous avons défini quatre critères d'évaluation, à savoir le temps de convergence réseau, Les frais de trafic de contrôle (ou l'overhead), la latence moyenne et le taux de paquets reçus.

1. Temps de convergence réseau :

Les nœuds dans un réseau de capteurs doivent former une topologie pour communiquer. Le temps de convergence de RPL est défini comme le temps nécessaire pour que tous les nœuds de réseau rejoignent le DODAG.

Pour calculer ce critère il faut calculer la différence entre l'instant où le premier DIO ou DIS est envoyé sur le réseau par le DODAG root (noté T_{root}) et l'instant où le premier DIO du dernier nœud qui rejoint le DODAG (noté T_{last}). La formule de calcul est donnée comme suit :

Temps de convergence réseau = T_{last} - T_{root}

2. Les frais de trafic de contrôle (Overhead) : cela inclut les messages DIO, DIS et DAO générés par chaque nœud et il est impératif de limiter le trafic de contrôle en gardant à l'esprit les ressources limitées de réseau RPL.

Pour calculer ce critère, il faut diviser le nombre de massages de contrôle sur le nombre total de paquets qui circulent sur le réseau. La formule de calcul est donnée comme suit :

Overhead = le nombre des massages de contrôle / le nombre total de paquets

3. La latence moyenne de réseau : La latence est définie comme la durée de temps prise par un paquet entre le nœud expéditeur et le nœud récepteur.

Pour calculer ce critère il faut calculer la latence de chaque paquet puis additionner toutes les latences des paquets UDP et diviser sur le nombre total de paquet UDP reçu.

La latence moyenne $=\sum_1^N$(L'instant de réception du paquet UDP − L'instant d'envoi du paquet UDP)/nombre totale de paquet UDP reçu

N le nombre de paquet UDP reçu avec succès.

4. Le taux de perte de paquets : c'est le ratio de paquets UDP perdu.

Pour calculer ce critère il faut calculer la différence entre le nombre des paquets UDP envoie par les clients et le nombre total de paquets UDP reçu par le serveur puis diviser la résulta sur le nombre total de paquets UDP envoyés par les clients.

Le taux de paquets perdu = (Nombre total de paquet UDP envoyés par les clients- Nombre total de paquet UDP reçu par le serveur) / Nombre total de paquet UDP envoyés par les clients

IV.4 Conclusion

Dans ce chapitre nous avons présenté le contexte global de notre solution qui s'appuie sur Trickle timer en décrivant le fonctionnement et l'intérêt de l'algorithme du Trickle Timer. Nous avons également décrit notre approche qui vise à analyser les variations des paramètres Imin et Imax du trickle timer afin d'étudier leur impact sur le fonctionnement et les performances de RPL. Enfin, nous avons défini les critères d'évaluation de cette approche.

Dans le chapitre suivant nous aborderons la partie implémentation de l'approche proposée ensuite nous allons analyser les résultats obtenus selon différents scénarios de simulations pour évaluer notre approche.

CHAPITRE V

Étude expérimentale

V.1 Introduction

Dans le chapitre précédent, nous avons présenté le principe de notre approche ainsi que la méthodologie d'évaluation.

Dans ce qui suit, nous allons aborder la partie mise en œuvre de notre solution en utilisant l'environnement de développement Cooja qui sera présenté dans ce qui suit. Nous allons également montrer comment ont été implémenté les critères d'évaluation dans cet environnement ainsi que les résultats obtenus.

V.2 L'environnement d'évaluation

Le simulateur Cooja fournit des outils permettant de diffuser des données de chaque test très facilement dans un format lisible.

Une autre raison d'utiliser Cooja est qu'il fournit une grande quantité de code d'exemple qui peut être utilisé comme base pour le code de mise en œuvre de ce projet. En outre, Cisco a développé le Contiki IPv6 stack et il est entièrement certifié sous le programme Logo IPv6 Ready. Pour connaître les étapes d'installation de l'environnement d'évaluation Cooja, nous invitons le lecteur à se reporter à l'annexe 1.

V.2.1 Contiki OS

Plusieurs systèmes d'exploitation pour capteurs ont été implémentés [19] tel-que Contiki OS,TinyOS, Mantis, LiteOS et autre, chacun offre sa propres fonctionnalités.

Contiki est un système d'exploitation open source léger et flexible pour les nœuds de capteurs WSN créé par Adam Dunkels écrit en langage C. Contiki connecte de petits microcontrôleurs peu coûteux et peu performants à internet. [29] [30]

La configuration Contiki typique consomme 2 KB de RAM et 40 KB de ROM, une installation Contiki complète comprend des fonctionnalités telles que : noyau événementiel, multitâche préemptif, proto-threads, chargeur de programme, réseau TCP / IP, IPv4, IPv6, Support UDP, et contient un Shell de ligne de commande et une, interface utilisateur graphique a Navigateur web, un serveur web personnel, un simple client telnet, un économiseur d'écran et un serveur virtuel. [30] [31]

Le choix de Contiki OS pour l'implémentation et l'évaluation de notre approche est dû à sa fonctionnalités qui comprend le protocole de routage RPL et supporte le standard 6LowPAN.

V.2.1.1 Architecture

Le système d'exploitation Contiki est basé sur une architecture modulaire contrairement à un système d'exploitation monolithique où le système complet ainsi que les applications sont enregistrées dans un fichier binaire qui est chargé d'un seul bloc dans l'EEPROM du capteur, sur Contiki le fichier binaire contient uniquement le système, les applications sont enregistrées séparément sous forme de modules. [30]

Contiki utilise un ordonnanceur événementiel, déclenche l'exécution d'un processus qui sera mené à son terme avant de rendre la possibilité à l'ordonnanceur de démarrer l'exécution d'une nouvelle tâche. Ce type d'ordonnancement est le plus économe en ressource et est généralement adapté à l'utilisation des capteurs. Le multi-threading est plus flexible, mais il est consommateur de ressources mémoire, car chaque thread nécessite son contexte d'exécution en mémoire. Pour permettre l'exécution en parallèle de plusieurs tâches, contiki apporte le concept des protothreads. Ce concept se situe entre les deux modèles, événementiels et multi-threads. Il permet de bloquer l'exécution d'un programme dans l'attente d'un événement ou d'une condition particulière. Le programme reprend son exécution lorsque l'événement attendu est survenu. Les protothreads partagent le même contexte d'exécution. Mais le multi-thread permet des exécutions en parallèle, ce qui peut être indispensable, par exemple pour des applications en temps réel. Contiki propose une bibliothèque qui peut être chargée par le programmeur si son application nécessite le multi-threading. **[30] [36]**

V.2.1.2 Contiki contient deux piles de communication : uIP et Rime

La pile uIP c'est une pile TCP / IP légère. Il est très optimisé, seules les fonctionnalités requises sont implémentées.

uIP supporte les protocoles IPv4,IPv6, TCP, UDP ,ICMP et implémente 6LoWPAN. Lors de communications radio suivant la norme IEEE 802.15.4 [32]

Rime est une Pile de communication légère conçue pour les radios de faible puissance. Rime fournit une large gamme de primitives de communication, de La diffusion de la zone locale à meilleur effort, à des inondations de données en vrac multi-hop fiables. [33]

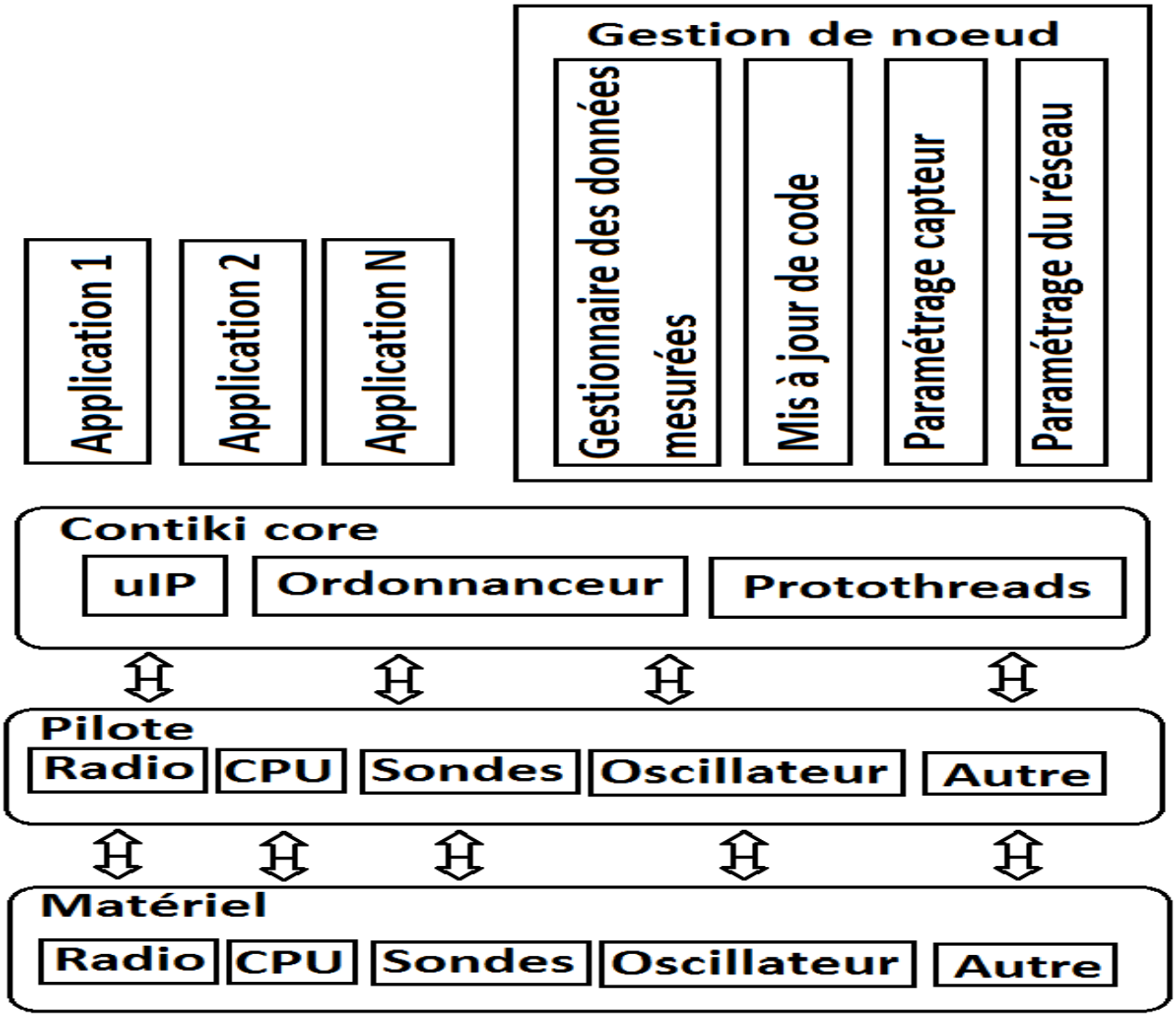

Figure IV.1 : Architecture de Contiki. [30]

V.2.1.3 ContikiRPL

ContikiRPL implémente le protocole RPL, comme spécifié dans la version 18 de la spécification RPL [34], et deux fonctions objectives OF0 et (MRHOF). La mise en œuvre de ContikiRPL sépare la logique du protocole, la construction et l'analyse des messages, et les fonctions objectives dans différents modules. Le module logique du protocole gère les informations DODAG, maintient un ensemble de parents candidats et leurs informations associées, communique avec les modules fonction objectifs et valide les messages RPL au niveau logique selon la spécification RPL. Le module de construction et d'analyse des messages se traduit par le format de message ICMPv6 de RPL et les structures de données abstraites de ContikiRPL. Enfin, les modules fonction objectifs implémentent une API fonctionnelle objective. Pour faciliter le remplacement et l'expérimentation de fonctions objectives, leur fonctionnement interne est opaque à ContikiRPL. [35]

V.2.2 Cooja et la suite Instant Contiki

Cooja permet de simuler de grands et petits réseau de capteurs (appelés motes) ayant pour systèmes d'exploitation Contiki, Cooja fait partie de la suite Instant Contiki qui est un environnement de développement complet de Contiki dans un seul téléchargement qui fonctionne dans VMWare player, possède Contiki OS et tous les outils de développement, compilateurs et simulateurs utilisés pour développer et tester le systèmes d'exploitation Contiki. [29]

Les Motes peuvent être émulés au niveau du matériel, ce qui est plus lent, mais permet une inspection précise du comportement du système. [29]

V.3 Variations des valeurs de Imin et Imax de l'algorithme Trickle timer

Le programme de la fonction « Trickle Timer » est implémentée sur le systèmes d'exploitation Contiki en cinq fichiers « trickle-timer.c », « trickle-timer.h », « trickle.c », « trickle.h » et « trickle-library.c », mais les deux paramètres de cette fonction présentent dans la section 2 de chapitre IV sont déterminée dans le fichier « rpl-conf.h » qui se trouve dans le chemins /home/user/contiki/core/net/rpl.

