Rapport d'analyse de performance du protocole TSCH et de l'ordonnancement Orchestra

Maxime Collette & Ethan Huret

Lien du dépôt Git :
https://github.com/EthanAndreas/Tsch-OrchestraPerformanceAnalysis

Table des matières

I. Structure du projet

I.1 Présentation du projet

L'objectif de ce projet est de comparer les performances du protocole TSCH à CSMA en mode non-beacon et de comparer les performances de l'ordonnanceur Orchestra à celui par défaut de Contiki : 6TiSCH, dans un réseau IoT avec différentes configurations. En effet, *Time Slotted Channel Hopping (TSCH)*[1] est un protocole MAC qui permet de réduire la consommation d'énergie et d'augmenter la fiabilité du réseau et *Orchestra*[2] est une solution d'ordonnancement autonome pour TSCH qui permet, avec l'utilisation de *RPL*[3], d'améliorer l'équilibre latence-énergie et de réduire la perte de paquets réseaux.

Pour réaliser ce projet, nous utilisons la plateforme de test *IoTLab* qui permet de déployer des expériences sur des nœuds IoT réels via une API ou en ligne de commande via une connexion SSH. Ainsi, nous avons créés des expériences contenant différents nœuds utilisant TSCH et Orchestra. Nous avons mis en place des groupes d'expériences ciblés sur l'analyse d'une métrique particulière, où chaque expérience contenait des configurations différentes. De ce fait, nous avons pu obtenir des résultats que nous avons pu analyser.

I.2 Type de nœuds et firmware

Pour ce projet, nous avons utilisé deux types de nœuds : *coordinateur* et *sender*. Les nœuds type *coordinateur* ont pour objectif de synchroniser les nœuds, ils reçoivent des trames de données des *sender* et leur renvoient une trame de données. Les nœuds type *sender* ont pour objectif d'envoyer des trames de données au coordinateur. Ainsi, chaque type de nœud possède un firmware écrit en C : *coordinateur.c* et *sender.c*. Un *Makefile* fournit avec les firmware permet de construire les exécutables (en *.iotlab*) adaptés aux nœuds IoT de la plateforme. Il permet de choisir le protocole MAC à implémenter ainsi que son orchestrateur. Un fichier de configuration nommé *project-conf.h* permet de modifier la configuration de TSCH et Orchestra. Pour chaque expérience, nous avons décidé d'implémenter un coordinateur et plusieurs sender afin de d'observer chaque métrique sur ce coordinateur en particulier. La principale raison de ce choix est que nous pensons que si plusieurs coordinateurs sont implémentés, la charge répartit entre eux ne sera pas forcément la même d'une expérience à une autre et ainsi, faussera les résultats.

I.3 Scripts d'automatisation

Pour automatiser le déploiement des expériences, nous avons créé des scripts qui nous permettent de déployer des expériences sur la plateforme IoTLab, de surveiller la consommation d'énergie des nœuds, de récupérer le trafic réseau et de l'analyser.

Script bash :
submit.sh : ce script permet de déployer une expérience sur la plateforme IoTLab. Il prend en paramètre le nom de l'expérience, la durée de l'expérience, le nombre de nœuds, le site sur lequel l'expérience doit être déployée et le type de protocole MAC utilisé (CSMA ou TSCH). Il crée un fichier

[1] https://docs.contiki-ng.org/en/develop/doc/programming/TSCH-and-6TiSCH.html
[2] https://docs.contiki-ng.org/en/develop/doc/programming/Orchestra.html
[3] https://docs.contiki-ng.org/en/develop/doc/programming/RPL.html

