République Algérienne Démocratique Et Populaire
Université Saad Dahlab Blida1
Faculté des Sciences
Département d'informatique

Thème :

Implémentation et Evaluation de Performances d'un Protocole de Routage Multi chemin dans l'Internet des Objets

En vue d'obtention du diplôme de Master

Domaine : MI

Filière : Informatique

Spécialité : Système informatique et réseaux

Organisme d'accueil : CERIST
Promoteur : Mr. Mohamed OULD-KHAOUA
Encadreur : Mr. Nadir BOUCHAMA

Rapport présenté par :
Amina BOUNAB
Maroua SIAKHENE

Année universitaire : 2018/2019

Remerciement

Nous tenons tout d'abord à remercier Dieu le tout puissant, qui nous a donné la force et la patience d'accomplir ce travail. En second lieu, nous tenons à remercier notre encadreur : **Mr. Nadir BOUCHAMA**, et notre promoteur : **Mr. Mohamed OULD-KHAOUA**, de leurs précieux conseils et leurs aides durant toute la période du travail.

Nos vifs remerciements vont également aux membres du jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail Et de l'enrichir par leurs propositions.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé de près ou de loin à la réalisation de ce travail.

Dédicaces

Nous dédions ce modeste travail :

À nos chers parents pour tous leurs sacrifices, leur amour, leur tendresse, leur soutien et leurs prières tout au long de nos études.

À tous nos collègues surtout : Asma HASNAOUI, Fatima RAHMANI, Mohammed MAHYOUB, Souhila KETTOUCHE qui étaient toujours disposés pour nous prêter main forte.

À toutes nos familles pour leur soutien tout au long de notre parcours universitaire en particulier Abd el malek MOUSSA et Said BENCHEIKHE

À tous ceux qui nous ont soutenues, qu'ils trouvent ici l'expression de notre amour et notre profonde gratitude

Amina /Maroua

Résumé

Contrairement au routage monochemin (*unipath routing*), le routage à chemins multiples (*multipath routing)* consiste à considérer non pas une seule route entre une source et une destination pour la transmission de données, mais un ensemble de chemins. Ces chemins choisis peuvent être soit à « nœuds disjoints » ou à « liens disjoints ». Le routage multi chemin offre plusieurs avantages. Par exemple, il permet une tolérance aux pannes, un équilibrage de charge, et augmentation de la bande passante… etc.

Le but de ce travail est d'implémenter et d'évaluer les performances d'un protocole de routage multichemin pour l'IoT (*Internet of Things*) afin d'améliorer les performances des applications sensibles en terme de délai, débit, et taux de réception.

Les résultats obtenus ont montré que notre solution peut améliorer les performances de LOADng en termes de débit, de taux de réception de paquets…etc.

Mots clés : Internet des Objets, routage multichemin, routage monochemin, LOADng, Cooja/Contiki.

Abstact

In order to transmit data, the multipath routing considers a set of paths between the source and the destination, these chosen paths can be either "disjoint nodes" or "disjoint links". This approach is more advantageous compared to the unipath routing one which considers only a single route between the source and the destination as it allows fault tolerance, load balancing and increased bandwidth.

Through this thesis, we aim at implementing and evaluating the performances of an IOT multipath routing protocol with an attempt to enhance the performance of applications with critical constraints in term of throughput, latency and reception rate.

The obtained results have demonstrated the effectiveness of our solution as it have enhanced the LOADng performances in terms of throughput, latency, reception rate…etc.

Key words: Internet of things, multipath routing, monopath routing, LOADng, Cooja/Contiki.

Table des matières

Liste des figures

Liste des tableaux

Abréviations

AODV	Ad-hoc On demand Distance Vector
AODVM	Ad-hoc On demand Distance Vector multipath
AODVM+	Ad-hoc On-demand Distance Vector Multipath Energy Plus
AOMDV	Ad-hoc On multipath demand Distance Vector
AOMR-LM	Ad-hoc On-Demand Multipath Routing with Lifetime Maximization
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DAO	Destination Avertissement Object
DAO-ACK	Accusé de réception d'Objet d'annonce de destination
DIO	DODAG Information Object
DIS	DODAG Information Sollicitation
DODAG	Destination Oriented Directed Acyclic Graph
DSDV	Destination-Sequenced Distance-Vector Routing
DSR	Dynamic Source Routing
ETT	Expected Transmission Time
ETX	Expected Number of Transmissions
FF-AOMDV	Fitness Function- Ad-hoc On-demand Ad hoc multipath Distance-vector
IEEE	Institute of Electrical and Electronics Engineers
IETF	International Engineering Task Force
IOT	Internet of Things
IP	Internet Protocole
IPV4	Internet Protocole version 4
IPV6	Internet Protocole version 6
ITU	International Telecommunication Union
LLN	Low power and Lossy Networks

LoWPAN Low-Power Wireless Personal Area Networks

6LoWPAN IPv6 over Low Power Wireless Personal Area Networks

LOAD The Lightweight On-demand Ad hoc Distance-vector Routing Protocol

LOADng The Lightweight On-demand Ad hoc Distance-vector Routing Protocol – Next Generation

M2M Machine-To-Machine

MAC Medium Access Control

MANET Mobile Ad Hoc Network

MIT Massachusetts Institute of Technology

MTU Maximum Transmission Unit

NS2 Network Simulator

NS3 Network Simulator

OF Objective Function

OLSR Optimized Link State Routing Protocol

OSPF Open Shortest Path First

RERR Route Error

RFID Radio Frequency Identification

ROLL-WG ROLL Working Group

RPL Routing Protocol for Low Power and Lossy Networks

RREP Route Reply

RREP-ACK Route Reply-acknowledgement

RREQ Route Request

RTOS Real Time Operating System

TLV Type-Length-Value

WMN Wireless Mesh Networks

WSN Wireless Sensor Networks

ZD-AOMDV Zone-Disjoint Ad-hoc On-demand Multi-path Distance Vector

Introduction générale

Internet n'a jamais cessé d'influencer voire de transformer la vie de l'être humain grâce à son évolution permanente. La création des capteurs intelligents, RFID (*Radio Frequency Identification),* les technologies de communication… etc ont induit à l'avènement de internet des objets (IoT) [1].

IoT est un nouveau paradigme et un concept qui permet à n'importe quel objet d'être connecté à internet. Le principe de base est de faire en sorte que ces objets intelligents sont en mesure d'interagir et coopérer les uns avec les autres pour atteindre des objectifs communs. L'IoT a un impact important sur plusieurs aspects de la vie quotidienne, tels que les soins de santé, le transport et les interventions d'urgence…etc.

Généralement dans les réseaux IoT, les périphériques sont contraints en terme de : puissance de traitement, capacité mémoire et énergie. Leurs interconnexions sont caractérisées par des taux de perte élevés, des débits faibles et une instabilité. Ce type de réseau est connu par LLN (*Low Power and Lossy Network*). Un environnement IoT mature implique une transmission de données importante générant des problèmes de goulots d'étranglement et une surcharge sur le réseau. Par conséquent, la communication dans IoT est devenue un défi, soumis aux contraintes citées ci-dessus. Un des défis majeurs dans l'IoT est le routage, qui nécessite des protocoles efficaces, capables de converger rapidement, même dans les très grands réseaux, tout en échangeant un trafic de contrôle limité et nécessitant une mémoire et une puissance de traitement limitées. Cependant, les protocoles de routage traditionnels comme AODV (*Ad-hoc On demand Distance Vector*) [2], OLSR (*Optimized Link State Routing Protocol*) [3], OSPF (*Open Shortest Path First)* [4] ...etc ne sont pas en mesure de satisfaire les exigences spécifiques de LLNs [5].

Dans ce contexte, les chercheurs [6] [7] ont proposé des protocoles de routage tels que RPL (*Routing Protocol for Low Power and Lossy Networks*) et LOADng (*The Lightweight On-demand Ad hoc Distance-vector Routing Protocol – Next Generation)* pour répondre aux exigences des LLNs. Ces protocoles sont conçus sur la base de la stratégie de routage à chemin unique. L'inconvénient de cette stratégie est qu'elle ne prend pas en compte les effets de diverses intensités de charge de trafic et elle a une faible flexibilité contre les défaillances de nœuds ou de liaisons et cela peut réduire considérablement les performances du réseau dans des situations critiques. Afin de faire face aux limitations de cette technique de routage à chemin unique, un

autre type de stratégie de routage, appelée approche de routage multichemin est devenu une technique prometteuse dans les réseaux LLNs.

Dans cette optique, plusieurs travaux se sont focalisés sur l'amélioration de RPL et LAODng. Comme RPL est standardisé par IETF (*International Engineering Task Force*), cela lui a donné le privilège d'être le centre d'intérêt des chercheurs d'où la profusion des protocoles basés sur lui dont les protocoles multichemin [8] [9] [10] [11]. Mais cela n'empêche que RPL ait des problèmes liés à son évolutivité. En particulier dans les réseaux bidirectionnels à grande échelle où il n'accorde pas beaucoup d'attention à l'optimisation du trafic descendant par rapport à celle du trafic ascendant [12]. En parallèle, nous avons constaté que LOADng n'a pas été traité de protocoles basés sur lui mais il vient de regagner l'intérêt grâce à ses caractéristiques qui peuvent répondre à certaines exigences dont le RPL est incapable comme sa capacité de prendre mieux en charge les flux de données bidirectionnels [13]. LOADng présente donc un vaste domaine de recherche [13] [14], c'est pourquoi nous nous sommes intéressées et avons choisi de faire une extension multichemin à ce protocole dans le but d'améliorer les performances (débit, taux de réception, temps de réponse…etc.) dans les réseaux LLNs.

L'organisation du mémoire est comme suit :

Chapitre 1 : est une introduction à l'internet des objets et ses différents concepts, ses principaux éléments ainsi que ses différents types des réseaux et ses domaines d'application.

Chapitre 2 : étale l'état de l'art sur le routage dans les réseaux IoT, plus précisément le routage multichemin, il décrit également quelques protocoles de routage et leurs classifications.

Chapitre 3 : est consacré à la conception de la nouvelle extension multichemin de LOADng.

Chapitre 4 : traite les détails de l'implémentation de la nouvelle extension sous le simulateur Cooja/Contiki.

Chapitre 5 : expose les résultats de simulation et leurs discussions.

Nous terminerons par une conclusion générale et quelques perspectives pour des travaux futurs.

Chapitre 01 : Introduction à l'Internet des Objets

1.1 Introduction

La prochaine vague de l'ère de l'informatique sortira du domaine des ordinateurs de bureau traditionnels et rentre dans le paradigme de l'internet des objets (IoT). Ce paradigme est un concept assez nouveau en champ informatique où un bon nombre d'objets qui nous entourent sera sur le réseau sous une forme ou une autre. Ces dernières années, l'internet des objets [1] a attiré une attention considérable de la part des chercheurs [15] [16] [17] car elle est considérée comme une partie de l'internet du futur qui pourra contribuer à l'amélioration de la qualité de vie de l'être humain.

Dans ce chapitre introductif, nous présentons une introduction à IoT, bien que loin d'être exhaustif il couvre une définition de ce concept, son architecture, les technologies fondamentales et les domaines d'applications de ce dernier.

1.2 L'internet des objets

Le terme Internet des Objets (de l'anglais *Internet of Things* : IoT) a été utilisé pour la première fois par K. Ashton co-fondateur d'Auto-ID Center au MIT (*Massachusetts Institute of Technology*), dans l'une de ses présentations en 1999 [18]. Il y décrit sa vision de la façon dont les objets et les personnes du monde physique peuvent être recensés et gérés informatiquement. IoT fait aujourd'hui référence à la possibilité de connecter tout objet du quotidien à l'Internet traditionnel. Les objets physiques ne sont plus déconnectés du monde virtuel, mais peuvent être contrôlés et surveillés à distance et agir comme des points d'accès à différents services

Le concept IoT a beaucoup évolué grâce aux énormes progrès effectués dans divers domaines technologiques : allant de celui des microcontrôleurs, des capteurs et actionneurs à celui des technologies sans fils. Ces équipements avec leur taille, peuvent être intégrés dans divers objets usuels (lunettes, porte-clés, barquettes alimentaires et jouets), transportés par des personnes (captage des paramètres physiques et surveillance des personnes vulnérables) ou incorporés dans l'environnement (habitat, forêt, centrale nucléaire et chaussée).

L'IoT permet la connectivité réseau entre appareils intelligents à tout moment, partout et à propos de tout. (Voir la figure 1.1).

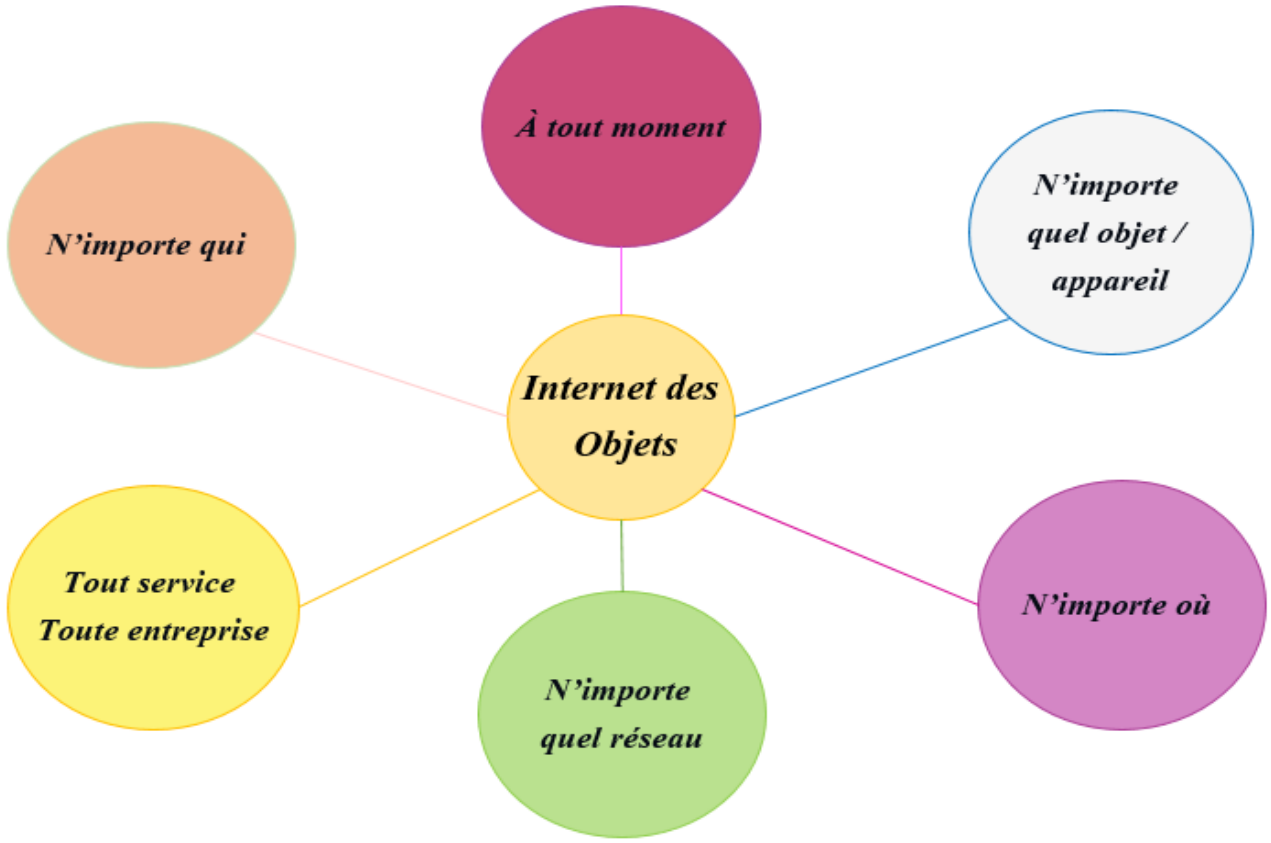


Figure 1.1: Définition de l'Internet des objets [16]

- **Définition de l'objet**

Les *objets* sont des participants actifs permettant d'interagir et de communiquer entre eux en échangeant des données et des informations relatives à l'environnement, tout en réagissant de manière autonome aux événements réels. Ils influencent et exécutent des processus qui déclenchent des actions et créent des services avec ou sans intervention humaine directe [19].

- **L'objectif de l'IoT**

L'un des principaux objectifs de l'IoT est de sensibiliser les utilisateurs à la situation et de permettre aux applications, aux machines et aux utilisateurs de mieux comprendre leurs environnements. La compréhension d'une situation ou d'un contexte permet potentiellement

aux services et aux applications de prendre des décisions intelligentes et de répondre à la dynamique de leurs environnements.

1.3 Les éléments de l'internet des objets

La compréhension des éléments de l'IoT permet d'approcher le sens réel ainsi que la fonctionnalité de ce concept. Ces éléments sont les suivants :

1.3.1 l'identification

L'identification est l'un des éléments les plus importants pour que les systèmes IoT puissent faire correspondre les services à leur demande et elle est utilisée pour fournir une identité claire et unique pour chaque objet appartenant à un réseau. Plusieurs méthodes d'identification sont utilisées dans l'IoT, tels que les codes de produits électroniques (EPC) et les codes omniprésents (uCode). De plus, les méthodes d'adressage incluent IPv6 (*Internet Protocole version 6*) et IPv4 (*Internet Protocole version 4*) [17].

1.3.2 La détection

La détection consiste à surveiller, contrôler l'environnement et collecter des données à partir d'objets associés au sein du réseau et à les envoyer à un système distribué ou centralisé pour l'analyser. Ces objets associés peuvent être des capteurs, RFID tag, des dispositifs de détection portables... etc. Ces derniers ne sont pas tous similaires en termes de portée et de volume de données à transférer. Ils doivent être capables de communiquer entre eux pour transmettre des données à une station de base et ils doivent être capables de communiquer dans des environnements difficiles. La détection peut être biométrique, biologique, environnementale, visuelle ou audible [17].

1.3.3 La communication

Après la détection des informations à partir des capteurs et pour les transmettre à la plateforme IoT (l'entrepôt de données, une base de données ou un cloud), Les appareils IoT nécessitent un moyen de communication en vue d'un traitement ultérieur [17]. Les technologies de communication IoT connectent des objets hétérogènes pour fournir des services intelligents spécifiques. Parmi les protocoles de communication utilisés :

- ✓ **IEEE 802.11 (*Wi-Fi*)**

  Cette technologie utilise les ondes radio pour échanger des données entre des objets situés dans un rayon de 50 m. Elle est prévue pour transférer de gros débits. Il existe de nombreuses normes dérivées de celle-ci. Les deux dérivées les plus connues sont la norme

802.11b qui offre un débit de 11 Mbit/s dans la bande de fréquence de 2,4 GHz et la norme IEEE 802.11a qui offre un débit de 54 Mbit/s dans la bande de fréquence de 5,3 GHz [20].

- ✓ **IEEE 802.15.1 (*Bluetooth*)**

Cette dernière présente une technologie de communication utilisée pour échanger des données entre des périphériques sur de courtes distances en utilisant une radio à courte longueur d'onde afin de minimiser la consommation d'énergie et permettant un débit d'environ 1 Mbit/s [20].

- ✓ **3G/4G/5G**

Les technologies qui désignent les générations de normes de téléphonie mobile. Cette technologie vise à faciliter les usages interactifs sur les terminaux numériques mobiles comme les smartphones, les tablettes…etc. En permettant la transmission simultanée de la voix et de paquets de données. 3G/4G/5G ont une plus grande portée, mais demandent beaucoup d'énergie [21].

- ✓ **IEEE 802.15.4 (*Zigbee*)**

C'est le standard de communication dominant dans IoT. Il spécifie à la fois une couche physique et une couche liaison de données pour les réseaux sans fil à faible puissance, ciblant des communications fiables et évolutives. Cette norme est maintenue par le groupe 4 du comité IEEE 802.15. La première version de ce standard a été publiée en 2003 suivie d'une révision en 2006. La conception de ce dernier a été réalisée pour permettre des communications de faibles portées en supportant des faibles débits 20 kbit/s, 40 kbit/s et 250 kbit/s. Au niveau de la couche physique, le dispositif matériel est constitué d'un émetteur/récepteur de radiofréquences et intègre les mécanismes de contrôle bas niveau tels que le contrôle de la qualité du signal et la détection de l'activité sur le canal où la gestion de la consommation énergétique dans ce niveau est optimisée. La couche liaison de données gère la transmission des trames de données sur le médium physique et le contrôle d'accès à ce support. Elle peut fonctionner selon deux modes : un mode asynchrone et un mode synchrone. Dans le premier mode, les nœuds utilisent le mécanisme de CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) pour

envoyer leurs trames. Dans le mode synchrone, un échange des balises contenant des informations pour synchroniser les nœuds [22].

- ✓ **RFID (*Radio Frequency Identification*)**

La RFID est la première technologie [23] utilisée pour réaliser le concept M2M (Machine-to-Machine). Il s'agit d'une technologie permettant de mémoriser et de récupérer à distance des données. L'objectif initial étant celui d'identifier et de réaliser un suivi minutieux des biens. Un système RFID assurera deux fonctions basiques pour l'Internet des Objets qui sont l'identification et la communication.

Un système d'identification par radiofréquence est constitué de :

- **RFID tag :** c'est un circuit intégré mémorisant l'information sur l'objet auquel la puce est incorporée. Il est muni d'une antenne pour la réception/transmission des signaux. L'étiquette RFID représente une simple puce ou une étiquette attachée pour fournir l'identité de l'objet. Les étiquettes RFID peuvent être actives, passives ou semi-passives / actives. Les tags actifs sont alimentés par batterie alors que les tags passifs n'ont pas besoin de batterie. Les balises semi-passives / actives utilisent l'alimentation de la carte au besoin.
- **RFID reading :** est utilisé pour envoyer le signal radio à la puce RFID, capturer la réponse de cette dernière et transmettre le signal à la base de données. Son principe de fonctionnement général est le suivant : RFID reading initie la communication en diffusant une requête. Les radio-étiquettes du voisinage répondent à ce dernier en fournissant leur identifiant et leurs données stockée.

1.3.4 La collection, l'analyse, la gestion et le développement des applications

La plateforme IoT permet de recevoir, collecter, analyser et stocker les résultats dans la base de données**.** Les unités de traitement (les microcontrôleurs, les microprocesseurs), les applications logicielles représentent le cerveau et la capacité de calcul de l'IoT, il existe diverses plates-formes IoT qui interagissent entre elles pour délivrer un service digital à l'utilisateur final [17] :

- ✓ Les plates-formes matérielles ont été développées pour exécuter des applications IOT comme Intel Galileo, Arduino [24], Raspberry PI [24].

- ✓ Les plates-formes logicielles sont utilisées pour fournir des fonctionnalités IoT et embarquent une intelligence qui offre de multiples possibilités. Parmi ces plates-formes, les systèmes d'exploitation, car ils fonctionnent pendant toute la durée d'activation d'un périphérique. Plusieurs systèmes d'exploitation temps réel (RTOS) (Real Time Operting System) sont de bons candidats pour le développement d'applications IoT. Par exemple, le RTOS Contiki [25] a été largement utilisé dans les scénarios IoT.
- ✓ Les plates-formes en nuage constituent une autre partie informatique importante de l'IoT. Ces plates-formes fournissent des installations permettant aux objets intelligents d'envoyer leurs données dans le cloud, au traitement des métadonnées en temps réel et éventuellement aux utilisateurs finaux de bénéficier des connaissances extraites des métadonnées collectées.

Le tableau 1.1 ci-dessous donne un résumé des éléments IoT ainsi que leurs différentes technologies.

Les éléments du IoT		**Exemple**
Identification		UPC, uCode IPV4, IPV6 , Rime
Détection		Capteur inteligent, dispositifs de détéction,, RFID tag, actionneurs ...
Communication		RFID, Bluetooth, Z-Wave, Wi-Fi, 802.15.4, 3G/4G/5G…
Plateforme	**Logicielle**	OS (Contiki, TinyOS, Android) Cloud …..
	Matérielle	Téléphones intelligents, Arduino, Raspberry Pi …….

Tableau 1.1: les téchnologies des éléments du IOT [17]

1.4 L'architecture de l'internet des objets

IoT possède ses propres attributs tels que la complexité technique et le style d'application, où l'architecture Internet traditionnelle doit être révisée pour répondre aux défis de l'IoT. L'IoT est un système composé de nombreuses fonctions incluant la perception, le traitement, la transmission, la décision et la fourniture de services. Cette dernière exige une architecture différente à l'architecture de l'internet traditionnel. L'architecture IoT varie d'une solution à

l'autre, en fonction du type de solution que nous avons l'intention de construire, où tous les projets n'adoptent pas une architecture formellement identique [26]. La Figure1.2 montre une des architecture IoT qui est composée principalement de trois couches : perception, réseau et application.

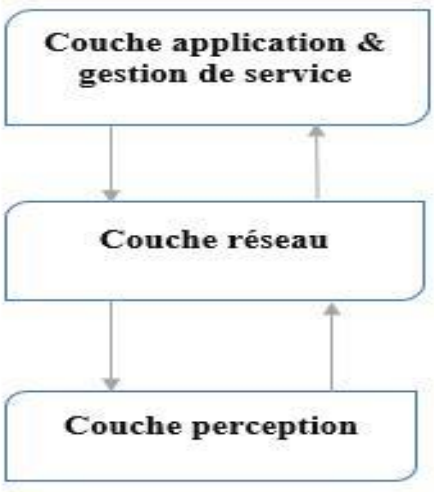


Figure 1.2 : Une architecture IoT [26]

1.4.1 La couche Perception

Cette couche représente les capteurs physiques de l'IoT avec différents types et états qui permettent d'exécuter différentes fonctionnalités où la fonction principale consiste à échantillonner collecter et à traiter des données relatives à l'environnement tels que l'interrogation de l'emplacement, la température, le poids, le mouvement, les vibrations, l'accélération, l'humidité, etc. Elle traite les données avec un moyen d'accès coopératif afin d'obtenir des informations utiles, puis de les transmettre à la couche réseau via les dispositifs d'accès.

1.4.2 La couche réseau

La fonction principale de cette couche est de fusionner les données produites par la couche perception et de les transmettre aux plates-formes correspondantes de la couche supérieure via des canaux sécurisés. Les données peuvent être transférées via diverses technologies tels que RFID, 3G, Wi-Fi, Bluetooth, ZigBee, Internet, un réseau de communication mobile, un réseau sans fil…etc.