Imin est calculé à partir de la constante « RPL_DIO Interval Minimum » tandis que Imax est déterminée par la constante « RPL_DIO_INTERVAL_DOUBLINGS ».

V.3.1 Variations de Imin

Selon la spécification, ContikiRPL définit par défaut la variable « RPL_DIO_Interval_Minimum » par la valeur 12, ce qui signifie que le Imin = 2 ^ 12 ms = 4.096 s.

Dans notre étude expérimentale, nous allons faire varier les valeurs de RPL_DIO_Interval_Minimum comme suit : 3, 8, 12, 15 et 18.

Ceci va donner les valeurs de Imin suivantes : 2^3 = , 2^8= ,2 ^ 12 ms = 4.096 s, 2 ^ 15 ms = 32.768 s et 2 ^ 18 ms = 262.144 s.

V.3.2 Variation de Imax

Selon la spécification, ContikiRPL définit par défaut la variable « RPL_DIO_INTERVAL_DOUBLINGS » par la valeur 8 qui signifie que la durée entre deux DIO peut être doublée jusqu'à 8 fois. A partir de ça, Imax vaut par défaut :

Imax = 2^(12+8) ms = 1048.576 s

Ceci signifie que l'intervalle de temps qui sépare deux DIO vaut au maximum 1048.576 s

Dans notre expérimentation, nous allons choisir les valeurs suivantes 3, 8 et 16 de RPL_DIO_INTERVAL_DOUBLINGS ce qui va donner des valeurs de Imax suivantes : 2^(12+3) ms = 32.768 s, 2^(12+8) ms = 1048.576 s et 2^(12+16) ms = 268435.456 s.

Donc selon les valeurs par défaut, Trickle Timer commence par un intervalle de temps minimum qui vaut 4.096 s et se stabilise sur un intervalle de 1048.576s si le réseau reste consistant après avoir doublé 8 fois l'intervalle minimum comme le montre la figure **V.1**.

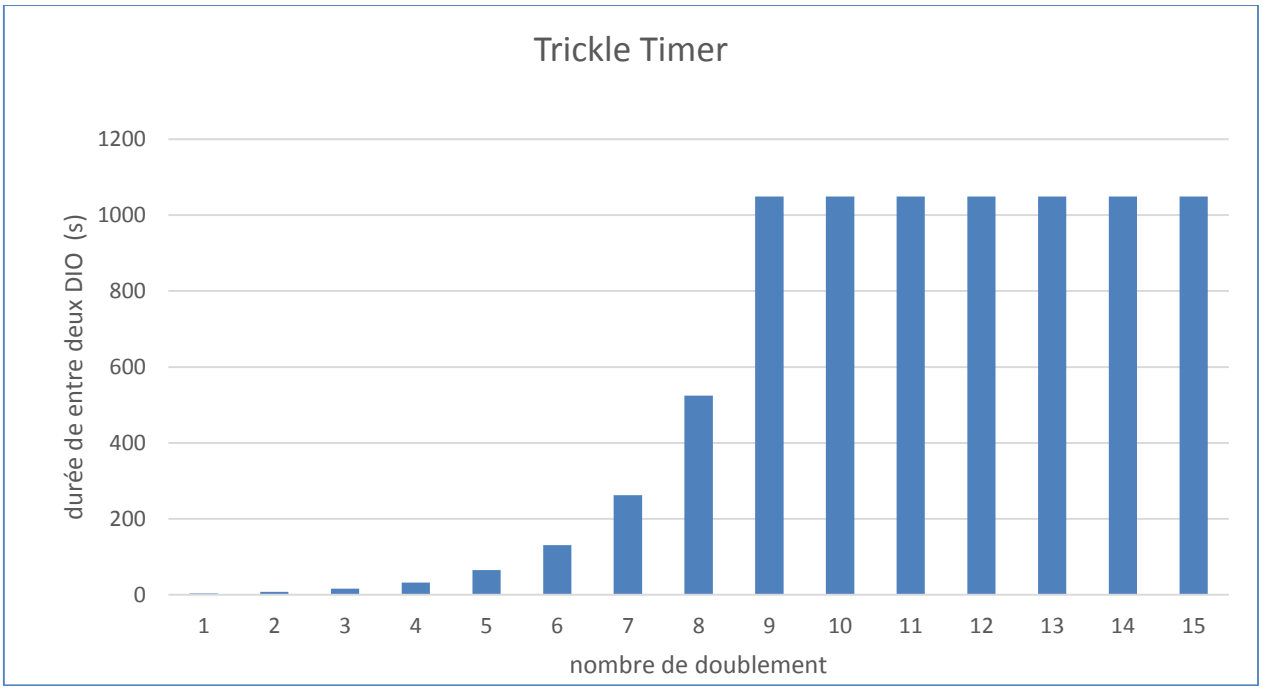

Figure V.1 : Fonctionnement de Trickle Timer selon les valeurs par défaut dans un réseau consistant.

V.4 Implémentation des critères d'évaluation

Dans cette section, nous allons décrire comment récupérer, traiter et visualiser les quatre critères d'évaluation présentés dans la section 4 du chapitre IV.

V.4.1 Récupérer les données brutes à partir de Cooja à l'aide de l'outil awk

Pour la partie récupération Tout d'abord il faut activer le débogage en remplaçant le macros « #define DEBUG DEBUG_NONE » par le macro « #define DEBUG DEBUG_PRINT » dans les trois fichiers « rpl-icmp6.c », « udp-client.c » et « udp-server.c » afin de visualiser les procédures d'exécution de ces fichiers dans le fichier journal de Cooja « COOJA.testlog », qui se trouve dans le chemin « /home/user/contiki/tools/cooja/build ».

Nous allons activer le débogage pour visualiser les messages de contrôle de RPL dans le premier fichier, pour visualiser les messages UDP envoyés par le client dans le deuxième fichier et pour visualiser les messages UDP reçu par le serveur dans le troisième fichier.

Le fichier journal « COOJA.testlog » contient les messages de l'exécution de tous les fichiers de la simulation, nous intéressons aux messages de contrôle de RPL et les données UDP envoient par les nœud client.

```
148686854 7 RPL  Sending prefix info in DIO for aaaa
148687467 2 RPL  Incoming DIO (id, ver, rank) = (30,240,1280)
148687629 52 RPL  DIO option 4, length  14
148687799 48 RPL  DIO option 8, length  30
148687901 30 RPL  DAG MC  type 7, flags 8, aggr 0, prec 0, length 2, ETX 0
148688002 26 RPL  Sending a multicast-DIO with rank 512
148688561 55 RPL  Received a DIO from fe80  212 742b 2b 2b2b
148688889 44 RPL  DIO option 2, length  6
148688908 13 RPL  Neighbor already in neighbor cache
148689979 48 RPL  Copying prefix information
148690157 30 RPL  DIO option 4, length  14
148690816 7 RPL  Sending a multicast-DIO with rank 1280
148691657 39 DATA send to 1 Hello 2
148692443 2 RPL  Incoming DIO (dag_id, pref) = (aaaa  ff fe00 1, 0)
148693018 13 RPL  Incoming DIO (id, ver, rank) = (30,240,1280)
148693503 55 RPL  Neighbor already in neighbor cache
148694042 44 RPL  DAG MC  type 7, flags 8, aggr 0, prec 0, length 2, ETX 1536
148694547 2 RPL  DIO option 2, length  6
148694807 52 RPL  DAG conf dbl=8, min=0 red=10 maxinc=1792 mininc=256 ocp=1 d_l=255 l_u=65535
148696299 44 RPL  DIO option 4, length  14
148697065 52 RPL  DIO option 8, length  30
148697256 30 RPL  DAG conf dbl=8, min=0 red=10 maxinc=1792 mininc=256 ocp=1 d_l=255 l_u=65535
148697346 15 RPL  Sending prefix info in DIO for aaaa
148697612 55 RPL  Incoming DIO (id, ver, rank) = (30,240,1280)
148697970 13 RPL  Incoming DIO (dag_id, pref) = (aaaa  ff fe00 1, 0)
```

Figure V.2 : Le fichier journal « COOJA.testlog ».

AWK est l'un des outils de traitement de texte les plus importants sur Linux. Il est très puissant et utilise un langage de programmation simple. Il peut résoudre des tâches de traitement de texte complexes avec quelques lignes de code. En commençant par un aperçu d'AWK, les instructions, le fichier de travail et le fichier de sorties.

Après la fin de la simulation, en lançant les deux commande « awk » sur le terminal pour récupérer les messages suivants.

```
user@instant-contiki:~$ awk '{if($3=="DATA"){if($
4=="send"){print $2,$4,$6,$8,$1} else {print $2,$
4,$6,$10,$1}}}' COOJA.testlog >UDP-min12-period4

user@instant-contiki:~$ awk '{if($4=="Sending" &&
 $8=="DIO"){print $8,$1,$2} else if($4=="Sending"
 && $5=="DAO") {print $5,$1,$2} else if($4=="Send
ing" && $6=="DIS") {print $6,$1,$2} }' COOJA.test
log >CONTROLE-min12-period4
```

1

2

Figure V.3 : Les deux commande awk.

1) la première commande nous permettons de récupérer les messages UDP

A) if ($4== 'send') pour les messages imprimer par les clients : $2 id de nœud client, $4 type

de l'opération « send » dans ce cas, $6 id de serveur, $8 numéro de message et $1 la date d 'envoi de message,

B) else pour les messages imprimer par le serveur : $2 id de serveur, $4 type de l'opération « recv » dans ce cas, $6 numéro de message, $10 id de client et $1 la date d 'envoi de message.

Les données qui résultent de cette commande seront enregistrées dans un fichier nommé « UDP-XN-periodM » comme le montre l'exemple de la figure **V.3**.

2) La deuxième commande nous permettons de récupérer les messages de contrôles

if ($4=='sending' && $8== 'DIO') pour trouver les messages de contrôle DIO, if ($4=='sending' && $5== 'DAO') pour trouver les messages de contrôle DAO, if ($4=='sending' && $6== 'DIS') pour trouver les messages de contrôle DIS, $1 la date envoie de message et $2 le nœud qui imprime le message de contrôle.

Les données de cette commande seront enregistrées dans un fichier nommé « CONTROLE-XN-periodM » comme l'illustre la figure **V.3.**

N indique les valeurs des deux paramètre Trickle Timer, M indique la valeur des trois scénarios de simulation.

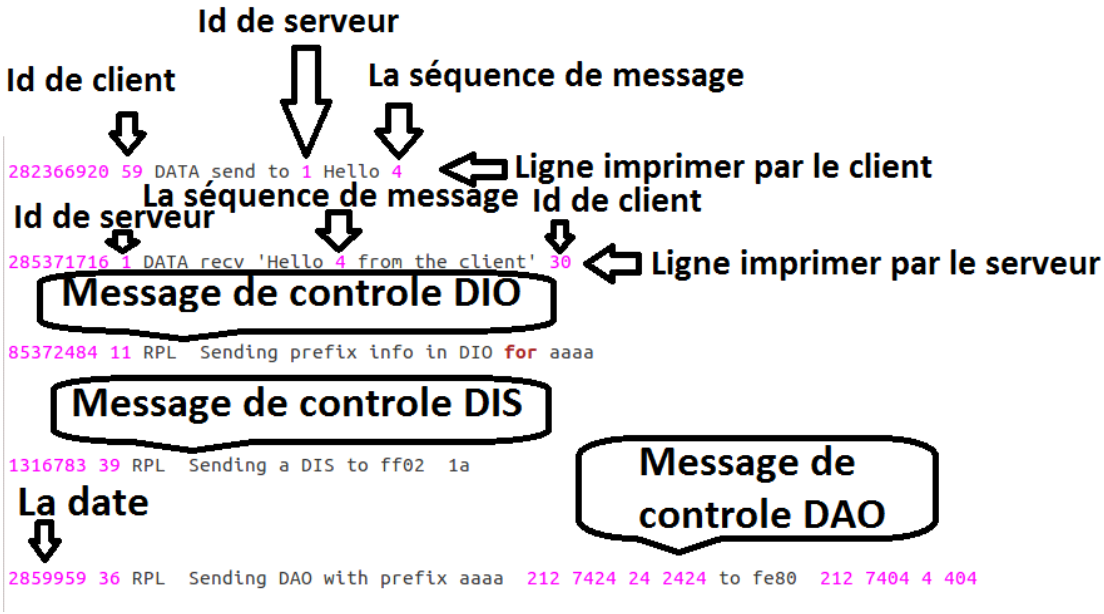

Figure V.4 : Les informations récupérer depuis le fichier journal « COOJA.testlog ».