JSON contenant les informations de l'expérience et l'envoie à l'API IoTLab. Il attend ensuite que l'expérience soit déployée et affiche l'ID de l'expérience.
check_free_node.sh : ce script permet de vérifier le nombre de nœuds disponibles sur un site. Il prend en paramètre le nom du site. Il affiche les nœuds disponibles sur le site.
stop.sh : ce script permet d'arrêter une expérience. Il prend en paramètre l'ID de l'expérience à arrêter. Il envoie une requête à l'API IoTLab pour arrêter l'expérience.
monitor.sh : ce script permet de lancer une expérience avec une métrique à observer (puissance consommée ou activité radio). Il prend en paramètre le nom de l'expérience, la durée de l'expérience, le nombre de nœuds, la métrique à observer et le type de protocole MAC utilisé (CSMA ou TSCH). Il est ensuite possible d'afficher les résultats de l'expérience avec le script *monitor.py*. *Ce script ne peut pas sélectionner le site de l'expérience, car via une connexion SSH, il est uniquement possible de récupérer les informations de puissance et d'activité radio sur le site lié à son compte IoTLab.*
netcat.sh : ce script permet de récupérer le trafic réseau. Il prend en paramètre le nom de l'expérience, la durée de l'expérience, le nombre de nœuds, le site sur lequel l'expérience doit être déployée et le type de protocole MAC utilisé (CSMA ou TSCH). Il récupère les données du trafic réseau de chaque nœud jusqu'à la fin de l'expérience. Les données sont écrites dans des fichiers texte sauvegardés dans un répertoire nommé *netcat*.
monitor_netcat.sh : ce script permet de lancer une expérience avec une métrique à observer (puissance consommée ou activité radio) et de récupérer le trafique réseau. Il prend en paramètre le nom de l'expérience, la durée de l'expérience, le nombre de nœuds, la métrique à observer et le type de protocole MAC utilisé (CSMA ou TSCH). Il est basé sur *monitor.sh* et *netcat.sh*.
sniffer.sh : ce script permet de sauvegarder le trafic réseau global via la commande *serial_aggregator*. Il prend les mêmes paramètres que le script *netcat.sh*. Les résultats sont sauvegardés dans des fichiers textes dans le dossier *sniffer*.

Il est important de préciser que chaque script est indépendant l'un de l'autre ainsi, *submit.sh* n'est pas obligatoire pour exécuter *netcat.sh* ou encore *monitor.sh*.

Script python :
monitor.py : ce script permet d'observer la puissance consommée et/ou l'activité radio des nœuds. Il prend en paramètre l'ID de l'expérience, la durée, si l'on souhaite observer la puissance consommée ou l'activité radio, le type de nœud à observer (coordinateur ou sender) et le résultat souhaité, si l'argument est laissé pour vide une valeur est affichée dans le terminal (puissance moyenne consommée ou duty cycle de chaque channel), sinon il affiche directement les graphes (*plot*).
netcat.py : ce script permet d'observer le temps de connexion au réseau, le pourcentage de réussite de transmission de données (PDR) et la durée d'un ping en fonction du nombre de nœuds. Il en résulte trois graphes, où les résultats pour CSMA et TSCH sont affichés. Il ne prend pas de paramètre.

Chaque scripte python nécessite d'être exécuté dans le répertoire courant du Git.

II. Description des scenarios

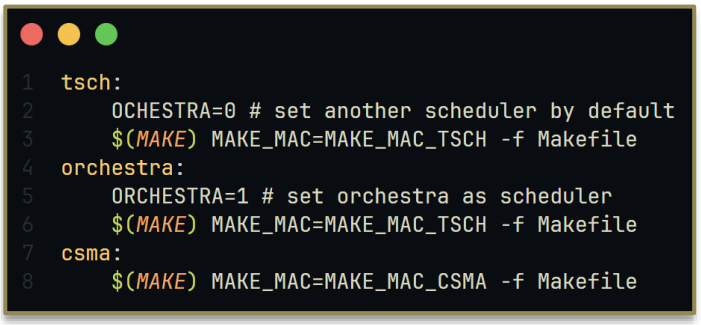
Pour les scénarios choisis, nous avons modifiés plusieurs aspects de la configuration des firmwares (protocole MAC, orchestrateur etc…) et réalisés plusieurs expériences avec des caractéristiques différentes.

II.1 Configuration des firmwares

Le but de ce projet est d'analyser les performances de TSCH et Orchestra, nous avons donc décidé d'adopter comme scénario, un cas où ils seraient utilisés et un autre cas où ils ne seraient pas. Ainsi, nous avons découpés nos scénarios de configuration en trois grands cas :

	CSMA	TSCH	Orchestra
Cas n°1	✓	✗	✗
Cas n°2	✗	✓	✗
Cas n°3	✗	✓	✓

Pour faciliter le passage d'un cas à l'autre, nous avons apportés au *Makefile*, 3 règles. La première vise à compiler les fichiers avec CSMA, la deuxième avec TSCH et un orchestrateur par défaut fournit par Contiki : 6TiSCH, et la dernière permet de compiler TSCH avec Orchestra comme orchestrateur.



```
tsch:
    OCHESTRA=0 # set another scheduler by default
    $(MAKE) MAKE_MAC=MAKE_MAC_TSCH -f Makefile
orchestra:
    ORCHESTRA=1 # set orchestra as scheduler
    $(MAKE) MAKE_MAC=MAKE_MAC_TSCH -f Makefile
csma:
    $(MAKE) MAKE_MAC=MAKE_MAC_CSMA -f Makefile
```