1.4.3 La couche d'application et de service

Cette couche est divisée en deux sous-couches. L'une est la sous-couche de service, l'autre est la sous-couche d'application :

1.4.3.1 La sous-couche de service (Middleware)

Cette couche permet aux programmeurs d'applications IoT de travailler avec des objets hétérogènes sans tenir compte d'une plate-forme matérielle spécifique, associe un service à son demandeur en fonction des adresses et des noms. C'est une plate-forme de base fiable et crédible pour prendre en charge les applications de l'industrie qui fournissent de nombreux services.

1.4.3.2 La sous-couche d'application

Cette couche intègre toutes les fonctions nécessaires des couches inférieures et fournit un service spécifique à toutes les industries. L'importance de cette couche pour l'IoT réside dans sa capacité à fournir des services intelligents de haute qualité pour répondre aux besoins des clients. Par exemple, la couche d'application peut fournir des mesures de température et d'humidité de l'air au client qui demande ces données. Cette couche intègre les différents domaines d'application et couvre de nombreux marchés verticaux tels que la maison intelligente, la construction intelligente, les transports, l'automatisation industrielle et les soins de santé intelligents où cette dernière est l'interface par laquelle les utilisateurs finaux peuvent interagir avec un périphérique et interroger des données intéressantes.

La figure 1.3 donne un exemple des outils et des technologies utilisés dans chaque couche de l'architecture IoT.

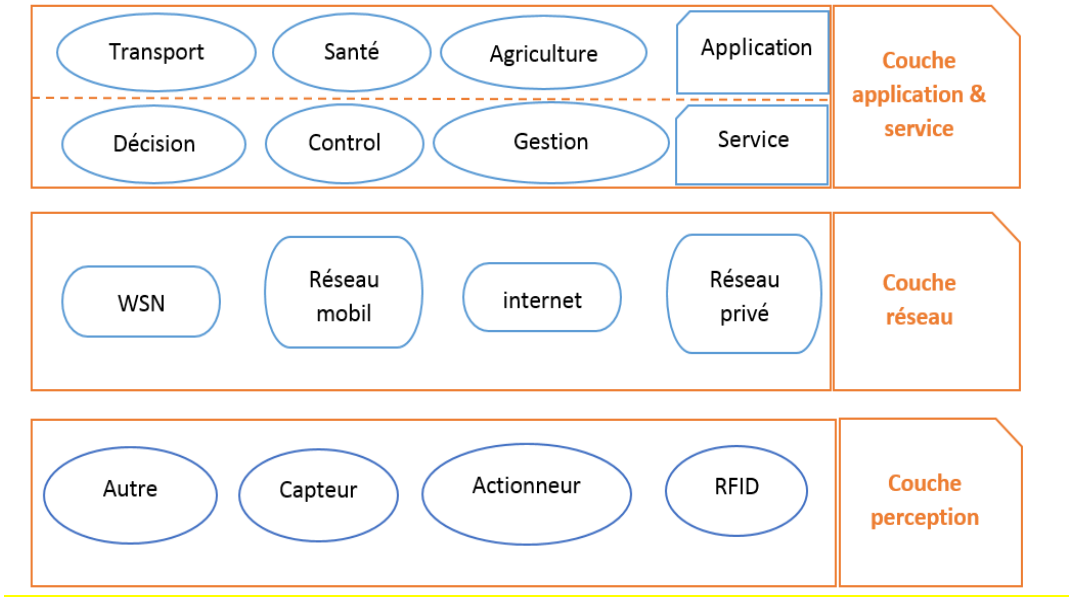


Figure 1.3 : Les outils et les technologies d'une architecture IoT [26]

1.5 Les réseaux IoT

L'internet des objets englobe plusieurs types des réseaux dont LLN, WSN (*Wireless Sensor Networks*), WMN (*Wireless Mesh Networks*) qui sont traités dans ce qui suit :

- **Les réseaux à faible consommation et à perte *(LLN)***

  Les organismes de normalisation ont introduit le terme de réseaux à faible consommation et à perte (LLN) pour désigner une classe de réseaux câblés et sans fil où les hôtes sont étroitement limités en ressources et en technologies de communication [40]. Bien que les limitations de ressources incluent des réserves de puissance et des capacités de traitement et de stockage limitées, les technologies de communication sous-jacentes peuvent présenter des débits binaires faibles, des caractéristiques de liaison fortement asymétriques, une perte de données importante et une variabilité élevée de la perte de données (liaisons avec perte) et de courtes distances de communication. Les hôtes LLN présentent généralement des caractéristiques similaires. Cependant, des différences peuvent exister entre les capacités de calcul et de stockage des nœuds.

  Les domaines d'application des LLN sont variés : surveillance industrielle, automatisation du bâtiment (ventilation et climatisation, éclairage, contrôle d'accès, incendie), maison connectée, soins de santé, surveillance de l'environnement, etc.

- **Les réseaux de capteurs sans fils *(WSN)***

Les réseaux WSN consistent en un certain nombre de nœuds de capteurs (quelques dizaines à des milliers), travaillant ensemble pour surveiller une région afin d'obtenir des données sur l'environnement où sont déployés de manière très dense à l'intérieur ou à proximité du phénomène. La position des nœuds de capteurs n'a pas besoin d'être conçue ni prédéterminée cela permet un déploiement aléatoire sur des terrains inaccessibles ou des opérations de secours en cas de catastrophe. D'autre part, cela signifie que les protocoles et les algorithmes de réseau de capteurs doivent posséder des capacités d'auto-organisation. Une autre caractéristique unique des réseaux de capteurs est l'effort de coopération des nœuds de capteurs. Les nœuds de capteurs sont équipés d'un processeur intégré. Au lieu d'envoyer les données brutes aux nœuds responsables de la fusion, les nœuds capteurs utilisent leurs capacités de traitement pour effectuer localement des calculs simples et ne transmettre que les données requises et partiellement traitées [27].

Un WSN a ses propres contraintes de conception et de ressources. Les contraintes de ressources incluent une quantité d'énergie limitée, une faible portée de communication, une faible bande passante et un traitement et un stockage limités dans chaque nœud. Les obstacles dans l'environnement peuvent également limiter la communication entre les nœuds, ce qui affecte la connectivité du réseau [27] [28].

Les WSN offrent un grand potentiel pour de nombreuses applications, tels que le suivi et la surveillance de cibles militaires, les secours en cas de catastrophe naturelle, la surveillance de la santé biomédicale, l'exploration d'environnements dangereux et la détection sismique.

- **Les réseaux mailles *(WMN)***

WMN sont des réseaux sans fil auto-organisés et auto-configurés de manière dynamique, les nœuds du réseau établissent automatiquement un réseau ad hoc. Ils sont composés de deux types de nœuds : les routeurs maillés et les clients maillés, où un routeur maillé contient des fonctions de routage supplémentaires pour prendre en charge le réseau maillé et transfère le trafic vers et depuis les passerelles qui peuvent mais ne doivent pas nécessairement être connectées à Internet. Lorsqu'un nœud ne peut plus fonctionner, les autres nœuds peuvent toujours communiquer entre eux, directement ou par le biais d'un ou de plusieurs nœuds intermédiaires. Les réseaux maillés sans fil

peuvent se former et se soigner eux-mêmes, ils subissent des modifications rapides. Cette fonctionnalité apporte aux WMN de nombreux avantages, tels que des coûts initiaux bas, une maintenance facile du réseau, la robustesse, une couverture de service fiable…etc. WMN se trouve dans diverses applications tels que les réseaux domestiques à large bande, l'automatisation des bâtiments, les réseaux métropolitains à grande vitesse et les réseaux d'entreprise. [29] [30]

1.6 Les domaines d'applications

L'Internet des objets (IoT) attire une attention considérable de la part des universités, des gouvernements, des industries et des citoyens pour des différentes applications. Il existe plusieurs domaines et environnements dans lesquels IoT peut jouer un rôle remarquable et améliorer nos vies. Ces applications comprennent les transports, les soins de santé, la surveillance de l'environnement, les villes intelligentes, l'automatisation industrielle et l'agriculture…etc.

Les applications IoT ont un impact important sur la qualité de vie des personnes comme ils génèrent également d'énormes avantages. La figure 1.4 montre quelque domaine d'application du IoT.

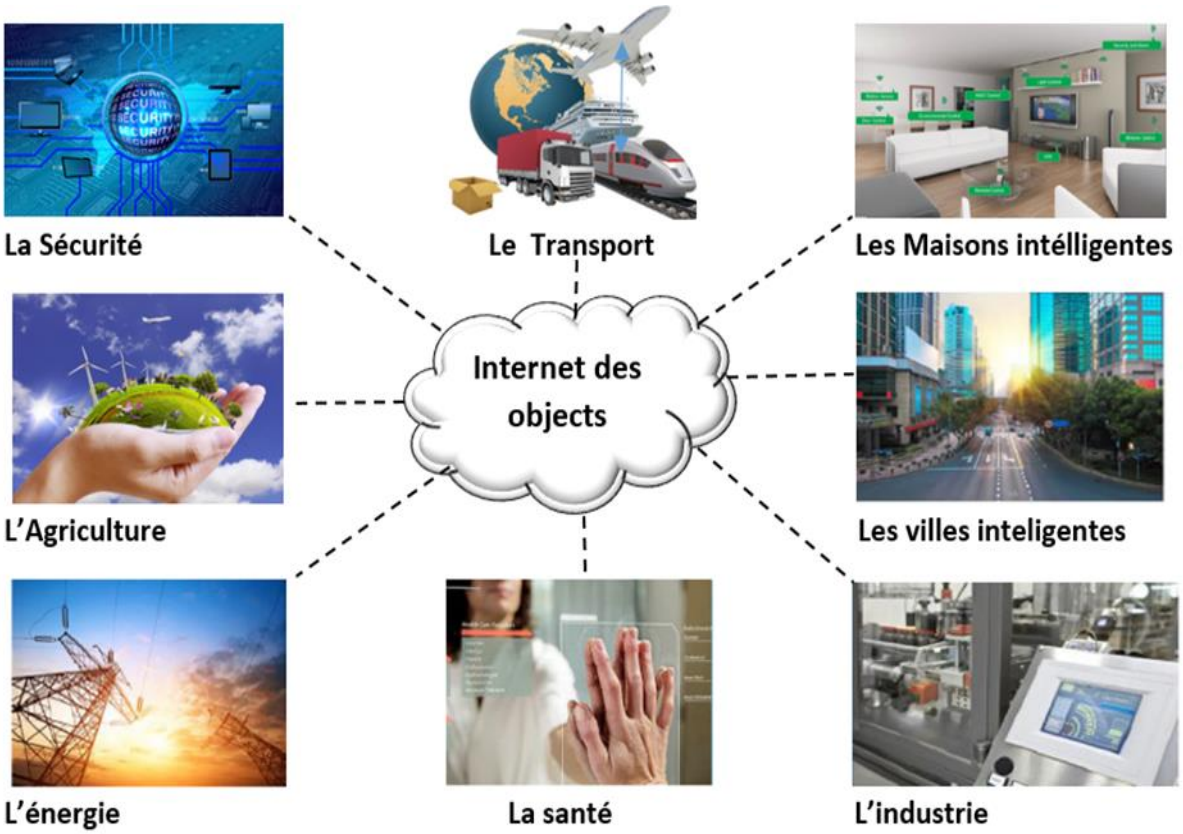


Figure 1.4: Exemples des applications d'IoT

1.7 Conclusion

L'internet des objets présente un intérêt considérable. En effet, plus les nouvelles technologies augmentent plus l'IoT évolue et prend de la place dans nos vies.

Dans ce premier chapitre, nous avons décrit les éléments d'IoT, ainsi que leurs principaux domaines d'application, l'architecture IoT et les différents types des réseaux tels que LLN, WSN, WMN…etc. Cependant, nous avons remarqué que plusieurs facteurs compliquent la gestion de ce type de réseaux alimentation insuffisante, puissance de calcul et capacité mémoire très limitées. Le chapitre suivant est dédié à une étude sur le routage dans les réseaux IoT, plus précisément dans les réseaux LLNs.

Chapitre 02 : Etat de l'art sur le routage dans les réseaux IoT

2.1 Introduction

Le routage dans les réseaux LLNs présente de nombreux défis pour la communauté scientifique à cause des caractéristiques uniques de ces derniers telles qu'une alimentation insuffisante, puissance de calcul et capacité mémoire très limitées, des caractéristiques de liaison fortement asymétriques, une perte de données importante et de courtes distances de communication…etc. De ce fait, plusieurs protocoles de routage furent proposés pour répondre aux exigences des réseaux LLNs.

Ce chapitre est consacré au routage d'une manière générale et plus spécifiquement au routage multichemin qui a intéressé les chercheurs dans le but d'améliorer les performances de routage dans les réseaux. Dans cette optique, nous allons introduire les principaux protocoles de routage, selon les différentes classifications en détaillant plus particulièrement les protocoles AODV [2], LOADng [7] et RPL [6].

2.2 Routage

Le routage est un mécanisme par lequel un ou plusieurs chemins sont sélectionnés pour acheminer les paquets de données entre n'importe quelle paire de nœud appartenant au réseau. Il consiste à assurer une stratégie qui garantit à n'importe quel moment un établissement des chemins qui soient correctes et efficaces, ce qui assure l'échange des paquets d'une manière continue.

2.2.1 Métriques d'efficacité des protocoles de routage

Une métrique est un algorithme qui traite le coût associé à un chemin de routage pour mesurer l'efficacité des protocoles de routage. Ces derniers permettent aux nœuds de comparer les métriques calculées afin de déterminer les routes optimales à emprunter.

- **Métrique basée sur le nombre de sauts :** Les protocoles de routage utilisent cette métrique pour minimiser le nombre de sauts pendant le routage. L'idée est de calculer le nombre de nœuds intermédiaires pouvant être traversés lors d'une transmission d'un paquet du nœud source vers le nœud destination. La route choisie est celle qui contient un nombre minimum de nœuds.
- **Métrique basée sur la qualité de lien :** Cette métrique est très populaire dans les réseaux sans fil. La qualité d'un lien est estimée par la qualité des canaux utilisés, il existe plusieurs façons pour mesurer la qualité d'un lien comme :
    - ✓ **ETX (*Expected Transmission Count*) [31]** : Cette métrique mesure le nombre moyen d'une transmission et retransmission nécessaires pour assurer une bonne livraison de paquets. Les chemins fiables sont ceux qui ont le moins nombre de retransmission.
    - ✓ **ETT (*Expected Transmission Time*) [32]** : Cette métrique correspond au temps nécessaire pour que la transmission d'un paquet sur un lien soit réussie.

2.3 Routage monochemin

Routage monochemin est une technique qui sélectionne un seul chemin pour transmettre les données de la source à sa destination.

2.3.1 Les limitations du routage monochemin

Parmi les limitations souvent reprochées au routage monochemin [33] :

- Répartition de charge non équilibrée vu que ce type de protocole choisit la même route pour acheminer les paquets où certains nœuds sont peu impliqués alors que d'autres sont fortement congestionnés et acheminent la plupart du trafic du réseau, de ce fait les nœuds surchargés consomment rapidement leurs ressources et peuvent être amenés à accuser une forte congestion ce qui diminue la durée de vie du réseau.
- En raison d'une alimentation insuffisante, d'un espace de stockage limité et d'une communication peu fiable, les pannes sont courantes. En cas d'échec, la plupart des protocoles de routage monochemin ne pourraient pas transmettre les données détectées au récepteur en raison de l'absence de mécanismes de tolérance aux pannes.
- Dans le routage à chemin unique, la présence d'un nœud malveillant sur le chemin peut manipuler et corrompre les données sans attirer l'attention du nœud récepteur.

2.4 Routage multichemin

Le routage multichemin est une technique de routage alternative qui a été conçue pour corriger les faiblisses de protocole monochemin. Il sélectionne plusieurs chemins pour transmettre les données de la source à la destination.

2.4.1 Avantages du routage multichemin

Le routage multichemin vise les objectifs suivants [32] :

- **Tolérance aux pannes** la tolérance aux pannes peut souvent être réalisée via deux approches comme l'envoi de paquets redondants sur des chemins multiples ou bien l'établissement à l'avance d'un ou plusieurs chemins de secours en cas de panne.
- **Equilibrage de charge** : le partage de charge consiste en l'utilisation efficace des ressources disponibles dans le réseau. La répartition des données d'un flux sur des chemins multiples, au lieu d'acheminer le flux sur un seul chemin, pouvait réduire la congestion et les goulots d'étranglement.
- **Augmentation de la bande passante offerte à flux** : un routage multichemin peut offrir une bande passante agrégée, via les différents chemins utilisés en parallèle, plus importante que la bande passante offerte par un chemin unique. Ceci peut permettre de satisfaire les besoins en bande passante d'une application qui n'aurait peut-être pas été satisfaite avec l'utilisation d'un seul chemin ayant une bande passante limitée.

2.4.2 Mécanismes de routage multichemin

Le routage multichemin est basé sur trois mécanismes [34] :

- **La découverte des chemins :** consiste à rechercher plusieurs routes entre un nœud source et un nœud de destination.
- **Maintenance de la route :** Dans les protocoles de routage multichemin, la disponibilité des routes alternatifs peut être utilisée en cas de défaillance de la route et cela donne la possibilité d'attarder la maintenance jusqu'à l'échec de N route tandis que dans l'uni-chemin la maintenance doit être effectuée immédiatement.
- **L'allocation du trafic** : décide comment les données à transmettre vont être distribuées sur les chemins disponibles entre une source et une destination dans le réseau.

2.4.3 Allocation du trafic dans le routage multichemin

L'utilisation des routes disponibles diffère d'un protocole à un autre selon l'objectif :

- Utilisation de tous les chemins trouvés simultanément en introduisant la redondance des paquets afin d'améliorer la fiabilité [35], comme AODVM (*Ad-hoc On demand Distance Vector multipath*) [36].
- Utilisation d'un seul chemin et les autres chemins construits servent comme chemins de secours en cas de rupture de route afin de minimiser la génération des paquets redondants et de réduire la charge sur le réseau, comme AOMDV (*Ad-hoc On multipath demand Distance Vector*) [37].
- Division des paquets de données en fragments au niveau du nœud source et leur transmission aux différents chemins découverts dans le but d'économiser l'énergie et réduire la surcharge [33].

2.4.4 Types de multichemin

Après la phase de découverte, le choix des chemins construits entre deux nœuds est très nécessaire car il aura un impact important sur la performance du routage multichemin. L'approche multichemin est basée sur le principe de la disjonction des chemins. Ce principe a pour but d'assurer l'indépendance des chemins, par exemple si des chemins, utilisés pour la transmission des données entre une source et une destination partagent des liens, la rupture d'un lien partagé entrainera la rupture de plusieurs chemins passant par celui-ci, ce qui induira la dégradation des performances du réseau. Deux types de chemins disjoints sont considérés [32] [38] :

- **Chemins à nœuds disjoints** : les chemins sont choisis d'une manière où il n'y a que la source et la destination comme nœuds en commun. L'utilisation de ce type de chemins assure que les coupures des liens se font de manière indépendante entre les différents chemins. Cependant le nombre de chemin trouvé est restreint. La figure 2.1 donne un exemple de chemins à nœuds disjoints.

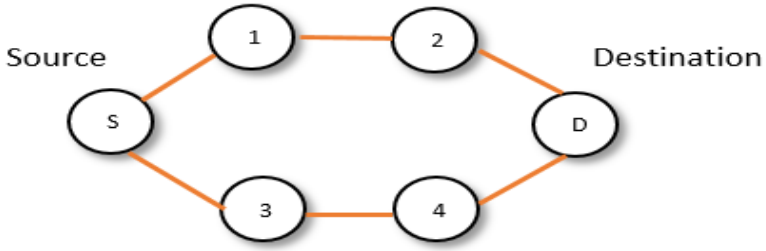


Figure 2.1 : Deux chemins à nœuds disjoints

- **Chemins à liens disjoints** : les chemins sont choisis d'une manière où il n'y a aucun lien en commun entre eux. Cette notion ne garantit néanmoins pas que les chemins pourront être utilisés simultanément. En effet, si certains nœuds sont partagés par différents chemins, ils ne pourront transmettre que sur un chemin à la fois et la rupture d'un nœud provoque la défaillance de tous les chemins passant par ce nœud. La figure 2.2 donne un exemple de chemins à liens disjoints.

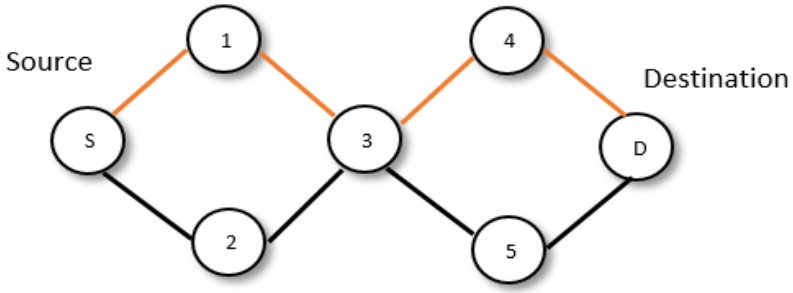


Figure 2.2 : Deux chemins à liens disjoints

2.5 Classification des protocoles de routage

Les protocoles de routage ont été largement étudiés, ces derniers peuvent être classifiés selon plusieurs critères. Nous proposons une classification pour les différents protocoles existants :

2.5.1 Routage à la source vs routage saut par saut

- **Routage à la source (*source routing*)** : C'est une technique utilisée dans les protocoles de routage qui impose à la source de prédominer le parcours de chemin. L'entête de paquet va donc contenir la liste des différents nœuds relayeurs vers la destination pour que chaque nœud intermédiaire lit l'entête et détermine quel est le prochain saut destiné à

recevoir le paquet. Ce type de protocole conçu pour une bonne utilisation de la bande passante en éliminant les messages périodiques de mise à jour de la table de routage. Le protocole le plus connu basant sur cette classe est RPL [6].

- **Le routage saut par saut (*hop by hop*)** : Dans cette stratégie chaque nœud de réseau, qu'il soit un nœud source ou un nœud intermédiaire, choisit d'une manière indépendante le meilleur saut menant vers la destination. L'avantage de ce type de routage est qu'il permet une meilleure adaptabilité aux changements fréquents de topologie. LOADng [7] fait partie des protocoles qui utilisent cette technique.

2.5.2 Protocole proactif vs Protocole réactif

- **Protocole proactif** : Dans ce type de protocole, Chaque nœud maintient une table de routage qui liste le nombre de chemins vers chaque destination, en se basant sur un échange périodique des messages qui contiennent les informations nécessaires pour mettre à jour la table de routage. Comme les tables de routage sont prêtes à être utilisées cette catégorie de protocole donne les latences les plus faibles. Or la mémorisation permanente de toutes ces destinations induit une surcharge dans la table de routage et l'échange régulier des messages de contrôle implique une consommation d'énergie et une utilisation de la bande passante importantes. Parmi les protocoles proactifs OLSR [3].

- **Protocoles réactifs** : dans ce protocole, les nœuds ne génèrent aucun trafic de contrôle sans qu'il soit nécessaire c'est-à-dire il reste en mode veille jusqu'à ce qu'il soit déclenché par une demande de transmission. Ceci permet d'une part, une réduction de la charge du trafic dans le réseau et d'autre part, une conservation d'énergie et de bande passante. Mais l'inconvénient majeur de cette catégorie est l'augmentation radicale de la latence et les retards de transmissions. Parmi les protocoles réactifs AODV [2].

- **Protocole hybride** : C'est un compromis entre les deux catégories de protocoles de routage. Ces protocoles combinent les deux idées des protocoles proactifs et réactifs. Le principe est de connaitre le voisinage de manière proactive jusqu'à une certaine distance (trois ou quatre sauts) et si jamais un nœud cherche à envoyer quelque chose à une destination qui n'est pas dans cette zone, il va effectuer une recherche réactive à

l'extérieur. Avec ce système, on dispose immédiatement des routes dans notre voisinage proche, et lorsque la recherche doit être étendue plus loin, elle en est optimisée.

2.5.3 Protocole monochemin vs Protocole multichemin

- **Protocole monochemin :** est un type de protocole qui sélectionne un seul chemin depuis une source vers une destination.
- **Protocole multichemin :** est un type de protocole qui sélectionne plusieurs chemins depuis une source vers une destination.

2.6 Quelques protocoles de routage

2.6.1 Contexte et histoire

Depuis la fin des années 90, l'IETF (*Internet Engineering Task Force*) s'est engagé dans la mise au point de protocoles de routage pour les réseaux avec des liaisons de plus en plus fragiles et de faible capacité et des ressources de routeur de plus en plus limitées. Et cela, en créant en 1997, le groupe de travail MANET (*Mobile Ad Hoc Network*), puis en 2006 et 2008 en créant les groupes de travail 6LowPAN (*IPv6 over Low Power Wireless Personal Area Networks*) et ROLL (*ROLL Working Group*) [14].

Le groupe de travail MANET s'est concentré sur le développement de deux familles de protocoles : les protocoles réactifs, y compris AODV [2], et les protocoles proactifs, y compris OLSR [3]. Protocole de vecteur de distance AODV [2] fonctionne de manière à la demande. Un protocole d'état de liaison OLSR [3], repose sur des échanges périodiques de messages de contrôle [14].

Le groupe de travail 6LowPAN a été chargé d'adapter IPv6 à un fonctionnement supérieur à IEEE 802.15.4. Une partie de la charte initiale de ce groupe de travail consistait à développer des protocoles pour le routage dans des topologies à sauts multiples dans des conditions aussi contraignantes et sur cette couche de liaison de données particulière (couche d'adaptation). Deux philosophies initiales ont été explorées pour ce type de routage : « *mesh-under* » et « *route over* » [14]. Si la décision de routage est prise dans la couche réseau, nous l'appelons « *route-over* » et si la décision est prise dans la couche d'adaptation, nous l'appelons « *mesh-under* » [39]. Plusieurs propositions de routage ont été présentées dans 6LowPAN, pour chacune de ces philosophies, y compris LOAD [40]. LOAD (*The Lightweight On-demand Ad hoc Distance-*

vector Routing Protocol) était un dérivé de AODV, adapté aux adresses de la couche 2 et au routage « *mesh-under* ».

Cependant, 6LowPAN traitait d'autres problèmes relatifs à l'adaptation d'IPv6 pour IEEE 802.15.4, tels que la compression d'en-tête de paquet IP (*Internet Protocol*), et la résolution des problèmes de routage a été suspendue, déléguée à un groupe de travail ROLL qui a produit un protocole de routage appelé RPL (*Routing Protocol for Low-power lossy networks*) en 2011 [13].