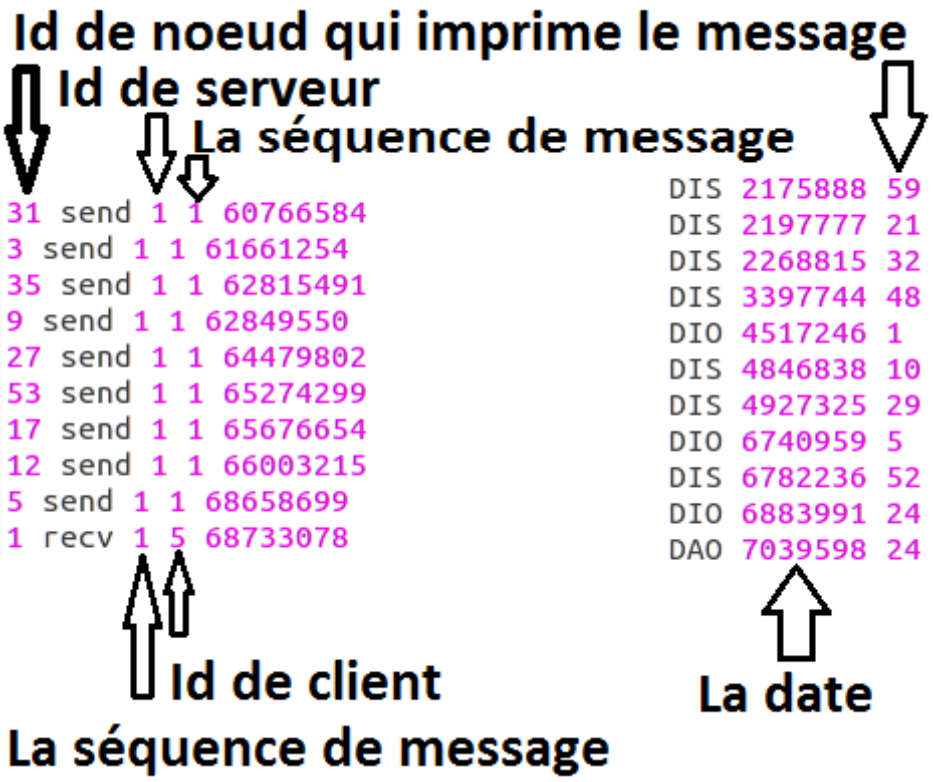

Figure V.5 : Exemple des deux type fichier obtenu par le script awk.

V.4.2 Implémentation du module de traitement et visualisation des résultats

Pour la partie traitement et visualisation des résultats nous avons implémenter une petite application en langage java. Cette application se charge de calculer les critères d'évaluation et enfin visualise ces critères en fonction des deux paramètres trickle « RPL_DIO_INTERVAL_MIN », « RPL_DIO_INTERVAL_DOUBLINGS » sous forme des graphes à l'aide de la bibliothèque jfreechart.

Notre application se compose de quatre classe Fichier, Traitement, Graphe et Contrôle.

La classe « Fichier » charge de récupérer les données depuis les deux types de fichier CONTROLE-XN-periodM et UDP-XN-periodM.

La classe « Traitement » appelle la classe fichier sur les deux fonction mudp() et mcon(), la fonction mudp() charge de calculer la latence moyenne, la ration de perte de paquet et overhead, la fonction mcon() charger de calculer le temps de convergence réseau.

```
75              for(int j=1;j<=nm;j++)
76                  {
77                   if(T[j][1]==0)
78                     {
79                      for(int h=j+1;h<=nm;h++)
80                          {
81                            if(T[h][1]==1 && T[j][0]==T[h][3] && T[j][3]==T[h][2])
82                               {
83                                double res =T[h][4]- T[j][4];
84                                ls=ls+res;
85                               }
86                          }
87                     }
88              }
```

Figure V.6 : Partie de programme qui calcule la latence.

Le principe des programmes qui calcule la latence et la ration de perte de paquet.

Le client imprime un message 'send to (id serveur) Hello (séquence de message dans chaque nœud)' dès qu'il envoie un paquet et cela nous permet de noter l'heure d'envoi. De même, le serveur imprime un message 'recv Hello (séquence de message dans chaque nœud) from the client (id de client)' et cela nous permet de noter l'heure de réception du paquet sur le serveur. Les messages d'impression sont stockés dans un fichier journal à partir de la simulation. À partir du temps d'envoi et du temps de réception, nous calculons la latence pour tous les paquets.

Nous lisons le fichier journal par ligne et notons l'identifiant des nœuds (clients et serveur), le numéro de paquet et la date l'envoi. Lorsque nous lisons une ligne de 'recv Hello (séquence de message dans chaque nœud) from the client (id de client)', nous recherchons le Send pour l'ID de client et la séquence de message pour trouver l'heure d'envoi de ce paquet. En cas de correspondance, nous calculons la latence de chaque paquet. Toutes les messages envoient par les clients et non imprimer par le serveur indiquent les paquets perdus

L'interface graphique « Graphe » utilise la bibliothèque JfreeChart pour visualiser les courbes.

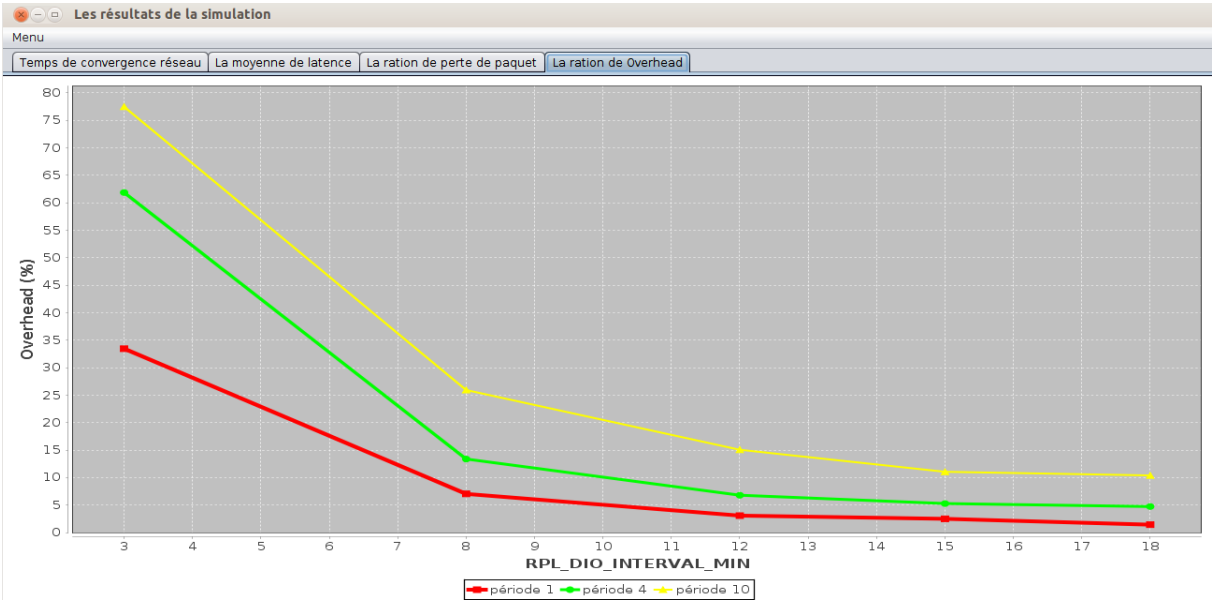

Figure V.7 : Les résulta des trois scénarios de critère « Overhead » en fonction de paramètre Imin.

L'interface graphique principale « Contrôle », cette interface contient deux buttons « Minmum » et « Doublings » pour afficher les courbes des critères d'évaluation, la premier en fonction du paramètre « RPL_DIO_INTERVAL_MIN » et la deuxième en fonction du paramètre « RPL_DIO_INTERVAL_DOUBLINGS ».

Figure V.8 : L'interface graphique principale « Contrôle ».

V.5 Description des simulations

Pour lancer Cooja, ouvrez d'abord une fenêtre de terminal.

Dans la fenêtre du terminal, accédez au répertoire Cooja avec la commande :

cd contiki/tools/cooja

Lancer Cooja avec la commande :

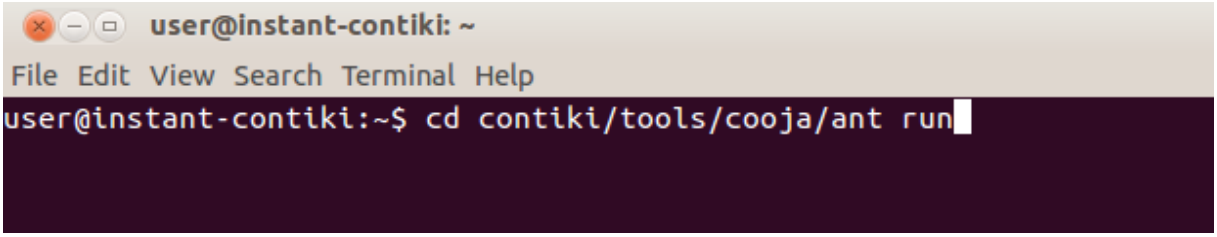

Figure V.9 : lancer Cooja.

Cliquez sur le menu « File » puis cliquez sur « New simulation » pour créer une nouvelle simulation.

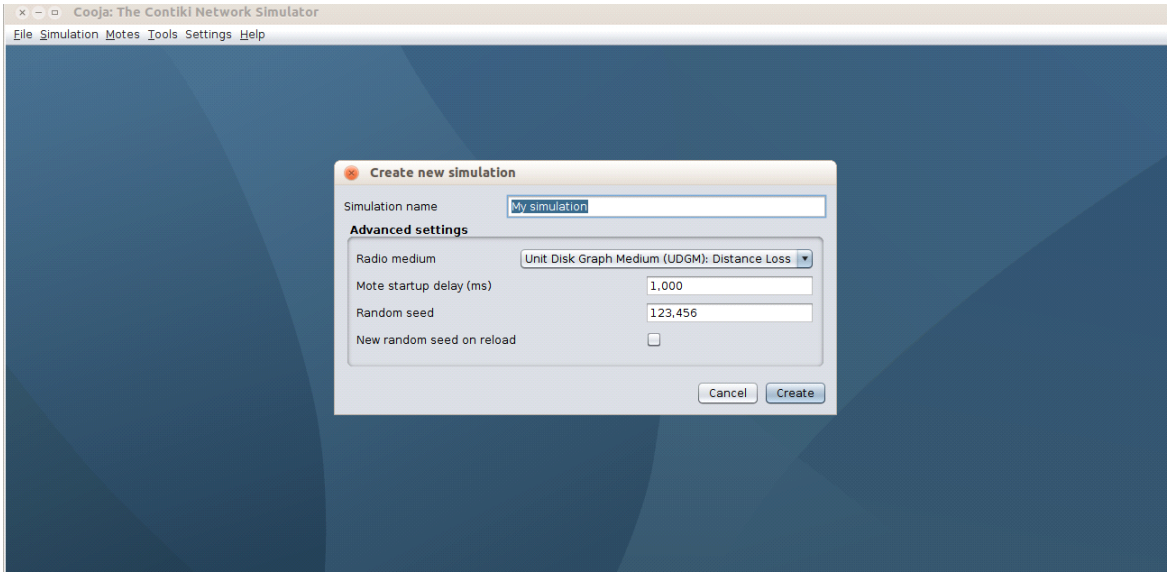

Figure V.10 : Créer une nouvelle simulation.

Cooja ouvre maintenant la boîte de dialogue « Create new simulation » comme l'indique la **Figure V.10** Nous pouvons choisir de donner à notre simulation un nouveau nom, le modèle radio est également choisi parmi une liste

Le modèle radio par défaut est le Unit Disk Graph Medium ou UDGM « UDGM Distance Loss ».

Une fois la configuration terminée on clique sur le bouton « Create ».

Cooja fait apparaître la nouvelle simulation. La fenêtre « Network », en haut à gauche de l'écran, affiche tous les motes dans le réseau simulé (il est vide maintenant, car nous n'avons pas de mote dans notre simulation). La fenêtre « Timeline », en bas de l'écran, affiche tous les événements de communication dans la simulation au fil du temps très pratique pour comprendre ce qui se passe dans le réseau. La fenêtre « Mote Output », sur le côté droit de l'écran, affiche toutes les impressions de port série de tous les motes. La fenêtre « Notes » en haut à droite est l'endroit où nous pouvons mettre des notes pour notre simulation. Et la fenêtre « Simulation Control » est l'endroit où on peut lancer une simulation, la mettre en pause ou bien la charger.