II.2 Configuration des expériences

Pour donner du sens aux différentes configurations des firmwares, nous avons décidé de réaliser plusieurs expériences sur chaque cas. Principalement, nous avons utilisé un format similaire pour chaque cas qui est le suivant :

- 1 coordinateur / 1 sender
- 1 coordinateur / 3 sender
- 1 coordinateur / 9 sender
- 1 coordinateur / 24 sender

Nous avons parfois réalisé une expérience en plus avec 49 sender pour tenter d'observer des accentuations de certaines métriques.
Ensuite, nous avons décidé d'adapter les durées des expériences selon les métriques que nous mesurions. Nous avons très bien pu réaliser des expériences de 2 et 10 min comme des expériences d'une heure.

III. Métriques retenues

Dans un premier temps, nous avons observés les consommations de puissance et l'activité radio du coordinateur ou d'un sender. Pour cela, nous avons utilisé un graphe montrant l'évolution de la consommation de puissance dans le temps, un graph montrant l'activité radio dans le temps ainsi que la puissance consommée moyenne et la période moyenne entre chaque onde radio reçue.

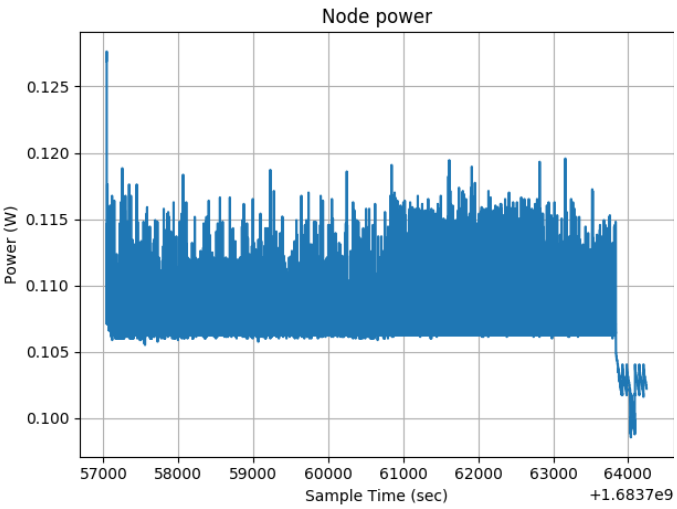
Dans un second temps, nous avons analysé le trafique réseau. Nous avons évalué la durée de connexion au réseau de tous les nœuds, le pourcentage de réussite applicatif de transmission de données pour tous les nœuds ainsi que la durée moyenne applicative d'un ping entre deux nœuds.

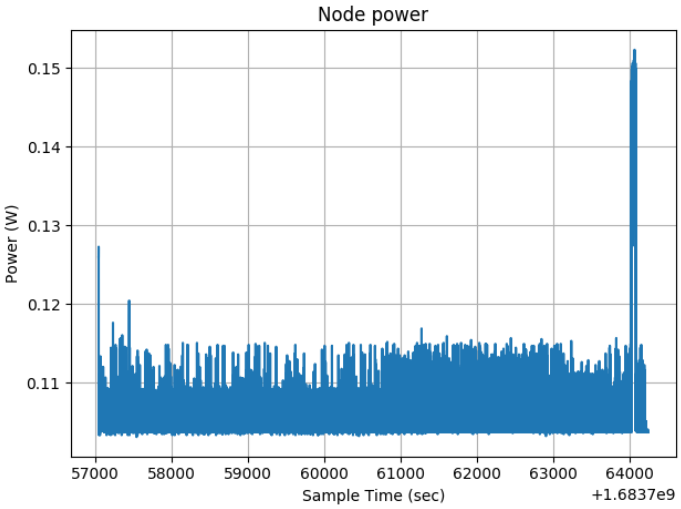
IV. Résultats

IV.1 Observation de l'évolution de la puissance

Nous avons réalisé une expérience par cas (n°1, n°2, n°3), où nous avons observé l'évolution de la puissance consommée par le coordinateur (ci-dessous à gauche) et par le sender (ci-dessous à droite). Nous avons lancé chaque cas durant 120 minutes et avec un coordinateur et un sender.