Bien que LOAD [40] ait été suspendu par le groupe de travail 6LoWPAN et les dérivés de protocole réactifs restent actifs. La norme ITU-T G3-PLC [41], publiée en 2011, spécifient l'utilisation de LOAD sur la couche 2 ou 2.5, pour fournir un routage *mesh-under* aux réseaux de comptage des services publics (électricité). L'utilisation d'un protocole de routage réactif plutôt que RPL est justifiée par le fait que ces protocoles prennent mieux en charge les flux de données bidirectionnels tels que la demande / réponse d'une lecture de compteur. L'émergence de réseaux LLN a donc suscité un regain d'intérêt pour les protocoles de routage réactif pour des scénarios spécifiques, ce qui a entraîné des travaux au sein de l'IETF aux fins de la normalisation d'un successeur de LOAD noté LOADng (*the Lightweight On-demand Ad hoc Distance vector Routing Protocol –Next Generation*)). LOADng intègre les expériences acquises lors du déploiement de LOAD, y compris, mais pas uniquement, dans les réseaux LLN. Elle a été incluse dans une révision ultérieure de la norme ITU-T G3-PLC pour la communication dans le réseau intelligent [13].

2.6.2 AODV (*Ad hoc on demand Distance Vector)*

Le protocole de routage avec vecteur de distance à la demande AODV [2] est un protocole de routage saut par saut réactif. Il combine l'utilisation des messages périodiques HELLO propres à DSDV (*Destination-Sequenced Distance-Vector Routing*) [42] et la technique de découverte de chemin à la demande de DSR (*Dynamic Source Routing*) [44]. Ce protocole est destiné à être utilisé dans les réseaux MANET car il offre une adaptation rapide aux conditions de liaison dynamique, réduction du trafic de contrôle et réduction de la surcharge mémoire.

AODV maintient les chemins d'une manière distribuée en gardant une table de routage au niveau de chaque nœud intermédiaire appartenant au chemin découvert. Chaque entrée dans la table de routage comprend, une adresse destination, le prochain saut et une métrique indiquant le nombre de saut pour atteindre la destination, liste des voisins qui utilisent ce chemin et un

numéro de séquence. Dans les réseaux MANET, les nœuds se déplacent fréquemment ce qui fait que les chemins maintenus deviennent invalides. Le numéro de séquence permet l'utilisation de la route la plus récente afin de forcer les mis à jour si nécessaires et d'éviter la formation de boucle [38].

2.6.2.1 Le principe de fonctionnement

Le protocole AODV s'appuie sur deux mécanismes [43] : la découverte de route (*Route Discovery*) et la maintenance de route (*Route Maintenance*).

- **Découverte de route**

  Lorsqu'un nœud désire émettre des données, il vérifie tout d'abord sa table de routage, s'il ne dispose d'aucune route vers la destination, il déclenche la procédure de découverte de route. Le nœud émetteur initie la procédure de découverte de route en inondant un paquet de type RREQ (*route request*) dans le réseau contenant l'adresse de destination recherchée, le dernier numéro de séquence connu pour cette destination, l'adresse de la source, le dernier numéro de séquence connu pour cette source, l'identifiant de la requête et le nombre de saut. La figure 2.3 présente la recherche d'une route par inondation.

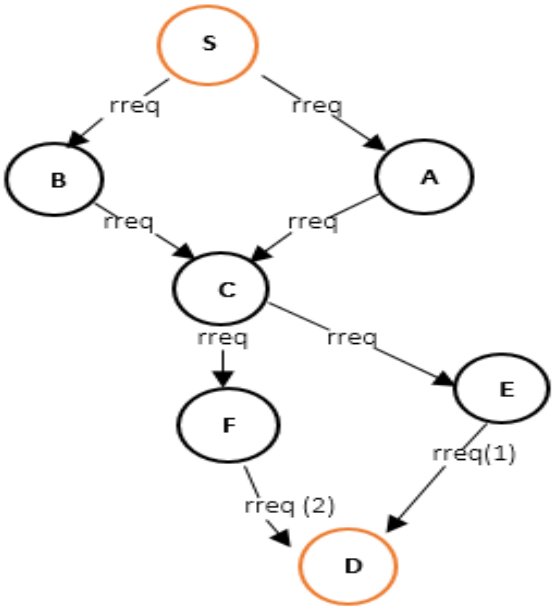


Figure 2.3 : Recherche de route par inondation

En recevant un paquet RREQ par un nœud récepteur N, le nœud N peut faire face à deux possibilités :

- ✓ Si l'adresse de N n'est pas la même que l'adresse destination du paquet RREQ donc il s'agit d'un nœud intermédiaire. D'abord, ce dernier traite la requête en stockant une route de retour vers la source (*Reverse Path*). La figure 2.4 montre la table de routage du nœud E après la construction du chemin inverse. Ensuite, il renvoie un paquet RREP (*Route reply*) s'il possède une route avec un numéro de séquence supérieur ou égal à celui de la RREQ. Alors un paquet RREP-gratuit est également envoyé pour informer la destination et pour prévenir les nœuds qui se trouvent au long du chemin menant à cette destination. Une rediffusion de paquet RREQ doit être effectuée si le nœud N ne possède plus une route vers cette destination ou s'il dispose d'une route avec numéro de séquence inférieur à celui de la RREQ.

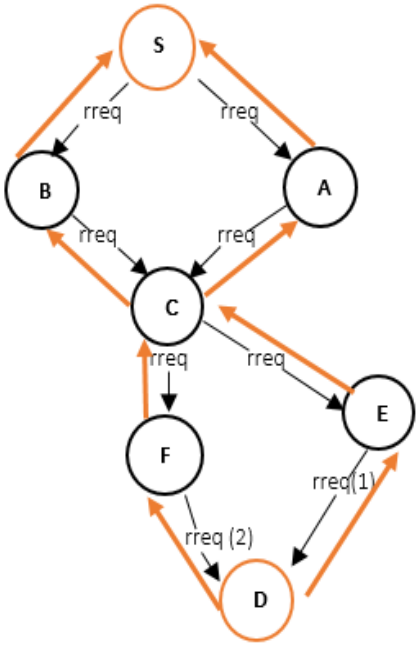


Table de routage			
Destination	Prochain saut	N° seq	Nombre de saut
S	**C**	**1**	**3**

Figure 2.4 : Construction des chemins inversés

- ✓ Sinon si l'adresse de N est la même que l'adresse destination de RREQ alors il s'agit d'un nœud destination. L'enregistrement de la route pour atteindre le nœud source de RREQ. Dès lors, une copie de chemin est envoyé dans un paquet réponse RREP de route vers la source. Chaque nœud reçoit le paquet RREP fait la mise à jour de sa table de routage en introduisant une route vers la destination. Voir la figure 2.5

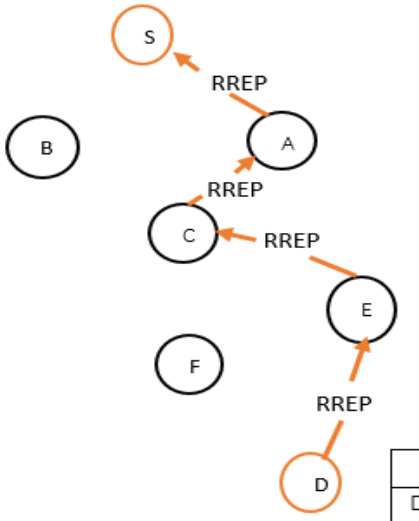


Table de routage			
Destination	Prochain saut	N° Seq	Nombre de saut
S	C	1	3
D	D	1	1

Table de routage			
Destination	Prochain saut	N° Seq	Nombre de saut
S	E	1	4

Figure 2.5 : La propagation d'un paquet RREP

Dès la réception de paquet RREP le nœud source peut utiliser le chemin trouvé pour transmettre les paquets de données (Voir la figure 2.6).

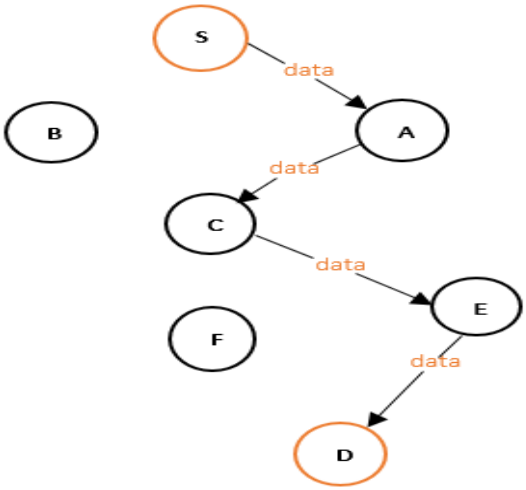


Figure 2.6 : Acheminement des données dans le réseau.

- **Maintenance des chemins**

  Dès qu'un chemin est établi entre un nœud source et un nœud destination, un mécanisme de maintenance est déclenché automatiquement. Ce mécanisme gère essentiellement les ruptures de liens entre les nœuds. Il est basé sur la transmission périodique des messages HELLO. Si trois massages Hello ne sont pas reçus

consécutivement à partir d'un nœud voisin, ce lien est considéré défaillant. Lorsqu'un nœud détecte une rupture de lien dans un chemin actif, ce dernier envoie un paquet d'erreur RERR (*Route Error*) à son nœud prédécesseur. Ce mécanisme est répété jusqu'au nœud source, voir figure 2.7. Dans le cas échéant il lance une nouvelle découverte de route [37].

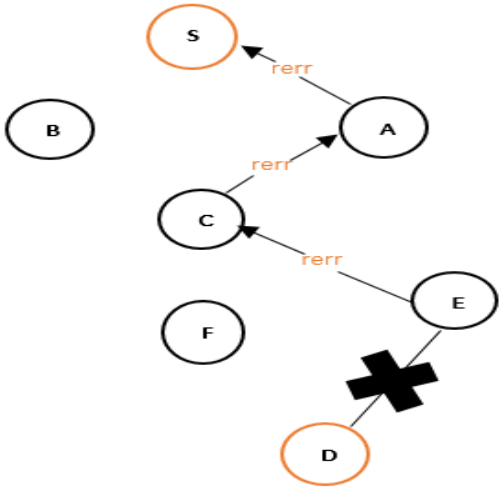


Figure 2.7: Maintence de la route

2.6.3 LOADng (*The Lightweight On-demand Ad hoc Distance-vector Routing Protocol – Next Generation*)

LOADng est un protocole de routage vecteur à distance réactif dérivé de AODV spécifié par IETF draft [7]. LOADng contient un petit ensemble d'opérations de protocole, et il a des exigences se prêtant à une implémentation simple pour l'utiliser dans les périphériques basés sur IEEE 802.15.4 dans les réseaux LoWPAN (*Low-Power Wireless Personal Area Networks*) et LLN [13].

En tant que protocole réactif, LOADng garde le même principe de fonctionnement de base de AODV y compris les procédures de découverte de route et maintenance de route sauf que LOADng a apporté des modifications pour répondre aux exigences des réseaux LLN. LOADng n'utilise pas le numéro de séquence de destination afin de réduire la taille des messages de contrôle. Il propose une simplification pour le processus de découverte de route où seule la destination qui est autorisée à répondre à un paquet RREQ en envoyant un paquet RREP en unicast, saut par saut à la source, voir la figure 2.8. Comme les nœuds intermédiaires sont explicitement interdits de répondre, même s'ils disposent des itinéraires actifs vers la destination, et donc les paquets RREP gratuit deviennent inutiles [14].

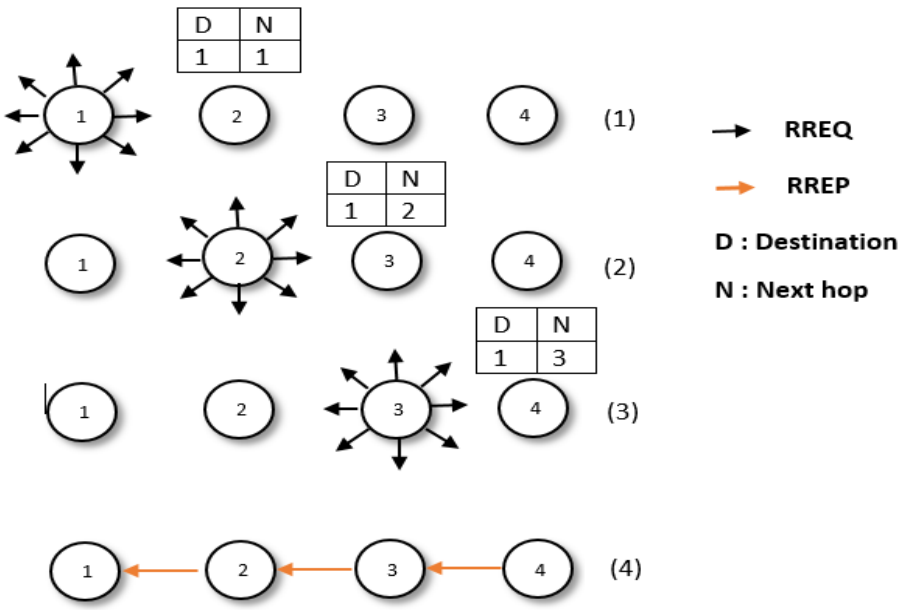


Figure 2.8 : Découverte de chemin LOADng sans RREP intermédiaire

LOADng n'utilise pas la "liste de précurseurs" de AODV, il ne se soucient que du prochain saut pour transférer le paquet actuel à sa destination. En cas d'échec, il transmit le paquet RERR uniquement à l'expéditeur de la livraison de données ayant échoué [14]. Ce protocole est agnostique de couche « layer-agnostic ». Cela signifie qu'il peut être utilisé en tant que protocole de routage à la couche réseau « *route-over* » ou à la couche 2 « *mesh-under* » [7].

2.6.3.1 Les caractéristiques de LOADng

LOADng se caractérise par [13] :

- Conception modulaire : L'implémentation LOADng a été développée pour être aussi modulaire que possible. Toutes les fonctions appelées lors de la réception ou de l'envoi d'un paquet, ou la gestion des tables LOADng sont indépendantes et transparentes les unes des autres, de sorte que des modifications supplémentaires peuvent être apportées de manière simple et directe sans affecter le reste du code.
- Adressage flexible : Différentes longueurs d'adresses sont prises en charge (IPv6, IPv4, Rime), la seule exigence est que dans un domaine de routage donné, toutes les adresses soient de la même longueur.

- Métriques : Prise en charge de différents types de métriques, la plus simple est le nombre de sauts.

2.6.3.2 L'objectif de LOADng

L'objectif de ce protocole est que chaque routeur LOADng puisse [14] :

- Être capable de découvrir un chemin bidirectionnel vers n'importe quelle destination.
- Établir les chemins uniquement lorsqu'il y a un trafic de données à envoyer.
- Conserver les chemins uniquement s'il y'a un trafic utilisant ce chemin.

2.6.3.3 Structure de LOADng

2.6.3.3.1 Les paquets LOADng

Le format de tous les paquets LOADng suit la même structure, composé du champ type, du bloc TLV (*Type-Length-Value*) et du champ message. Le tableau 2.1 présente le format d'un paquet LOADng [7].

Le champ	La taille / bit	La description
Type	4	Encode le type de message dans le champ <message>
Tlv-block	Variable	Contient les TLV requises
Message	Variable	Contient le message RREQ, RREP, RERR ou RREP – ACK conformément au champ <type>

Tableau 2.1: format de paquet LOADng [7]

- **Champ Type :** Ce champ code le type de message inclus dans le paquet LOADng. Tableau 2.2 affiche l'encodage pour chaque type de message LOADng.

Message	Type
RREQ	0
RREP	1
RERR	2
RREP–ACK	3

Tableau 2.2 : Encodage du champ Type [7]

- **Bloque TLV :** Ensemble d'éléments permettant de développer des extensions de protocoles flexibles et de l'adapter aux besoins spécifiques de l'application. Le format d'un bloque TLV est spécifié dans le tableau 2.3.

Champ	description
<tlv-count>	Spécifie le nombre de TLV inclus, chacun contenant les champs suivants.
<tlv-type>	Spécifie le type de TLV.
<tlv-length>	Spécifie la longueur du champ <valeur-tlv> suivant, en octets.
<tlv-value>	Champ de longueur tlv – longueur, en octets.

Tableau 2.3 : le format de bloque TLV [7]

- **Champ message :** LOADng utilise quatre types de message :

  - ✓ **Le message RREQ** *:* Généré par nœud LOADng lorsqu'il est présenté avec un paquet de données à une destination pour laquelle il n'a pas de route valide.
  - ✓ **Le message RREP :** Généré par le nœud de destination après la réception et le traitement d'un paquet RREQ. Le tableau 2.4 illustre les champs d'un message RREQ et RREP.

Champ	Taille/bits	Description
Msg-Type	2	RREQ ou RREP
Addr-Length	4	Taille de l'adresse
Hop-Limit	4	Limite de saut de message
Hop-Count	4	Nombre de saut
Metric-Type	8	Type de la métrique
Seq-Num	16	Numéro de séquence
Flag	4	1 si le msg RREP – ACK est requis, de 1 à 3 Sont réservés
Metric	4	Métrique
Originator	8-128	Expéditeur du message
Destination	8-128	Destination du message

Tableau 2.4 : Champs RREQ / RREP et valeurs initiales [7]

- ✓ **Les messages RREP-ACK (***Route Reply-acknowledgement***):** Générés par nœud LOADng après une réception de la RREP pour informer la source de RREP que son

message a été reçu avec succès. Le tableau 2.5 illustre les champs d'un message RREP-ACK.

Champ	Taille/bits	Description
Msg-type	2	RREP-ACK
Addr-length	4	Taille de l'adresse
Seq-num	16	numéro de séquence
Destination	8-128	Destination du message

Tableau 2.5 : Champs d'un message RREP-ACK [7]

- ✓ **Les messages RERR :** générés par un nœud LOADng lorsqu'un lien vers la destination est rompu. Le tableau 2.6 illustre les champs d'un message RERR.

Champ	Taille/bits	Description
Msg-type	2	RRER
Addr-length	4	Taille de l'adresse
Hop-limit	8	Limite de saut de message
Hop-count	8	Nombre de saut
Errorcode	8	Spécifie le type d'erreur
Unreachable-address	variable	Destination du paquet ayant échoué
Originator	variable	Expéditeur du message
Destination	variable	Destination du message

Tableau 2.6 : Champs d'un message RERR [7]

2.6.3.3.2 Les tables LOADng

- **La table de routage (*routing set*)**

La table de routage stocke les informations de routage pour tous les nœuds accessibles dans le réseau. Selon les informations qui existent dans la table de routage, un nœud peut vérifier la nécessité de démarrer un nouveau processus de découverte de route**.** Le tableau 2.7 illustre les champs de chaque entrée de la table de routage.

Champ	description
R_dest_addr	L'adresse de la destination
R_next_addr	L'adresse du prochain saut sur l'itinéraire sélectionné vers la destination
R_dist	La distance entre l'itinéraire sélectionné et la destination
R_metric	Spécifie comment la distance est défini et calculé
R_seq_num	Le numéro de séquence
R_valid_time	Spécifie le temps nécessaire pour que les informations contenues dans ce tuple soient considérées comme valides
route_cost	
weak_links	

Tableau 2.7 : les champ d'une entrée de la table de routage [7]

- **La table des voisins en liste noire (*Blacklisted Neighbor Set*)**

Cette table contient l'ensemble des voisins pour lesquels la connectivité a été détectée comme étant unidirectionnelle. Un lien unidirectionnel peut être détecté lorsqu'un RREP – ACK attendu n'est pas reçu. Il est possible de recevoir des données de leur part, mais il est impossible de les communiquer. Le tableau 2.8 illustre les champs pour chaque entrée de la liste noire.

Les champs	la description
B_neighbor_address	L'adresse du voisin de la liste noire
B_valid_time	Spécifie le temps nécessaire pour que ce nœud reste dans la liste noir

Tableau 2.8 : les champ d'une entrée de la liste noire [7].

- **La table d'attente de confirmation (*Pending Acknowledgement Set*)**

Cette table contient des informations sur les RREP transmises avec l'indicateur ACK-REQUIRED et pour lesquelles aucun RREP – ACK n'a pas été encore reçu. Le tableau 2.9 illustre les champs pour chaque entrée de la liste d'attente.

Les champs	la description
P_next_hop	L'adresse du voisin auquel le RREP a été envoyé
P_originator	L'adresse de l'expéditeur du RREP
P_seq_num	Numéro de séquence
P_ack_timeout	Spécifie le temps nécessaire pour que la liaison avec le voisin correspondant ne soit pas considérée comme bidirectionnelle

Tableau 2.9 : les champ d'une entrée de la liste d'attente de confirmation [7]

2.6.4 RPL (*Routing Protocol for Low Power and Lossy Networks*)

RPL [6] est un protocole de routage Vecteur de distance proactif, il a été conçu par le groupe de travail IETF ROLL pour les réseaux LLNs. Ce protocole de routage prend en charge des modèles de trafic simples et complexes tels que *Point-to-Multipoint* (trafic de données de la racine au nœud du réseau), *Multipoint-to-Point* (trafic de données d'un nœud à la racine) et *Point-to-Point* (trafic de données d'un nœud à un nœud).

2.6.4.1 Composants RPL

Les composants du protocole RPL sont les suivants [45] **:**

1) Graphe acyclique dirigé par destination (*DODAG*)

Le DODAG (*Destination Oriented Directed Acyclic Graph*) fait référence à un graphe acyclique dirigé avec une seule racine, comme illustré à la figure 2.9. Chaque nœud de DODAG connaît ses parents mais ne dispose d'aucune information sur les enfants associés. De plus RPL conserve au moins un chemin d'accès pour chaque nœud vers la racine.

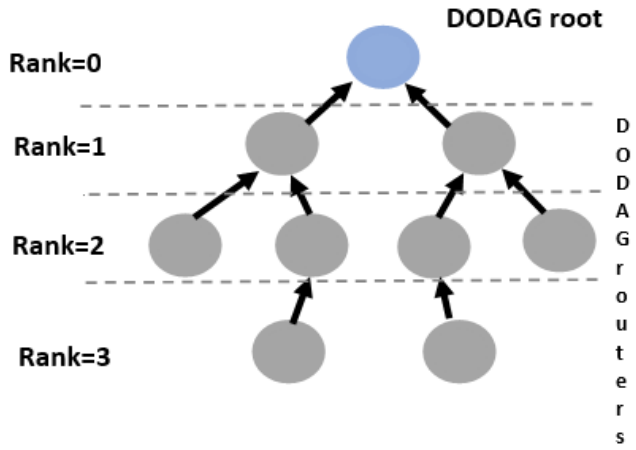


Figure 2.9 : Graphique acyclique dirigé orienté vers la destination

2) Messages de contrôle RPL

Afin de maintenir la topologie de routage et de garder ses informations à jour, RPL utilise cinq types de messages de contrôle :

DIO (*DODAG Information Object*)

C'est le premier message émis par la racine pour maintenir la topologie et les itinéraires ascendants. Il contient quatre paramètres importants (RPLinstanceID, DODAGID, DODAGVersion et Rank) :

- **RPLInstanceID :** C'est un identifiant unique d'une instance RPL. Les DODAG avec le même RPLInstanceID partagent la même fonction objective (Voir la figure 2.10).

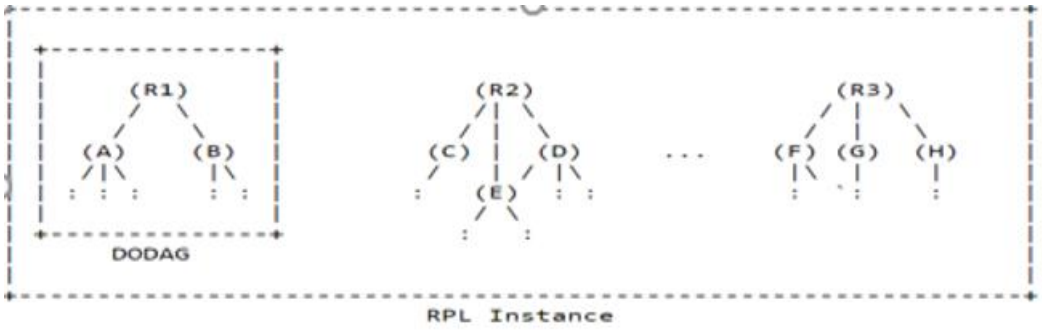


Figure 2.10 : une instance RPL [6]

- **DODAGID** : C'est l'identifiant d'une racine DODAG. Le DODAGID est unique dans le cadre d'une instance RPL dans le LLN. Le couple (RPLInstanceID, DODAGID) identifie le DODAG d'une manière unique.
- **DODAGVersion :** Chaque fois qu'une reconstruction de DODAG est nécessaire, une nouvelle version de la topologie est identifiée (Voir la figure 2.11).

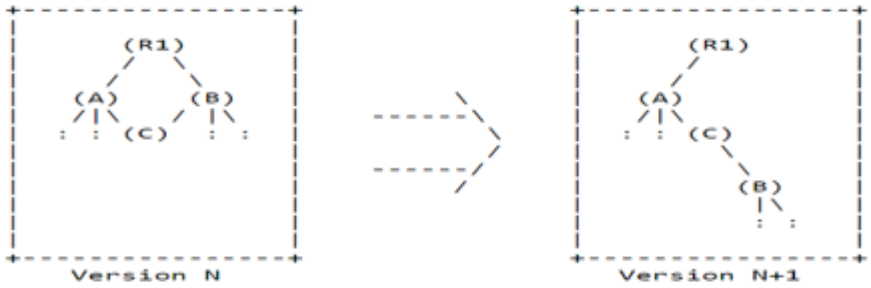


Figure 2.11 : version d'un DODAG [6]

- **Rank :** Le rang d'un nœud définit sa position individuelle par rapport à sa racine DODAG, le rang augmente strictement dans la direction Bas et diminue

strictement dans la direction Haut, Le mode de calcul exact du rang dépend de la fonction objectif (OF).

DIS (*DODAG Information Sollicitation*)

Ce type de message est utilisé par les nœuds qui souhaitent joindre le DODAG pour solliciter une DIO auprès des nœuds RPL.

DAO (*Destination Avertissement Object*)

Pour acheminer le trafic vers le bas, la racine DODAG doit connaitre les chemins pour atteindre n'importe quelle destination, les messages DAO sont utilisés pour construire des routes dans la direction descendante.

DAO-ACK (*Destination Avertissement Object- Acknowledgment)*

Le destinataire d'un message DAO réponds par un message DAO-ACK pour assurer la bonne réception de message DAO.