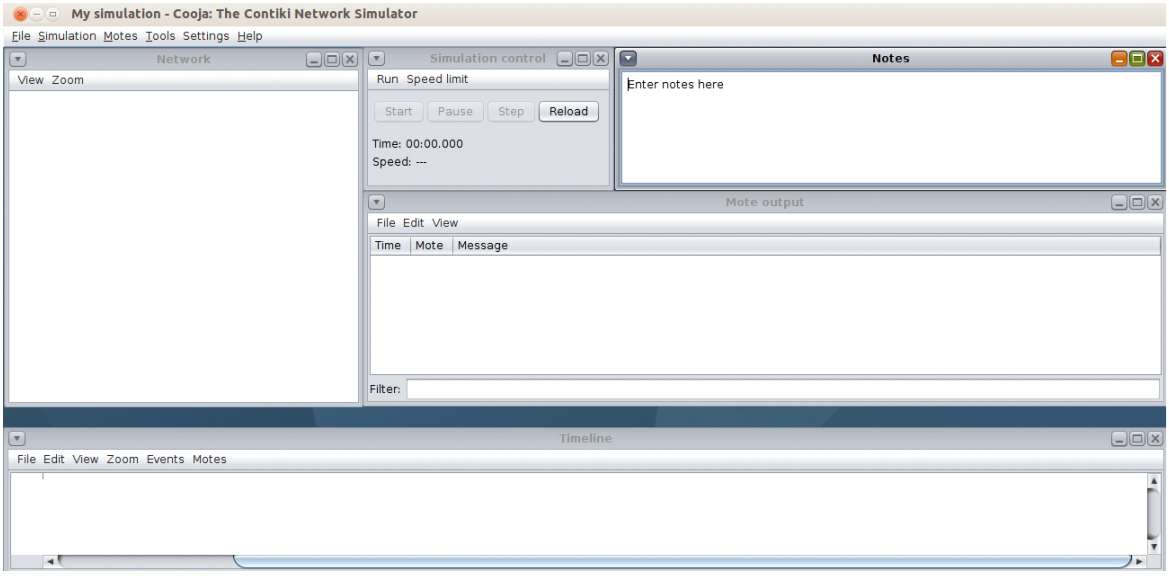

Figure V.11 : la fenêtre principale de Cooja.

Avant de pouvoir simuler notre réseau, nous devons ajouter plusieurs motes. Nous faisons cela via le menu « Motes », en cliquant sur « add motes » .... Puisque c'est la première fois que nous ajoutons, nous devons d'abord créer un type de mote à ajouter. Il faut cliquer sur « Create new mote type », et sélectionner l'un des types de mote disponibles. Pour ce projet, nous choisissons le type « Sky mote » ... qui permet d'émuler le « Tmote Sky mote ».

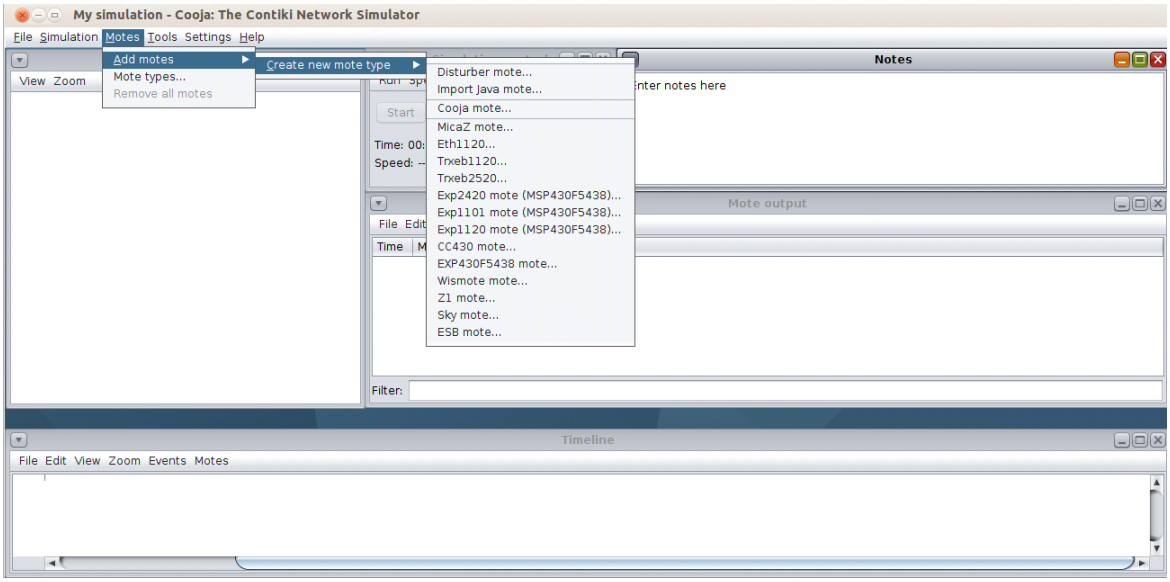

Figure V.12 : Créer un nouveau type de mote.

V.6 Description des scénarios d'évaluation

Nous avons défini trois scénarios pour tester l'impact de l'algorithme Trickle sur les performances de RPL.

Ces scénarios sont dépendants du nombre de messages utiles envoyés par les clients vers le serveur (messages UDP).

Le premier scénario correspond à un grand nombre de messages UDP envoyés puisque l'intervalle entre 2 messages UDP successifs.

Le second scénario contient un nombre moyen de messages UDP

Enfin le troisième scénario contient peu de messages UDP.

Selon ces scénarios, nous allons mener des expérimentations afin d'étudier et d'analyser l'impact de la variation des paramètres de Trickle sur les différents critères que nous avons définis auparavant.

V.7 Configuration des paramètres de simulation

Pour tester notre approche, nous allons choisir l'exemple «/home/user/contiki/examples/ipv6/rpl-udp», le serveur utilise un exemple d'application « udp-server.c » alors que tous les autres nœuds utilisent « udp-client.c », dans cet exemple chaque nœud client envoie un exemple de paquet UDP au serveur après chaque intervalle d'envoi (selon le scénario de simulation) et après un délai de démarrage initial (65s). Le client imprime un message 'send to (id serveur) Hello (séquence de message dans chaque nœud)' dès qu'il envoie un paquet. De même, le serveur imprime un message 'recv Hello (séquence de message dans chaque nœud) from the client (id de client)'. Les messages d'impression sont stockés dans un fichier journal à partir de la simulation.

Chapitre V : Étude expérimentale

Notre expérience se divise en deux phase, dans la premier nous allons fixer le paramètre «RPL_DIO_INTERVAL_DOUBLINGS » sur la valeur par défaut et nous allons choisir les valeurs suivantes 3, 8, 12, 15, 18 pour le paramètre « RPL_DIO_Interval_Minimum », dans la deuxième nous allons fixer le paramètre « Trickle Timer » « RPL_DIO_Interval_Minimum» sur la valeur par défaut et nous allons choisir les valeurs suivantes 3, 8, 16 pour le paramètre « RPL_DIO_INTERVAL_DOUBLINGS »

Avant le lancement de la simulation nous devons configurer certains paramètres du protocole RPL et du simulateur Cooja voir le tableau suivant.

Paramètre	Valeur	Le fichier	Le macros
Intervalle d'envoi	Varie selon les scénarios (1, 4, 10)	udp-client.c	#define PERIOD
Délai de départ	65 s	udp-client.c	#define START_INTERVAL
RX Ratio	40%	UDGM.java	public double SUCCESS_RATIO_RX
TX Ratio	100%	UDGM.java	public double SUCCESS_RATIO_TX
TX Range	50m		
RX Range	55m		
RDC Chanel Check Rate	16	NETSTACK.h	#define NETSTACK_RDC_CHANNEL_CHECK_RATE 16
OF	rpl_mrhop	rpl-conf.h	#define RPL_OF
Métrique	ETX	rpl-conf.h	#define RPL_DAG_MC
Mode d'opération RPL	DOWNWARD_ROUTES		
Client nœud	20		
Temps de simulation	600s (6-40 Min temps réel)		

Table V.1 Configuration des paramètres de simulation.

« Intervalle d'envoi » représente l'intervalle entre deux messages UDP successifs envoie par le même nœud.

Cet intervalle correspond à la valeur 1 pour le premier scénario, à la valeur 4 pour le second scénario et à l'intervalle 10 pour le troisième scénario.

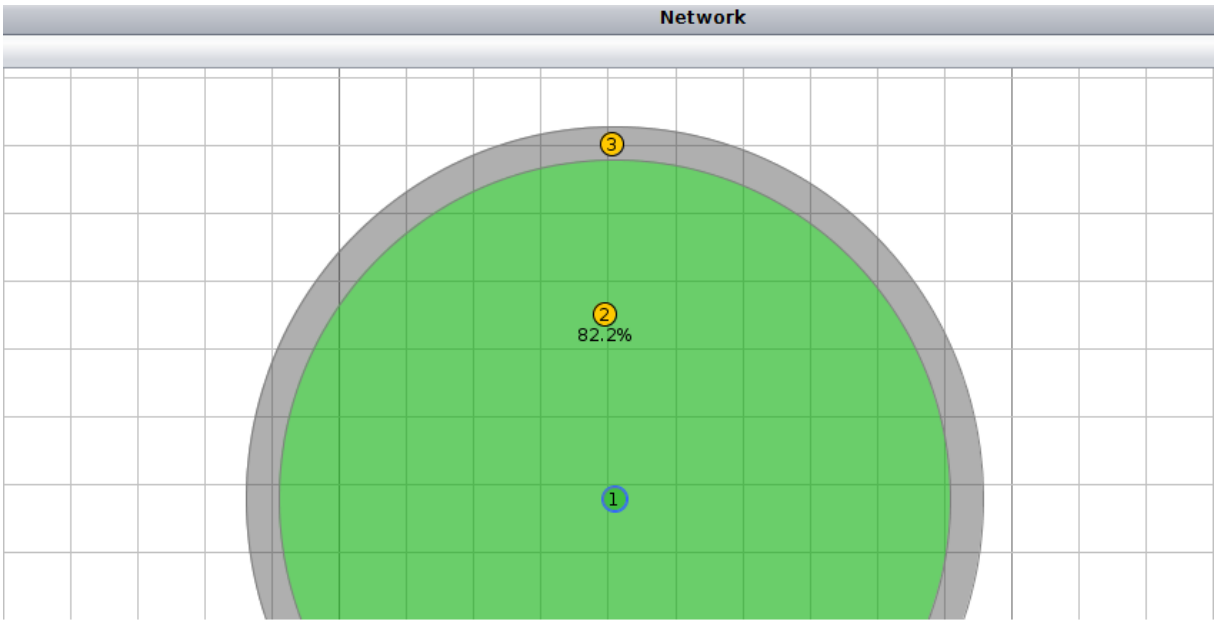

Figure V.13 : Le taux des deux paramètres radio UDGM égale à 82.2%.

Les taux d'émission et de réception peuvent être configurés avec le modèle UDGM. Nous gardons le taux d'émission à 1 (100%) dans ces simulations car nous ne visons pas à introduire des pertes à l'extrémité de l'émetteur mais uniquement à l'extrémité de réception avec un taux de réception de 40%.

La probabilité de succès de la réception des paquets sur un nœud augmente lorsque la distance (D) du nœud diminue vers l'autre nœud dans sa plage de transmission (R), la probabilité minimale de succès serait au bord de la plage de transmission R et égal au ratio RX.

Il y'a deux façon pour modifier les paramètres de transmission :

1) sur interface de Cooja : juste cliquez droit sur la fenêtre « Network » et sélectionné sur le menu déroulant « Change TX/RX success ration »

2) dans le code de cooja : juste aller vers le fichier « UDGM.java » sur le chemin « /home/user/contiki/tools/cooja/java/org/contikios/cooja/radiomediums »

La plage TX est réglée sur 50m voire cercle vert **Figure V.13** et la plage des interférences RX à 55m voire le cercle **Figure V.13**.

La plage TX représente la portée de transmission et RX représente la portée de collision.

Pour modifier le paramètre radio « RDC Chanel Check Rate » aller vers le fichier « NETSTACK.h» sur le chemin « /home/user/contiki/core/net » puis chercher le macros #define « NETSTACK_RDC_CHANNEL_CHECK_RATE 16 »

Pour la fonction objective de protocole RPL, nous allons choisir la fonction MRHOF, en utilisant la métrique ETX pour modifier aller vers le fichier de configuration de RPL « rpl-conf.h » sur le chemin « /home/user/contiki/core/net/rpl », « #define RPL_OF », #define RPL_DAG_MC.

Nous définissons le mode de fonctionnement RPL dans les itinéraires non descendants car nous sommes intéressés à utiliser le trafic multipoint à point pour cette évaluation.

RPL Configuration réseau de 20 nœuds clients et 1 nœuds serveur en tant que racine du DODAG., Le nœud serveur marqué vert, tous les autres nœuds aux cercles orange sont les nœuds clients.

