Université de Strasbourg
UFR mathématiques informatique strasbourg

Rapport de TER (Master 1 SIRIS)

L'Internet des Objets Industriels : fiabilité et délai

May 8, 2020

Encadrant :

Fabrice Théoleyre theoleyre@unistra.fr

Responsables de parcours :

Pierre David pda@unistra.fr
Julien Montavont montavont@unistra.fr

Etudiant :

Lahmer Seyyid-Ahmed lahmer.seyyidahmed@etu.unistra.fr

Résumé :

L'Internet industriel des objets (IIoT) se concentre sur les besoins industriels (des professionnels et des entreprises). Deça, IIoT vise les systemes industriels qui ont des besoins tangible (des indicateurs mesurables et concrets) comme la continuité de service pour une longue durée, économies sur les coûts de production et gain de production. IIot tends de connecter des objets avec des limitations en capacités de traitement et en puissance pour des applications industrielles pour le but de fournir des garanties en terms de délai ET fiabilité .
Le groupe de travail IPv6 over the TSCH mode of IEEE 802.15.4e (6TiSCH) a défini une pile protocolaire qui represent actuellement les standards utilisés pour l'IIoT et qui comprend : IEEE 802.15.4e TSCH comme méthode d'accès au médium , la sous-couche IETF 6top pour la manipulation d'ordonnanceur, IETF 6loWPan comme une couche pour adaptater les paquets ipv6 à la couche MAC-802.15.4e, IETF rpl comme protocole de routage, udp pour la couche transport et IETF CoAP pour la couche application.
Dans le cadre de cette TER, nous allons en premier temps étudier la pile protocolaire 6TiSCH, ensuite nous focaliserons sur le problème d'ordonnancement temps-fréquence des transmissions pour fournir des garanties, ainsi quelques solutions proposées et finalement nous finirons par l'implémentation d'une solution et l'evaluer.

Keywords: IIot ; Délai maximum ; Fiabilité minimale; TSCH ; Groupe de travai 6-TiSCH ; 6top ; CoAP ; Ordonnancement

Contents

List of Figures

Introduction

L'Internet des objets est l'ensemble des systèmes, des applications connectées avec le réseau afin de fournir un ou plusieurs services, ce ou ces services peuvent prendre différentes formes, comme la collecte de données auprès d'autres entités ou avec un capteur attaché à la même entité , fournissant des données à d'autres objets. Cette idée peut être appliquée à divers domaines qui trouvent ce type de réseaux utile afin d'effectuer certaines tâches plus performantes, ou à cause de la nature de ces tâches qui nécessitent ce type de réseau. Parmi de nombreuses applications iot, nous pouvons citer les applications industrielles et personnelles.
Internet industriel des objets (IIoT): ensemble d'éléments ou de substance mettant en place une partie industrielle communiquant entre eux en partageant des informations, en synchronisant certaines actions afin de finir par effectuer une ou plusieurs tâches en parallèle ou de manière séquentielle. tout cela avec pour principaux objectifs de réduire la probabilité de défaillance, d'augmenter les performances du système, d'accélérer les tâches, d'éliminer les états d'arrêt inutiles et de fournir l'automatisation.
le groupe ietf 6tisch a normalisé une pile protocolaire qui est considérée comme la pile standard pour IIot, cette pile comprend: ieee 802.15.4e Time-Slotted Channel Hopping (TSCH) qui se concentre uniquement sur les deux couches mac et phy, la couche IETF 6LoWPAN qui permet aux longs paquets ip (jusqu'à 1280 octets) de s'adapter dans des trames IEEE 802.15.4 (jusqu'à 127 octets), IETF RPL comme protocole de routage qui fournit une connectivité de bout en bout IPV6 et IETF Constrained Application Protocol (CoAP) qui est implémenté au-dessus d'udp et qui autorise les nœuds dans les réseaux de capteurs sans fil (WSN) pour agir en tant que client ou serveur Coap. 6tisch fournit également une couche entre les deux couches 6LoWPAN et Mac appelée 6top qui permet aux nœuds voisins de négocier des cellules (ajouter / supprimer / réallouer des cellules dans le scheduler TSCH). tout cela dans le but d'atteindre des performances élevées en fournissant des garanties dans le monde industriel en termes de consommation d'énergie, fiabilité .

Problématique

Dans les réseaux de type Low power and lossy networks (LLN) et plus précisément dans le domaine industriel (IIot), les systèmes nécessitent une continuité de services ,une fiabilité minimale et une contrainte de délai à respecter qui devront être garantis. En d'autres termes, IIot nécessite qu'une bande passante minimale soit disponible pour que les applications industrielles fournissent leurs paquets de données et aient parallèlement un temps de service maximal sans interruption.

IEEE802.15.4e TSCH fait partie de la pile 6tisch, et qui définit la couche responsable pour l'accés au médium, il a été conçu pour ce type de réseaux principalement pour les applications industrielles, visant à fournir un accès moyen fiable avec une faible consommation d'énergie. TSCH utilise une combinaison de Time division multiple access (TDMA) et de Frequency division multiple access (FDMA), il met également en œuvre un saut de canal qui permet de faire face aux effets des interferences externe, en changeant la fréquence à chaque transmission.

Le groupe de travail 6tisch a défini l'architecture de la pile pour IIoT, visant à fournir des performances élevées pour les applications industrielles en termes de fiabilité de réseau, consomation d'énergie et disponibilité des services. Cependant, pour la partie ordonnanceur, il a spécifié uniquement la partie exécution et a laissé la partie de construction d'ordonnanceur libre.

Ce travail vise à implémenter et évaluer un algorithme de ceux qui existent dans la littérature [1] à l'aide d'une solution qui implémente la pile 6-TISCH "OpenWsn" [2].

1 IPv6 over the TSCH mode of IEEE 802.15.4 (6TISCH)

Le groupe de travail IETF 6tisch a travaillé activement à la normalisation d'une pile de protocoles performante pour l'iot industriel (IIot), dans le but d'intégrer la connectivité IPv6 avec le mode TSCH [3], il fini finalement avec la pile protocolaire qui est appelée 6TiSCH [3]. Cette dernière combine la fiabilité et la faible consommation d'énergie du mode TSCH tout en offrant une connectivité IPv6 de bout en bout [4]. La figure 1 montre les differentes couche de la pile 6tisch (Ref [1]).

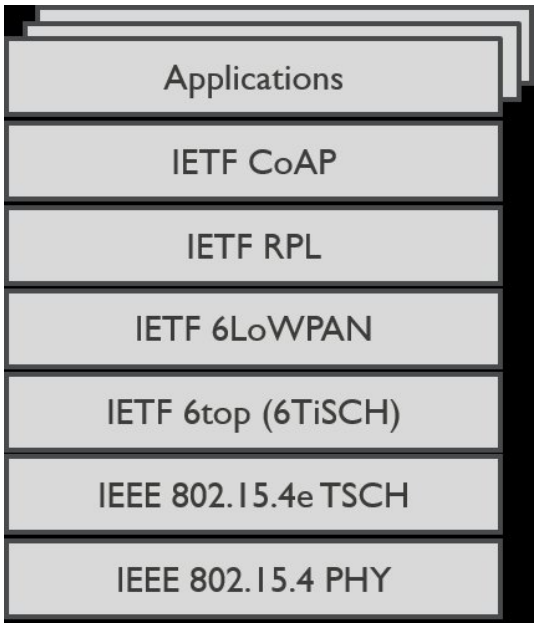


Figure 1: La pile protocolaire 6TISCH

1.1 IEEE 802.15.4e Time Slotted Channel Hopping

TSCH est une méthode déterministe d'accès au support (sans contention) qui est connu pour sa fiabilité (atténuer les effets des interférences et multipath fading) pour les réseaux industriel tout en réduisant la consommation d'énergie.

TSCH adopte à la fois TDMA (multiplexage temporel) et FDMA (multiplexage fréquentiel). TSCH maintient (pour la partie TDMA) une structure SlotFrame qui est constituée d'une collection de TimeSlot, ce dernier sera associé à un channel offset.