3) La fonction objective

En RPL, le processus de construction du graphe DODAG est l'un des principaux défis. Chaque nœud souhaite participer au réseau doit sélectionner un ou plusieurs parents préférés parmi un ensemble de nœuds existants. Le choix se fait selon la fonction objective OF (*Objective Function*). Plus spécifiquement, OF explique comment une position de nœud, des métriques de liaison ou des contraintes sont utilisées pour définir le rang du nœud. La fonction objectif est indiquée dans le message DIO.

2.6.4.2 Construction du DODAG

- Configurer certains nœuds pour être des racines DODAG.
- Découvrir et maintenir les itinéraires ascendants :
  - ✓ La racine DODAG diffuse un paquet DIO annonçant une nouvelle version de DODAG
  - ✓ À la réception d'un message DIO, les nœuds choisissent le parent qui porte le rang le plus bas, ils mettent à jour le paquet DIO par leur nouveau rang calculé selon OF, et ils le transmettent aux autres.
- Découvrir et maintenir des itinéraires descendants :

- ✓ Chaque nœud du DODAG transmet un paquet DAO contenant ses informations vers la racine DODAG pour maintenir les itinéraires descendants. RPL a spécifié deux modes pour la création et la gestion des itinéraires descendants « storing » et « non-storing » : dans le mode « Storing », les nœuds non racine stockent les tables de routage de leurs sous-DODAG (le sous-DODAG d'un nœud représente l'ensemble des autres nœuds dont les chemins de routage vers la racine passent par ce nœud). Et dans le mode « non-storing », seule la racine DODAG (la racine DODAG) qui stocke la table de routage et elle est responsable de tous les itinéraires descendants [6].

2.7 Comparaison entre RPL et LOADng

Les caractéristiques	**RPL**	**LOADng**	**AODV**
La catégorie	Vecteur de distance *proactif.*	Vecteur de distance *réactif.*	Vecteur de distance *réactif.*
Proposé par	IETF	IETF	IEEE (*Institute of Electrical and Electronics Engineers*)
Proposé en	2011	2012	1999
Standardisé par	IETF	ITU(*International Telecommunication Union*)	
Type du réseaux	les réseaux *LLN*s	les réseaux *LLN*s	Les réseaux Ad hoc
Message de contrôle	Les messages **DIO** Les messages **DIS** Les messages **DAO** Les messages **DAO-ACK**	Les messages **RREQ** Les messages **RREP** Les messages **RERR** Les messages **RREP-ACK**	Les messages **RREQ** Les messages **RREP** Les messages **RERR** Les messages **RREP-ACK** Les messages **HELLO**

Le type du routage	Le routage source pour le trafic descendant	Le routage saut par saut.	Le routage saut par saut.
Le type de Trafic	Multipoint à point Point à multipoint. Point à point.	Point à point.	Point à point.
Les avantages	Consomme moins de délai.	Moins de surcharge dans le réseau car il est un protocole léger.	La maintenance de la disponibilité de la bande passante.
Les inconvénients	Consomme plus d'énergie et de surcharge dans le réseau.	L'établissement des chemins à la demande exige plus de délai.	L'exécution du processus de découvert de chemin occasionne un délai important.

Tableau 2.10 : Comparaison entre RPL vs LOADng

2.8 Limitations de RPL

De nombreuses études ont montré que RPL souffre de problèmes qui limitent son efficacité et son domaine d'application, citant quelques limitations [12] :

- RPL exige que chaque nœud exécutant un mode « stornig » conserve l'état de routage de tous les nœuds de son sous-DODAG. Bien que RPL est spécialement conçu pour les nœuds avec mémoire limitée, le protocole a l'ambition de gérer des réseaux denses comprenant jusqu'à des milliers de nœuds. Dans de tels réseaux à haute densité, il est très probable que l'état de routage qui doit être maintenu va déborder de la capacité de stockage de ces périphériques contraints. Par conséquent, un nœud débordé ne pourra pas gérer toutes les entrées de routage devant être conservées dans sa table de routage,

rendant plusieurs destinations de son sous-DODAG inaccessibles par la racine DODAG.

- Dans le mode de non-stockage, la racine doit associer un en-tête de route source (la liste des différents nœuds relayeurs vers la destination) pour chaque paquet de données transmis dans la direction descendante. Cependant, RPL est conçu pour fonctionner sur des couches de liaison avec une unité de transmission maximale MTU (*Maximum Transmission Unit*) limitée qui permet une longueur de chemin de huit sauts au maximum, de la source à la destination. Ceci impose une contrainte stricte à la transmission à sauts multiples.
- RPL souffre encore de nombreux problèmes liés à son évolutivité, en particulier dans les réseaux bidirectionnels à grande échelle. Cela inclut l'inefficacité des deux modes d'exploitation spécifiés pour les itinéraires descendants.

2.9 Travaux connexes sur les protocoles de routages multi chemin

Dans cette étape nous nous sommes concentr**és** sur les protocoles de routage multichemin basé sur AODV car il y'a un manque des protocoles sur LOADng multichemin et AODV est le protocole de base LOADng.

- **AOMDV**

  Le protocole réactif AOMDV [37] basé sur AODV, se consacre à la recherche de chemins multiples à liens disjoints. Parmi les chemins multiples trouvés, AOMDV n'utilise que le meilleur en terme de nombre de sauts pour la transmission de données entre une source et une destination, les autres chemins calculés n'étant utilisés que lorsque la route principale est rompue. AOMDV est basé sur deux mécanismes :

  - ✓ Une règle de mise à jour des chemins pour maintenir des chemins multiples sans boucle de routage.
  - ✓ Un mécanisme distribué entre les différents nœuds du réseau pour calculer des chemins disjoints.

- **AODVM**

  Le protocole AODV-Multipath [36] est une extension d'AODV dans laquelle plusieurs chemins à nœuds disjoints sont recherchés. A la différence d'AODV, AODVM n'autorise pas à un nœud intermédiaire de répondre à une requête à la place de la

destination. Les nœuds intermédiaires ne rejettent pas les copies des RREQ et les enregistrent dans une table de requêtes de route. Le nœud destination répond à chaque copie RREQ qui traverse des chemins différents. Chaque nœud intermédiaire, recevant à son tour une copie de la réponse de route RREP, enregistre le chemin par lequel cette copie provient, choisit dans sa table de requêtes l'entrée avec la plus courte distance pour retourner à la source.

- **FF-AOMDV (***Fitness Function- Ad-hoc On-demand Ad hoc multipath Distance-vector*)

  C'est un protocole de routage multichemin appelé vainqueur de trajets multiples ad- hoc à la demande avec la fonction de précision (FF-AOMDV) [46]. Le FF-AOMDV utilise la fonction objective comme méthode d'optimisation qui recherche le niveau d'énergie et la distance de l'itinéraire afin de sélectionner les chemins optimaux.

  Le protocole de routage FF-AOMDV a apporté une amélioration en termes de débit, rapport de livraison des paquets, délai de bout en bout, consommation d'énergie et durée de vie du réseau.

- **AODVM+** (*Ad-hoc On-demand Distance Vector Multipath Energy Plus*)

  C'est un protocole de routage réactif multichemin pour les réseaux ad hoc basé sur le protocole AODV et AODVM. Il est conçu principalement pour les nœuds ayant des batteries électriques limitées ou les ruptures des liens et les défaillances des chemins se produisent fréquemment. L'idée principale de ce protocole est de découvrir plusieurs chemins entre un nœud source et un nœud de destination en exploitant l'énergie résiduelle minimale de nœuds pour empêcher qu'un nœud critique épuise sa réserve d'énergie et se déconnecte du réseau. Après le processus de découverte, les chemins multiples peuvent avoir des énergies minimales très proches. Un nouveau facteur peut être combiné avec cette valeur minimale qui est l'énergie moyenne des batteries des nœuds constituant les chemins [47].

- **ZD-AOMDV** (*Zone-Disjoint Ad-hoc On-demand Multi-path Distance Vector*)

  C'est un protocole de routage multichemin basé sur AOMDV qui envoie des paquets de données à travers ces chemins simultanément. Ces chemins ne sont pas totalement indépendants les uns des autres à cause des mécanismes d'accès au canal partagé dans les réseaux sans fil. A cet effet, le nombre de voisins actifs est calculé suivant une stratégie bien déterminée. La technique proposée compte le nombre de voisins actifs pour chaque chemin, et choisit les chemins avec moins de voisins actifs pour l'envoi des

données. Les voisins actifs d'un nœud sont définis comme des nœuds qui ont déjà reçu la requête RREQ. Ce protocole consomme moins d'énergie et améliore le délai de bout en bout [48].

Le tableau 2.11 compare quelques protocoles du routages mutichemin basé sur AODV.

Protocole	**Date**	**Métrique**	**Implement-ation**	**Avantages**	**Inconvénients**
AODV [2]	1999	Nombre de saut	NS2 (*Network Simulator2*) NS3 (*Network Simulator3*) Cooja	-La maintenance de la disponibilité de la bande passante.	-L'exécution du processus de découvert de chemin occasionne un délai important. -Un délai d'attente et une mémorisation des paquets de données en cas de changement de topologie.
AOMDV [37]	2001	Nombre de saut	NS2 NS3 Cooja	- Trouver plusieurs chemins alternatifs disjoints lors du processus de découverte d'itinéraire [38]. -Lors d'une rupture de lien au niveau du chemin, la source change ce	-Le calcul de plusieurs chemins induit à une surcharge dans la table de routage (consommation mémoire) [38]. -Accroit la taille de la table de routage alors qu'il bénéfice d'un

				chemin par un des chemins alternatifs au lieu de déclencher un nouveau processus de découverte des chemins ce qui induit une minimisation le délai [38].	seul chemin pour la transmission [38]. -Utilise des liens disjoints, la défaillance d'un nœud en commun entre plusieurs chemins provoque la rupture de tous les chemins passant par ce nœud [38].
FF-AOMDV [46]	2017	Le niveau d'énergie des nœuds mobiles et la distance de la route	NS2	-Prolonge la durée de vie du réseau. -Consommation moins d'énergie. -démineur la surcharge [38].	
ZD-AOMDV [48]	2011	Le nombre des voisins actifs après RREQ	NS2		

AODVM [36]	2001	Nombre de saut	NS2	-La défaillance d'un nœud dans un chemin n'a aucune influence sur les autres chemins découverts [38]. -Consomme moins d'énergie	-Le nombre de chemins trouvé est restreint par rapport aux protocoles liens disjoints. -Utiliser tous les chemins trouvés qui induit à une surcharge dans le réseau [38].
AODVME+ [47]	2010	L'énergie résiduelle minimale et l'énergie moyenne des nœuds constituant les chemins.	NS2	-Minimise le surcout de routage. -minimiser le délai et prolonger la durée de vie du réseau [38].	

Tableau 2.11 : Comparaison entre les protocoles multichemin basées sur AODV

Les protocoles du routage multichemin cités dans le tableau ont pour but de corriger les faiblesses du protocole de base AODV, ou ils ont sélectionné plusieurs chemins disjoints entre la source et la destination selon différentes métriques qui induit à la deminuation de delai de bout en bout, la surcharge dans le réseau et la consommation d'énergie.

2.10 Conclusion

Dans ce chapitre, nous avons parlé du routage d'une manière générale puis nous avons donné un petit aperçu sur le routage monochemin ainsi que ses limitations qui ont abouti à une nouvelle catégorie de routage nommé routage multichemin. Nous avons traité le routage multi chemin d'une manière plus détaillée car il est la base de notre travail. Pour cela nous avons

étalé ses avantages et ses mécanismes qui sont comme suit : la découverte des chemins, la maintenance des chemins et l'allocation du trafic. Nous avons distingué deux types des chemins dans cette stratégie : chemins à nœuds disjoints, chemins à liens disjoints.

Après avoir compris le concept du routage multichemin, nous avons étudié les solutions proposées pour les réseaux LLN (RPL et LOADng) et pour pouvoir mettre en œuvre notre proposition nous nous sommes inspirées des protocoles multichemin existants dans la littérature. Deux tableaux comparatifs sont joints pour résumer tous les protocoles de routage étudiés. Dans le chapitre qui suit, nous nous intéressons à l'implémentation d'une extension multichemin nommée **µLOADng** en se basant sur l'étude effectuée dans ce chapitre.

Chapitre 03 : Proposition d'un protocole multichemin pour l'IoT

3.1 Introduction

Avec les limitations reprochées au routage monochemin, et les contraintes des périphériques dans les réseaux LLN, qui ont un impact crucial sur les performances du réseau (délai, débit et taux de réception…etc.), des solutions de routage multichemin ont été développées pour faire face à ces problèmes. Dans cette optique, nous avons tenté de concevoir un protocole de routage multichemin à partir d'un protocole réactif monochemin.

Une panoplie de protocoles de routage multichemin basés sur RPL existe. Contrairement au LOADng qui vient de susciter l'intérêt des chercheurs ces derniers temps d'où le manque des protocoles basés sur lui. Nous avons opté pour ce protocole car il présente un vaste domaine de recherche et pour contribuer à son développement.

LOADng a les mêmes limitations que n'importe quel protocole monochemin. Pour cette raison, nous avons trouvé qu'il est important de créer une extension de ce dernier en apportant des modifications pour le rendre multichemin dans le but corriger les faiblesses et améliorer les performances du réseau.

Dans les protocoles de routage à liens disjoints, la rupture d'un nœud en commun peut provoquer dans la plupart des cas la défaillance de plusieurs chemins. Pour cette raison notre extension de protocole sélectionne les chemins à nœuds disjoints où aucun nœud n'est en commun entre les chemins de la même destination. Cette technique utilise les ressources du réseau de manière efficace et elle est plus tolérante aux pannes.

Ce chapitre comme son titre l'indique est dédié à la conception de notre extension du protocole LOADng. Cette conception passe par les étapes suivantes :

- Restructuration de la table de routage
- Découverte de routes
- Estimation de la qualité de lien
- Allocation du trafic

3.2 Le protocole de routage multichemin µLOADng

Dans le contexte de ce projet, nous avons proposé une extension du protocole LOADng qui est un protocole multichemin conçu dans le but d'améliorer les performances du réseau (débit, taux de perte, temps de réponse, délai de bout en bout).

Nous nous sommes inspirées dans notre solution des protocoles de routage multichemin AOMDV [37], et AODVM [36].

Nous avons nommé notre extension µLOADng dont le µ (lettre mu de l'alphabet grec) fait référence au concept **mu**ltichemin.

Afin de rendre LOADng un protocole multichemin nous avons suivi la démarche ci-dessous :

3.2.1 Restructuration de la table de routage

La mise en place d'un protocole de routage multichemin nécessite une structure de table de routage différente à celle des protocoles de routage monochemin. Pour cela nous avons restructuré la table de routage de LOADng afin de pouvoir introduire plusieurs chemins à une seule destination. Le tableau 3.1 ci-dessous montre notre nouvelle structure de table de routage.

Champ	Description
R_Dest_Addr	Adresse de la destination
Adv_Hop_Count	Nombre de saut le plus grand entre les chemins
Precursors_List	Liste des précurseurs
R_Metric	Type de métrique
R_Seq_Num	Numéro de séquence
Route_List	Liste des chemins disponible pour cette destination

R_Next_Addr
R_Last_Addr
R_Valid_Time
Route_Cost
Weak_links
Link estimate

Tableau 3.1 : Structure de la table de routage de µLOADng

- **Adv_Hop_Count :** Le champ Adv-Hop-Count (*Advanced Hop Count)* qui signifie nombre de sauts annoncés indique le nombre de sauts le plus élevé d'un chemin. Sachant que dans les protocole multichemin le coût (nombre de saut) des routes doit être approximatif. De ce fait le cout des nouveaux chemins est comparé avec **Adv_Hop_Count** et tout chemin ayant un cout très élevé est rejeté. La valeur de ce champ se met-à-jour à chaque fois qu'un nouveau chemin est ajouté à la table de routage.

- **Precursors_List :** Le champs **Precursors_List** contient les adresses des nœuds construisant les chemins d'une entrée dans la table de routage. Nous l'avons utilisé pour pouvoir construire des chemins à nœuds disjoints.
- **Route_List :** Route_list remplace le champ next_hop dans le protocole monochemin**,** elle définit essentiellement plusieurs next_hop avec leurs coûts respectifs. Chaque chemin dans la liste est caractérisé par un ensemble de propriétés :
  - ✓ **Valid_Time :** indique le temps nécessaire pour que la route soit valide.
  - ✓ **R_Next_Addr :** indique l'adresse du prochain saut (next_hop) pour atteindre la destination.
  - ✓ **R_Last_Addr :** indique le dernier saut pour atteindre la destination
  - ✓ **Route_Cost** désigne la distance (nombre de saut) entre le nœud source et le nœud de destination.
  - ✓ **Link_estimate** représente la qualité du lien du nœud voisin (next_hop).

3.2.2 Découverte des chemins

La découverte des chemins se fait par les messages de contrôle. Afin de pouvoir découvrir plusieurs chemins à nœuds disjoints, nous avons modifié le format de message RREQ, en ajoutant deux nouveaux champs qui sont "**First_Hop**" et "**Precursors_List**" comme le montre le tableau 3.2. Le premier contient l'adresse du voisin (à un saut) de l'émetteur, le deuxième contient l'adresse des nœuds traversés par le message RREQ.

Champ	Taille(bit)	Description
Msg-Type	8	RREQ ou RREP
Addr-Length	4	Taille de l'adresse
Hop-Limit	8	Limite de saut de message
Hop-Count	8	Nombre de saut
Metric-Type	8	Type de la métrique
Seq-Num	16	Numéro de séquence
Flag	8	1 si le msg RREP – ACK est requis, de 1 à 3 sont réservés
Metric	variable	Métrique
Originator	8-128	Expéditeur du message
Destination	8-128	Destination du message
First-Hop	8-128	adresse du premier saut
Precursors-List		Liste des précurseurs

Tableau 3.2 : les champs d'extension du message RREQ/RREP

Lorsqu'un nœud veut envoyer un paquet de données à une destination, il consulte sa table du routage pour vérifier s'il existe des routes valides vers cette destination, si oui il va l'utiliser directement sinon il va déclencher le processus de découverte de route en inondant un paquet RREQ dans le réseau. La figure 3.1 schématise ce processus.

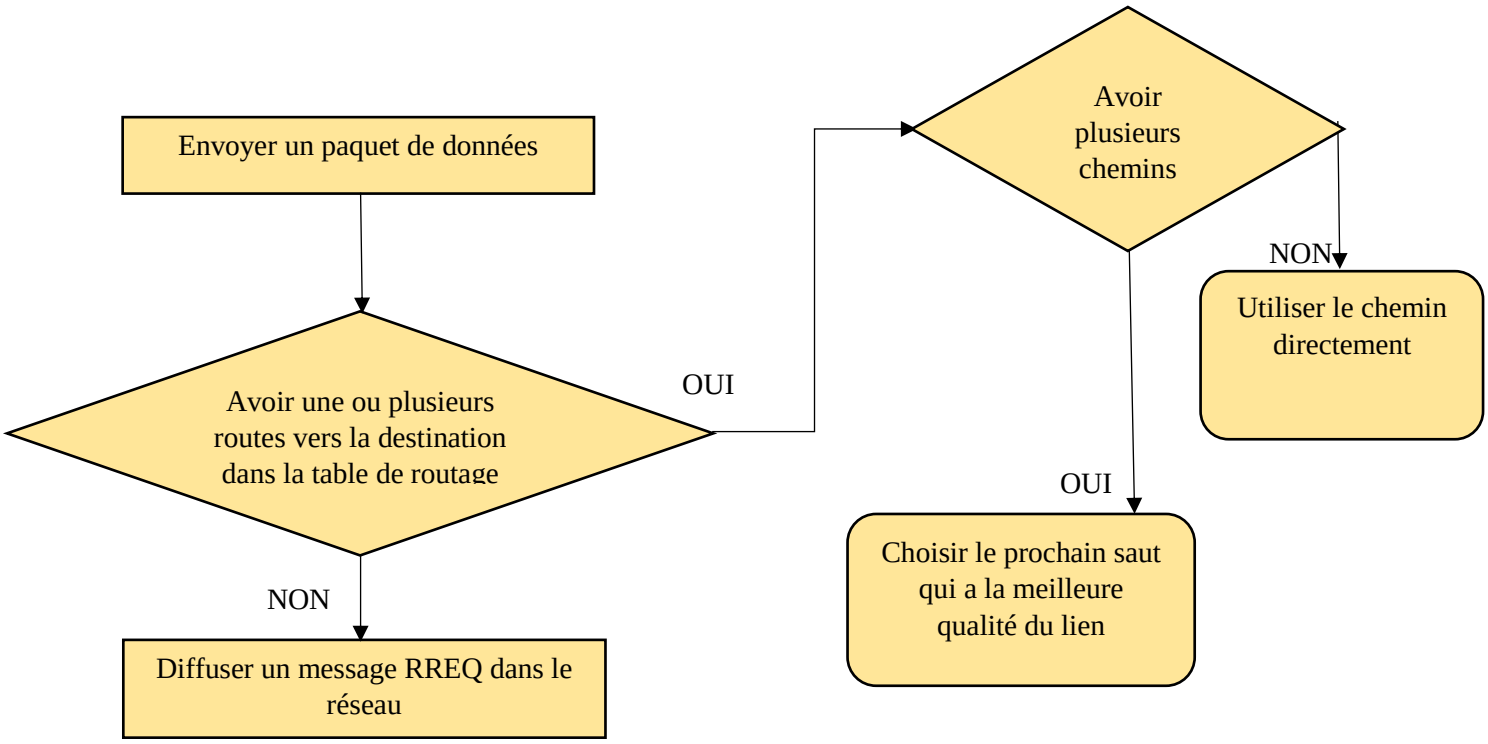


Figure 3.1 : Processus global du fonctionnement de µLOADng

La diffusion d'un message RREQ dans le réseau permet aux nœuds de recevoir plusieurs copies qui viennent des chemins différents. Nous avons utilisé ces messages dupliqués pour construire plusieurs routes.

Dans une première phase le nœud récepteur vérifie le message RREQ reçu, s'il trouve qu'il est son propre message RREQ, il l'ignore directement. Mais s'il confirme que ce n'est pas le sien il consulte sa table de routage pour voir s'il dispose d'une entrée vers la source de ce message. Pour qu'un nouveau chemin soit ajouté à la table de routage le nœud récepteur analyse si ce chemin répond aux conditions suivantes :

- La disjonction qui se fait en comparant entre **Precursors_List** de la table de routage et **Precursors_List** du message RREQ.
- Le cout approximatif par rapport aux chemins existants en utilisant le champ **Adv_hop_count**.

Dans une deuxième phase, le nœud récepteur vérifie s'il est un nœud intermédiaire ou un nœud destination. Pour ce faire, il compare son adresse avec l'adresse de destination du message RREQ. S'ils sont les mêmes adresses c'est un nœud destination et il est un nœud intermédiaire si ces adresses sont différentes.

Quand le nœud récepteur est un nœud intermédiaire, il met à jour le champ **First_Hop** du message RREQ après avoir vérifié que l'émetteur est son voisin. Puis il ajoute son adresse dans **Precursors_List** du message RREQ et continue à le diffuser. Et quand le nœud récepteur est un nœud destination, il répond par un message RREP.

L'organigramme ci-dessous illustre les opérations effectuées par un nœud à la réception d'un message RREQ (Dans la figure 3.2).

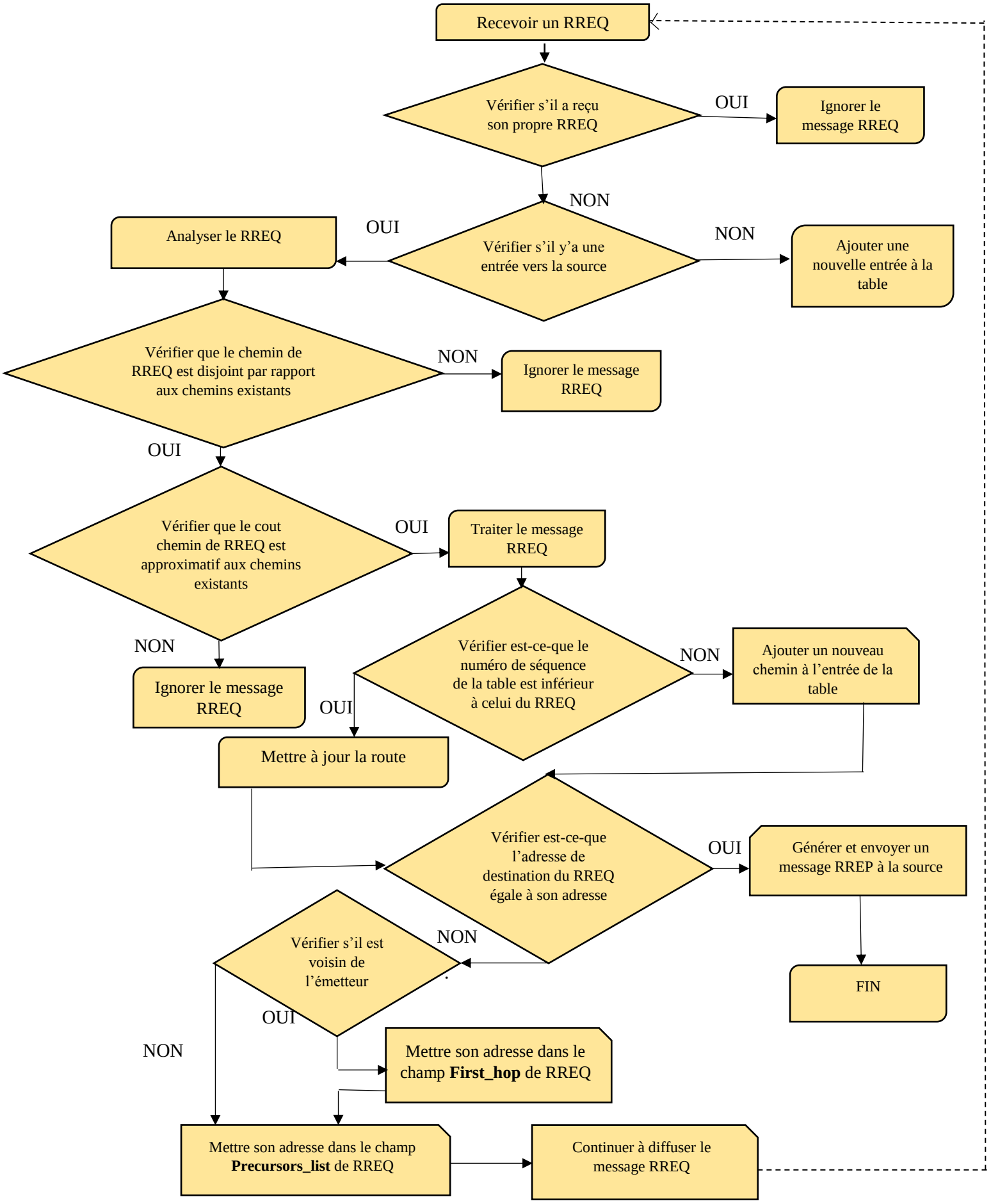


Figure 3.2 : Acheminement d'un message RREQ dans µLOADng

Dans notre extension du protocole, le nœud destination répond à tous les message RREQ qui viennent des chemins disjoints par des messages RREP. Nous avons exigé que chaque message RREP achemine son propre message RREQ car dans notre proposition le traitement de la disjonction est effectué au niveau du nœud destination dans le but de minimiser le traitement et réduire la surcharge dans le réseau.