Cas n°1 : *CSMA*





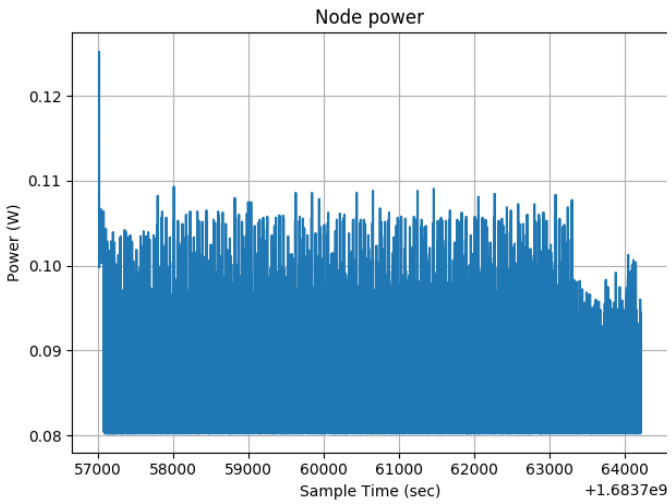
Nous avons mesuré une puissance moyenne consommée pour chaque nœud de :

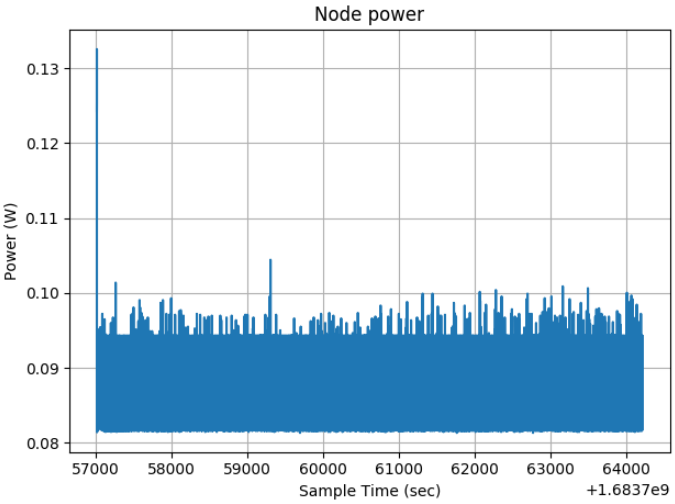
coordinateur : 31.76 mW

sender : 31.85 mW

Pour ce cas-ci, 49 messages ont été transmis correctement, ce qui représente 0,65 mW par message.

Cas n°2 : *TSCH*





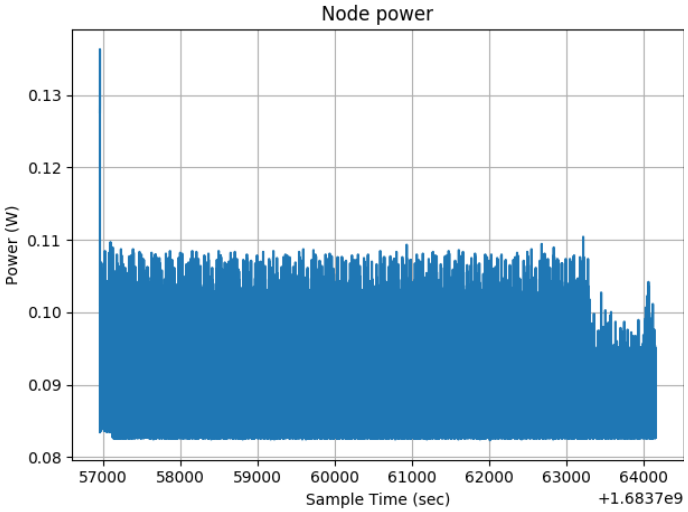
Nous avons mesuré une puissance moyenne consommée pour chaque nœud de :