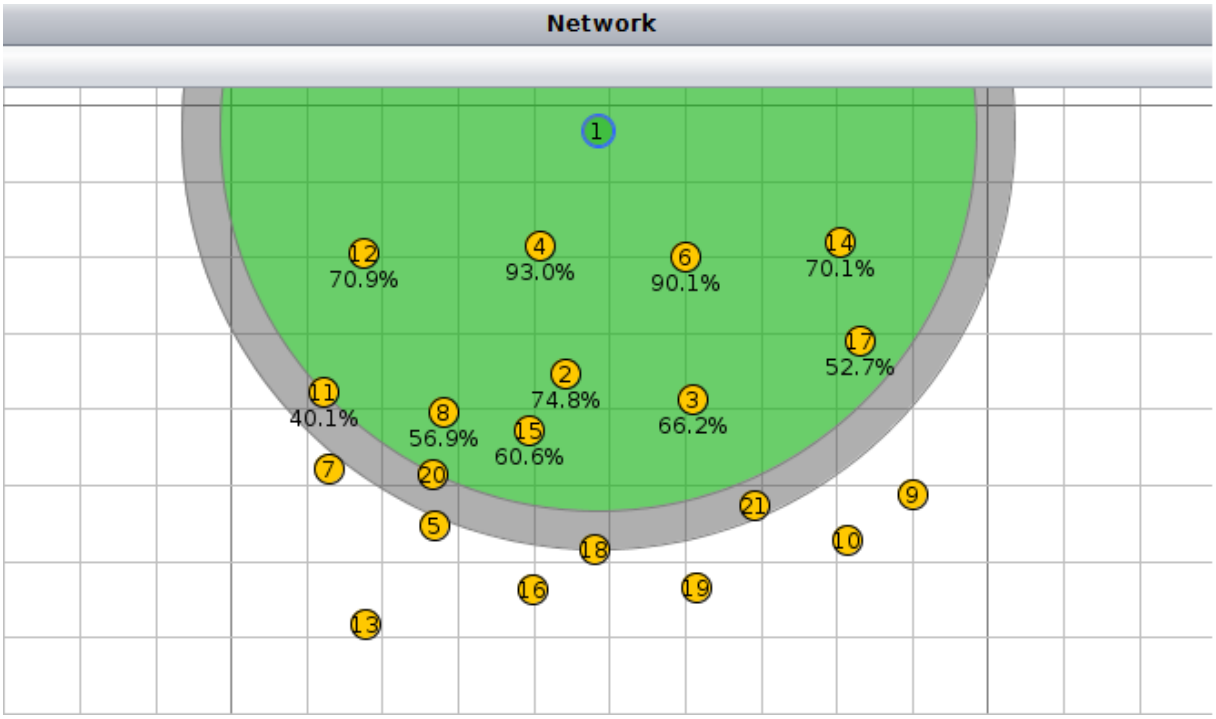

Figure V.14 RPL Configuration réseau de 20 nœuds clients et 1 serveur dans COOJA Simulator,

Nous utilisons « Éditeur de script » pour mesurer le temps de simulation et arrêter la simulation après l'heure spécifiée. Ce plugin crée également un fichier journal (COOJA.testlog) pour toutes les sorties de la simulation que nous analyserons à la fin de la simulation, pour lancer la simulation avec « Éditeur de script r » et en mode non GUI aller dans le menu Éditeur de script, sélectionnez le Menu déroulant « Run » puis sélectionnez « Save simulation and run with script».

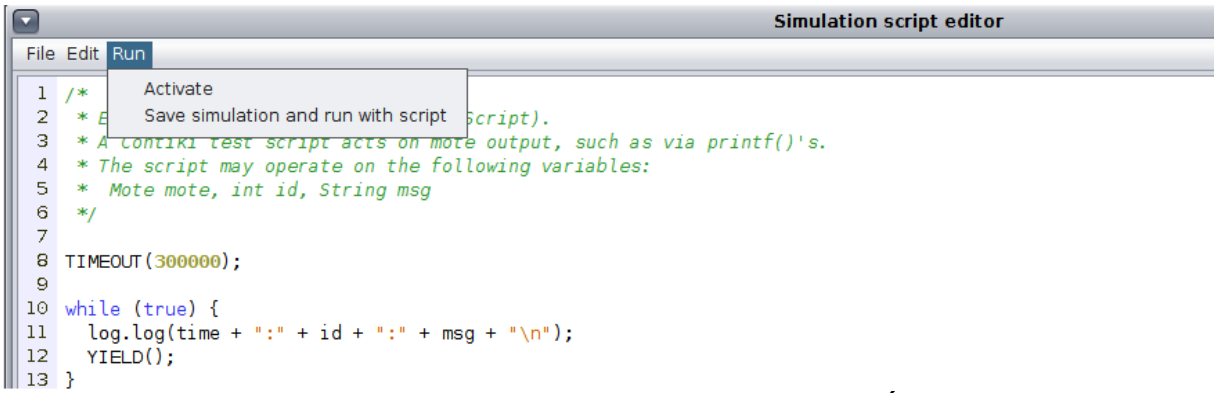

Figure V.15 : Lancer la simulation en mode non GUI avec Éditeur de script.

V.8 Résultats des expérimentations et analyse

Après environ 6-45 Min de simulation nous avons obtenu les résultats suivants pour la 1ère phase et la seconde phase.

V. 8.1 Première phase : variation du Imin

1) Temps de convergence réseaux

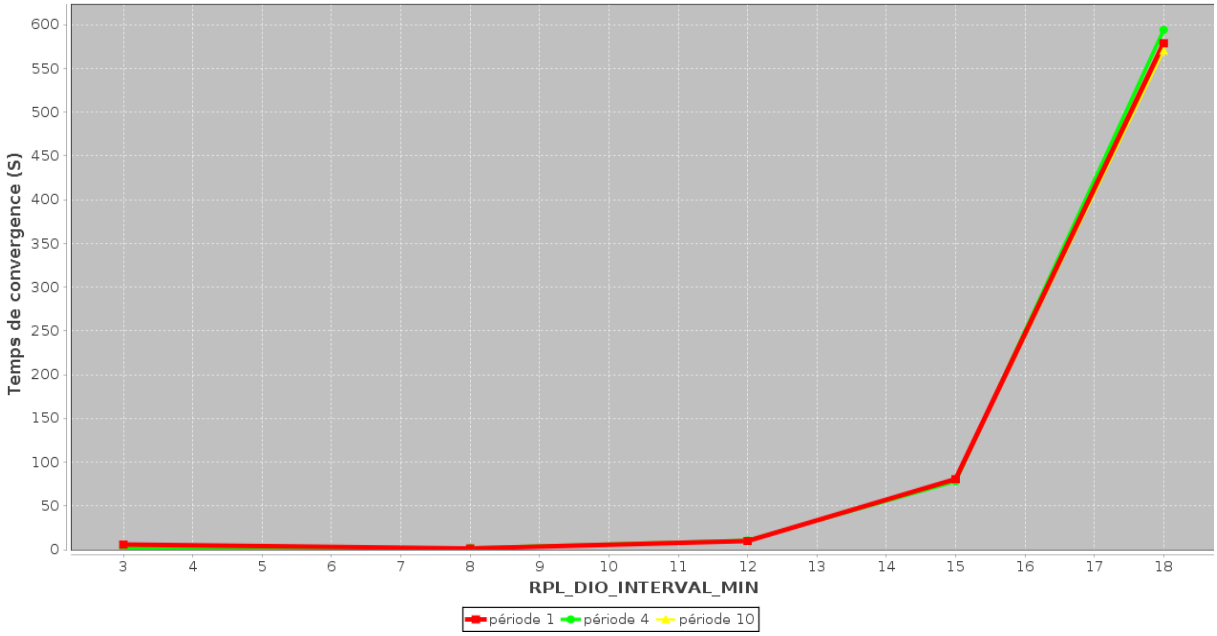

Figure V.16 : les courbes pour le temps de convergence réseaux.

À mesure que nous augmentons l'intervalle RPL_DIO_Interval_Min, le temps de convergence stable également, entre 3, 8, 12, il est approximativement égal.

Pour RPL_DIO_Interval_Min = 15, le réseau converge moins bien avec cette valeur.

Pour RPL_DIO_Interval_Min = 18 et plus, il devient grand et donc inacceptable dans certaines applications.

Les trois courbes des différentes périodes sont égales donc l'augmentation de l'intervalle d'envoie 1, 4, 10 n'a pas influé sur le comportement de ce critère.

Ce comportement représente clairement l'importance du paramètre RPL_DIO_Interval_Min sur la convergence du réseau. Un calibrage approprié de ce paramètre est important pour tout réseau.

Nous en déduisons que les valeurs du RPL_DIO_Interval_Min dans la gamme (3-12) donne les résultats les plus satisfaisant en terme de convergence du réseau.

2) Les frais des messages de contrôle (l'overhead) :

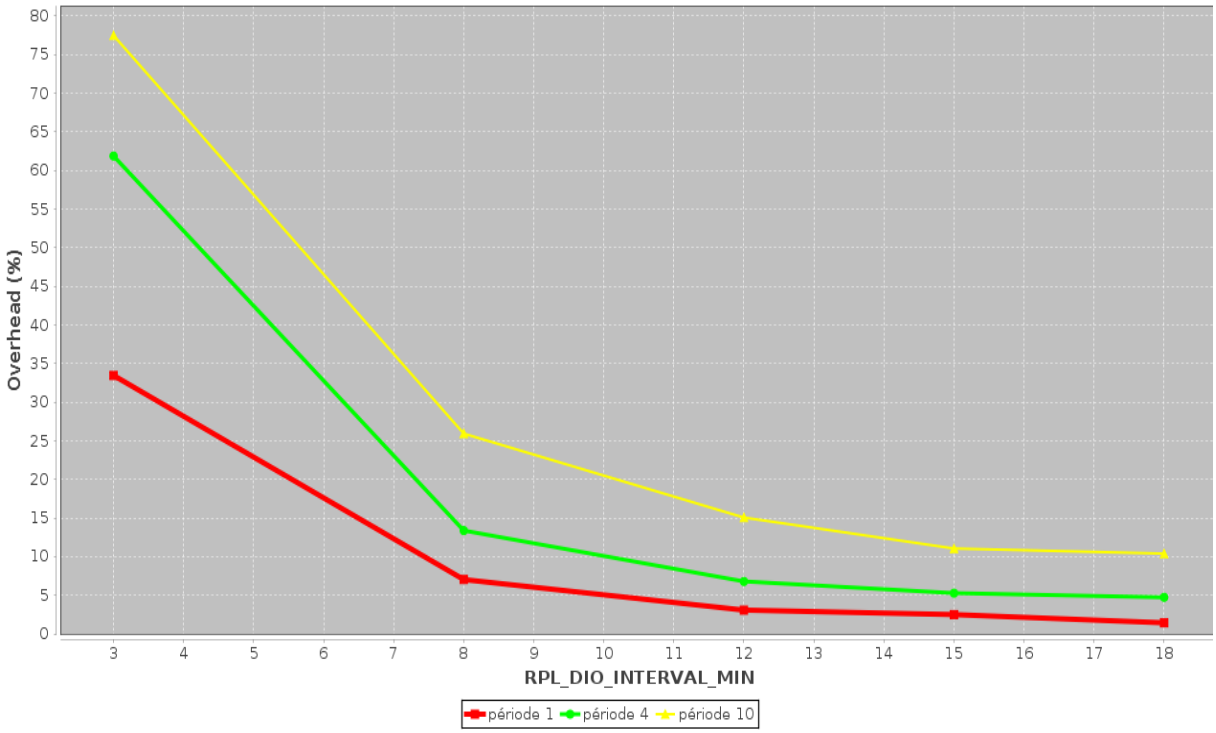

Figure V.17 : les courbes pour les frais des messages de contrôle.

Le ratio de l'overhead est très élevé pour RPL_DIO_Interval_Min inférieur à 8, comme le montre la figure **V.17**, ces valeurs produisent des intervalles Imin inférieurs à 1 seconde qui signifie le temps entre deux DIO donc plus des messages de contrôles. Nous observons également que les frais de contrôle sont inférieurs pour RPL_DIO_Interval_Min supérieur de 8 car le temps entre deux DIO augmente.

Le ratio de l'overhead de la courbe jaune (période 10) est très élevé par rapport aux autres courbes car dans cette période les clients envoient moins de paquet UDP (les clients envoient les paquets UDP chaque 10s tandis que pour la courbe rouge (période 1) les clients envoient les paquets UDP chaque 1s), notez que :

Overhead = nombre des messages de contrôle / (nombre des messages de contrôle + nombre des messages UDP).

Nous en déduisons que les valeurs du RPL_DIO_Interval_Min dans la gamme (8-18) donne les résultats les plus satisfaisant en terme de quantité de trafic de contrôle.

3) Le taux de perte de paquet

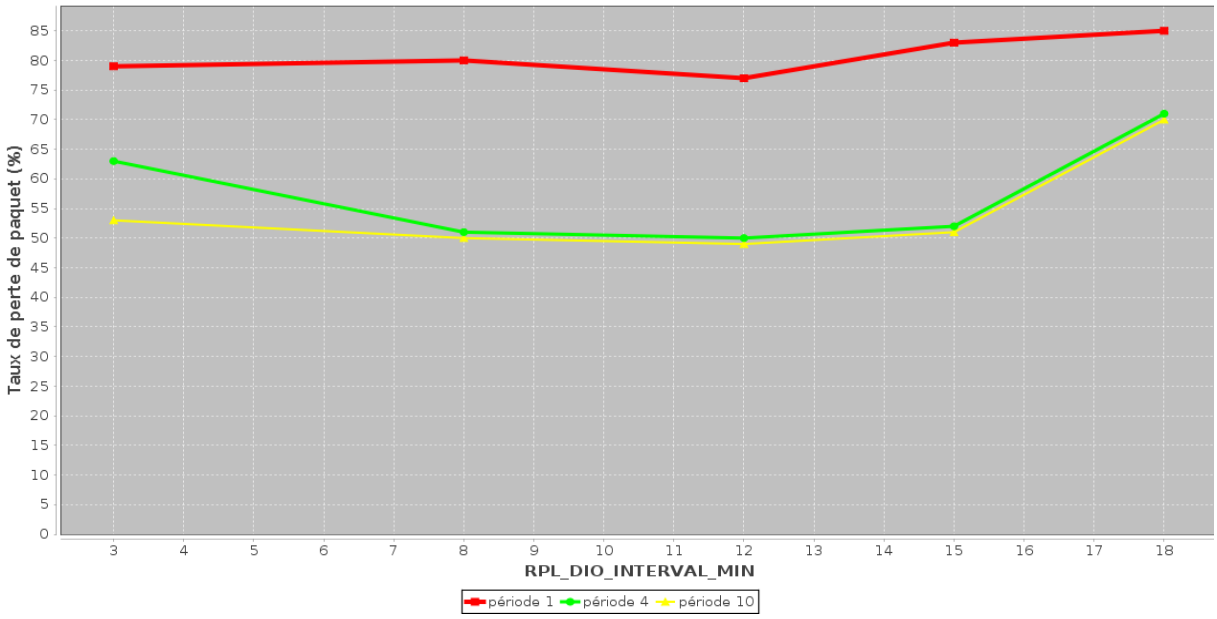

Figure V.18 : les courbe pour le taux de perte de paquet.