Si le message RREP change le chemin de retour cela risque que la source du message RREQ construit des chemins non disjoints. Dans ce cas, le champ **First_Hop** est utilisé pour assurer l'acheminement du massage RREP vers la même route traversée par son massage RREQ.

L'organigramme ci-dessous résume les opérations effectuées par un nœud à la réception d'un message RREP (Dans la figure 3.3).

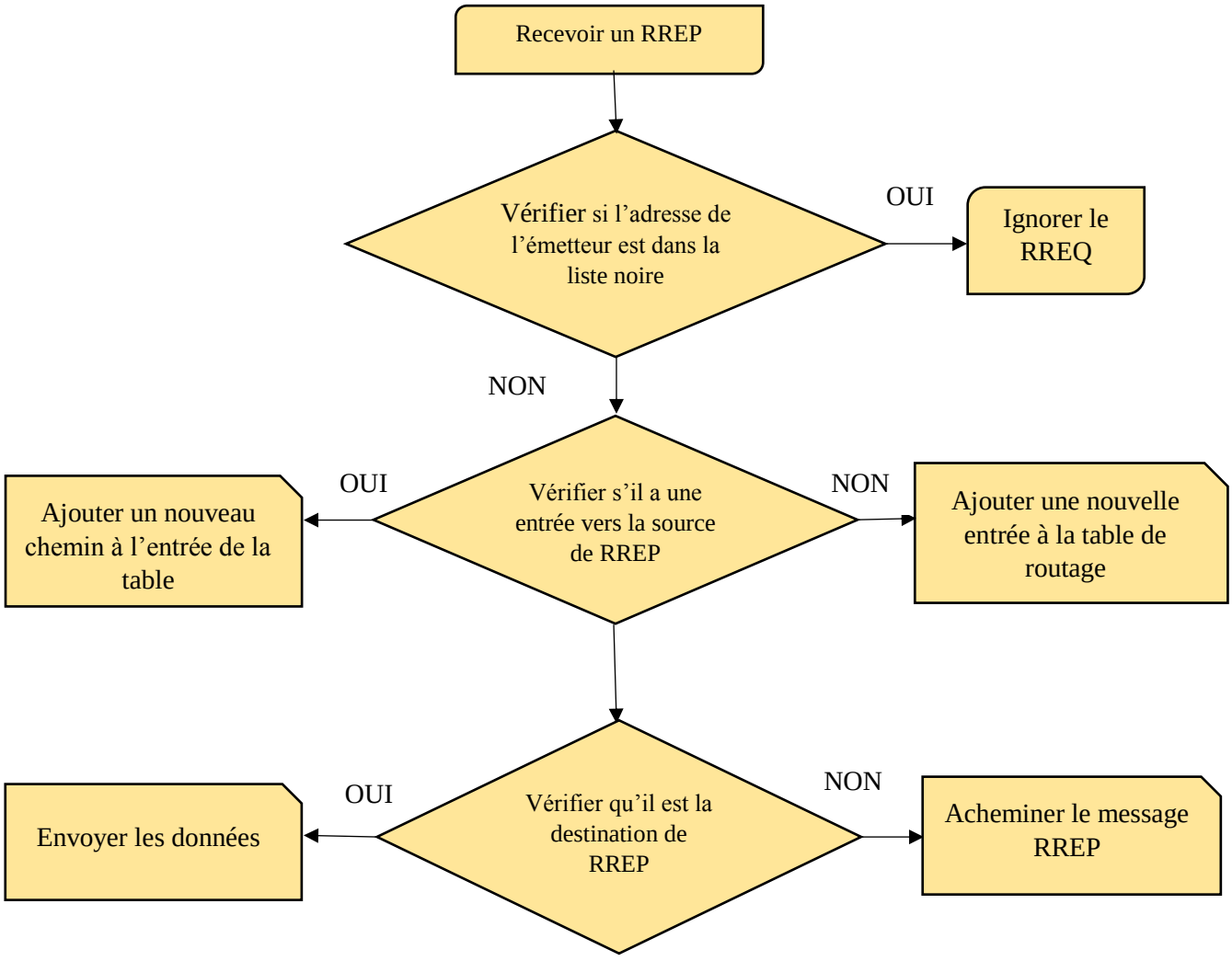


Figure 3.3 : Acheminement d'un message RREP dans µLOADng

3.2.3 Estimation de la qualité du lien

Nous avons suggéré l'utilisation de feedback de la couche MAC pour estimer la qualité de la liaison et pour la mettre à jour en permanence. Cette mise à jour dépend essentiellement des messages de données et de contrôle transmis.

La couche de contrôle d'accès au support MAC est responsable de transmettre les messages au prochain saut via un canal partagé, de détecter les erreurs de transmission et de retransmettre les données en cas d'échec.

Dans notre protocole, nous avons utilisé la métrique ETX où la qualité d'un lien est estimée par rapport au nombre de retransmission effectuée au niveau de la couche MAC. ETX calcule la qualité de la liaison d'un saut entre deux nœuds voisins.

L'équation 3.1 ci-dessous montre le calcul de la valeur de ETX.

$$ETX_{new} = (num_{Tx} * \alpha) + ETX_{old} * (1 - \alpha) \qquad (3.1)$$

- num_{tx} : Nombre de retransmissions
- α : Détermine dans quelle mesure l'ancienne valeur ETX contribue au calcul de l'e actuel valeur de ETX
- ETX_{new} : Nouvelle valeur de ETX
- ETX_{old} : Ancienne valeur de ETX

3.2.4 Allocation de trafic

Contrairement au protocole de routage qui utilise tous les chemins trouvés simultanément, notre protocole sélectionne à chaque fois le chemin qui a la meilleure qualité du lien pour la transmission. Ceci donne la possibilité à chaque chemin d'être actif quand il est choisi. Par conséquent, notre proposition assure la répartition de la charge sur tous les chemins trouvés grâce au basculement effectué entre ces derniers et évite d'avoir des paquets redondants dans le réseau.

3.3 Conclusion

Dans ce chapitre, nous avons présenté les détails de notre conception, en évoquant les principes de base greffés au protocole de base LOADng afin de le rendre multichemin. Pour concevoir notre solution, nous avons commencé par la restructuration de la table de routage, puis nous avons changé le processus de découverte des chemins. L'estimation de la qualité du lien est utilisée pour choisir le meilleur chemin. Le prochain chapitre abordera en détail l'aspect implémentation de notre solution.

Chapitre 04 : L'implémentation du protocole multichemin

4.1 Introduction

Ce présent chapitre couvre tous les détails relatifs à l'aspect implémentation de notre extension. Pour ce faire, nous avons eu recours au simulateur Cooja et le système d'exploitation Contiki qui nous ont permis de simuler notre extension du protocole. Dans ce qui suit, nous présenterons le simulateur Cooja et le système d'exploitation Contiki. Nous aborderons également d'une manière détaillée la démarche suivie pour implémenter notre protocole.

4.2 Aperçu sur le système d'exploitation Contiki

ContikiOS est un système d'exploitation open source léger écrit en langage C. Contiki a été créé par Adam Dunkels en 2002 [49], il est conçu pour être utilisé dans les nœuds de capteurs et les petits périphériques soumis à des contraintes de mémoire, d'alimentation et de traitement. Contiki fournit des fonctionnalités standards tels que les threads, les timers, le générateur de nombres aléatoires, l'horloge, un système de fichiers et un shell de ligne de commande. Il inclut une pile IPv4, les supports TCP et UDP, ainsi que la pile de communication radio Rime. Ses dernières versions prennent en charge la pile IPv6 [7].

Rime est une pile de communication légère conçue pour les radios à faible puissance. Elle fournit une large gamme de primitives de communication adaptées à la mise en œuvre d'applications liées à la communication ou de protocoles réseau [25].

La pile réseau mise en œuvre dans *ContikiOS* diffère légèrement du modèle à 5 couches adopté dans TCP / IP. Entre la couche physique et la couche réseau, nous avons 3 couches différentes : Framer, Radio Duty Cycle (RDC) et Medium Access Control (MAC) [50], voir la figure 4.1.

Figure 4.1 : L'organisation des couches dans ContikiOS [50]

- **La couche MAC (** *Medium Access Control***)** Contiki fournit deux protocoles mac ***csma*** et ***nullmac.*** ***nullmac*** ne fait aucun traitement au niveau de la couche mac alors que ***csma*** programme une retransmission de paquet si la couche radio détecte une collision ou absence d'un accusé de réception.
- **La couche RDC (*Radio Duty-Cycle*)** implémente un mécanisme qui prend en charge la période de sommeil des nœuds (allumer et éteindre l'émetteur-récepteur radio) pour économiser de l'énergie. Il s'agit de la couche la plus importante car c'est à elle qu'il incombe de décider du moment exact où les paquets seront transmis et de s'assurer que le noeud est réveillé lors de la réception des paquets.
- **La couche Framer** se compose d'un ensemble de fonctions auxiliaires appelées avant la transmission d'un paquet et après leur réception. Il existe deux types de framer : ***framer-nullmac*** qui devrait être utilisé avec ***nullmac*** (couche MAC) et ***framer-802154*** qui crée et analyse des paquets compatibles avec la norme IEEE 802.15.4

  La figure 4.2 ci-dessous montre les protocoles implémentés dans ces différentes couches.

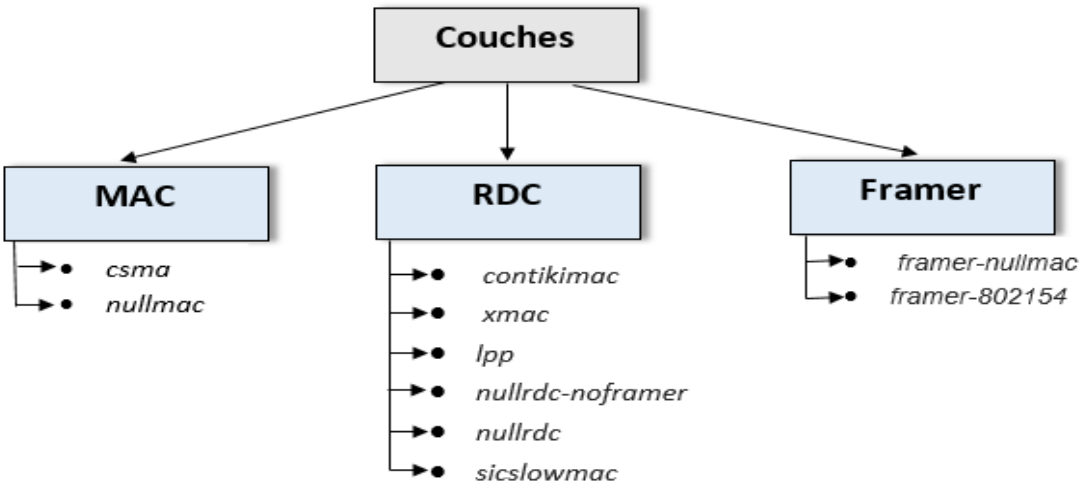


Figure 4.2 : protocoles implémenter dans les couches MAC, RDC et Framer

4.3 Aperçu sur le simulateur Cooja

Cooja (*Contiki Os Java Simulation*) est un simulateur implémenté en Java conçu pour simuler des réseaux de capteurs exécutant le système d'exploitation Contiki. Il permet l'émulation des plates-formes matérielles réelles (Skymote, Z1mote, MicaZ ...etc.) et une simulation simultanée au niveau du réseau et du système d'exploitation.

COOJA simule des réseaux de nœuds de capteurs où chaque nœud peut être d'un type différent. Diffèrent non seulement par le logiciel embarqué, mais aussi il est capable de simuler des nœuds non Contiki, tels que des nœuds implémentés en Java ou même des nœuds exécutant un autre système d'exploitation. Ceci permet de simuler les réseaux hétérogènes. Il est aussi flexible dans le sens que de nombreuses parties du simulateur peuvent être facilement remplacées ou étendues avec des fonctionnalités supplémentaires. Des exemples de pièces pouvant être étendues incluent le support radio simulé et le matériel de nœud simulé [51].
La figure 4.3 montre l'interface de simulation qui se compose de cinq fenêtres : Network, Mote Output, Timeline, Simulation Control, Note.

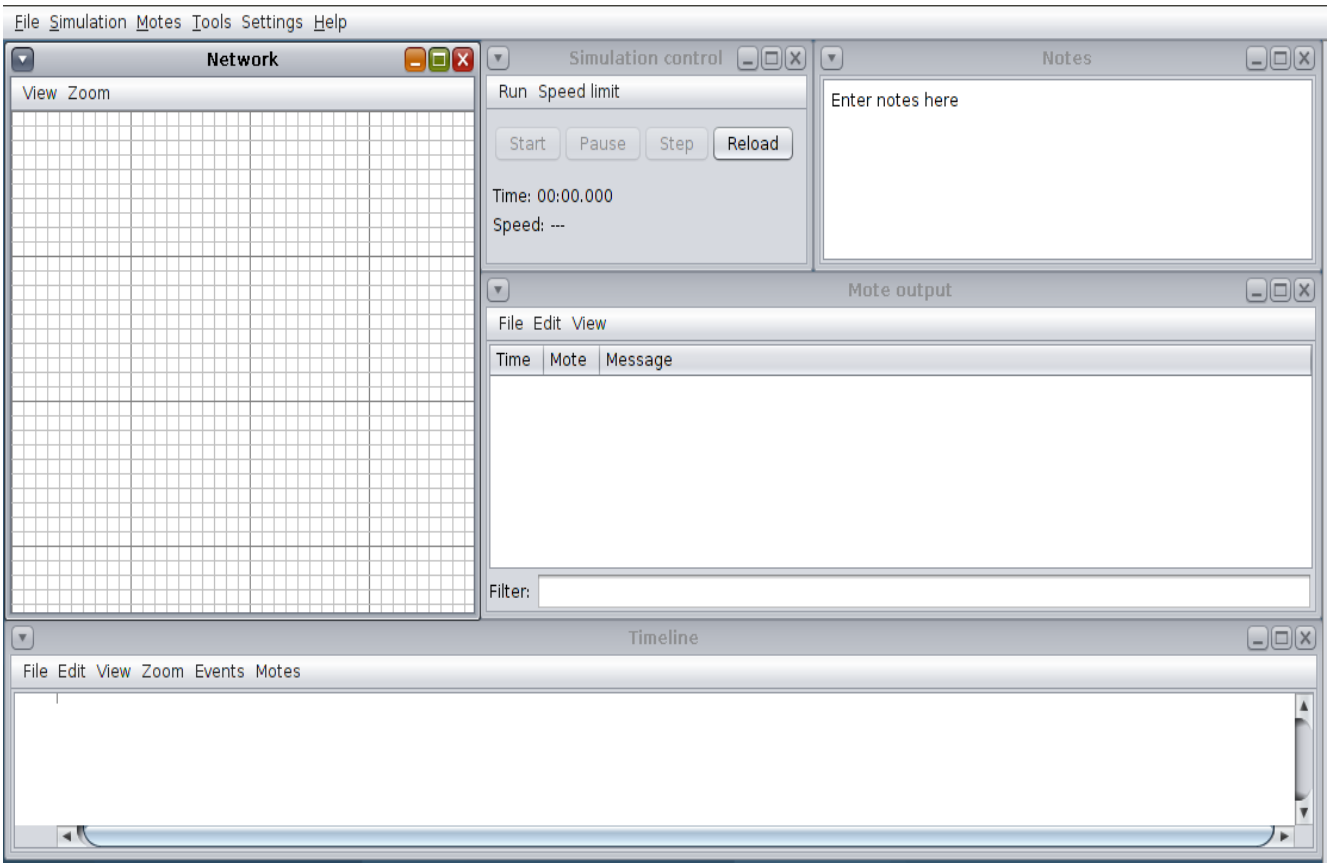


Figure 4.3 : L'interface de simulation

- **La fenêtre Network** : est l'endroit où nous pouvons déployer les nœuds pour former la topologie. Cette zone permet de visualiser l'état de chaque nœud du réseau (Type, Identifiant, position …).
- **La fenêtre Mote Output** : est l'endroit où nous obtenons le résultat de notre simulation, où nous pouvons voir tous ce qui se passe entre les nœuds du réseau et les messages échangés entre eux.
- **La fenêtre Timeline :** c'est l'endroit où on peut visualiser les communications radio (transmission, réception, collision) ainsi que les états réveillés et endormis des nœuds capteurs par rapport au temps.
- **La fenêtre Note :** permet de garder toutes les notes importantes concernant la simulation.

4.4 Détails de l'implémentation de notre extension multichemin

Le protocole LOADng a été implémenté pour s'exécuter en tant que protocole de routage dans Contiki OS. L'implémentation de LOADng se trouve dans **core/net/rime**.
Dans cette partie, nous présenterons les changements majeurs apportés au code source du protocole LOADng afin de le rendre multichemin. La démarche est comme suivie :

4.4.1 Restructuration de la table de routage

Afin de pouvoir introduire plusieurs chemins vers la même destination dans la table de routage :

- Tous d'abord nous avons modifié la structure de la table de routage au niveau du fichier **core/net/rime/route.h** (voir la figure 4.4)

```
75 struct dist_tuple {
76         uint8_t route_cost;     // cout du chemin
77         struct collect_link_estimate le;  // la qualité du lien
78         uint8_t weak_links:4;
79         uint8_t padding:4;      //not used, initialized to 0;
80 };
81 struct route_list {   // route liste
82         struct route_list* next;
83         rimeaddr_t R_last_addr;  // adresse du dernier saut
84         rimeaddr_t R_next_addr; //adresse du prochain saut
85         struct dist_tuple R_dist;
86 };
87 //same as struct routing_tuple
88 struct route_entry {
89         struct route_entry* next;
90         rimeaddr_t R_dest_addr;
91         struct route_list *list; // la liste des chemins
92         uint8_t adv_hop_count; // le nombre de saut annoncé
93         rimeaddr_t list_p[SIZE]; // la liste des precurseur
94         uint16_t R_seq_num;
95         clock_time_t R_valid_time;
96         uint8_t R_metric:4;     //R_metric: type of routing metric. 0, by default, means using hop-count
97         uint8_t padding:4;      //not used, initialized to 0;
98 };
```

Figure 4.4 : Restructuration de la table de routage

- Les changements que nous avons apportés à la structure de la table de routage nous ont poussés à modifier les fonctions ***route_add (), route_remove ()*** qui se trouvent dans le fichier **core/net/route.c**. La première fonction permet d'introduire une nouvelle entrée à la table de routage. La deuxième permet la suppression d'une entrée dans la table, dans le cas d'un multichemin la suppression est effectuée si tous les chemins deviennent inactifs (Voir les figures 4.5 et 4.6).

```
struct routing_entry *
route_add(const rimeaddr_t *dest,const rimeaddr_t *last_hop,const rimeaddr_t  tb[],const rimeaddr_t *next_hop,uint16_t weak_links,uint16_t route_cost, uint16_t seqno)
{
	struct route_entry *e;
	struct route_list *r;
	int i=0;
	/* Avoid inserting duplicate entries. */
	e = route_lookup(dest);
	if(e != NULL && rimeaddr_cmp(&e->list->R_next_addr, next_hop)) {
		list_remove(route_set, e);
	} else {
		/* Allocate a new entry or reuse the oldest entry with highest cost. */
		e = memb_alloc(&route_set_mem);
		if(e == NULL) {
		  /* Remove oldest entry.  XXX */
		  e = list_chop(route_set);
		  PRINTF("route_add: removing entry to %d.%d \n",
			e->R_dest_addr.u8[0], e->R_dest_addr.u8[1]);}
	}
	for(i=0;i<SIZE;i++)
	   rimeaddr_copy(&e->list_p[i], &tb[i]); //remplir la liste des precurseurs
	rimeaddr_copy(&e->R_dest_addr, dest); // remplir le champ destination
	e->R_seq_num=seqno;
	e->R_metric = METRICS;
	r = memb_alloc(&route_list_mem);   // allouer un espace pour le champ route_list
	rimeaddr_copy(&r->R_next_addr, next_hop); // remplirle champ next_hop
	rimeaddr_copy(&r->R_last_addr, last_hop); // remplir le champ last_hop
	collect_link_estimate_new(&r->R_dist.le); // initialiser la qualité de lien
	r->R_dist.route_cost=route_cost;          //remplir le champ route_cost
	r->R_dist.weak_links=weak_links;
	r->next=NULL;
	e->adv_hop_count = route_cost;          //remplir le champ adv_hop_count
	e->list=r;
	/* New entry goes first. */
	list_push(route_set, e); // ajouter a la table de routage
	 PRINTF("route_add: new entry to %d.%d with nexthop %d.%d Cost: %d ETX accumulator: %d  seq num : %d \n",
		e->R_dest_addr.u8[0], e->R_dest_addr.u8[1],
		e->list->R_next_addr.u8[0], e->list->R_next_addr.u8[1],
		e->list->R_dist.route_cost,
		collect_link_estimate(&r->R_dist.le),
		e->R_seq_num);
	return (struct routing_entry*)e;
}
```

Figure 4.5 : Insertion d'une nouvelle entrée à la table de routage

```
void
route_remove(struct route_entry *e)
{
    struct route_list *rl;
     for (rl =e->list ; rl != NULL ; rl = rl->next)
          route_list_remove (e,rl);
     if (e != NULL) {
            PRINTF("\nroute_remove: removing entry to %d.%d \n",
                  e->R_dest_addr.u8[0], e->R_dest_addr.u8[1]);
            list_remove(route_set, e);
            memb_free(&route_set_mem, e);

        }
}
```

Figure 4.6 : Suppression d'une entrée dans la table de routage

- Par la suite nous avons ajouté les fonctions ***route_list_add (), route_list_remove ()*** dans le fichier **core/net/route.c** qui assurent l'insertion et la suppression d'un chemin à une destination déjà existante dans la table de routage. Cette étape présente l'une des étapes clé de la mise en place de notre protocole multi chemin (Voir les figures 4.7 et 4.8).

```
struct route_list *
route_list_add(const rimeaddr_t *dest,const rimeaddr_t *last_hop,const rimeaddr_t tab[],const rimeaddr_t *next_hop,uint16_t weak_links,uint16_t route_cost)
{
    struct  route_list *r, *rr;
    struct route_entry *e;
    int i =0, j=0;
     rimeaddr_t addrr;
         addrr.u8[0] = 0;
         addrr.u8[1] = 0;
    e=route_lookup(dest); //chercher la destination dans la table de routage
     while(!rimeaddr_cmp(&e->list_p[i],&addrr))
              i++;
     while (j<SIZE && (!rimeaddr_cmp(&tab[j],&addrr)))
       {
         rimeaddr_copy(&e->list_p[i], &tab[j]); // remplir la liste des precurseurs
         i++;
         j++;
       }
     r = memb_alloc(&route_list_mem); // allouer un espace memoire pour le champ route liste
     if (e->adv_hop_count< route_cost)  // mettre à jour adv_hop_count si c'est nécessaire
     e->adv_hop_count=route_cost;
     r->R_dist.weak_links=weak_links;
     e->R_valid_time = 0;                    // initialiser R_valid_time pour la nouvelle route
     rimeaddr_copy(&r->R_next_addr, next_hop); // remplir le champ next_hop
     rimeaddr_copy(&r->R_last_addr, last_hop);   // remplir le champ last_hop
     r->R_dist.route_cost=route_cost;           // remplir le champ route_cost
     collect_link_estimate_new(&r->R_dist.le);  // initialiser la qualité du lien
     // ajouter à l'entrée de la table de routage
     rr = e->list;
     e->list=r;
     r->next=rr;

   printf("Route list add : Destination %d.%d  Next address %d.%d  Cost: %d  ETX  %d \n",
          e->R_dest_addr.u8[0],e->R_dest_addr.u8[1],
          rr->R_next_addr.u8[0],rr->R_next_addr.u8[1],
          e->list->R_dist.route_cost,
          collect_link_estimate(&r->R_dist.le));
 return (struct route_list*)r;
}
```

Figure 4.7 : Insertion d'un nouveau chemin à une destination déjà existante

```
468 void route_list_remove ( struct route_entry *e , struct route_list *r)
469 {
470 struct route_entry *ee;
471 struct route_list *rr,*p,*q,*t;
472   for ( rr=e->list ; rr != NULL ; rr = rr->next)
473    {
474     if(rimeaddr_cmp(&rr->R_next_addr,&r->R_next_addr))
475        if(rr->next == NULL)
476         {
477         list_remove(e->list, rr);
478         memb_free(&route_list_mem, rr);
479         e->list=NULL;
480         printf("\n Removing the head of the route list");
481         }
482     else
483        {
484         t=rr;
485         q=rr->next;
486         list_remove(e->list, t);
487         memb_free(&route_list_mem,t);
488         p->next=q;
489         printf("\nRemove route with next hop = %d.%d ",rr->R_next_addr.u8[0],rr->R_next_addr.u8[1]);
490        }
491     }
492     p=rr;
493  }
494 }
```

Figure 4.8 : Suppression d'un chemin pour une destination dans la table

4.4.2 Découverte des chemins

Afin de découvrir des chemins multiples à nœuds disjoint plusieurs fonctions ont été modifiées au niveau du fichier **core/net/rime/route-discovery.c** :

- Nous avons ajouté les champs *First_hop* et *precursor_list* dans le paquet RREQ (Voir la figure 4.9).

```
58 typedef struct general_message{
59       //uint8_t addr-length:4;
60       uint8_t type;
61       uint8_t seqno;
62       /*if metric_type set to 0 hop_count is used otherwise route_metric is used*/
63       uint8_t metric_type;
64       uint32_t route_metric;
65       uint8_t hop_limit;
66       uint8_t hop_count;
67       rimeaddr_t originator;
68       rimeaddr_t destination;
69       rimeaddr_t precursor_list[SIZE];
70       rimeaddr_t first_hop;
71       uint8_t ackrequired;
72 }rreq_message,rrep_message;
```