SlotFrame

Comme dit, TSCH utilise un accès multiple par répartition temps-fréquence qui est une combinaison de FDMA et TDMA. Un SlotFrame est composé d'un nombre fini de timeSlot (SLOTFRAME_LENGTH 101 [openwsn]), chaque timeSlot peut être soit:

- contention-based TS ou (Shared Slot) .
- contention-free TS ou (Dedicated Slot) .

[1] Reprinted from "Time-critical communication in 6TiSCH networks," by A. Karaagac, 2018, IEEE WCNCW

Chaque TimeSlot est associé a un channel offset qui sera traduit ultérieurement en une fréquence de communication (décrite dans la section suivante), chaque TimeSlot est également associé à l'un de ces modes:

- TX : mode de transmission, au moins un paquet doit être en file d'attente pour que le nœud allume la radio.
- RX : mode de réception
- TX/RX : mode hybride, le nœud sera tel qu'il est en mode transmission, à moins qu'il n'ait pas de paquet à envoyer, alors il sera en mode réception
- OFF : mode veille, la radio est éteinte

Un TimeSlot suffit pour envoyer un paquet et recevoir l'acquittement correspondant, l'expéditeur commence à transmettre après TxOffset ms depuis le début de TimeSlot, tandis que le récepteur commence à écouter après T/2 - TxOffset, où T est le temps de garde qui tient compte de la perte de synchronisation dans les deux sens positive et négative (en raison de la dérive de l'horloge) [5]. Comme TSCH utilise des techniques TDMA, les nœuds du réseau TSCH sont tenus de conserver une synchronisation globale, chaque nœud synchronise son horloge à la réception des paquets en calculant la différence entre l'heure de réception attendue et l'heure de réception réelle, puis il corrige son horloge[]. Deux approches sont adoptées pour la synchronisation []:

- Acknowledgment-based : dans ce cas, le nœud émetteur synchronise son horloge en fonction du récepteur (en utilisant le paquet d'ack).
- Frame-based synchronization : dans ce cas, le récepteur synchronise son horloge en fonction de l'expéditeur (en utilisant le paquet de données)

Dans les deux cas, si le nœud ne synchronise pas son horloge en 30 secondes, un paquet vide sera envoyé afin de maintenir l'horloge synchronisée,TSCH conserve ASN (absolute sequence number) comme compteur pour les TimeSlots écoulés, ce paramètre est globalement connu de tous les nœuds de réseaux (obtained in join time).

Channel Hopping

Pour chaque TimeSlot, TSCH peut utiliser un canal de fréquence différent, qui est défini par un channel offset, un entier variant entre 0,15, ce qui rend 16 canaux disponibles au total (2.4 GHz). Toutes les communications entre les nœuds sont identifiées par la paire [slot offset, channel offset]. Le canal de transmission (réception) correspondant est calculé à l'aide de l'équation eq.1.

$$Frequence(channelOffset) = F\{(ASN + channelOffset) \mod nFreq\} \quad (1)$$

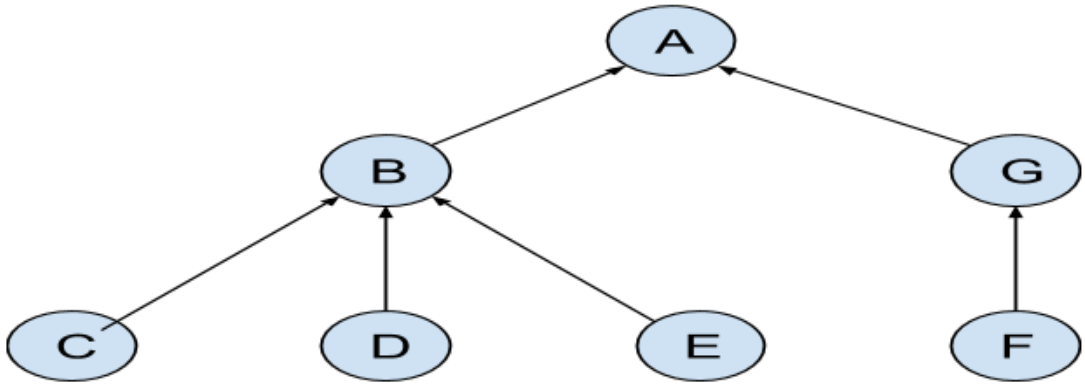
où :

- F : où F se compose de la table qui fait l'association entre le channel offset et le canal de communication réel. elle permet également le saut de canal en changeant l'association dans chaque temps écoulé.
- ASN : nombre de timeSlot écoulées.
- nFreq : Nombre de canaux disponibles (16 par défaut).

1.2 Ordonnancement et IETF 6top

Comme discuté dans la section précédente, toutes les communications d'un nœud avec ses voisins sont guidées avec un slot-frame qui est répétée dans le temps. Ce slot-frame peut être formé de manières différentes:

- Ordonnancement centralisé : là où il y a un orchestrateur de réseau (PAN) qui a une vue globale du réseau, il construit localement la table d'ordonnancement (fig 2) puis pour chaque cellule sa la table, il l'annonce aux nœuds concernés, et enfin les nœuds mettent à jour leur slot-frame selon les annonces reçues.
- Ordonnancement distribuée: là où aucune entité dans le réseau détient ou construit la table d'ordonnancement, chaque nœud négocie avec ses voisins pour ajouter / supprimer des cellules entre eux. Une table d'ordonnancement abstraite est construite lorsque toutes les slot-frame sont combinées.



	slot-i	slot-i+1	slot-i+2	slot-i+3	.....	slot-N
Freq-1	B -> A	C -> B D -> B E -> B	G -> A	B -> A		
Freq-2	F -> G	G -> A	D -> B	F -> G		
Freq-3						

Shared-cell
Dedicated-Cell

Figure 2: Table d'ordonancement

La nécessité d'un mécanisme pour gérer la table d'ordonancement d'un nœud, une nouvelle sous-couche a été créée pour gérer la table d'ordonancement du nœud en fournissant une interface pour la manipuler [6]. Cette interface fournit de nombreuses commandes [6], parmi eux :

- Adding Cells : cette commande permet à un nœud de réserver un lien pour un ensemble de TimeSlots avec son voisin.
- Deleting Cells : cette commande permet à un nœud de désallouer un TimeSlot (ou plusieurs) réservé à son voisin.

- Listing Cells : cette commande permet de lister les TimeSlots alloués
- Clearing the Schedule : cette commande permet à un nœud de désallouer tous les TimeSlots réserves avec son voisin

Cette couche garantit la cohérence entre un nœud et son voisin après l'exécution des commandes ci-dessus.

1.3 IPv6 Routing Protocol for LLNs

L'IETF a examiné les protocoles existants comme ospf, isis, bgp et d'autres protocoles qui peuvent répondre aux exigences de l'IOT, ils ont constaté qu'il n'y a pas de protocole qui correspond à ces exigences [7]:

- Stabilité pour les réseaux de type LLN.
- Considérant les contraintes des nœuds et pas seulement le coût de la liaison.
- Permettre l'utilisation de différentes fonctions objectives

Généralités

La RFC 6550 définit RPL, qui est une distance vectorielle. RPL a ajouté quelques métriques de routage par rapport à celles existantes comme: l'énergie, la latence, la fiabilité de la liaison (ETX), l'état du nœud (s'il est alimenté par batterie ou non) [7].