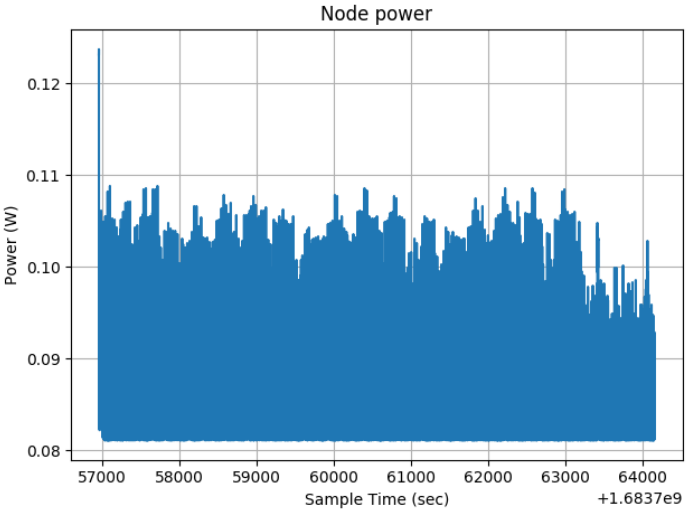
coordinateur : 26.93 mW

sender : 27.21 mW

Pour ce cas-ci, 48 messages ont été transmis correctement, ce qui représente 0,57 mW par message.

Cas n°3 : *TSCH + Orchestra*





Nous avons mesuré une puissance moyenne consommée pour chaque nœud de :

coordinateur : 26.96 mW

sender : 26.84 mW

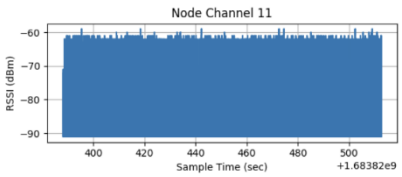
Pour ce cas-ci, 36 messages ont été transmis correctement, ce qui représente 0,75 mW par message

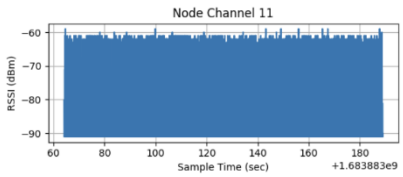
Nous avons réalisé cette expérience plusieurs fois afin de vérifier que les ordres de grandeur des puissances consommées étaient respectés pour chaque expérience.

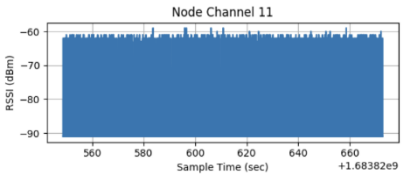
IV.2 Observation de l'activité radio

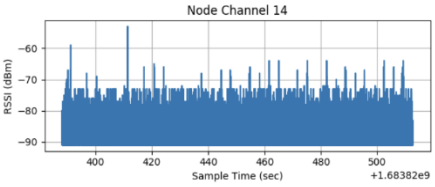
Nous avons réalisé plusieurs expériences par cas (n°1, n°2, n°3), où nous avons observé l'activité radio par channel (11 et 14). Nous avons lancé en tout 4 expériences par cas durant 2 minutes avec 2, 4, 10 et 25 nœuds. Les graphes pour CSMA sont ci-dessous à gauche, ceux pour orchestra ci-dessous au milieu et ceux de TSCH ci-dessous à droite.