Comme le montre la figure V.18, le taux de perte des paquets est très élevé au début pour les trois courbes, pour RPL_DIO_Interval_Min (3-8), ce qui signifie qu'en raison d'une surcharge de contrôle élevée, le réseau RPL subit des collisions surtout pour le scénario 1 où le trafic est dense. Cependant, à mesure que nous augmentons RPL_DIO_Interval_Min (entre 8-12) RPL fournit un bon taux de livraison de paquets. Nous pouvons également observer que le taux de perte des paquets augmente considérablement avec RPL_DIO_Interval_Min de 15 et plus. La raison en est que la valeur de RPL_DIO_Interval_Min supérieur à 12 ne fournit pas une convergence de réseau rapide comme nous pouvons le voir à partir du résultat obtenu dans la figure **V.16**. En conséquence, nous avons une perte de paquets sur certaines des destinations du réseau car le routage ne s'effectue pas correctement.

Le taux de perte de la courbe rouge (période 1) est très élevé par rapport aux autres courbes car il produit plus des paquets UDP donc plus de collisions.

Nous concluons que, pour obtenir un taux de perte faible, les valeurs de RPL_DIO_Interval_Min recommandées se situe entre [8-15].

4) La latence moyenne

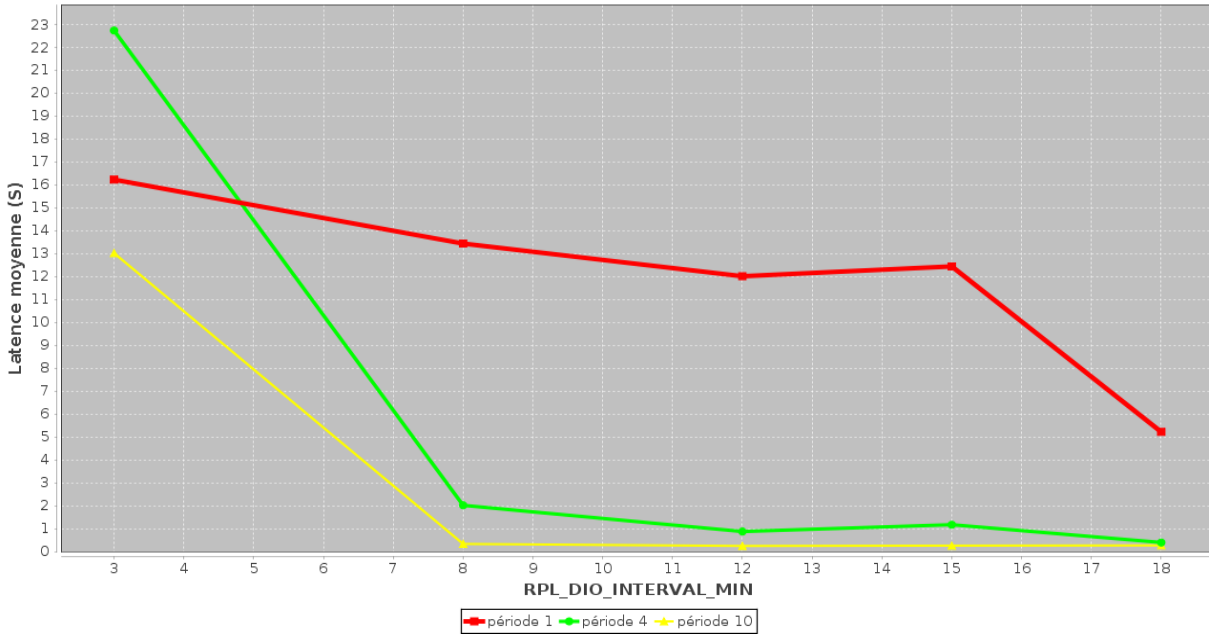

Figure V.19 : les courbes pour La latence moyenne.

La latence moyenne du réseau diminue pour RPL_DIO_Interval_Min entre 3 et 8. Cette diminution soudaine est due à la raison pour laquelle les valeurs inférieures RPL_DIO_Interval_Min ont généré un gros trafic de contrôle comme nous l'avons observé dans la figure V.18 sur le taux de perte de paquet. le trafic de contrôle dense provoque plus de collisions et les paquets doivent être mis en mémoire tampon avant la résolution de la collision radio ce qui augmente la latence du paquet. Dès que le trafic de contrôle diminue pour RPL_DIO_Interval_Min de plus de 8, le tampon de paquets diminue et les collisions diminuent également. Par conséquent, le paquet atteint la destination relativement rapidement qu'avant.

La latence moyenne du réseau de la courbe rouge (période 1) est très élevée dans la plupart des cas par rapport aux autres courbes car il produit plus des paquets UDP donc plus de collisions et plus de stockage de paquets dans les tampons.

Nous concluons que, pour une latence moyenne faible, les valeurs de RPL_DIO_Interval_Min recommandées se situe entre [8-18].

V. 8.2 La deuxième phase : variation du Imax

1) Temps de convergence réseaux

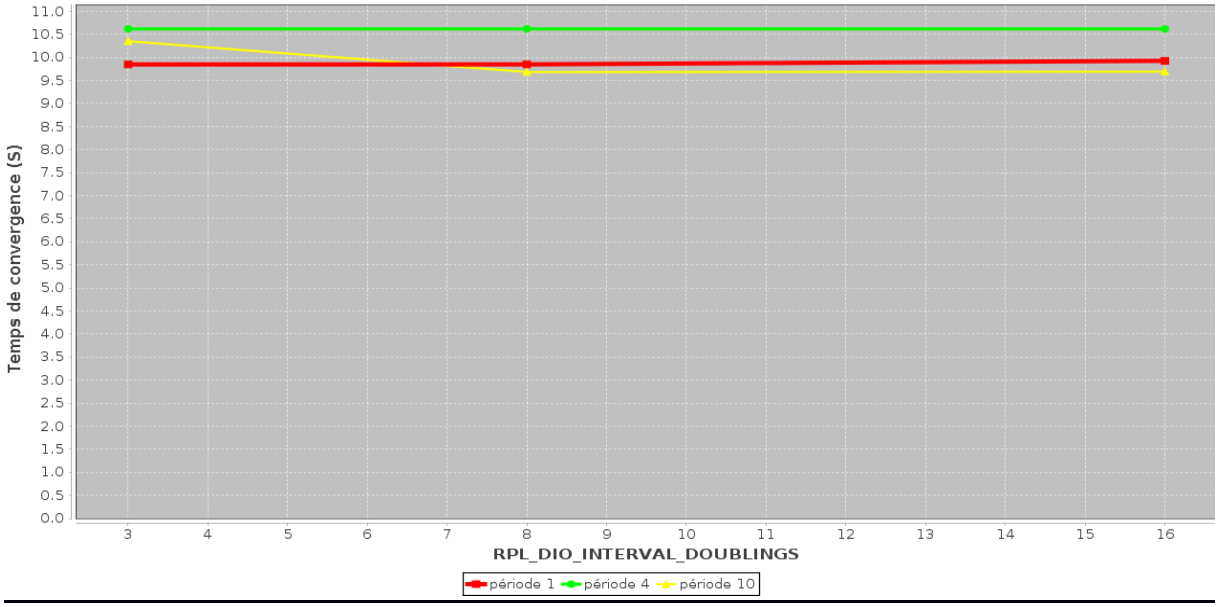

Figure V.20 : **les courbe pour le temps de convergence réseaux.**

Après avoir changé l'intervalle de DIO Doublings 3, 8 et 16, nous observons que le temps de configuration du réseau n'est pas affecté et que le temps de convergence du réseau est stable 9.9 et 10.6 secondes pour les courbes rouge et vert respectivement dans tous les cas et presque stable pour le courbe jaune comme montre la Figure V.20.

Imax = Imin * 2 ^ DIO Doublings

Cela implique que le minimum Imax dans cette simulation sera pour DIO Doublings 3 et calculé comme :

Imax = 4096 * 2 ^ 3 = 32,768s

Ce temps Imax est plus grand que le temps de convergence du réseau de 9.5 à 10.5 comme le montre la Figure donc tous les nœuds rejoignent le réseau avant d'atteindre le minimum de Imax.

2) Les frais des messages de contrôle (l'overhead) :

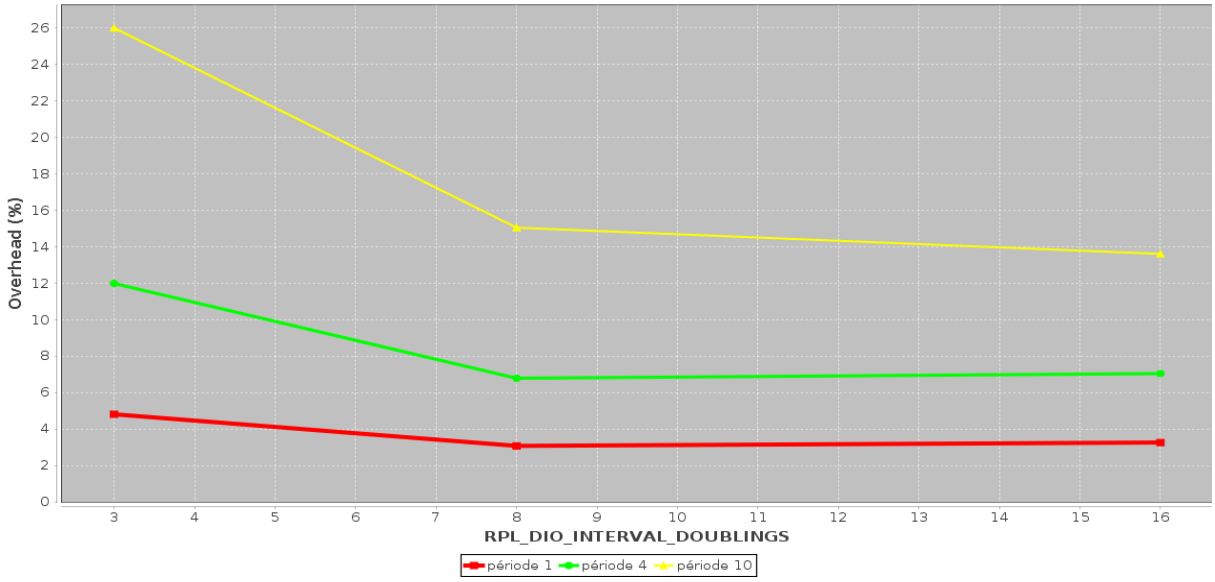

Figure V.21 : les courbes pour frais des messages de contrôle.

Le ratio de l'overhead est très élevé pour RPL_DIO_Interval_Doublings inférieur à 8, comme le montre la figure, ces valeurs produise moins des doublements donc stabilise à des intervalles plus cour, c'est intervalle signifie le temps entre deux DIO donc plus des messages de contrôles.

Le ratio de l'Overhead de la courbe jaune (période 10) est très élevé par rapport à les autres courbes car dans cette période les clients envoient moins de paquet UDP (les clients envoient les paquets UDP chaque 10s tandis que pour la courbe rouge (période 1) les clients envoient les paquets UDP chaque 1s), notez que :

Overhead = nombre des messages de contrôle / (nombre des messages de contrôle + nombre des messages UDP).

Nous en déduisons que les valeurs du RPL_DIO_Interval_ Doublings dans la gamme (8-16) donne les résultats les plus satisfaisant en terme de quantité de trafic de contrôle.