RPL utilise DoDag ou (Destination Oriented Directed Acyclic Graph) (fig.3)(Ref [2]) où tous les nœuds WSN pointent vers une direction qui est appelée passerelle, cette passerelle peut être par exemple Cisco CGR [8].
RPL a principalement deux modèles de fonctionnement [8]:

- Storing mode : la où chaque nœud dans DoDag RPL conserve une table de routage pour son sous-arbre de nœuds
- Non-storing mode : la où seule la passerelle qui conserve une table de routage pour tous les nœuds du RPL dDoDag

Le protocole RPL utilise principalement quatre types de messages de contrôle [8] (d'autres messages existent pour Secured-RPL):

- DIS ou (DODAG Information Solicitation) : ce message est utilisé par les nœuds afin de découvrir des voisins qui parlent RPL.
- DIO ou (DODAG Information Object) : ce message est utilisé par des nœuds qui font déjà partie du réseau RPL, ce message diffuse des informations sur le réseau RPL.
- DAO ou (Destination Advertisement Object) : en Storing-Mode, chaque nœud utilise ce message pour annoncer ses routes accessibles, y compris lui-même à son parent RPL. En Non-Storing mode, ce message est utilisé par chaque nœud pour annoncer à lui-même la passerelle.
- DAOA ou (Destination Advertisement Object Acknowledgment) : ce message est utilisé comme accusé de réception pour le message DAO

[2] Reprinted from https://learning.oreilly.com/videos/internet-of-things/

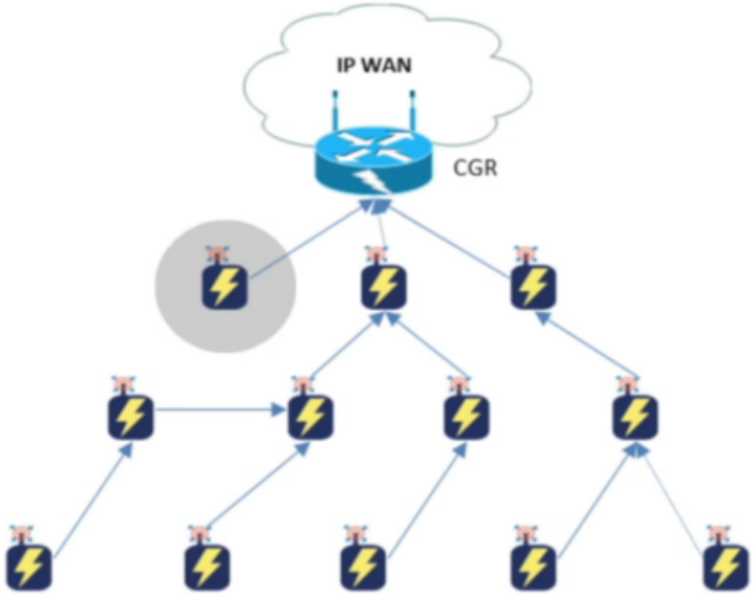


Figure 3: RPL DoDag

1.4 IETF 6LoWPAN

Traditionnellement, les réseaux de type WSN, tels que 802.15.4 étaient considérés comme incapables de parler IP. Par exemple dans le monde des réseaux d'automatisation industriels, la situation est comparable à la situation des réseaux locaux d'entreprise dans les années 80: "dois-je utiliser Token-Ring, ATM ou TokenBus?" se traduit par "dois-je utiliser ZigBee, WeMo ou Bluetooth?". La même transition vers IP qui s'est produit pour les LANs, se produit maintenant dans les mondes de l'automatisation domestique et industrielle. 6LoWPAN et RPL ont rendu cela possible.
Le groupe de travail 6LoWPAN a été formé pour le but de créer une couche d'apatation de IPV6 pour les réseaux de type LLN [9].

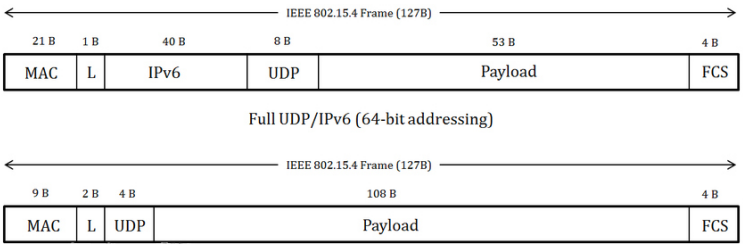


Figure 4: 6LowPAN

6LoWPAN nous permet d'effectuer certaines modifications pour l'entête IPv6, afin qu'on puisse adapter le paquet IPv6 dans un réseau de contraintes. Lorsque 802.15.4 a été publié, il supportait une trame de 127 octets. Nous pensons maintenant à un MTU IPV6 normal égal à 1280 octets qui ne peuvent pas être inséré directement dans une trame dans 127 octets. Une solution était de diviser le paquet IPV6 en un fragment et de compresser l'en-tête IPv6, donc au lieu de 40 à 60 octets, nous avons maintenant un entête IPV6 de 1 à 3 octets. La figure fig.4(Ref [3]), montre

[3]Reprinted from https://www.researchgate.net/publication/303524762

un exemple d'un fragment d'un paquet IPv6 encapsulé dans une trame 802.15.4. On peut voir le fragment avec entête compressé en bas, et celui sans compression en haut. On peut voir qu'avec une entête sans compression, le ratio maximum de bande passante utilisée qu'on peut atteindre avec une version non-compressé est de 41%, et jusqu'à 85% pour la version avec compression , ce qui a augmenté considérablement l'efficacité de la transmission [9].

1.5 Udp et CoAP

Le protocole Coap ou (Constrained Application Protocol) a été inspiré par http, c'est un sort d'implémentation légère de http pour les environnements contraintes et RESTful (standard Core under IETF) qui a été faite par l'ietf. Bien qu'il soit dérivé de Http, mais coap est un protocole basé sur udp.
Il y avait plusieurs modifications qui ont été apportées à partir de Http, incluent:

- La réduction de la taille de l'en-tête à 10 octets, ce qui est très léger par rapport à Http.
- Coap comprend uniquement les opérations http essentielles: GET, POST, DELETE, PUT.

Coap a ajouté quelques autres opérations qui n'étaient pas spécifiques à http, ces opérations comprennent publier et s'abonner (PUBLISH & SUBSCRIBE), ce qui permet un modèle observable. Ce modèle permet aux clients coap de s'abonner aux événements (par choix), puis le serveur Coap informera ces clients de la survenance des événements.

Comme Coap est basé sur udp, qui est connu pour être un protocole de transport non fiable, coaps ajoute quelques fonctionnalités pour garantir une fiabilité minimale par-dessus udp. Pour faire cela, Coap introduit deux types de messages:

- Confirmable message : où le message doit être acquitté par le destinataire, un processus de retrasmittion aura lieu au cas contraire (Coap limite le nombre de retransmission à 5).
- Non-Confirmable message : la où le message est comme un paquet udp normal, il n'a pas besoin d'un accusé de réception

Pour la sécurité, coap permet également le chiffrement du trafic de bout en bout par l'utilisation de DTLS.

2 Implementation

Dans le cadre de ce TER, on était demandé d'implémenter un algorithme d'ordonnancement centralisé. On choisi d'implémenter l'algorithme proposé dans [10] toute en effectuant des modifications qu'on a vu pertinentes qui seront décrite en détail dans cette section.
Dans la solution openwsn ou on va expérimenter notre implémentation, il existe déjà une fonction minimale d'ordonnancement distribué MSF (minimal scheduling function). Dans le cadre de notre implémentation on va laisser cette fonction mais en limitant sa point de vue sur la table d'ordonnancement interne pour chaque nœud a 20 premiers slots, comme ça on assure le lancement correct de notre réseau, et donc notre implémentation va manipuler les slots de 30 a 101, a noté aussi que la solution openwsn limite le nombre de slot qu'on peut utiliser à 30.
L'algorithme proposé dans [10] a pour l'objective de construire une table d'ordonnancement qui sera distribué a tous les nœuds de réseau IOT (chaque nœud reçoit sa partie concernant), l'algorithme suppose qu'il a la vue total de topologie de réseau, cette vue comporte :

- Les connectivitées des neouds
- Le pére (RPL Parent) choisi par chaque noeud
- Le nombre de paquets dans la queue (L'algorithme suppose que chaque nœud a un nombre connu de paquets qu'il souhaite transmettre à chaque slot-frame)

2.1 Collection de topologie et la distribution de la table d'ordonnancement

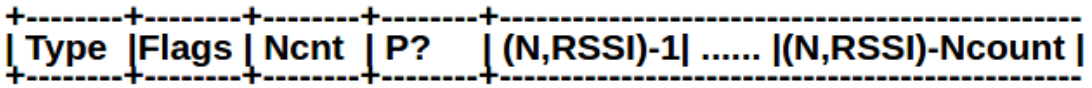
Afin de collecter et de construire le graphe de la connectivité physique, On a créé une application Coap dans tous les nœuds, cette application a pour l'objective d'annoncer la connectivité physique a un orchestrateur connu préalablement, et annoncer même le père RPL choisi. Pour faire cela on avait défini une structure simple d'un paquet coap qui permet au orchestrateur de la parser et de construire le graphe de connectivité. Chaque nœud a annonce tous les changements avec une période prédéfini.