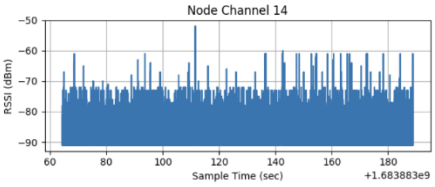
Expérience n°1 : 2 nœuds

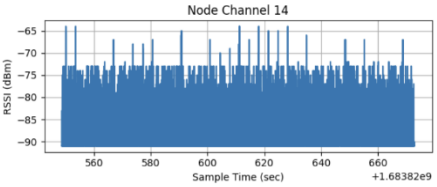




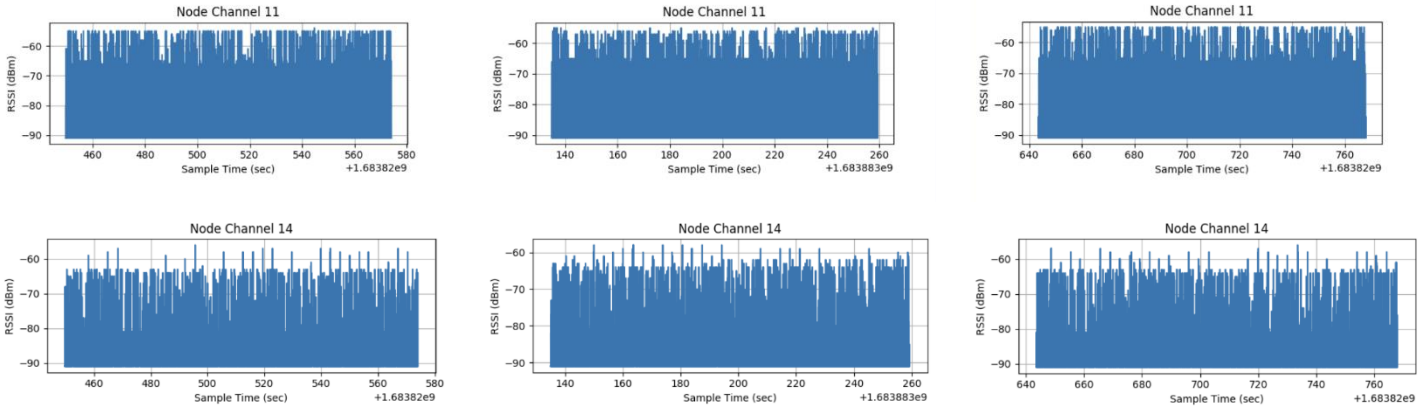




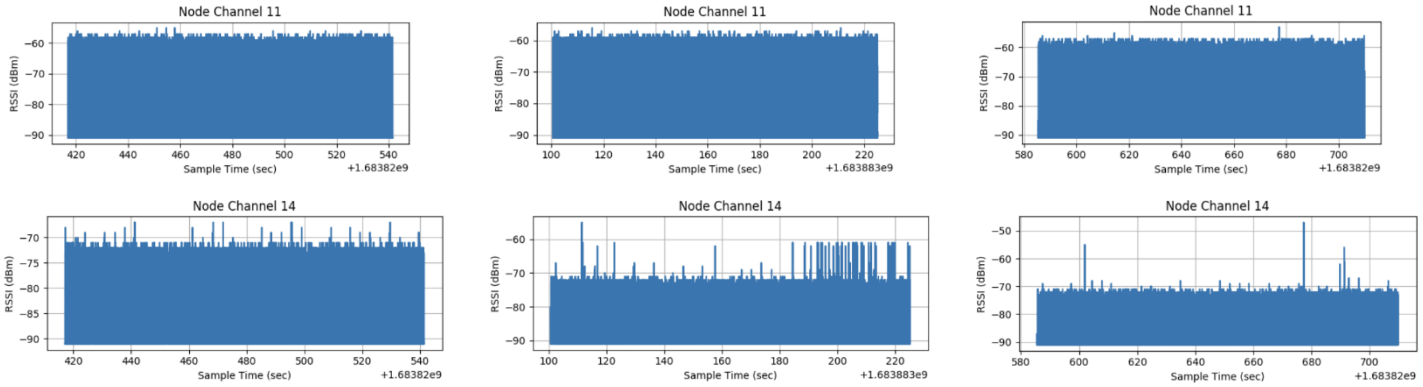




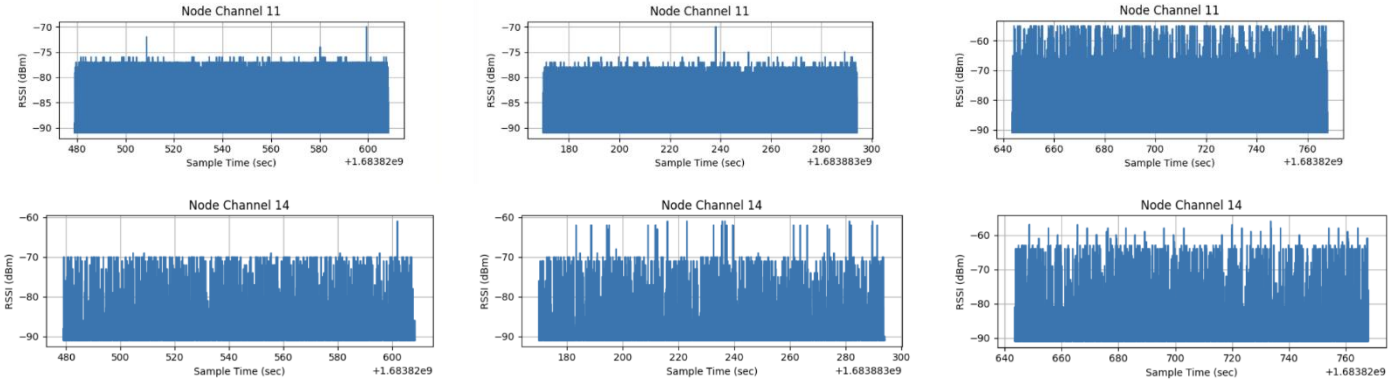
Expérience n°2 : 4 nœuds



Expérience n°3 : 10 nœuds

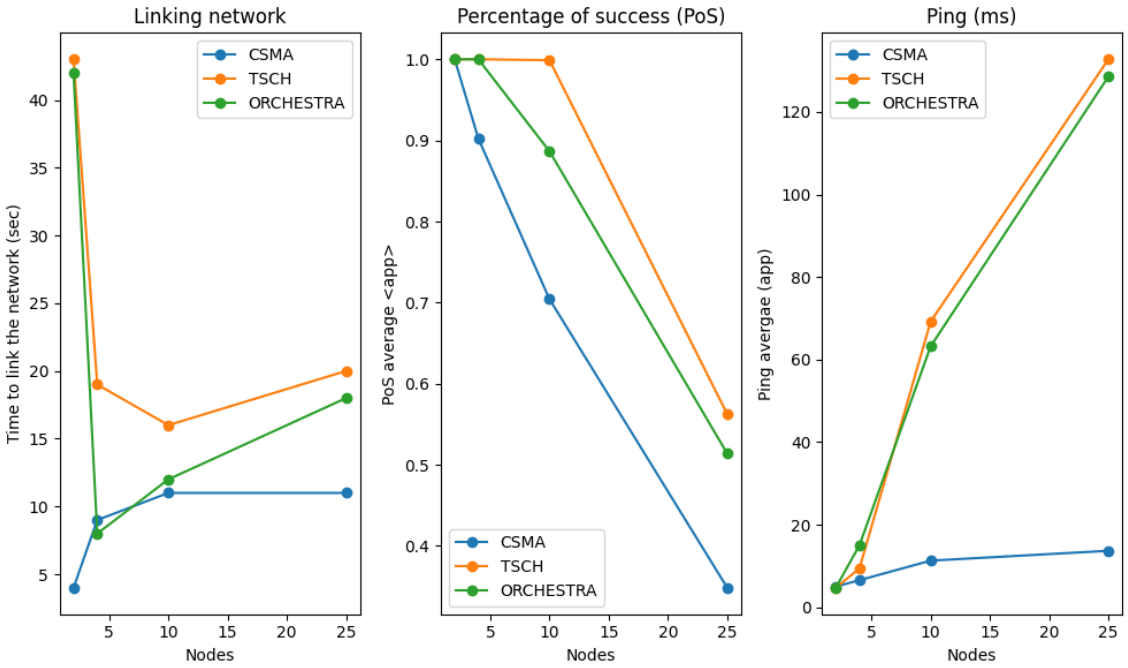


Expérience n°4 : 25 nœuds



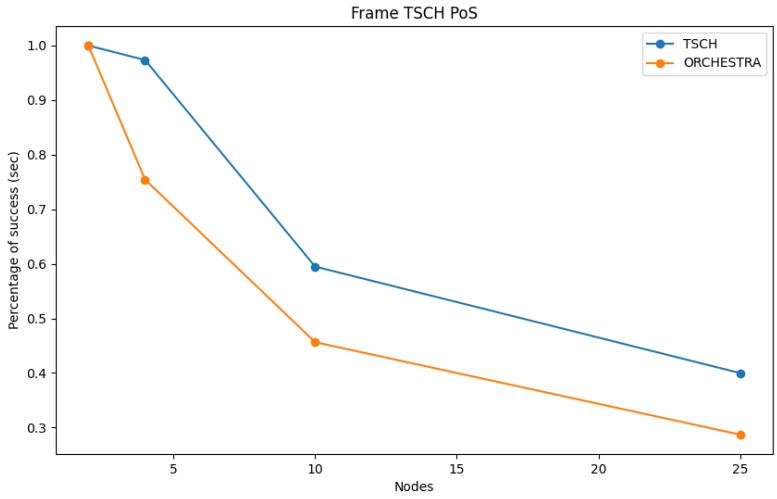
IV.3 Observation des délais et de pourcentage de réussite

Nous avons réalisé plusieurs expériences par cas (n°1, n°2, n°3), où nous avons observé le temps de connexion au réseau (durée d'initialisation de RPL), le pourcentage de réussite applicatif de transmission de données et le délai de communication applicatif entre coordinateur et sender (ping). Nous avons tracé des graphes montrant l'évolution de ces caractéristiques selon le nombre de nœuds présents (2, 4, 10 et 25 nœuds).