3) Le taux de perte de paquet

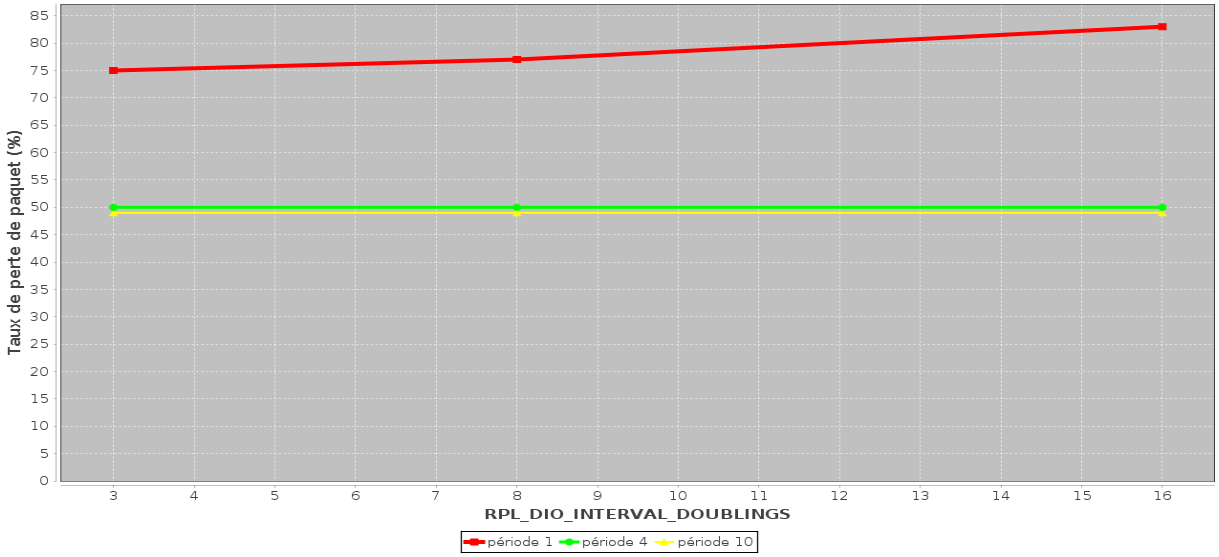

Figure V.22 : **les courbes pour le taux de perte de paquet.**

La ration de perte de paquet et presque stable sur les RPL_DIO_Interval_Doublings 3, 8, 16 pour les trois courbes des différentes périodes donc la variation de paramètre RPL_DIO_Interval_Doublings, n'a pas influé sur le comportement de ce critère.

Pour la variation des périodes d'envoi, le taux de perte du courbe rouge (période 1) est très élevé par rapport aux autres courbes car il produit plus des paquets UDP donc plus de collisions.

4) La latence moyenne

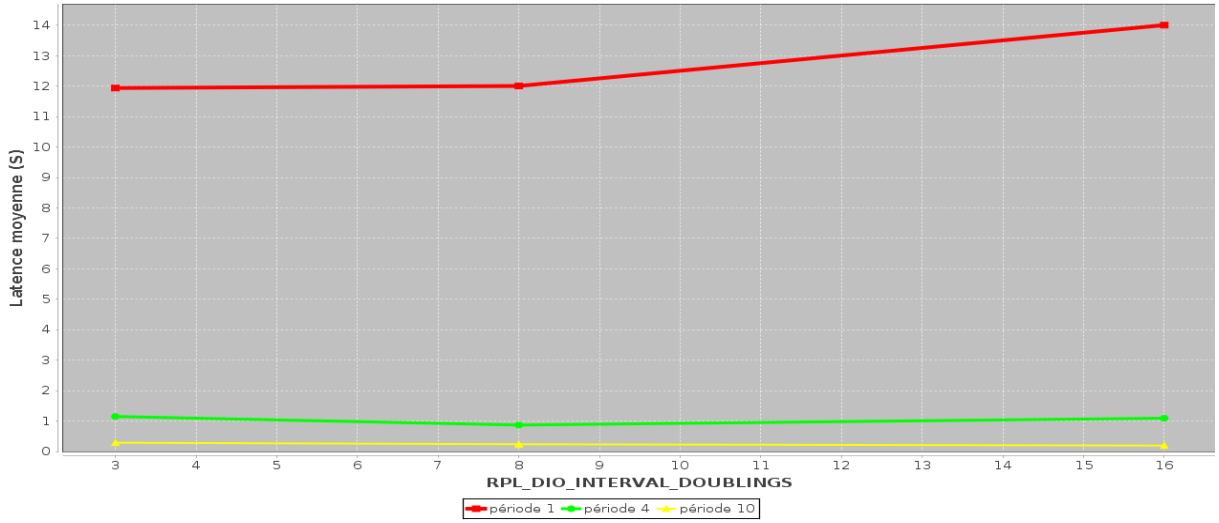

Figure V.23 : les courbes de la latence moyenne.

Après avoir changé l'intervalle de DIO Doublings 3, 8 et 16, nous observons que la latence moyenne reste stable pour les deux courbe jaune et vert et il n'y a pas de changement considérable pour le courbe rouge, différence de deux seconde entre RPL_DIO_Interval_Doublings 3 et 16, donc RPL_DIO_Interval_Doublings n'a pas influé sur le comportement de ce critère.

La latence moyenne du réseau de la courbe rouge (période 1) est très élevée par rapport aux autres courbes car il produit plus des paquets UDP donc plus de collisions.

V.9 Discussion

Pour la première phase :

Ce comportement représente clairement l'importance du paramètre RPL_DIO_Interval_Min sur la convergence du réseau. Un calibrage approprié de ce paramètre est important pour tout réseau, les meilleures valeurs du DIO min Interval se situent entre 3 et 12

Nous en déduisons que les valeurs du RPL_DIO_Interval_Min dans la gamme (8-18) donne les résultats les plus satisfaisant en terme de quantité de trafic de contrôle.

Nous concluons que, pour obtenir un taux de perte faible, les valeurs de RPL_DIO_Interval_Min recommandées se situe entre [8 -15].

La latence moyenne du réseau est inférieure à 0,5 s pour le RPL_DIO_Interval_Min de (8-18), comme le montre la Figure V.19.

Pour la deuxième phase :

Nous en déduisons que les valeurs du RPL_DIO_Interval_ Doublings dans la gamme (8-16) donne les résultats les plus satisfaisant en terme de quantité de trafic de contrôle.

V.10 Conclusion

Dans cette dernière partie de ce projet nous avons abordé la partie mise en œuvre de notre solution en utilisant Cooja comme environnement de développement pour implémenter et observer les causes et les effets de l'approche proposée selon plusieurs scénarios de simulations ensuite nous avons présenté et analysé les résultats obtenus.

Les résultats obtenus montrent que les valeurs de Imin qui satisfont à la fois les 4 critères d'évaluation se situent entre [12-15] pour le scénario 1 avec une forte densité de messages.

Elles se situent entre [8-15] pour les applications qui envoient un nombre relativement moyen de messages.

Elles se situent entre [8-15] pour les applications qui n'envoient que très peu de messages.

Les résultats obtenus montrent que les valeurs de Imax qui satisfont à la fois les 4 critères d'évaluation se situent entre [3-8] pour le scénario 1 avec une forte densité de messages.

Elles se situent entre [8-16] pour les applications qui envoie un nombre de messages moyenne.

Elles se situent entre [8-16] pour les applications qui n'envoient que très peu de messages.

Conclusion générale et perspectives

Dans ce projet, nous avons étudié le routage dans le contexte 6LoWPAN plus particulièrement avec le protocole de routage RPL. Ce protocole a été développé pour répondre aux limites des réseaux LLN telles que la faible puissance de traitement, de batterie et de mémoire [17].

Dans le premier chapitre nous avons introduit le contexte et les concepts clés de notre travail à savoir les réseaux LLN et le protocole 6LoWPAN. Ces réseaux utilisent souvent l'IPv6 comme protocole de réseau qui est caractérisé par une taille de paquet d'au moins 1280 octets.

La problématique c'est que les réseaux LLN s'appuient sur des couches de liaison avec des tailles de trame restreintes comme celles définies dans le standard IEEE 802.15.4, donc L'IETF a créé une couche d'adaptation 6LoWPAN afin de résoudre cette incompatibilité. [2]

Le second chapitre est consacré quant à lui au routage dans les réseaux ad hoc avec les protocoles proposés par le groupe de travail MANET puis nous avons abordé le routage dans les réseaux LLN avant et après l'apparition de ROLL.

Dans la troisième partie de ce projet, nous nous sommes intéressés au protocole RPL et à son fonctionnement et ses caractéristiques afin de mieux cerner certaines de ces limitations. Nous avons également examiné une partie des travaux d'amélioration de RPL que nous avons classé selon plusieurs critères.

Dans la quatrième partie de ce projet nous avons proposé une méthode d'optimisation de RPL permettant d'améliorer la fonction Trickel Timer afin de réduire la surcharge des messages de contrôle tout en assurant le bon fonctionnement de RPL. Notre méthode s'appuie sur une démarche expérimentale visant à analysant le fonctionnement de RPL sur différentes variantes de Trickle et selon différents scénarios.

Dans la dernière partie de ce projet, nous avons abordé la partie mise en œuvre de notre solution en utilisant Cooja comme environnement de développement pour implémenter et observer les causes et les effets de l'approche proposée selon plusieurs scénarios de simulations ensuite nous avons présenté et analysé les résultats obtenus.

Comme perspective de ce travail, nous envisageons de rajouter un autre critère d'évaluation qui est celui de la consommation énergétique afin d'analyser les différentes configurations de Trickle suivant ce critère qui est particulièrement important dans le cadre de l'Internet des objets.

Dans les futurs travaux nous comptons également explorer les effets de la valeur de la constante de redondance (k). Nous avons déjà commencé à tester cela mais les résultats n'étaient pas assez concluants pour tirer de solides conclusions sur l'impact de k dans les différents scénarios.

Installations des outils utilisés

1) La machines virtuelles

La machine virtuelle est un environnement d'application ou de système d'exploitation qui exécute un ou plusieurs systèmes d'exploitation sur le même ordinateur sans redémarrer.

Dans notre projet nous allons utiliser VMware Workstation Player comme une application de virtualisation recommande par la communauté Contiki pour exécuter l'environnement de développement « instant Contiki », nous allons installer VMware Workstation Player sous linux « ubuntu 16.04 » il est aussi possible d'installer sous Microsoft Windows.

VMWare Player il est gratuit de télécharger pour des usages non commerciales et facile à installer, mais nécessite une inscription. Il pourrait être nécessaire de redémarrer votre ordinateur, ce qui est malheureux, mais il est nécessaire de faire fonctionner le réseau.

Voici le lien de téléchargement sur le site officiel. « https://my.vmware.com/en/web/vmware/free#desktop_end_user_computing/vmware_workstation_player/12 »

Pour l'installation sur UNIX, ouvrir un terminal, aller dans le répertoire ou se trouve le fichier avec la commande cd, donner les droits d'exécution au fichier « chmod +x Vmware-Player-12.5.2-4638234.x86_64.bundle », exécuter le fichier « sudo ./Vmware-Player-12.5.2-4638234.x86_64.bundle ».

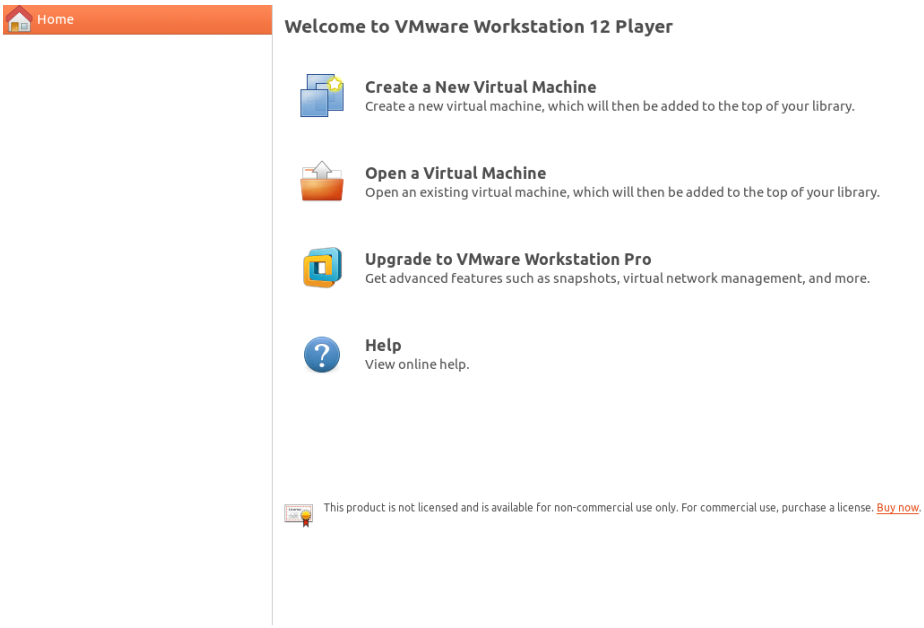

Figure 1 : La fenêtre principale de VMWare Player.

2) Instant Contiki

Il est gratuit de télécharger, c'est un grand fichier, un peu plus d'un 1 gigaoctet. À la fin de téléchargement, décompressez le fichier, il y a plusieurs versions de « instant Contiki » nous avons choisi d'installer la dernier version « Instant Contiki 3 ».

Voici le lien de téléchargement sur le site officiel « https://sourceforge.net/projects/contiki/files/Instant%20Contiki/ »

Après l'installation de VMWare player et la décompression de « Instant Contiki » ouvrir le répertoire ou o n'a décompresser « l'instant Contiki » puis lancer l'installation de l'instant Contiki en cliquant sur le fichier « Instant_Contiki_Ubuntu_12.04_32-bit.vmx ».