2.1.1 Annonce de topologie

Chaque nœud possède une application coap qui est chargée d'annoncer sa connectivité physique et le parent rpl choisi s'il existe, ainsi que d'autres mesures à l'orchestrateur. Dans cette section, nous décrirons en détail comment cette application est construite.
Le nœud faisant partie de RPL DoDag, il commencera à annoncer ses connectivités physiques à l'orchestrateur. Tous les nœuds sont préconfigurés pour connaître l'adresse de l'orchestrateur, chaque nœud est également équipé d'un temporisateur qui est responsable aux événements d'annonce. Donc, sur le déclenchement du temporisateur, le nœud commence par vérifier s'il est bien un membre d'un RPL-DoDag, si oui, le nœud vérifie ensuite s'il peut obtenir un paquet, si oui, il construit un paquet coap de type PUT (avec l'orchestrateur comme destination dans la partie ip dest). Ensuite, Le nœud commence à construire le msg (voir fig.5)en incluant toutes ses connectivités, rssi pour chaque lien ... etc (d'autres informations peuvent être incluses comme la qualité du lien, la taille de la file d'attente en définissant un nouveau type de message)

la figure 5, montre le format de message utilisé par les nœuds pour annoncer leurs connectivités physiques. ce message sera bien sûr encapsulé dans un paquet Coap plus tard et envoyé à la destination (orchestrator). Certains champs du message sont obligatoires et certains sont facultatifs:

- Type : définit le type de message, trois valeurs sont définies (add_link: 0x20, add_links: 0x21, statistics1: 0x78), d'autres valeurs peuvent être ajoutées pour d'autres types de messages



Field	Short description
Type	Defines type of msg, Len=8bits, Vals=[0x21,0x20]
Flags	Defines flags applied to the msg, Len=8bits, Vals=[10000000]
Ncnt	Defines the number of (N,RSSI) elements that exist in the msg
P?	This field exists if FirstBit(flags) == 1, indicates the parent index in the elements list.

Figure 5: format de message d'annonce

- Flags : contient la liste des drapeaux appliqués au message, le premier bit est utilisé pour indiquer si un parent RPL existe, les 7 autres bits ne sont pas définis.
- Ncnt : selon le type de message, le payload contiendra différentes données qui sont généralement représentées par une liste de nuplet (eg. tuple (node_addr,rssi)), ce champ définit le nombre d'éléments existant dans le payload.
- P? : ce champ n'existera que si le bit parent est défini dans le champ flags, il indique l'index du nœud parent dans la liste des n-uplets.

2.1.2 Ordonnanceur

Dans le côté de l'ordonnanceur, on a choisi d'utiliser une bibliothèque de graphe pour bien concentré sur l'implémentation de l'algorithme d'ordonnancement. L'ordonnanceur agit comme un serveur coap qui reçoit des annonces depuis les nœuds, il mets à jour le graphe de connectivités et même le DoDag RPL.
L'orchestrateur commence par créer un serveur coap, il attend les requêtes des nœuds (add_links, add_link, _statistics) (voir fig.6). Il crée le graphe de connectivités et continue de le mettre à jour, on a pas implémentée une méthode pour détecter la terminaison où l'orchestrateur peut calculer la table d'ordonancement, donc cela doit être fait manuellement.

Jusque-là on a expliqué comment les nœuds annoncent leurs connectivités physique et leur parents RPL choisi. L'ordonnanceur a été mentionné comme une boite noire qui prends un graphe et exécute ref (section suivante), ensuite annonce la table d'ordonnancement a tous les nœuds, mais a fin de permettre ça, les nœuds doivent avoir une interface qui permettre a l'orchestrateur de manipuler leur table d'ordonnancement interne. Pour faire cela, une autre application coap vient. 6t-coap permet pour le moment d'allouer des slots pour chaque nœud. Chaque nœud agit comme un serveur coap 6t, qui reçoit les requêtes depuis l'ordonnanceur (coap 6t client)(voir fig.7). Quand l'ordonnanceur fini son calcule de table d'ordonnancement, il envoie pour chaque nœud la partie concernée (on envoie uniquement pour les nœuds qui ont besoin d'un slot TX), a la réception de ce message, le nœud ensuite parse le message, et utilise sa couche 6top pour négocier les slots qu'il a reçu avec son père RPL.

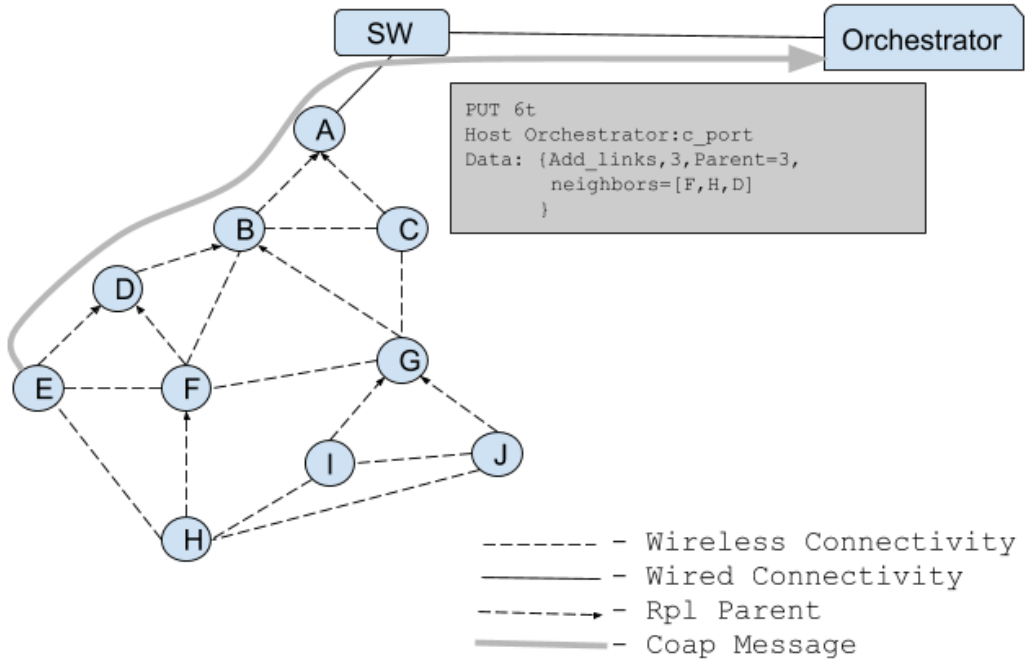


Figure 6: Add links message

	S1	S2	S3	.....	.....	SN
F1	J-G;C-A	I-G;F-B;C-A	G-B;H-F	.....	.....	.....
F2	H-F;	E-D	E-D;C-A	.....	.....	.....
F3	E-D;			.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
.....	.....	.....	.....	.....	.....	.....
F16						