Figure 4.9 : Ajout des champs first_hop et precursor_list au paquet RREQ

- Nous avons apporté des modifications au niveau de la fonction ***rreq_msg_received ()*** pour permettre à chaque nœud, qu'il soit nœud destination ou intermédiaire, de traiter tous les messages RREQ dupliqués afin de construire plusieurs chemins inverses vers la source et de mettre à jour les champs **First_Hop** et **Precursors_List** s'il s'agit d'un nœud intermédiaire (Voir les figures 4.10 et 4.11).

```
371
372  ret_val = valid_check(msg, from);  // vérifier la disjonction et detecter les boucles
373  if(ret_val!=0){
374  »       return ret_val;
375  }
376
377  rt = route_lookup(&msg->originator);  // verifier s'il existe une entrée dans la table de routage
378  if(rt==NULL){  // pas de entrée pour cette destination
379           //remplir les informations nécessaire à partir du message RREQ
380          rimeaddr_copy(&last_addr,&new_msg.first_hop);
381          rimeaddr_copy(&next_addr,from);
382          weak_links = 0;
383          route_cost = msg->hop_count + 1;
384          route_add(&msg->originator,&last_addr, msg->precursor_list,&next_addr,weak_links,route_cost ,msg->seqno); // ajouter une nouvelle entrée à table de routage
385  }
386  else{   // il y'a une entrée dans la table de routage
387    if (rt->R_seq_num < msg->seqno)
388      {
389          rimeaddr_copy(&last_addr,&new_msg.first_hop);
390          rimeaddr_copy(&next_addr,from);
391          weak_links = 0;
392          route_cost = msg->hop_count + 1;
393          route_remove(rt);
394          route_add(&msg->originator,&last_addr,msg->precursor_list,&next_addr,weak_links,route_cost ,msg->seqno);
395      }
396     else
397      {
398          rimeaddr_copy(&last_addr,&msg->first_hop);
399          rimeaddr_copy(&next_addr,from);
400          weak_links = 0;
401          route_cost = msg->hop_count + 1;
402          route_list_add(&msg->originator,&last_addr,msg->precursor_list,&next_addr,weak_links,route_cost);// ajouter un nouveau chemin à une destination dèja existante
403
404       }
405
406  }
```

Figure 4.10 : Traiter tous les messages RREQ dupliqués

```
if(rimeaddr_cmp(&msg->destination, &rimeaddr_node_addr)) //vérifier si ce noeud est la destination
{
  PRINTF("rreq_msg_received: %d.%d: route_packet_received: route request for our address\n",rimeaddr_node_addr.u8[0], rimeaddr_node_addr.u8[1]);
  PRINTF("rreq_msg_received: from %d.%d rssi %d lqi %d\n",from->u8[0], from->u8[1],packetbuf_attr(PACKETBUF_ATTR_RSSI),packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY));
        new_msg.type = RREP_TYPE; //
        new_msg.metric_type = 0;
        new_msg.route_metric = 0;
        new_msg.seqno = rrep_seqno;
        rrep_seqno++;
        new_msg.hop_count = 0;
        new_msg.hop_limit = MAX_HOP_LIMIT;
        rimeaddr_t temp_dest;
        rimeaddr_copy(&temp_dest,&new_msg.destination);
        rimeaddr_copy(&new_msg.first_hop,&msg->first_hop); //  remplir le champ first_hop
        rimeaddr_copy(&new_msg.destination,&new_msg.originator);
        rimeaddr_copy(&new_msg.originator,&temp_dest);
   send_rrep(c, &new_msg);      // envoyer un message RREP
  return SENDREP; /* Don't continue to flood the rreq packet. */ }
  else {   // si ce noeud n'est pas la destination
  PRINTF("rreq_msg_received: from %d.%d rssi %d lqi %d\n",from->u8[0], from->u8[1],packetbuf_attr(PACKETBUF_ATTR_RSSI),packetbuf_attr(PACKETBUF_ATTR_LINK_QUALITY));
  if(rimeaddr_cmp(&msg->first_hop,&addrr)) // si le champ first_hop est vide
        rimeaddr_copy(&new_msg.first_hop,&rimeaddr_node_addr); //ce noeud est le first_hop
  else // si ce noeud n'est pas la destination
        rimeaddr_copy(&new_msg.first_hop,&msg->first_hop);
  while(i<SIZE)
       {if(rimeaddr_cmp(&msg->precursor_list[i],&addrr)&&(copy==0)){
        rimeaddr_copy(&new_msg.precursor_list[i],&rimeaddr_node_addr);// mettre l'adresse de ce noeud dans precursor_list
        copy=1;}
       else {
        rimeaddr_copy(&new_msg.precursor_list[i],&msg->precursor_list[i]);
        i++;}
        rimeaddr_copy(&new_msg.destination,&msg->destination);
        rimeaddr_copy(&new_msg.originator,&msg->originator);
       if(msg->hop_count < MAX_HOP_COUNT && msg->hop_limit >0){
          new_msg.hop_count = msg->hop_count+1;
          new_msg.hop_limit = msg->hop_limit - 1;
          new_msg.route_metric = msg->hop_count + 1;
          new_msg.seqno = msg->seqno;
          new_msg.metric_type = msg->metric_type;
          new_msg.type = RREQ_TYPE;
          send_rreq(c,&new_msg); //continuer à diffuser le message RREQ
          return FORWARD;
  }
```

Figure 4.11 : mettre à jour les champs **Precursors_List** et **First_Hop**

- Nous avons apporté des modifications au niveau de la fonction ***send_rrep ()*** pour permettre à la destination de répondre à chaque message RREQ par un message RREP. Nous avons utilisé le champs *First hop* pour que chaque message RREP achemine la même route traversée par le massage RREQ (Voir la figure 4.12).

```
static void
send_rrep(struct route_discovery_conn *c, rrep_message *input)
{
        struct  route_list *r;   // déclarer une variable de type route_list
        struct route_entry *rt;
        rrep_message *msg;
        packetbuf_clear();
        msg = packetbuf_dataptr();
        packetbuf_set_datalen(sizeof( rrep_message));
        msg->type = RREP_TYPE;
        msg->metric_type = input->metric_type;
        msg->route_metric = input->route_metric;
        msg->seqno = input->seqno;
        msg->hop_limit = MAX_HOP_LIMIT;
        rimeaddr_copy(&msg->destination,&input->destination);
        rimeaddr_copy(&msg->first_hop,&input->first_hop); // remplir le champ first_hop
        rimeaddr_copy(&msg->originator,&input->originator);
        if(rimeaddr_cmp(&rimeaddr_node_addr,&input->originator))
          msg->hop_count = 0;
        else
        msg->hop_count =input->hop_count;
        rt = route_lookup(&msg->destination);
      if(rt != NULL)
        {
              r=rt->list ; // la liste de tous les chemins
             for(r = rt->list; r != NULL; r = r->next)   // parcourir toute la liste des chemins
             {
                if(rimeaddr_cmp(&input->first_hop, &r->R_last_addr)) // utiliser le first_hop pour le bon acheminement des message RREP
                  {
                    unicast_send(&c->rrepconn, &r->R_next_addr); // envoyer le message RREP
                    PRINTF("%d.%d Send_RREP , Originator %d.%d ,  Destination  %d.%d Via %d.%d \n",
                    rimeaddr_node_addr.u8[0], rimeaddr_node_addr.u8[1],
                    msg->originator.u8[0], msg->originator.u8[1],
                    msg->destination.u8[0],msg->destination.u8[1],
                    r->R_next_addr.u8[0],r->R_next_addr.u8[1]);
                    }
                 }
}
        else {
              PRINTF("send_rrep: no route entry from %d.%d to %d.%d\n",
                    rimeaddr_node_addr.u8[0], rimeaddr_node_addr.u8[1],
                    msg->destination.u8[0],msg->destination.u8[1]);
         }
```

Figure 4.12 : Répondre à chaque message RREQ via le chemin qu'il a traversé

- Nous avons apporté des modifications au niveaux de la fonction ***rrep_msg_received ()*** pour permettre au nœuds récepteur de RREP de traiter tous les messages RREP reçus (Voir la figure 4.13).

```
503         route_add(&msg->originator,&last_addr,msg->precursor_list,&next_addr,weak_links,route_cost ,msg->seqno); /* ajouter une nouvelle ent à la table si
504                                                                                                                     il y'a pas d'une entrée dans la table de routage */
505       }
506        else
507       {
508        rimeaddr_copy(&last_addr,&msg->first_hop);
509        rimeaddr_copy(&next_addr,from);
510        weak_links = 0;
511        route_cost = msg->hop_count + 1;
512        route_list_add(&msg->originator,&last_addr,msg->precursor_list,&next_addr,weak_links,route_cost); // ajouter un chemin à une destination existante
513       }
514   }
515   if(!rimeaddr_cmp(&msg->destination, &rimeaddr_node_addr)) { // si ce noeud est la destination
516         new_msg.hop_count = msg->hop_count + 1;
517         new_msg.hop_limit = msg->hop_limit - 1;
518         new_msg.seqno = msg->seqno;
519         new_msg.route_metric = msg->hop_count + 1;
520         new_msg.type = msg->type;
521         new_msg.metric_type = msg->metric_type;
522         rimeaddr_copy(&new_msg.destination,&msg->destination);
523         rimeaddr_copy(&new_msg.originator,&msg->originator);
524         rimeaddr_copy(&new_msg.first_hop,&msg->first_hop); // remplir le champ first_hop
525         send_rrep(c, &new_msg); // cont a acheminer le message RREP
526       return SENDREP; /* Don't continue to flood the rreq packet. */
527   }else // si ce noeud est la destination
528     {
529      PRINTF("rrep_msg_received: rrep for us!\n");
530      rrep_pending = 0;
531      ctimer_stop(&c->t);
532      if(c->cb->new_route) {
533        rimeaddr_t originator;
534        rimeaddr_copy(&originator, &msg->originator);
535        c->cb->new_route(c, &originator); // la route est trouvée
536      }
537   }
538   return TRUE;
```

Figure 4.13 : Traiter tous les message RREP

- Pour empêcher les nœuds de traiter les requêtes qui viennent des chemins non disjoints ou qui forment une boucle, nous avons utilisé le champ *precursor_list* au niveau de la fonction ***valid_check ().***

```
144 int valid_check(struct general_message *input, const rimeaddr_t *from){
145       //address check is skipped;
146       struct route_entry *rt;
147       struct route_list *r;
148       struct blacklist_tuple *bl;
149
150       if(rimeaddr_cmp(&input->originator,&rimeaddr_node_addr)){
151             PRINTF("valid_check: Receive RREQ from itself \n");
152             return FALSE;
153       }
154
155       rt = route_lookup(&input->originator);
156       if(rt!=NULL){
157        int j;
158        for(j=0;j<SIZE;j++)
159            if(rimeaddr_cmp(&rimeaddr_node_addr,&input->precursor_list[j])){
160                PRINTF("valid_check: Loop \n");
161                return FALSE;
162             }
163        }
164       if((rt!=NULL)&&(rt->R_seq_num == input->seqno)&&((input->hop_count+1) - rt->adv_hop_count >= MAX_VAL)){
165             PRINTF("valid_check:I have a bater root \n");
166             return FALSE;
167        }
168       if(rt!=NULL){
169           int i=0, j;
170           rimeaddr_t addrr;
171           addrr.u8[0] = 0; addrr.u8[1] = 0;
172           while((i<SIZE) && (!rimeaddr_cmp(&input->precursor_list[i],&addrr))){
173           for(j=0;j<SIZE;j++)
174             if(rimeaddr_cmp(&input->precursor_list[i],&rt->list_p[j]))
175              {
176              PRINTF("valid_check: Non Disjoint Path \n");
177              return FALSE;
178              }
179            i++;
180         }
181        }
```

Figure 4.14 : La vérification de la disjonction

4.4.3 Estimation de la qualité de la liaison

- Afin de pouvoir récupérer le nombre total des transmissions nécessaires pour l'envoi d'un paquet, nous avons apporté des changements dans le fichier **core/net/mac/csma.c**

```
188 packet_sent(void *ptr, int status, int num_transmissions)
189 {
190   struct neighbor_queue *n;
191   struct rdc_buf_list *q;
192   struct qbuf_metadata *metadata;
193   clock_time_t time = 0;
194   mac_callback_t sent;
195   void *cptr;
196   int num_tx,tx;
197   int backoff_transmissions;
198   n = ptr;
199   if(n == NULL) {
200     return;
201   }
202   switch(status) {
203   case MAC_TX_OK:
204   case MAC_TX_NOACK:
205     n->transmissions++;
206     break;
207   case MAC_TX_COLLISION:
208     n->collisions++;
209     break;
210   case MAC_TX_DEFERRED:
211     n->deferrals++;
212     break;
213   }
214   for(q = list_head(n->queued_packet_list);
215       q != NULL; q = list_item_next(q)) {
216     if(queuebuf_attr(q->buf, PACKETBUF_ATTR_MAC_SEQNO) ==
217        packetbuf_attr(PACKETBUF_ATTR_MAC_SEQNO)) {
218       break;
219     }
220   }
221   if(q != NULL) {
222     metadata = (struct qbuf_metadata *)q->ptr;
223   if(metadata != NULL) {
224       sent = metadata->sent;
225       cptr = metadata->cptr;
226       num_tx = n->transmissions;
227      tx=n->transmissions+n->collisions;   //récuperer le nombre total des transmissions
228       if(status == MAC_TX_COLLISION ||
229          status == MAC_TX_NOACK) {
```

Figure 4.15 : récupérer le nombre total des retransmissions

```
258         time = time + (random_rand() % (backoff_transmissions * time));
259         if(n->transmissions < metadata->max_transmissions) {
260           PRINTF("csma: retransmitting with time %lu %p\n", time, q);
261           ctimer_set(&n->transmit_timer, time,
262                      transmit_packet_list, n);
263           /* This is needed to correctly attribute energy that we spent
264              transmitting this packet. */
265           queuebuf_update_attr_from_packetbuf(q->buf);
266         } else {
267           PRINTF("csma: drop with status %d after %d transmissions, %d collisions\n",
268                  status, n->transmissions, n->collisions);
269           free_packet(n, q);
270           mac_call_sent_callback(sent, cptr, status, tx);
271         }
272       } else {
273         if(status == MAC_TX_OK) {
274           PRINTF("csma: rexmit ok %d\n", n->transmissions);
275         } else {
276           PRINTF("csma: rexmit failed %d: %d\n", n->transmissions, status);
277         }
278         free_packet(n, q);
279         mac_call_sent_callback(sent, cptr, status, tx);
280       }
281     }
```

Figure 4.16: retourner le nombre de retransmissions

- Après avoir récupéré le nombre de transmission, nous avons mis à jour la qualité du lien dans la table de routage au niveau du fichier **core/net/rime/rime.c**

```
146 static void
147 packet_sent(void *ptr, int status, int num_tx)
148 {
149   struct route_entry *rt;
150   struct route_list *r;
151   struct channel *c = ptr;
152   struct rime_sniffer *s;
153   rt = route_lookup(packetbuf_addr(PACKETBUF_ADDR_ERECEIVER)); // utiliser les information du buffer pour chercher l'entrée de la table de routage
154   switch(status) {
155   case MAC_TX_COLLISION: // si le message de données n'a pas été transmit à cause des collision
156     PRINTF("rime: collision after %d tx\n", num_tx);
157        if(rt != NULL)
158     for(r=rt->list;r!=NULL;r=r->next) //parcourir route_list
159        if(rimeaddr_cmp(&r->R_next_addr,packetbuf_addr(PACKETBUF_ADDR_RECEIVER))) // utiliser les information du buffer pour trouver le bon chemin
160          collect_link_estimate_update_tx_fail(&r->R_dist.le,num_tx); // mettre à jour la qualité du lien.
161     break;
162   case MAC_TX_NOACK:  // si le message de données n'a pas été transmit à cause d'absence des accusés de réceptions
163     PRINTF("rime: noack after %d tx\n", num_tx);
164        if(rt != NULL)
165     for(r=rt->list;r!=NULL;r=r->next) ////parcourir route_list
166        if(rimeaddr_cmp(&r->R_next_addr,packetbuf_addr(PACKETBUF_ADDR_RECEIVER))) // utiliser les information du buffer pour trouver le bon chemin
167          collect_link_estimate_update_tx_fail(&r->R_dist.le,num_tx); // mettre à jour la qualité du lien.
168     break;
169   case MAC_TX_OK: // si le message de données a été bien transmit
170     PRINTF("rime: sent after %d tx\n", num_tx);
171     if(rt != NULL)
172     for(r=rt->list;r!=NULL;r=r->next) // parcourir route_list
173        if(rimeaddr_cmp(&r->R_next_addr,packetbuf_addr(PACKETBUF_ADDR_RECEIVER))) // utiliser les information du buffer pour trouver le bon chemin
174          collect_link_estimate_update_tx(&r->R_dist.le,num_tx); // mettre à jour la qualité du lien
175     break;
176   default:
177     PRINTF("rime: error %d after %d tx\n", status, num_tx);
178   }
179  /* Call sniffers, pass along the MAC status code. */
180   for(s = list_head(sniffers); s != NULL; s = list_item_next(s)) {
181     if(s->output_callback != NULL) {
182       s->output_callback(status);
183     }
184   }
185   abc_sent(c, status, num_tx);
186 }
```

Figure 4.17 : Mettre à jour la qualité de lien

- La fonction qui met à jour la qualité du lien se trouve **core/net/rime/collect-link-estimate.c**

```
72  void
73  collect_link_estimate_update_tx(struct collect_link_estimate *le, uint8_t tx)
74  {
75    if(le == NULL) {
76      return;
77    }
78    if(tx == 0) {
79      return;
80    }
81    if(le != NULL) {
82      if(le->num_estimates == 0) {
83        le->etx_accumulator = tx * COLLECT_LINK_ESTIMATE_UNIT;
84      }
85      if(le->num_estimates < MAX_ESTIMATES) {
86        le->num_estimates++;
87      }
88      le->etx_accumulator = (((uint32_t)tx * COLLECT_LINK_ESTIMATE_UNIT) *
89                             COLLECT_LINK_ESTIMATE_ALPHA +
90                             le->etx_accumulator * (COLLECT_LINK_ESTIMATE_UNIT -
91                                                    COLLECT_LINK_ESTIMATE_ALPHA)) /
92        COLLECT_LINK_ESTIMATE_UNIT;
93    }
94  }
```

Figure 4.18 : Le calcul de la valeur ETX

4.4.4 Allocation de trafic

Après avoir conçu la table de routage avec le concept du multichemin.

- Tous d'abord nous avons modifié dans la fonction ***data_packet_forward*** dans le fichier **core/net/rime/mesh.c** pour pouvoir récupérer la liste complète des chemins.

```
86  struct route_list *
87  data_packet_forward(struct multihop_conn *multihop,
88  »       »           const rimeaddr_t *originator,
89  »       »           const rimeaddr_t *dest,
90  »       »           const rimeaddr_t *prevhop, uint8_t hops)
91  {
92    struct route_entry *rt;
93    struct mesh_conn *c = (struct mesh_conn *)
94      ((char *)multihop - offsetof(struct mesh_conn, multihop));
95
96    rt = route_lookup(dest); //chercher s'il y'a une entée dans la table de routage
97    if(rt == NULL) { // si non
98      if(c->queued_data != NULL) {
99        queuebuf_free(c->queued_data);// mettre le paquet dans buffer
100     }
101
102     PRINTF("data_packet_forward: queueing data, sending rreq\n");
103     c->queued_data = queuebuf_new_from_packetbuf();
104     rimeaddr_copy(&c->queued_data_dest, dest);
105     route_discovery_discover(&c->route_discovery_conn, dest, PACKET_TIMEOUT); // declancher le processus de decouverte de route
106
107     return NULL;
108   }
109   // si oui
110   return rt->list;   // retourner toute la liste des chemin
111 }
```

Figure 4.19 : récupération de la liste des chemins

- Après avoir récupéré toute la liste des chemins à partir de la table de routage, nous avons modifié dans les fonctions ***multihop_send ()*** et ***data_packet_received ()*** pour choisir le meilleur chemin à emprunter (Voir les Figures 4.20 et 4.21)

```
68  data_packet_received(struct unicast_conn *uc, const rimeaddr_t *from)
69  {
70    struct multihop_conn *c = (struct multihop_conn *)uc;
71   struct route_list *nexthops;
72    rimeaddr_t sender, receiver;
73     struct route_entry *rt;
74    rimeaddr_copy(&sender, packetbuf_addr(PACKETBUF_ADDR_ESENDER));
75    rimeaddr_copy(&receiver, packetbuf_addr(PACKETBUF_ADDR_ERECEIVER));
76    PRINTF("data_packet_received from %d.%d towards %d.%d len %d\n",
77  »       from->u8[0], from->u8[1],
78  »       packetbuf_addr(PACKETBUF_ADDR_ERECEIVER)->u8[0],
79  »       packetbuf_addr(PACKETBUF_ADDR_ERECEIVER)->u8[1],
80  »       packetbuf_datalen());
81    if(rimeaddr_cmp(packetbuf_addr(PACKETBUF_ADDR_ERECEIVER),
82  »       »       »       »       &rimeaddr_node_addr)) {
83      PRINTF("for us!\n");
84      if(c->cb->recv) {
85        c->cb->recv(c, &sender, from,
86  »       »       packetbuf_attr(PACKETBUF_ATTR_HOPS));}
87    } else {
88      if(c->cb->forward) {
89        packetbuf_set_attr(PACKETBUF_ATTR_HOPS,
90  »       »       »       packetbuf_attr(PACKETBUF_ATTR_HOPS) + 1);
91        nexthops = c->cb->forward(c, &sender, &receiver, // recuperer la liste des chemins
92  »       »       »       from, packetbuf_attr(PACKETBUF_ATTR_HOPS) - 1);}
93      if(nexthop) {
94       rt = route_lookup(&sender);
95       struct route_list *e;
96       uint16_t best =WORSE;
97       rimeaddr_t best_addr;
98         for(e = nexthop ; e != NULL; e =e->next)  // parcourir toute la liste pour trouver le lien avec la meilleur valeur ETX
99              {
100              if(collect_link_estimate_num_estimates(&e->R_dist.le) == 0)
101                {   best_addr=e->R_next_addr;
102                    best=collect_link_estimate(&e->R_dist.le);}
103               else
104                  if(collect_link_estimate(&e->R_dist.le) < best)
105                  {best_addr=e->R_next_addr;
106                   best=collect_link_estimate(&e->R_dist.le);}}
107        PRINTF("\n forwarding to %d.%d\n",best_addr.u8[0],best_addr.u8[1]);
108        unicast_send(&c->c, &best_addr); // acheminer le message de données
109  }}}
```

Figure 4.20 : choix de la route par les nœuds intermediaire

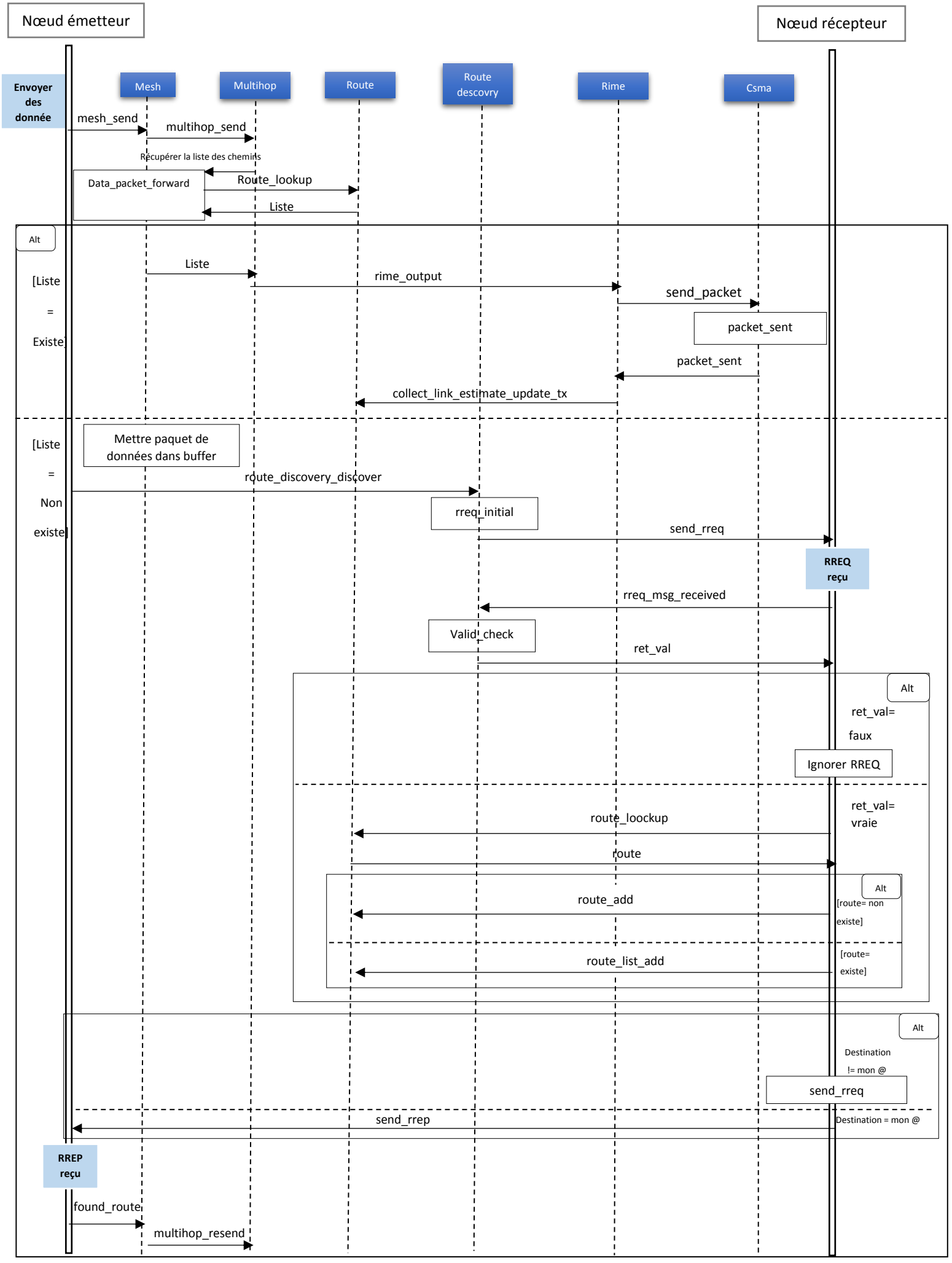
```
int
multihop_send(struct multihop_conn *c, const rimeaddr_t *to)
{
 struct route_list *nexthop , *e;

  if(c->cb->forward == NULL) {
    return 0;
  }
  packetbuf_compact();
  packetbuf_set_addr(PACKETBUF_ADDR_ERECEIVER, to); //mettre l'adresse de destination dans buffer
  packetbuf_set_addr(PACKETBUF_ADDR_ESENDER, &rimeaddr_node_addr); //mettre l'adresse source dans buffer
  packetbuf_set_attr(PACKETBUF_ATTR_HOPS, 1);
  nexthop = c->cb->forward(c, &rimeaddr_node_addr, to, NULL, 0);  // recuperer la liste des chemins
  if(nexthop == NULL) {
    PRINTF("multihop_send: no route\n");
    return 0;
  } else {
 int i;
 uint16_t best =WORSE;
  rimeaddr_t best_addr;
         for(e = nexthop ; e != NULL; e =e->next) //parcourir toute la liste des chemins
           {
            if(collect_link_estimate_num_estimates(&e->R_dist.le) == 0) // si il ya un nouveau chemin dans la table alors il faut le tester
               {
                PRINTF("\n multihop_send: sending data towards %d.%d\n",
                e->R_next_addr.u8[0], e->R_next_addr.u8[1]);
                unicast_send(&c->c, &e->R_next_addr); // envoyer les données
                return 1;
                }
            else   // choisir le meilleur chemin
                 if(collect_link_estimate(&e->R_dist.le) < best)
                {
                 best_addr=e->R_next_addr;
                 best=collect_link_estimate(&e->R_dist.le);
                }
            }
      PRINTF("\n multihop_send: sending data towards %d.%d\n",
                best_addr.u8[0], best_addr.u8[1]);
        unicast_send(&c->c, &best_addr);   // envoyer les données
    return 1;
  }
}
```

Figure 4.21 : choix de la route par la source

4.5 Diagramme de séquence

Le diagramme de séquence ci-dessous représente l'enchainement les différentes interactions entre les principales classes concernées par routage.