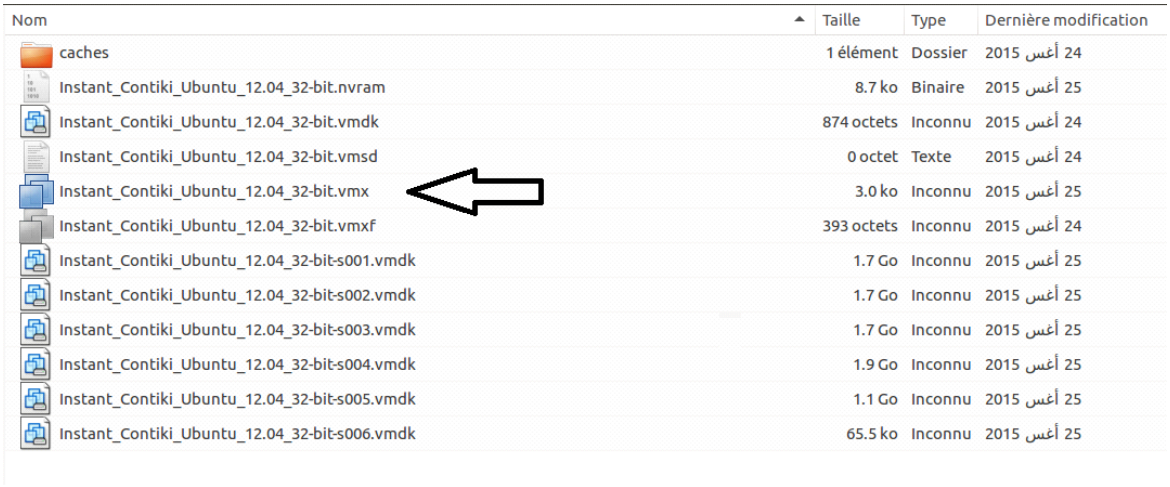

Figure 2 : Installer InstantContiki3.0.

Après la fine de l'installation en lançant VMWare player, puis lancer la machine virtuelle intitulée « InstantContiki3.0 », apparaît à gauche de la fenêtre dans la liste des machines

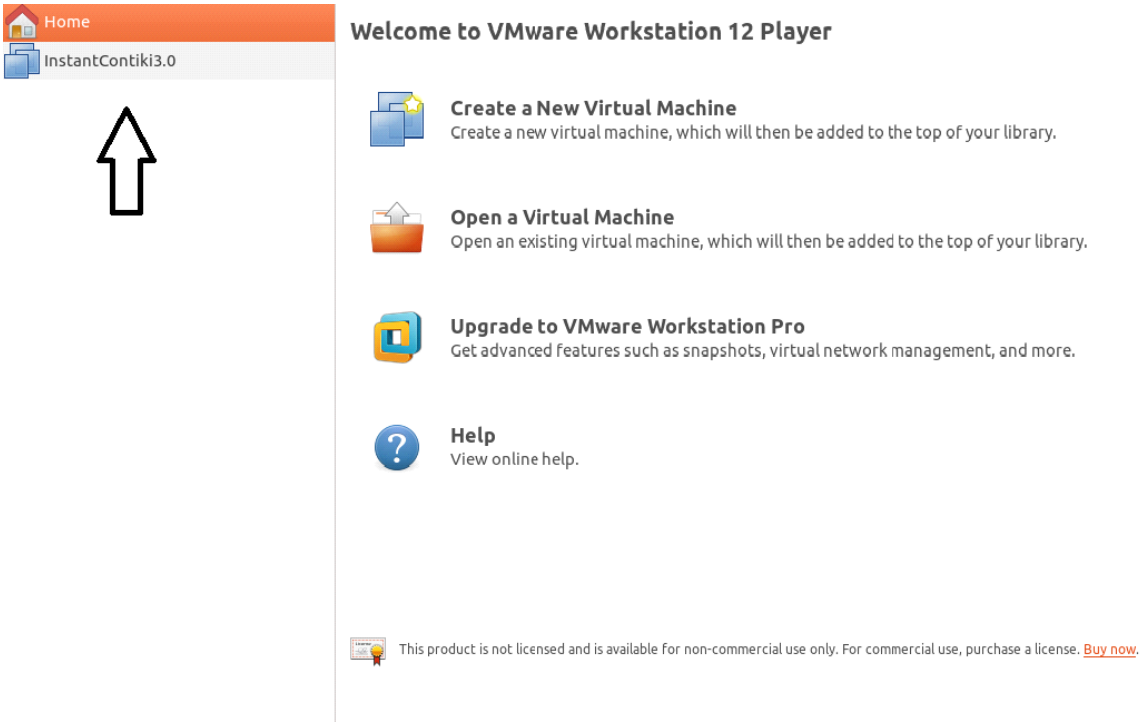

Figure 3 : Lancer Instant Contiki

Virtuelle disponibles, et après la fin de l'installation connectez-vous à « Instant Contiki ». Le mot de passe est « user ».

Et enfin le bureau « d'instant Contiki » s'affiche.

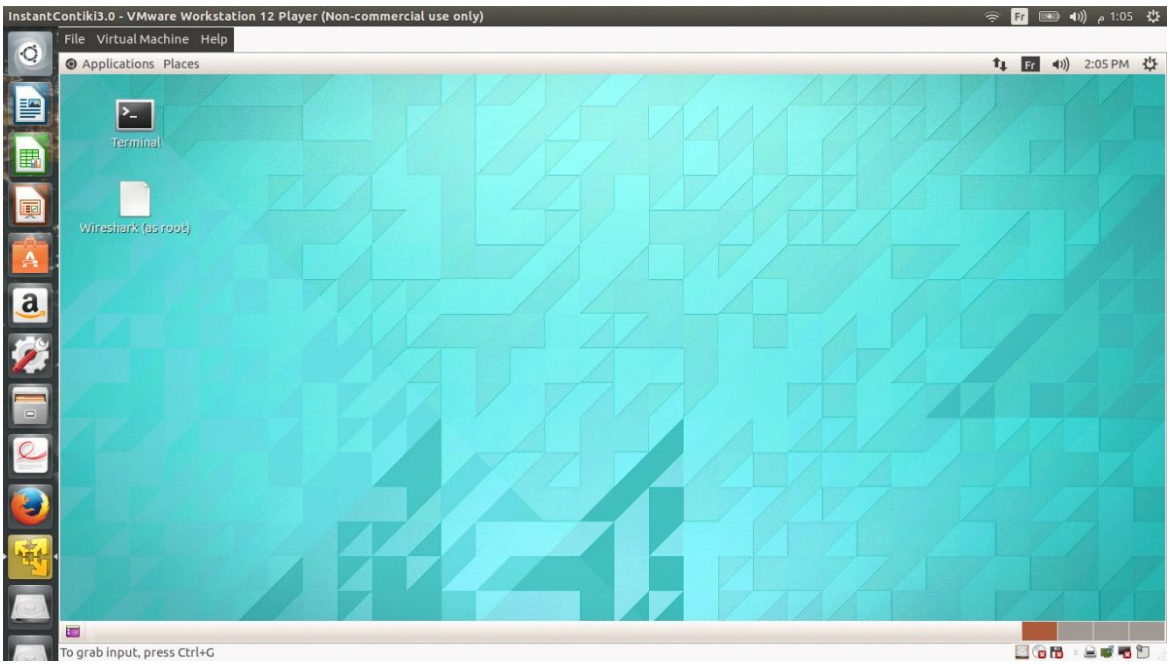

Figure 4 : le bureau « d'instant Contiki ».

Exécuter la commande suivante pour mettre à jour le système.

```
git submodule update –init
```

Bibliographie

[1] RFC 6550, T. Winter et al, IPv6 Routing Protocol for Low-Power and Lossy Networks, March 2012

[2] Jonas Olsson, LoWPAN demystified, Texas Instruments 2014

[3] Pierre-Jean Benghozi, Françoise Massit-Folléa, Sylvain Bureau, L'internet des objets Quels enjeux pour les Européens, Ecole polytechnique et TELECOM Paris Tech. 2008

[4] Gartner http://www.gartner.com/newsroom/id/2636073, 2013

[5] Blog.innovation-artisanat http://blog.innovation-artisanat.fr/linternet-des-objets-au-service-de-la-gestion-de-la-production-et-de-la-maintenance-2/, 2017

[6] Stéphane Lohier, Internet des objets, Lohier@univ-mlv.fr

[7] Alexandre Mouradian, Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil. Réseaux et télécommunications [cs.NI]. INSA de Lyon, 2013.

[8] Pat Kinney, Phil Jamieson, ieee802.org, http://www.ieee802.org/15/pub/TG4.html, 11 May 2017

[9] IEEE Standard for Local and metropolitan area networks, Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), PDF : ISBN 978-0-7381-6684-1 STDPD97126, 16 June 2011

[10] electronicdesign.com, http://electronicdesign.com/what-s-difference-between/what-s-difference-between-ieee-802154-and-zigbee-wireless, Mar 22, 2013

[11] datatracker.ietf.org, https://datatracker.ietf.org/wg/roll/charter/, 2017

[12] Muhammad Ikram, Ki-Hyung Kim, Hassen Redwan, Route-over vs mesh-under routing in 6LoWPAN, Conference Paper, January 2009

[13] Lucien Loiseau, Thèse de Doctorat De l'exploitation des réceptions opportunistes dans les mécanismes de relayage pour les réseaux sans-fil, Université de Rennes, 1 Décembre 2013

[14] Jean Robert HOUNTOMEY, Le Routage Statique AFNOG-NAIROBI–KENYA, 2006

[15]packetlife.net http://packetlife.net/blog/2008/oct/2/distance-vector-versus-link-state/, 2008

[16] Zach Shelby, Carsten Bormann, 6LoWPAN The Wireless Embedded Internet, Universität Bremen TZI Germany, 2009

[17] Aishwarya Parasuram, David Culler Ed, Randy Katz Ed, An Analysis of the RPL Routing Standard for Low Power and Lossy Networks University of California at Berkeley, May 14, 2016

[18] Olfa Gaddour, Anis Koubâa, RPL in a nutshell, Polytechnic Institute of Porto, 02.14.2013

Bibliographie

[19] Mr. Tall HAMADOUN, Réseau maillé sans fil à basse consommation énergétique, Mémoire de Stage de fin d'études Master Informatique, institut de la Francophonie Lyon 1

[20] RFC 6719, O. Gnawali, P. Levis, The Minimum Rank with Hysteresis Objective Function, September 2012

[21] Pietro Gonizzi, Riccardo Monica, Gianluigi Ferrari, Design and Evaluation of a Delay-Efficient RPL Routing Metric, Université of Parma Italie, 2012

[22] P. Karkazis, et al, RPL modeling in J-Sim platform, Ninth International Conference on Networked Sensing Systems, 2012

[23] Olfa Gaddour, Anis Koubâa, Nouha Baccour, Mohamed Abid, OF-FL : QoS-Aware Fuzzy Logic Objective Function for the RPL Routing Protocol, 2014

[24] Simon Duquennoy, Olaf Landsiede, Thiemo Voigt, Let the Tree Bloom : Scalable Opportunistic Routing with ORPL, Sweden, 11–15, 2013

[25] Olaf Landsiedel, Euhanna Ghadimi, Simon Duquennoy, Mikael Johansson, Low Power Low Delay : Opportunistic Routing meets Duty Cycling, Sweden, 16–20, 2012

[26] Baraq Ghaleb, Ahmed Al-Dubai, Elias Ekonomou, E-Trickle: Enhanced Trickle Algorithm for Low-Power and Lossy Networks, Oct.2015

[27] RFC 6206, P. Levis, T. Clausen, J. Hui, O. Gnawali, J. Ko, The Trickle Algorithm, March 2011

[29] http://www.contiki-os.org/, 2017

[30] Edosoft Factory, CONTIKI AND TINYOS, August 13, 2012

[31] Hazrat Ali, A Performance Evaluation of RPL in Contiki, 10-2012

[32] Alberto Camacho Martínez, Implementation and Testing of LOADng: a Routing Protocol for WSN, 2012

[33] Mälardalens Högskola, Optimizing Routing Protocol for Low power and Lossy Network, 07-09-2016

[34] T. Winter (Ed.), P. Thubert (Ed.), and RPL Author Team. RPL: IPv6 Routing Protocol for Low power andLossy Networks. Internet Draft draft-ietf- roll-rpl- 18, July 28, 2010

[35] Jeonggil Ko , Joakim Eriksson , Nicolas Tsiftes , Stephen Dawson-haggerty , Andreas Terzis , Adam Dunkels , David Culler ContikiRPL and TinyRPL: Happy Together, 2011

[36] Fadila Khadar, Contrôle de topologie dans les réseaux de capteurs, 15 décembre 2009

[37] Badis Djamaa and Mark Richardson. "Optimizing the Trickle Algorithm." *IEEE Communications Letters* 19.5 (2015): 819-822.