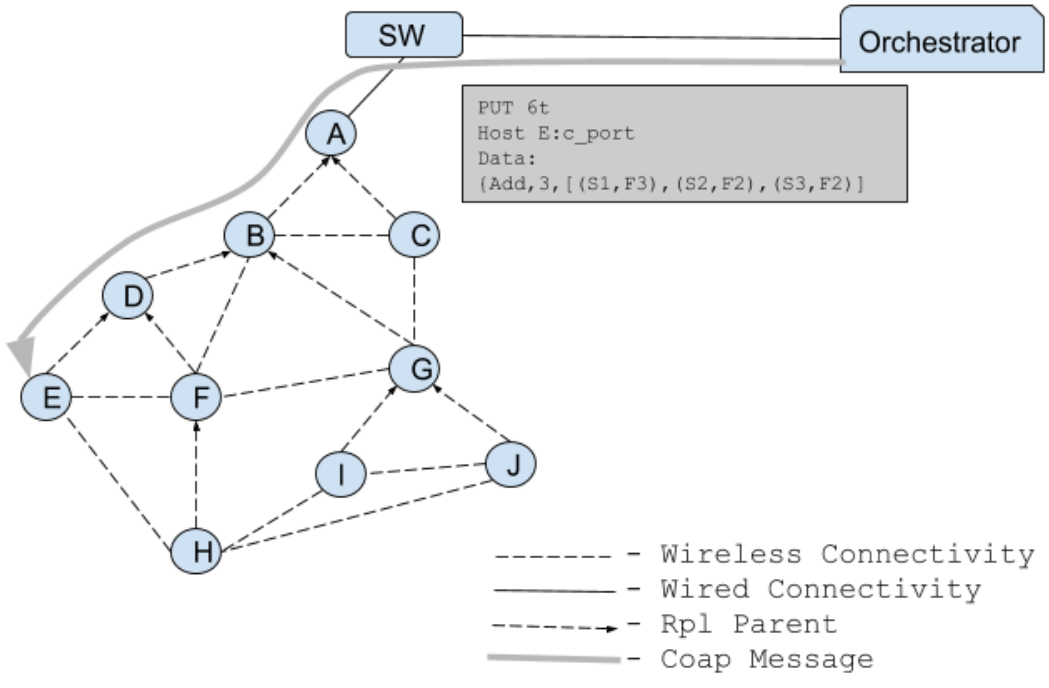


Figure 7: Diffusion de la table d'ordonancement

2.2 Description de l'implementation de l'algorithme

Dans cette section on va commencer par décrire les hypothèses de l'algorithme [10], ensuite on commencera par décrire l'algorithme proposé dans ref tout en expliquant comment on l'a implémenté.

hypothèse de l'algorithme [10]:

- Un nombre de paquets fixe soit disponible au début du super-trame par chaque nœuds, et qui doit être fourni à l'orchestrateur.
- L'algorithme ne prend pas en considération une taille fixe de super-trame. Donc pour un trafic qui nécessite un grand super-trame, l'algorithme ne prends pas en compte les parties calculées et qui sont hors de super-trame

L'algorithme proposé dans [10] prends en paramètre le graphe de connectivité P, le graphe orienté destination G (dans notre cas c'est le DoDag RPL), et l'état de queue pour chaque nœud. L'algorithme utilise principalement deux algorithmes de graphe pour construire la table d'ordonnancement globale, il modélise au premier temps les ensembles de nœuds qui peuvent envoyer en utilisant la même fréquence sous forme d'un problème de coloration, et utilise l'algorithme de coloration qui va affecter des colleurs a tous les nœuds, les nœuds qui auront la même couleurs peuvent utiliser la même fréquence, ces couleurs seront traduites en fréquences. l'algorithme modélise aussi les nœuds qui peuvent envoyer en même temps sous forme d'un problème de "graph-matching", vue qu'en réseau sans-fil tous les nœuds sont "hald-duplex" (un nœud peut pas envoyer et recevoir en meme temps, un nœud ne peux pas recevoir depuis plusieurs autres nœuds en même temps), donc il utilise un algorithme de "maximum edges matching" pour avoir la meilleure ensemble d'arêtes qui ont une somme maximale (chaque arête a un poids) et qui satisfait bien sur la condition qu'un nœud ne peut pas envoyer et recevoir en même temps, un nœuds ne peux pas recevoir depuis plusieurs autres nœuds en même temps. L'algorithme de "maximum edges matching" nécessite un graphe pondéré. L'algorithme [10] propose de mettre le poids pour un lien vers son père selon le nombre de paquets qu'il a dans sa queue interne, et la queue globale (la somme de paquets disponible pour tous les nœuds qui font partie de réseau) avec une condition qu'un lien sera alloué que si l'émetteur a au moins un paquet a envoyé. Dans le cadre de notre implémentation on a choisi de calculer le poids selon la formule 2, avec cette formule on peut aussi garantir qu'un lien sera alloué si et seulement si l'émetteur a un paquet a envoyé et aussi que la queue de récepteur n'est pas chargé, cette formule aussi permettre d'augmenter la priorité de lien quand l'émetteur est surchargé et l'inverse pour le récepteurs.

$$Weight(i,j) = Queue(i) * (MAX\_QUEUE\_SIZE - Queue(j))) \tag{2}$$

L'algorithme proposé dans [10] ne prends pas en considération la taille de slot-frame et le nombre de slots qu'un nœud peux rajouter dans sa table d'ordonnancement interne, l'algorithme simplement a pour l'objective de faire sortir tous les paquets disponibles dans le réseau, a la fin de l'exécution l'algorithme renvoi la taille de slot-frame nécessaire pour faire sortir tous les paquets dans du réseau. Dans notre implémentation la taille de slot-frame sera pris en considération et donc connu a préalable par l'orchestrateur.
L'algorithme décrit dans [10], a un processus qui sera répéter jusqu'à aboutir à une table d'ordonnancement qui permet à tous les nœuds de faire sortir leur paquets. ce processus commence par calculer le "Maximum matching graph" qui va renvoyer l'ensemble de nœuds qui peuvent envoyer en même temps, ensuite il exécute l'algorithme de coloration en donnant le graphe de connectivité et l'ensemble renvoyé par "Maximum matching graph" qui renvoi ensuite le nombre de colleurs (avec la contrainte que deux nœuds qui ont une connectivité physique ne partage pas la même couleur), le processus

finalement affecte les couleurs(fréquences) pour l'ensemble de lien renvoyé par "Maximum matching graph", il mets à jour également l'état de la queue pour chaque nœuds, et bien sur rajouter l'ensemble de liens chacun avec la fréquence correspondante dans la table d'ordonnancement pour le slot courant, ce processus sera répéter jusqu'à l'aboutissement a une somme de queue pour tous les nœuds égale à 0, en d'autres termes tous les paquets sont sorti du réseau.

3 Evaluation des performances

Dans cette section on va montrer l'évaluation de l'implementation de l'algorithme [10] avec celle qui est integrer a la solution openwsn (MSF).
Tous les résultat qui seront présenter sont obtenu de plusieurs topologies de taille varie (de 5 à 20 noeuds). Tous les expérimentations qui ont été éffectuer, utilisent la version openwsn non-modifier pour MSF, et notre implementation pour TASA.

Generation de données

Afin de tester et comparer l'implémentation dont on a discuté dans la section précédente, on a créé une application Coap rrt (modifiée de celle qui intégrait a openwsn). Chaque nœud utilise cette application pour simplement annoncer son ID, le temps de création de paquets Coap et finalement le nombre de places vide dans sa queue. Il envoie ce paquet à l'orchestrateur. Ce dernier, il prend le temps de réception et le rajoute au payload reçu, finalement il enregistre tout dans un fichier qui sera utilisé pour l'évaluation. (Pour nos tester, on a utilisé deux valeurs différentes pour la période d'annonce, 1 et 5 seconds)

Remarques pour les résultats suivants

Pour tous les résultats suivants, les tailles d'échantillons seront supérieures à 100. Selon la loi des grands nombres et le théorème central limite (et en supposant que les données soient indépendantes et identiquement distribuées) les moyennes des échantillons convergent vers une loi normale, nous avons donc pu calculer sereinement les écarts-types puis les intervalles de confiance. (Pour bien faire les choses, il aurait fallu tester la dépendance des données en utilisant des outils statistiques.)