Nœud émetteur
Nœud récepteur
Envoyer des donnée
Mesh
Multihop
Route
Route descovry
Rime
Csma
mesh_send
multihop_send
Récupérer la liste des chemins
Data_packet_forward
Route_lookup
Liste
Alt
[Liste = Existe]
Liste
rime_output
send_packet
packet_sent
packet_sent
collect_link_estimate_update_tx
[Liste = Non existe]
Mettre paquet de données dans buffer
route_discovery_discover
rreq_initial
send_rreq
RREQ reçu
rreq_msg_received
Valid_check
ret_val
Alt
ret_val= faux
Ignorer RREQ
ret_val= vraie
route_loockup
route
Alt
route_add
[route= non existe]
route_list_add
[route= existe]
Alt
Destination != mon @
send_rreq
send_rrep
Destination = mon @
RREP reçu
found_route
multihop_resend

4.6 Conclusion

Dans ce chapitre nous avons présenté et expliqué les modifications apportées dans les différentes fonctions du protocole LOADng pour le rendre multichemin. Des captures sont jointes pour montrer le travail réalisé.

Le chapitre suivant illustre les résultats obtenus ainsi qu'une étude comparative visant à vérifier les performances de notre solution.

Chapitre 05 : Evaluation des performances

5.1 Introduction

Dans les deux chapitres précédents, nous avons expliqué les détails de conception et d'implémentation du protocole µLOADng qui a été conçu dans le but d'améliorer les performances du réseau. Dans le présent chapitre, nous présentons les résultats obtenus lors de la simulation de notre protocole µLOADng et le protocole LOADng à l'aide du simulateur Cooja.

5.2 Configuration de réseau

Pour tester le comportement général de µLOADng par rapport à LOADng, plusieurs simulations ont été exécutées sous le simulateur Cooja. Ces simulations testent les communications *Point-to-Point (P2P)* dans un réseau où un ensemble de nœuds est déployé aléatoirement. Parmi ces nœuds il y en a un qui est un nœud racine et il y a un autre qui tente d'envoyer des paquets de données à ce dernier. Une communication s'est établie entre ces deux.

La procédure d'envoi d'un paquet de données est la suivante : lorsque le nœud du réseau envoie un nouveau paquet de données, la procédure de découverte de route est appelée pour obtenir la route recherchée. Après avoir reçu un paquet de données, le nœud racine renvoie un paquet de réponse au nœud émetteur. Au cours de la procédure de découverte de la route, la route inverse est installée afin que le nœud racine n'ait pas besoin de découvrir un nouveau chemin vers le nœud émetteur.

La Figure 5.1 montre la configuration d'un réseau de 35 nœuds. Le cercle vert indique la zone de transmission. Le cercle gris indique la zone d'interférence.

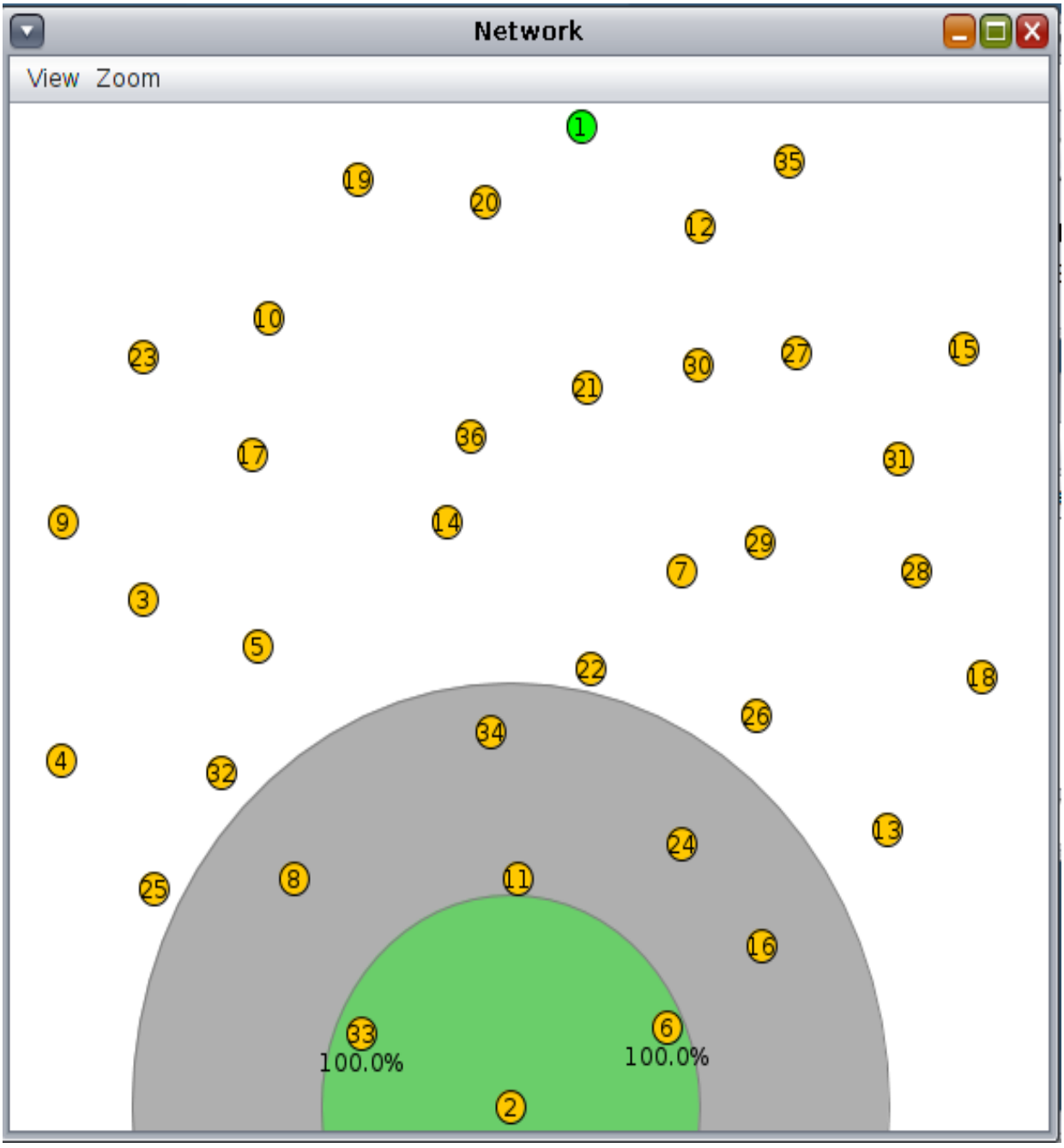


Figure 5.1 : Topologie aléatoire de 35 nœuds

Les paramètre de simulation sont indiqués dans le tableau 5.1

Paramètre	**Valeur**
Simulateur	COOJA
Contiki	Version 2.7
Platform	Skymote
Nombre de nœud	10, 20, 30,35, 40
Intervalle d'envoi des paquets de données	3 paquet / seconde
Modèle radio	UDGM Distance_loss
Zone de transmission	50 m
Zone d'interférence	100 m
Taille de la grille	300 * 300 m^2
Couche Mac	CSMA
Couche RDC	Contikimac
Framer	framer-802154
Taux de contrôle de canal (***Channel check rate***)	8 Hz
Ratio de réussite TX / RX	1.0 / 1.0
Pile réseau	Rime
Temps de simulation	900 s = 15 mn

Tableau 5.1 : Paramètres de simulation

Le scénario de simulation consiste en un ensemble de nœuds répartis d'une manière aléatoire sur une surface de 900 m^2, avec une zone de transmission de 50m et une zone d'interférence de 100m.

Le modèle radio **UDGM Distance_Loss** qui est un modèle dans lequel la plage de transmission est modélisée comme un disque idéal et où tous les nœuds situés derrière ce disque ne reçoivent pas de paquets, tandis que les nœuds situés dans la distance de transmission peuvent recevoir les paquets. Les interférences sont considérées si les paquets sont interférés, ils seront perdus par la suite. Le taux de réussite de l'émission et de la réception peut être défini par TX/RX. Les paquets sont transmis avec la probabilité TX et reçus avec la probabilité RX [52].

Le protocole *Contikimac* est utilisé pour économiser de l'énergie, car il fonctionne en mode de cycle de travail. Un paramètre important à définir pour le fonctionnement de ce protocole est **Channel check rate** qui définit la fréquence à laquelle les nœuds écouteront le support pour éventuellement recevoir des données de leurs voisins. Lorsqu'une activité est détectée, les nœuds restent éveillés pour recevoir des données. Lorsqu'il n'y a pas d'activité, les nœuds reviennent en mode veille pour une autre période de cycle de travail [50].

La pile Rime a été utilisée pour la communication entre les nœuds. Les nœuds simulés envoient des paquets avec le module **mesh**, qui exploite la communication multi-sauts à travers le module **multihop**.

Les simulations ont été effectuées à l'aide de plates-forme de nœud de capteur **Skymote** qui se caractérise par [53] :

- Mémoire : 48 KB ROM
- Très faible consommation d'énergie et réveil rapide du sommeil
- Interopérabilité avec de nombreux périphériques IEEE 802.15.4 différents

5.3 Métrique de performance

Différentes métriques sont définies pour évaluer les performances de µLOADng par rapport à LOADng selon le scénario mis en jeu :

- **Temps de routage** : le temps nécessaire pour trouver la route. Il est défini dans l'équation 5.1.

$$Temps\ de\ routage\ = \mathrm{T_{rep}} - \mathrm{T_{req}} \qquad (5.1)$$

  - ✓ T_{req} : temps d'initialisation du message RREQ
  - ✓ T_{rep} : temps de réception du premier message RREP

- **Délai moyen de bout en bout (E2ED) :** temps moyen requis par le paquet de données pour parcourir la distance de la source à la destination. Il est défini dans l'équation 5.2.

$$E2ED = \frac{1}{N}\sum_{i=1}^{i=N} Tr_i - Ts_i \qquad (5.2)$$

  - ✓ *N :* nombre total des paquets envoyés

- ✓ T_s : temps d'envoi du paquet
- ✓ T_r : temps d'arrivée du paquet

- **PRR (*Packet Reception Rate***) : est le rapport entre le nombre de paquets de données reçus et le nombre de paquets de données envoyés. Il est défini dans l'équation 5.3.

$$PRR = \frac{\sum_{i=1}^{i=N} P_r}{\sum_{i=1}^{i=N} P_s} \qquad (5.3)$$

- ✓ P_r : paquet reçu
- ✓ P_s : paquet transmis
- ✓ N : nombre totale des paquets

- **Débit :** le taux auquel les messages sont remis avec succès Il est défini dans l'équation 5.4.

$$Débit = \frac{\sum_{i=1}^{i=N} P_r}{T} \qquad (5.4)$$

- ✓ P_r : paquet reçu
- ✓ T : temps de simulation
- ✓ N : nombre totale des paquets

5.4 Evaluation de performance

Nous avons effectué nos simulations pour comparer notre protocole µLOADng au LOADng et vérifier s'il est plus performant en termes de délai, débit, temps de routage, taux de perte. Les deux protocoles ont été testés et évalués sur la moyenne de 15 simulations dans un scénario de test dont les nœuds sont placés sur une topologie aléatoire.

La figure 5.2 montre le chemin trouvé par le nœud 2 utilisant le protocole LOADng dans un réseau de 35 nœuds et la figure 5.3 montre l'entrée de la table de routage du même nœud.

La figure 5.4 montre les chemins trouvés par le nœud 2 utilisant le protocole µLOADng dans un réseau de 35 nœuds et la figure 5.5 montre les entrées de la table de routage du même nœud.

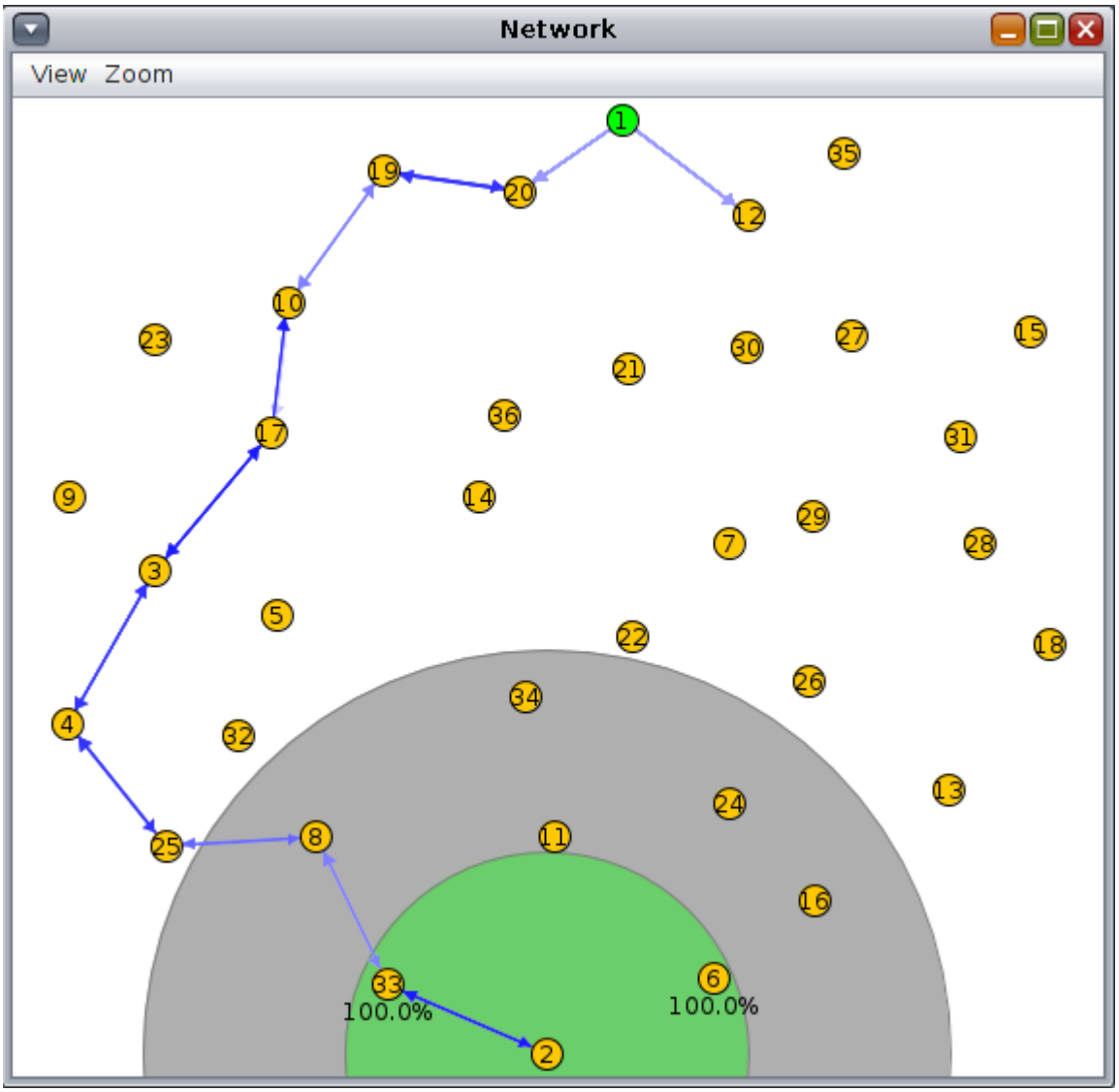


Figure 5.2 : Le chemin trouvé par LOADng

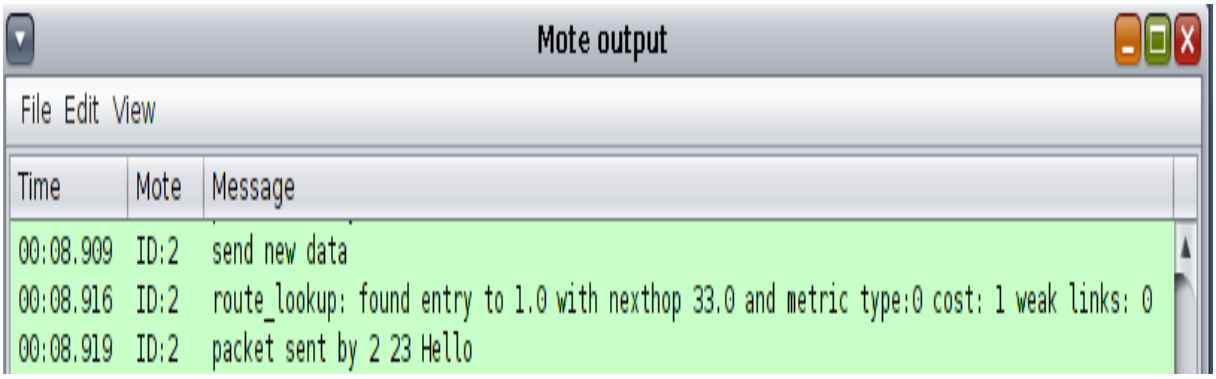


Figure 5.3 : L'entrée de la table de routage pour la destination 1 dans le nœud 2

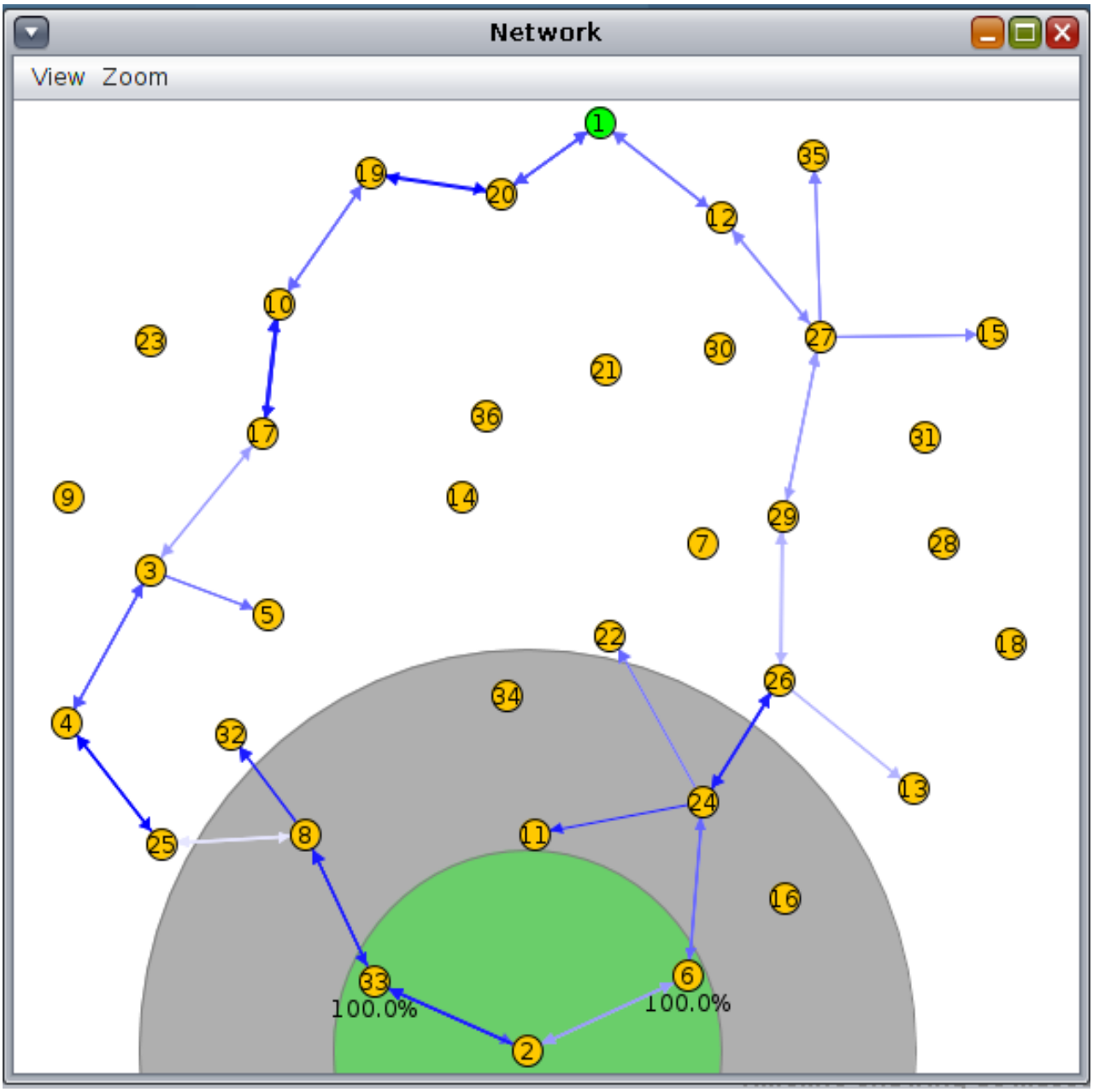


Figure 5.4 : Les chemins trouvés par µLOADng

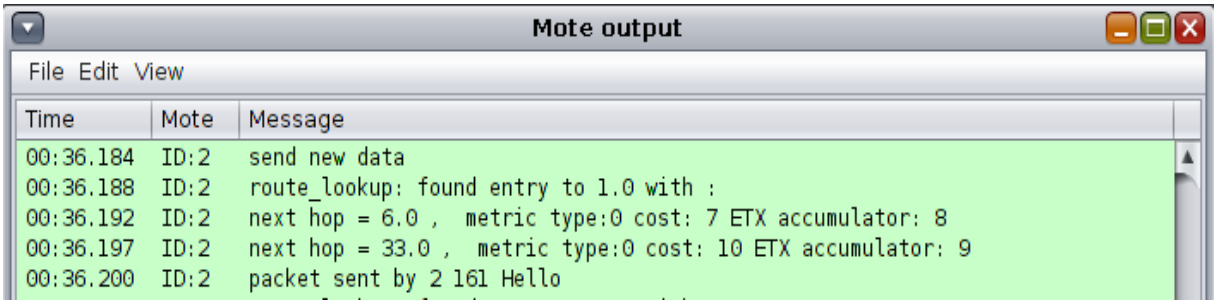


Figure 5.5 : Les entrées de la table de routage pour la destination 1 dans le nœud 2

- Dans un premier test de simulation, nous déterminons le temps de routage en fonction du nombre de nœuds (Figure 5.6)

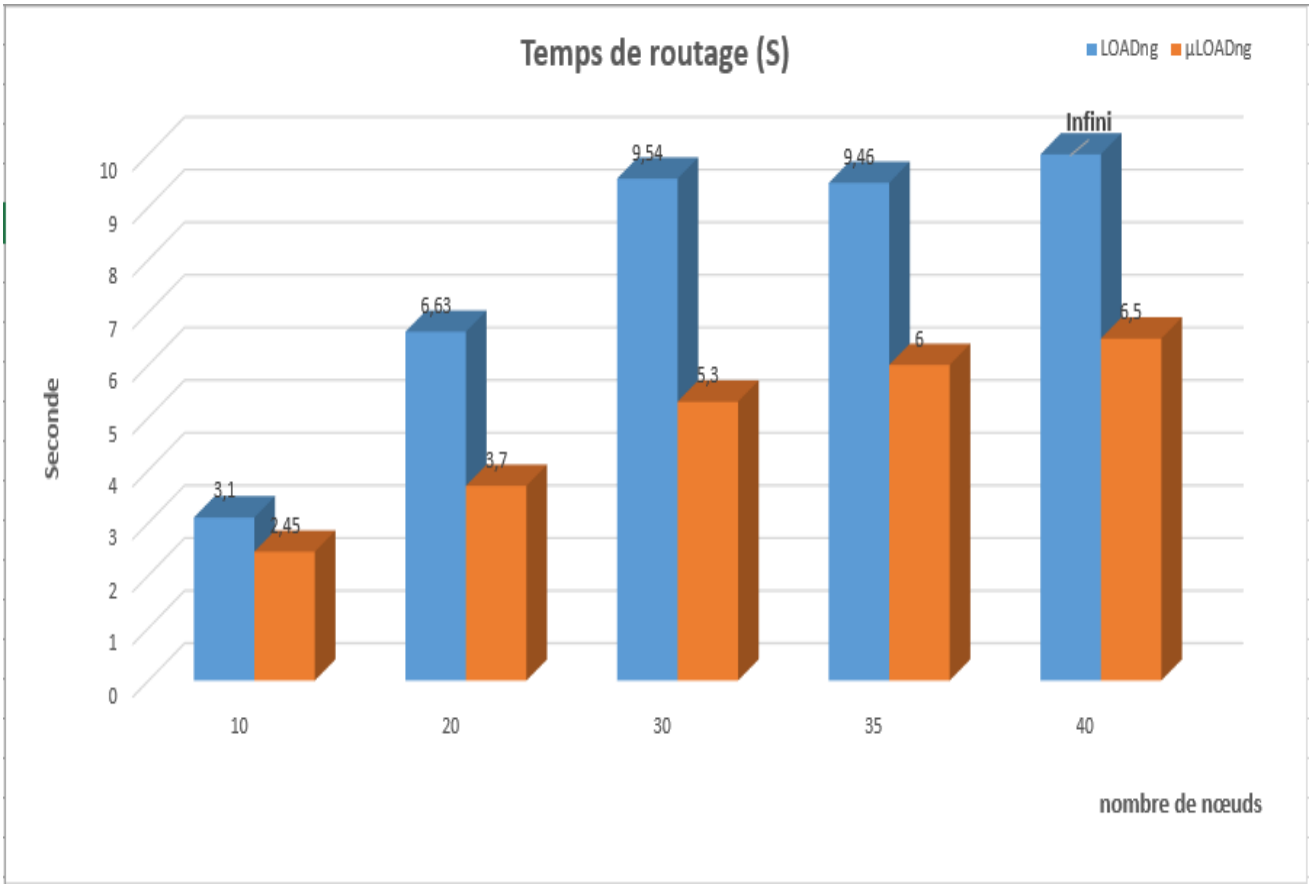


Figure 5.6 : Temps de routage LOADng vs µLOADng

Nous remarquons que ce soit pour LOADng ou µLOADng, le temps de routage augmente proportionnellement avec le nombre du nœud mais en arrivant au seuil de 40 nœuds LOADng cesse de répondre. Notons que plus le nombre de nœuds augmente plus le nombre des messages RREQ rediffusés augmente ce qui implique des congestions.