3.1 Temps d'exécution

Dans cette section, on a expérimenté le temps d'exécution de notre implémentation pour l'algorithme [10] sur Raspberry Pi 3 Modèle B+, on a expérimenté le temps d'exécution avec un nombre de nœuds qui varie de 5 a 70, la figure 8 montre l'évolution de temps d'exécution (ms) par rapport au nombre de nœuds présents.
Pour voir quelle fonction correspond à nos données, on a essayé des modèles de régression linéaires , polynomiaux et exponentiels. on a vu que seul le modèle polynomial de degré deux (voir eq.3) qui correspond à nos données avec une erreur (Square Error) minimale.

On peut remarquer que [10] prends un temps polynomial en exécution (plus de 2.4 (s) pour un réseau de 70 nœuds, 11 (s) pour 140 nœuds et 51(s) pour 300 ne ouds), d'où ça on peut dire que pour un grand nombre de nœuds, l'algorithme ne soit pas adapté au topologie qui change régulièrement et qui nécessite un ordonnanceur dynamique et rapide.

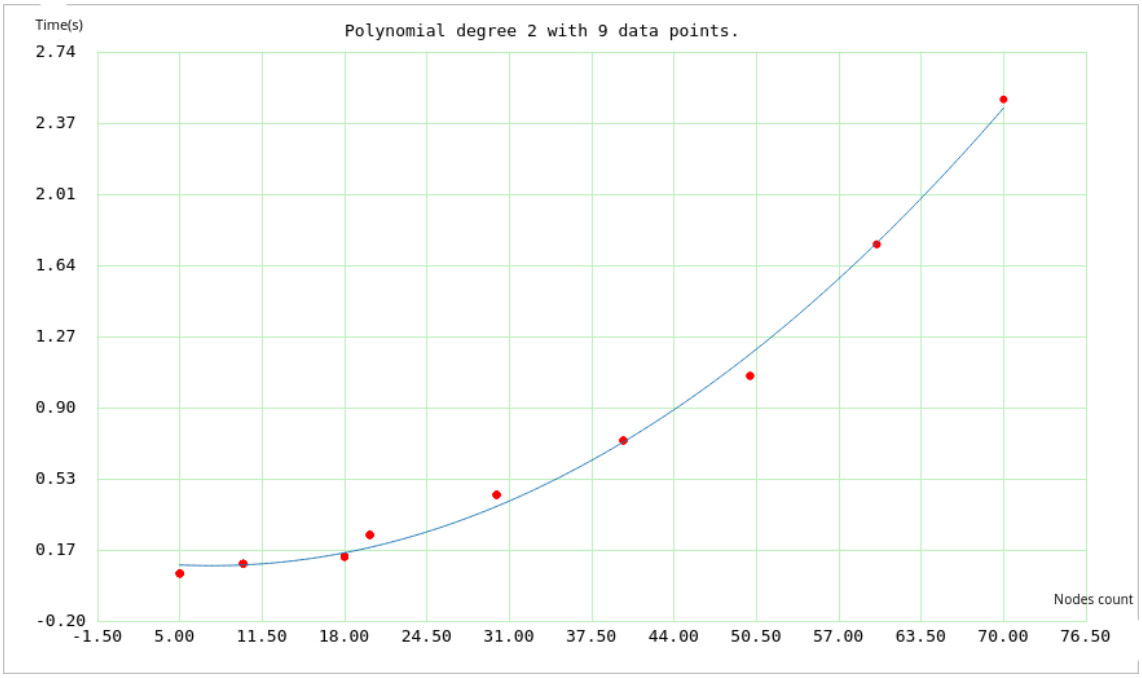


Figure 8: Temps d'exécution d'ordonnanceur

$$Exec - Time(N) = 6.06261607 * 10^{-4} * N^2 + 9.110331911 * 10^{-3} * N + 0.119062079 \quad (3)$$

3.2 Nombre de paquets livrés

Dans cette section, on cherche à savoir l'impact des algorithmes d'ordonnancement(Tasa [10],Msf) sur le nombre de paquets livrés au cours du temps, pour faire cela, on a utilisé l'application rrt Coap avec deux taux d'envoi différents (1 paquet-Coap/s et 1 paquet-Coap/5s) pour les deux algorithmes.

La figure 9 montre le ratio de paquets livrés au cours du temps, pour une topologie de 10 nœuds, et avec un taux d'envoi de 1-paquet/s. Étant donné que les nœuds n'ont pas commencé à envoyer des paquets en même temps, nous considérons la période de warm-up à partir de 80 secondes où tous les nœuds envoient 1 paquet par seconde, on peut voir que la pente de la fonction de livraison de l'algorithme [10] est supérieure à celle de msf, ce qui signifie la bande passante utilisée est également plus grande, et enfin on peut voir que [10] arrive à livrer jusqu'à 80 pour cent du total des paquets, tandis que msf jusqu'à 42 pour cent. Ça implique que msf n'arrivais pas à allouer des slots nécessaire pour le trafic génère par les nœuds. Avec une autre maniere, on avait utilisé une technique de régréssion lineare pour prédire la fonction de nombre de paquet envoié au cours du temps pour les deux ordonanceur, on a obtenu la fonction eq.4 pour l'algorithme [10] et fonction eq.5 pour l'algorithme msf, on peut voir que [10] a un slop qui est plus grand que celui de msf, ce qui signifie un plus grand tau de livraison.

$$Dtasa10(T) = 6.85428364 * T + 76.30956641 \quad (4)$$

$$Dmsf10(t) = 5.27118697 * T - 107.67140273 \quad (5)$$

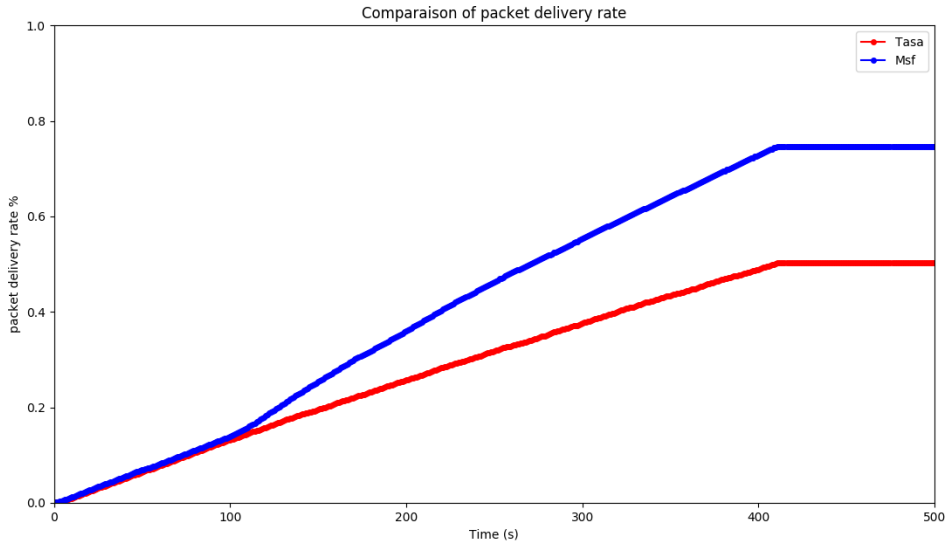


Figure 9: Nombre de paquets livrés (10 noeuds)

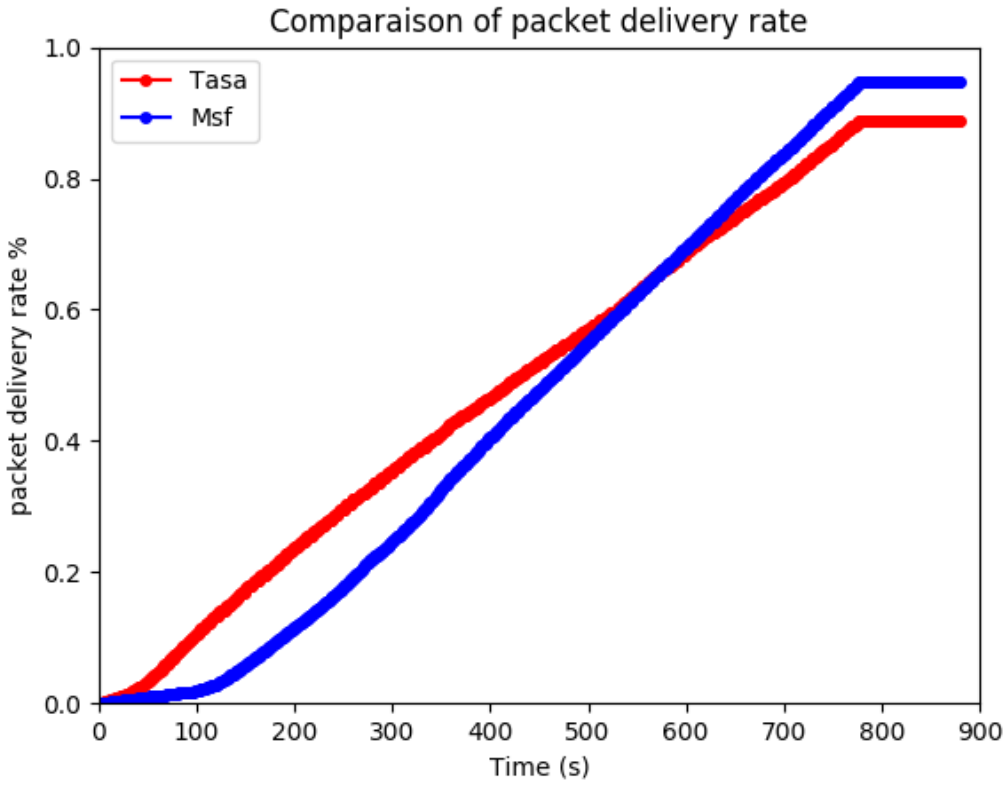


Figure 10: Nombre de paquets livrés (20 noeuds)

Pour une deuxième expérimentation on a changé le taux d'envoi a 1 paquet par 5 seconds. Vu qu'on

a un super-frame qui dure 1 second a peu prés dans la solution openwsn, notre implémentation de [10] ne pouvais pas établir une table optimale dans le sense ou aucun slot TX n'est établie si le nœud n'as pas un paquet a envoyé (meme pour RX) (L'algorithme [10] suppose que le nombre de paquets à envoyer par chaque nœud est fixe et connu au début de chaque super-frame). La figure 10 montre le ratio de paquets livrés au cours du temps, on peut voir que meme dans cette expérimentation [] a un ratio de livraison qui est plus grand que celui de msf, mais la, on peut même remarquer que msf a eu un ratio plus grand que celui de l'expérimentation précédente, en fait ça revient au fait que le DoDag RPL qui a été formé pour RPL avais un rank de 1 pour 17 nœuds (17 nœuds ont choisi le DoDag root comme parent), donc tous les ne ouds ont eu l'occasion d'établir un slot d'envoi avec le DoDag root, ce qui explique bien le tau d'envoi pris par msf. On a fait la meme chose pour cette expérimentation, on a utilisé la technique de régréssion pour les prédire la fonction de livraison par rapport du temps, on a obtenu les deux équations eq.(6 ET 7) qui correspondent a la figure 10.

$$Dtasa20(T) = 2.71223508 * T - 1.31832261 \tag{6}$$

$$Dmsf20(t) = 2.48616514 * T - 221.01568437 \tag{7}$$

3.3 Taille de fille d'attente

Dans cette section, on cherche a savoir l'impact des algorithmes d'ordonnancement(Tasa,Msf) sur la taille de fil d'attente. Un algorithm qui gére bien sa file d'attente aura moin de perte (les paquets à venir ont la chance de trouver une place vide). Pour faire cela, on a utilisé l'application rrt Coap avec deux taux d'envoi différents (1 paquet-Coap/s et 1 paquet-Coap/5s) pour les deux algorithmes.

Pour une première expérimentation, on a utilisé une application rrt coap 1 paquet/s, on a capturé la taille de files d'attente au cours de la simulation. On a calculé l'intervalle de confiance sur la moyenne de nombre de places vide dans les files d'attente avec niveau de confiance 99%, la figure fig.11 montre qu'on a deux intervalles qui ne se chevauche pas, de ça on peut déduire que pour une topologie de 10 nœuds, TASA a bien géré l'ensemble de files d'attente que MSF.

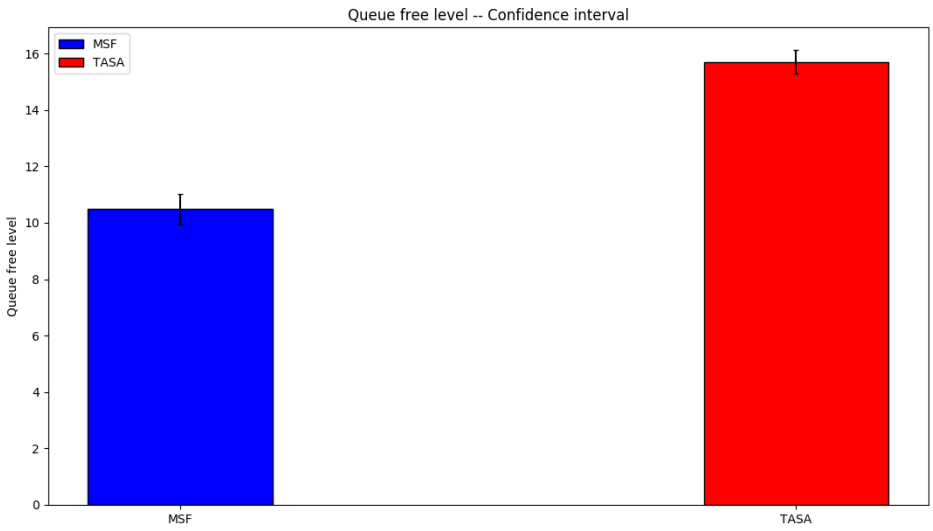


Figure 11: Queue free level (10 nodes)

Pour une deuxieme expérimentation, on a diminué le taux d'envoi a 1 paquets/ 5sec et augmenté le nombre de noeuds à 20, on a calculé l'intervalle de confiance sur la moyenne de nombre de places vide dans les files d'attente avec niveau de confiance 99%, la figure fig.12 montre que TASA gére bien l'ensemble de files d'attente mieux que MSF.

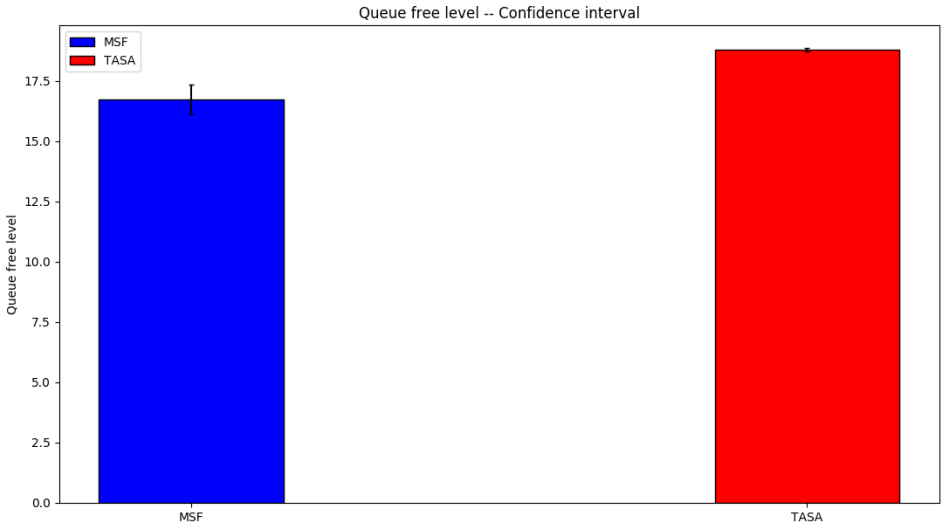


Figure 12: Queue free level (20 nodes)

De fig.(11 ET .12) On peut voir que les deux algorithmes pour un grand nombre de nœuds ont bien géré l'ensemble de leur files d'attente, pour TASA on peut expliquer sa par le fait qu'on a une table d'ordonnancement non-optimal au sens mentionné dans la sous-section précédente, il possible que les TXs attribués sans envoie ont été utilisé pour des paquets de contrôle (Comme 6top) qui permettait de dégager autres paquets de contrôles. Par rapport au MSF, comme dit dans la section précédente, le DoDAG-RPL formé pour cette topologie avait une profondeur de 1, ce qui permettait a tous les nœuds d'avoir un slot d'envoi vers leurs père qui est le nœud 1.