Nous remarquons également que le temps de routage de µLOADng est toujours meilleur que celui de LOADng car notre protocole élimine tous les messages RREQ qui forment une boucle grâce au champ **Precursors_List** ajouté au message RREQ. L'élimination de ces messages permet la réduction de la surcharge dans le réseau. LOADng n'arrive pas à détecter ces boucles et il rediffuse continuellement tous les messages RREQ reçus. Par conséquent, il devient saturé et incapable de répondre dans un réseau de 40 nœuds.

- Dans le deuxième test de simulation, nous déterminons le délai moyen en fonction du nombre de nœuds (figure 5.7).

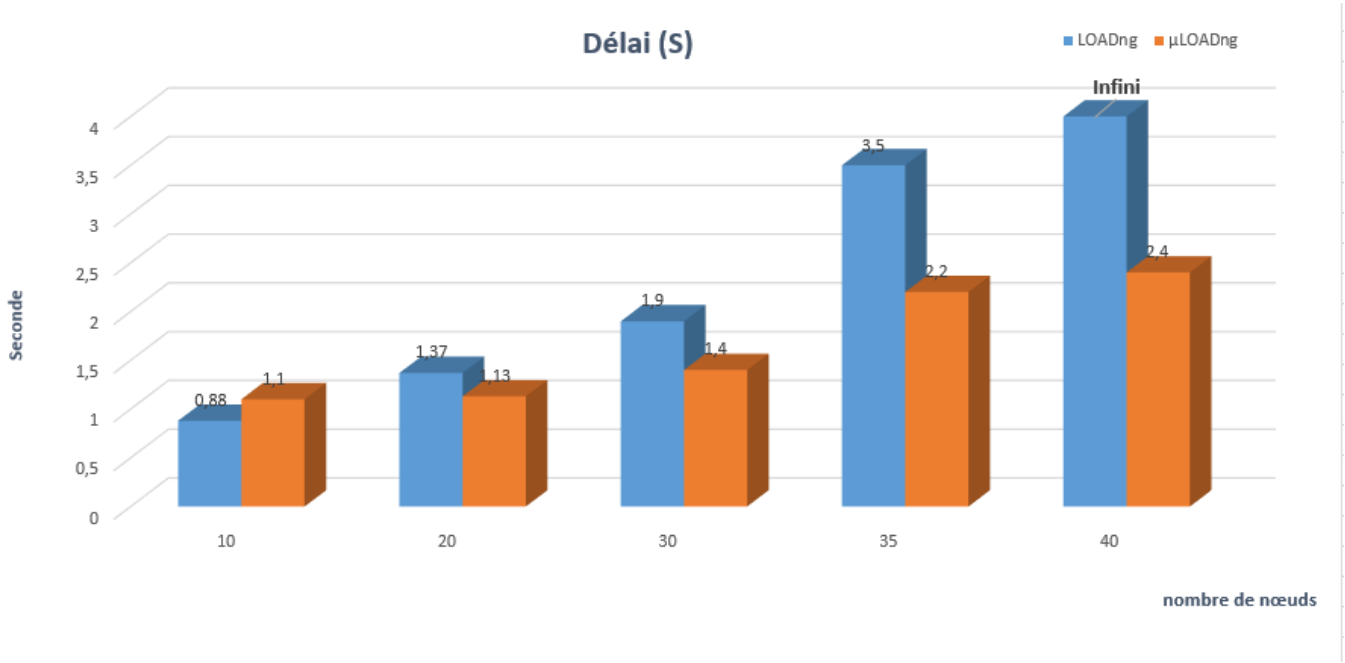


Figure 5.7: Délai LOADng vs µLOADng

LOADng marque son meilleur délai dans un réseau de 10 nœuds par rapport au µLOADng qui performe mieux à partir d'un réseau de 20 nœuds ou plus.

Le principe de la disjonction restreint le nombre des chemins donc dans les petites topologies (10 nœuds) µLOADng n'arrive pas parfois à trouver plusieurs chemins ce qui explique que les délais de LOADng sont meilleurs que ceux de µLOADng. Les délais de µLOADng s'améliorent dans les réseaux de 20 nœuds ou plus parce que la probabilité de trouver plusieurs chemins augmentent. Le basculement de données que nous avons proposé c'est avéré très utile car il permet d'opter la route là moins congestionnée ce qui induit une diminution dans les délais. Il reste à noter que l'incapacité de LOADng de trouver un chemin dans un réseau de 40 nœuds l'empêche impérativement d'envoyer des données.

- Dans le troisième test de simulation, nous déterminons le taux de réception en fonction du nombre de nœuds (Figure 5.8).

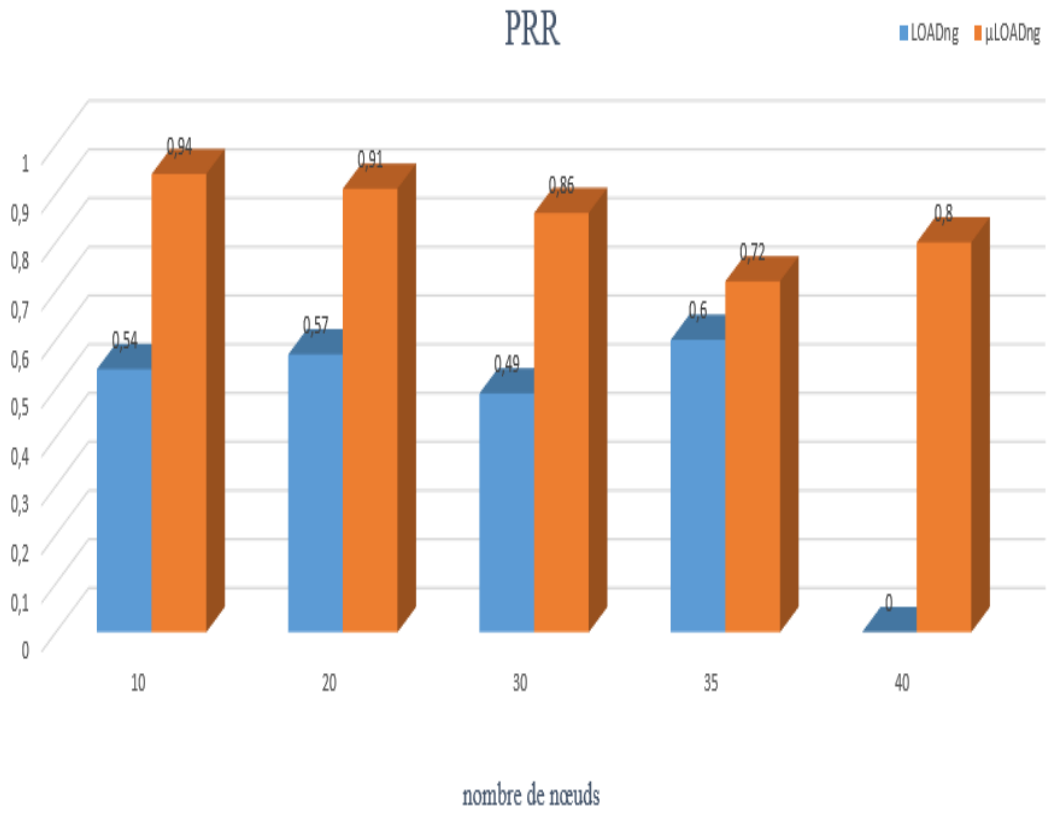


Figure 5.8: PRR LOADng vs µLOADng

Dans la plupart du temps µLOADng arrive à transmettre presque tous les paquets et cela est clair à partir de figure 5.8 où nous remarquons que la valeur de PRR est toujours proche de 1, tandis que LOADng enregistre des valeurs de PRR variant entre 0.5 et 0.6. Le basculement entre les chemins trouvés a également influencé le PRR d'une manière positive car le choix des chemins moins congestionnés diminue significativement la perte des paquets. LOADng utilise un seul chemin donc ce dernier sera congestionné ce qui augmente la probabilité de perdre des paquets.

- Nous déterminons les valeurs du débit en fonction du nombre de nœuds (Figure 5.9)

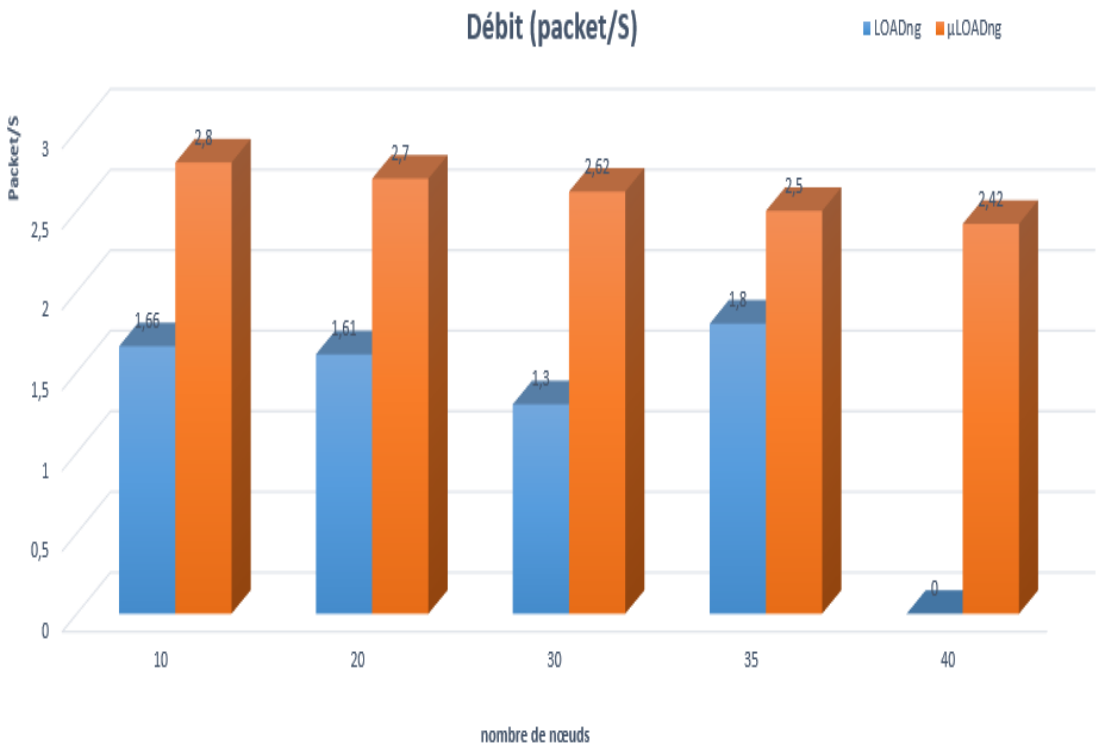


Figure 5.9 : Débit LOADng vs. µLOADng

D'après le graphe de la Figure 5.9 nous remarquons que le débit de µLOADng est toujours meilleur que celui de LOADng. Cette différence significative est due au fait que µLOADng a la capacité de découvrir les chemins rapidement donc la transmission des données se fait antérieurement par rapport à LOADng d'une part, et d'autre part les valeurs de PRR de µLOADng est supérieurs à celle de LOADng.

5.4 Conclusion

Dans ce chapitre, nous avons montré les différents résultats obtenus après avoir implémenté notre protocole de routage multi-chemins µLOADng sous Cooja. Nous avons fait une analyse comparative des performances entre notre proposition et LOADng. A travers les différents résultats, nous avons conclu que les modifications que nous avons apportées ont abouti à des résultats favorables et nous pouvons dire que µLOADng est plus performant en terme de débit, temps de réponse et taux de perte que son prédécesseur qui a montré ses limites dans un réseau de 40 nœuds. Le délai a fait l'exception où les valeurs enregistrées dans LOADng sont parfois meilleures que celles de µLOADng. Ces fluctuations au niveau de ce dernier sont dues au nombre de chemins trouvés et le temps nécessaire pour choisir le meilleur entre eux.

Conclusion générale

Les réseaux LLN trouvent des applications dans plusieurs domaines telles que la communication dans un champ de bataille, les communications de groupe et la gestion des catastrophes naturelles…etc. Les particularités uniques de ces réseaux ont dégagé plusieurs sujets de recherche voire des défis qui doivent être solutionnés ou améliorés. Pour ce faire des protocoles de communication sont proposés pour répondre à ces caractéristiques.

Le travail réalisé dans ce mémoire s'inscrit dans le cadre du routage dans les LLNs et, plus précisément, du routage multichemin afin d'améliorer les performances de tels réseaux. Pour mieux approcher ce thème, nous avons commencé par une étude des principales solutions multichemin présentées dans la littérature, puis nous avons enchainé avec les différents protocoles de routage proposés pour les réseaux LLN, y compris RPL et LOADng, enfin nous avons donné une taxonomie des différents protocoles de routage multichemin.

Nous nous sommes intéressées au protocole de routage LOADng pour proposer une extension basée sur le concept du multichemin, nous avons opté pour ce protocole car il vient de susciter l'intérêt des chercheurs ces derniers temps grâce à sa capacité de prendre mieux en charge les flux de données bidirectionnels, ainsi que la simplicité du protocole et sa réactivité aux changements de topologie.

L'indépendance des chemins est une propriété importante dans le routage multichemin. En se basant sur l'approche des chemins à nœuds disjoints, nous avons proposé µLOADng. Notre protocole est conçu dans le but de corriger les faiblesses liées au protocole uni-chemin et améliorer les performances dans les réseau LLN.

Dans l'étape de la réalisation, nous avons procédé à l'implémentation de notre protocole µLOADng sous le simulateur Cooja en réutilisant le code source de LOADng. Les résultats de simulation ont montré que notre extension a amélioré le débit, le taux de perte, temps de réponse et le délai moyen par rapport à son prédécesseur. Ces améliorations sont dues au fait que notre solution a permis de réduire la surcharge et de mieux utiliser les ressources disponibles dans le réseau.

Au bout du travail sur ce thème nous nous sommes familiarisées avec les réseaux LLNs, nous avons approfondi nos connaissances dans les protocoles de routage et nous avons découvert une nouvelle plateforme de simulation adéquate qui est Cooja. Nous avons réussi à mettre en place notre extension.

Le travail réalisé dans le cadre de ce projet peut être étendu de plusieurs façons. Nous pouvons citer à titre d'exemple :

- Comparer les performances de notre protocole µLOADng avec une extension multichemin de RPL.
- Tester l'impact de la mobilité sur les performances de notre protocole.
- Tester la scalabilité de µLOADng par rapport à : (a) nombre de nœuds, (b) nombre de flux de données, (c) nombre de chemins maintenus.
- Inclure une métrique de qualité de service telle que le délai dans nœud, la bande passante… etc.
- Enfin, il serait intéressant également de mener une étude expérimentale pour voir les performances réelles du protocole sur une plateforme IoT telle que IoT-LAB.

Nous espérons que ce modeste travail puisse apporter sa contribution pour d'autres travaux et étude concernant le routage dans les réseaux LLN.

Bibliographies

[1] A. Fallis, Internet of things architectures, protocols and standards, *Journal of Chemical Information and Modeling* vol. 53, no. 9. 2013.

[2] C. E. Perkins, E. M. Royer, S. Das. Ad hoc on-demand distance vector (AODV) routing, RFC 3561, July 2003.

[3] Clausen, Thomas, Philippe Jacquet, Anis Laouiti, Paul Muhlethaler, Amir Qayyum, Laurent Viennot, Thomas Clausen, Philippe Jacquet, Anis Laouiti, and Pascale Minet. 2010. "Optimized Link State Routing Protocol (OLSR)".

[4] MOY, John T. *OSPF:* anatomy of an Internet routing protocol. Addison-Wesley Professional, 1998.

[5] LEVIS, Philip, Tavakoli, Arsalan, and Dawson-Haggerty, Stephen. Overview of existing routing protocols for low power and lossy networks. *Internet Engineering Task Force, Internet-Draft draftietf-roll-protocols-survey-07*, 2009.

[6] T. Winter, P. Thubert, A. R. Corporation, and R. Kelsey, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," pp. 1–157.

[7] A. Camacho Martínez, "Implementation and Testing of LOADng: a Routing Protocol for WSN," 2012.

[8] M. Opportunistic, R. P. L. Routing, and B. Pavkovi, "Multipath Opportunistic RPL Routing over IEEE 802.15.4."

[9] Quynh, Thu Ngo, Nien Le Manh, and Khoi Nguyen Nguyen. 2015. "Multipath RPL Protocols for Greenhouse Environment Monitoring System Based on Internet of Things." *ECTI-CON 2015 - 2015 12th International Conference on Electrical Engineering/Electronics, Computer, Telecommunication and Information Technology*.

[10] T. N. Quynh, N. Le Manh, and K. N. Nguyen, "Multipath RPL protocols for greenhouse environment monitoring system based on Internet of Things," *ECTI-CON 2015 - 2015 12th Int. Conf. Electr. Eng. Comput. Telecommun. Inf. Technol.*, 2015.

[11] M. N. Moghadam and H. Taheri, “High throughput load balanced multipath routing in homogeneous wireless sensor networks,” *22nd Iran. Conf. Electr. Eng. ICEE 2014*, pp. 1516–1521, 2014.

[12] B. Ghaleb *et al.*, “A Survey of Limitations and Enhancements of the IPv6 Routing Protocol for Low-Power and Lossy Networks: A Focus on Core Operations,” *IEEE Commun. Surv. Tutorials*, vol. 21, no. 2, pp. 1607–1635, 2019.

[13] T. Clausen, J. Yi, and U. Herberg, “Lightweight On-demand Ad hoc Distance-vector Routing - Next Generation (LOADng): Protocol, extension, and applicability,” *Comput. Networks*, vol. 126, no. July, pp. 125–140, 2017.

[14] T. H. Clausen *et al.*, “ LOADng : Towards AODV Version 2 : Towards AODV Version 2,” pp. 1–5, 2019.

[15] L. Atzori, A. Iera, and G. Morabito, “The Internet of Things: A survey,” *Comput. Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[16.] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, “Context aware computing for the internet of things: A survey,” *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[17.] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[18.] K. Asthon, “That’ Internet of Things’ Thing,” *RFID J.*, p. 4986, 2010.

[19.] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[20] J.-S. Lee, Y.-W. Su, and C.-C. Shen, “A Comparative Study of Wireless Protocols : A SURVEY AND A COMPARISON,” pp. 46–51, 2007.

[21] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck, “Screen-off traffic characterization and optimization in 3G/4G networks,” *Proc. ACM SIGCOMM Internet Meas. Conf. IMC*, pp. 357–363, 2012.

[22] IEEE Task Group 4 (TG4) – IEEE 802.15.4 (Accédé 2019).
URL : http://www.ieee802.org/15/pub/TG4.html.

[23] P. O. Kamgueu, P. Olivier, and K. Configuration, “Configuration dynamique et routage pour l’internet des objets,” 2018.

[24] S. Ferdoush and X. Li, “Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications,” *Procedia Comput. Sci.*, vol. 34, pp. 103–110, 2014.

[25] [Online]. Available: http://www.eistec.se/docs/contiki/ . [Accessed: 2019].

[26] R. Duan, X. Chen, and T. Xing, “A QoS architecture for IOT,” *Proc. - 2011 IEEE Int. Conf. Internet Things Cyber, Phys. Soc. Comput. iThings/CPSCom 2011*, pp. 717–720, 2011.

[27] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Akyildiz et al._2002_Wireless sensor networks a survey.pdf,” *Comput. Networks*, vol. 38, pp. 393–422, 2002.

[28] J. Yick, B. Mukherjee, and D. Ghosal, “Wireless sensor network survey,” *Comput. Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[29] A. Ian F. and W. Xudong, “A Survey on Wireless Mesh Networks,” *IEEE Commun. Mag.*, vol. 43, no. September, pp. 23–30, 2005.

[30] J. Guerin, M. Portmann, and A. Pirzada, “Routing metrics for multi-radio wireless mesh networks,” *2007 Australas. Telecommun. Networks Appl. Conf. ATNAC 2007*, no. February, pp. 343–348, 2008.o. 4, pp. 2347–2376, 2015.

[31] D. De_Couto, D. Aguayo, J. Bicket, and R. Morris, “A high throughput path metric for multi-hop wireless routing. Mobicom Conference,” 2003.

[32] I. Doghri and S. De, “Stratégies de routage multi-chemin dans les réseaux sans fil multi-sauts,” 2013.

[33] K. Sha, J. Gehlot, and R. Greve, “Multipath routing techniques in wireless sensor networks: A survey,” *Wirel. Pers. Commun.*, vol. 70, no. 2, pp. 807–829, 2013.

[34] S. Mueller, R. P. Tsang, and D. Ghosal, “Multipath routing in mobile ad hoc networks: Issues and challenges,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 2965, pp. 209–234, 2004.

[35] M. Radi, B. Dezfouli, K. A. Bakar, and M. Lee, “Multipath routing in wireless sensor networks: Survey and research challenges,” *Sensors*, vol. 12, no. 1, pp. 650–685, 2012.

[36] A. Goel and A. K. Sharma, “A framework for routing in mobile ad hoc networks,” *Adv. Comput. Sci. Eng. Reports Monogr. - Innov. Appl. Inf. Technol. Dev. World - Proc. 3rd Asian Appl. Comput. Conf., AACC 2005*, vol. 2, no. C, pp. 230–232, 2007.

[37] M. K. Marina and S. R. Das, “Ad hoc on-demand multipath distance vector routing,” *Wirel. Commun. Mob. Comput.*, vol. 6, no. 7, pp. 969–988, 2006.

[38] U. Des, S. Et, and D. E. L. A. Technologie, “ Routage Multipath Dans Les Réseaux AD HOC,” 2015.

[39] A. H. Chowdhury *et al.*, “Route-over vs mesh-under routing in 6LoWPAN,” *Proc. 2009 ACM Int. Wirel. Commun. Mob. Comput. Conf. IWCMC 2009*, no. May 2014, pp. 1208–1212, 2009.

[40] KIM, K. 6LoWPAN ad hoc on-demand distance vector routing (LOAD). *draft-daniel-6lowpan-load-adhoc-routing-02. txt*, 2007.

[41] Systems, D, Narrowband orthogonal frequency division multiplexing power line communication transceivers for G3-PLC networks, SERIES G: Transmission Systems and Media, Digital Systems and Networks, “ITU-T,”2017.

[42] C. E. Perkins and P. Bhagwat, “Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computer,” *Proc. Conf. Commun. Archit. Protoc. Appl. SIGCOMM 1994*, pp. 234–244, 1994.

[43] C. E. Perkins, M. Park, and E. M. Royer, C. E. Perkins, E. M. Royer, Ad hoc On-Demand Distance Vector Routing, in Proc. 2n IEEE Wksp. Mobile Comp. Sys. and Apps., pp. 90100, Feb. 1999.

[44] D. B. Johnson and D. A. Maltz, “DSR : The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks,” *Comput. Sci. Dep. Carnegie Mellon Univ. Addison-Wesley*, pp. 139–172, 1996.

[45] E. Aljarrah, M. B. Yassein, and S. Aljawarneh, “Routing protocol of low-power and lossy network: Survey and open issues,” *Proc. - 2016 Int. Conf. Eng. MIS, ICEMIS 2016*, 2016.

[46] A. Taha, R. Alsaqour, M. Uddin, M. Abdelhaq, and T. Saba, “Energy Efficient Multipath Routing Protocol for Mobile Ad-Hoc Network Using the Fitness Function,” *IEEE Access*, vol. 5, pp. 10369–10381, 2017.

[47] O. Smail, Z. Mekkakia, B. Messabih, R. Mekki, B. Cousin, Energy Conservation for Ad Hoc On-Demand Distance Vector Multipath Routing Protocol, International Journal of Computer Network and Information Security (IJCNIS), Vol. 6, No. 6, pp. 1-8, 2014.

[48] H. Nasehi, N. T. Javan, A. B. Aghababa, Y. G. Birgani, Improving energy efficiency in manets by multi-path routing, International Journal of Wireless And Mobile Networks (IJWMN), Vol. 5, No. 1, Feb. 2013.

[49] A. Kurniawan, *Practical Contiki-NG: Programming for Wireless Sensor Networks*. 2018.

[50] [Online]. Available: https://anrg.usc.edu/contiki/index.php/MAC_protocols_in_ContikiOS/. [Accessed: 2019].

[51] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-level sensor network simulation with COOJA,” *Proc. - Conf. Local Comput. Networks, LCN*, pp. 641–648, 2006.

[52] M. Stehlík, “Comparison of Simulators for Wireless Sensor Networks,” pp. 1–92, 2011.

[53] [Online]. Available: https://slogix.in/difference-between-skymote-and-z1mote-and-wismote-in-contiki-cooja simulation. [Accessed: 2019].