3.4 Délai de bout en bout

Pour la même expérimentation qu'on fait avec 10 nœuds, avec un taux d'envoi a 1 paquet/s, on a calculé l'intervalle de confiance sur la moyenne de l'ensemble de paquet envoyés avec niveau de confiance 99%, la figure fig.13 montre une comparaison entre le délai pris par l'implémentation de TASA et MSF.

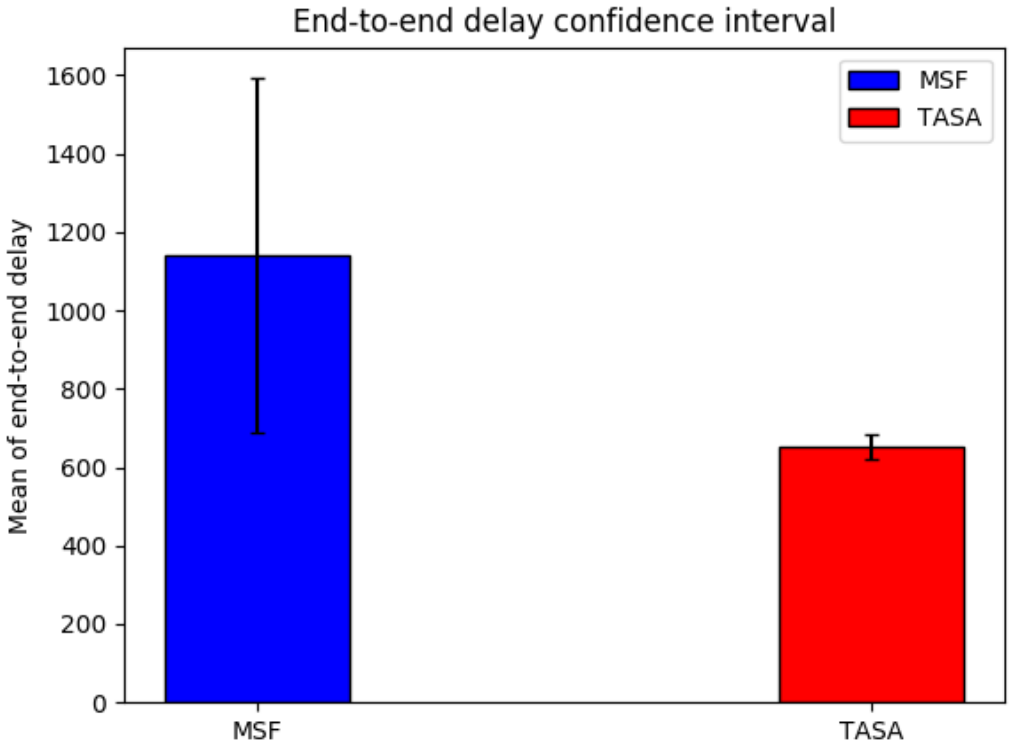


Figure 13: Délai de bout en bout (10 nodes)

On peut voir que l'algorithme tasa a géré plus efficacement le délai bout en bout pour l'ensemble des paquets transmis.

Annexe

Liens des dépôts git

Le projet Openwsn est divisé en plusieurs sous-projets:

- Openwsn-fw : Contient un firmware pour les motes
- Openwsn-Sw : Contient toutes les parties qui s'exécutent côté ordinateur.
- Openvisualizer : Une interface Web de contrôle et de visualisation
- Coap : Une implémentation python de Coap

Tous les sous-projets qu'on a utilisé dans le cadre de ce TER, proviennent du dépôt github officiel du projet :

- Official openwsn repository : `https://github.com/openwsn-berkeley`
- Forked Openwsn-fw : `https://github.com/stevlulz/openwsn-fw`
- Forked Openwsn-Sw : `https://github.com/stevlulz/openwsn-sw`
- Forked Openvisualizer : `https://github.com/stevlulz/openvisualizer`
- Forked Coap : `https://github.com/stevlulz/coap`

On a également utilisé python pour faire la partie d'analyse, vous trouverez également toutes les données qui ont été générées et utilisé : `https://github.com/stevlulz/analysis`

Teste

Etapes :

- A fin de compiler le firmware et lancer une simulation, il suffit just rentrer dans le dossier openwsn-fw : scons board=python toolchain=gcc oos_openwsn
- A fin de lancer une simulation il suffit juste de renter dans le dossier openvisualize : sudo scons runweb –sim –simCount=5
- Une fois la simulation est lancer, il faut rentrer dans le dossier openwsn-sw/software/openApps/scheduler_ta
  En exécutant python scheduler.py, ce script va lancer un serveur Coap, attends les annonces des objets, a la réception, il les affiches.

Scheduler.py offre trois commandes essentiel :

- topo : pour visualiser la vu de l'ordonanceur (connectivitées physique)
- routing : pour visualiser la vu de l'ordonanceur (RPL DODag)
- calc_sched : pour calculer et envoyer la table d'ordonancement à tous les objets (il faut assurer que tous les neouds font bien partie du DoDag-RPL).

vous trouverez également une petite démo sur le dépot git analyzer.

References

[1] R. Teles Hermeto, A. Gallais, and F. Theoleyre, "Scheduling for ieee802.15.4-tsch and slow channel hopping mac in low power industrial wireless networks: A survey," *Computer Communications*, vol. 114, pp. 84–105, Dec 2017. [Online]. Available: http://icube-publis.unistra.fr/2-TGT17

[2] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. D. Glaser, and K. Pister, "Openwsn: a standards-based low-power wireless development environment." *Trans. Emerg. Telecommun. Technol.*, vol. 23, no. 5, pp. 480–493, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/ett/ett23.html#WatteyneVKCWWGP12

[3] P. Thubert, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4," Internet Engineering Task Force, Internet-Draft draft-ietf-6tisch-architecture-28, Oct. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-6tisch-architecture-28

[4] M. V. T. C. X. Vilajosana, T. Watteyne and K. S. J. Pister, "6tisch: Industrial performance for ipv6 internet-of-things networks," *Proceedings of the IEEE*, vol. 107, pp. 1153–1165, June 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8685699/

[5] A. Mavromatis, G. Z. Papadopoulos, X. Fafoutis, A. Elsts, G. Oikonomou, and T. Tryfonas, "Impact of guard time length on ieee 802.15.4e tsch energy consumption," in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2016, pp. 1–3.

[6] Q. Wang, X. Vilajosana, and T. Watteyne, "6TiSCH Operation Sublayer (6top) Protocol (6P)," RFC 8480, Nov. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8480.txt

[7] R. Alexander, A. Brandt, J. Vasseur, J. Hui, K. Pister, P. Thubert, P. Levis, R. Struik, R. Kelsey, and T. Winter, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550, Mar. 2012. [Online]. Available: https://rfc-editor.org/rfc/rfc6550.txt

[8] O. Hersent, D. Boswarthick, and O. Elloumi, *The Internet of Things: Key Applications and Protocols*, 2nd ed. Wiley Publishing, 2012.

[9] P. Thubert, C. Bormann, L. Toutain, and R. Cragie, "IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) Routing Header," RFC 8138, Apr. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8138.txt

[10] M. R. Palattella, N. Accettura, M. Dohler, L. A. Grieco, and G. Boggia, "Traffic aware scheduling algorithm for reliable low-power multi-hop ieee 802.15.4e networks," in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, 2012, pp. 327–332.