IV.4 Observation du pourcentage de réussite selon l'orchestrateur

Nous avons réalisé une expérience où nous avons observé le pourcentage de réussite de transmission de données entre deux orchestrateurs : celui par défaut de Contiki, *6TiSCH* et *Orchestra*.



V. Analyse des résultats

V.1 Analyse de la puissance consommée

Nous pouvons observer que ces graphes nous donnent des évolutions similaires. Pour les deux types de nœud, nous observons un pic de puissance au début de l'expérience, ce qui représente la phase d'initialisation. Ensuite, la puissance se stabilise entre une valeur plus grande (pic) et une

valeur plus faible (vallée), ce qui montre l'alternance entre mode réveillé (écoute ou transmission) et mode endormi. De plus, au vu des puissances moyennes consommées, nous pouvons voir que TSCH consomme moins que CSMA, et qu'Orchestra n'apporte pas de changement majeur à la consommation énergétique vis-à-vis de 6TiSCH.

V.2 Analyse de l'activité radio

Nous pouvons observer que les graphes pour le channel 11, quelque soit l'expérience et le cas, possèdent une évolution similaire, seulement les RSSI moyen change selon l'expérience. Il est le même dans chaque cas et expérience sauf pour l'expérience à 25 nœuds, où CSMA et TSCH + Orchestra montre une baisse contrairement à TSCH + 6TiSCH. Ce qui est normal puisqu'il y a plus de nœuds et donc plus d'activité radio, ce qui par conséquent diminue l'intensité des signaux. Cependant, pour TSCH + 6TiSCH, nous pouvons supposer que c'est dû à 6TiSCH que le RSSI moyen reste aussi haut.

Nous pouvons observer que les graphes pour le channel 14, ont des évolutions similaires selon les cas, et qu'il y a de moins en moins de pic isolé plus le nombre de nœuds augmentent. Quant au RSSI moyen, il est équivalent dans chaque cas selon l'expérience. Nous pouvons voir que TSCH + Orchestra montre plus de disparités que les autres cas, ce qui peut être justifiable du fait d'une mauvaise configuration de notre part ou d'une influence d'autres expériences sur l'activité radio dans le channel 14.

De plus, nous avons calculé les duty cycle de chaque channel, cependant, nous obtenons des valeurs telles que 50,00 ± 0,05 % . Il est donc difficile d'en conclure quelque chose, d'autant plus qu'il n'y pas de proportionnalité selon les protocoles utilisés ou le nombre de nœuds.

V.3 Analyse des délais et de pourcentage de réussite

Pour le temps de connexion au réseau, nous voyons tout d'abord que pour TSCH et TSCH avec Orchestra qu'il y a un temps important de connexion au réseau pour un sender et un coordinateur, mais que cela redevient acceptable pour plus de nœuds. Nous remarquons que CSMA est le plus rapide à converger et qu'Orchestra est un peu plus rapide que 6TiSCH. Le temps de convergence prend en compte la convergence de RPL et aussi le temps de planification de TSCH (d'où l'écart mesuré).

Pour le pourcentage de réussite, ils sont tous sans erreurs avec un seul sender, mais ils décroissent tous significativement en fonction du nombre de nœuds. Cependant, TSCH et aussi dans de moindre mesure Orchestra, gardent un pourcentage de succès de 100% même au-delà de 1 sender (> 4 pour TSCH + Orchestra et > 10 pour TSCH + 6TiSCH). Nous pouvons faire l'hypothèse que TSCH grâce notamment à l'orchestrateur maintient un pourcentage de réussite de 100% jusqu'à un certain nombre de nœuds, mais qu'au-delà de ce nombre, la charge devient trop importante pour le coordinateur.

Pour le ping, CSMA garde pratiquement un ping constant alors que celui de TSCH et Orchestra augmente grandement avec le nombre de nœud. Cela est sûrement dû au fait que TSCH est un protocole plus lourd pour le coordinateur donc en fonction du nombre de nœuds, le temps pour répondre au sender est plus long, alors que le mode CSMA est beaucoup plus simple.

Nous pouvons donc en déduire que TSCH améliore la fiabilité au prix de la rapidité, alors que CSMA améliore la rapidité au prix de la fiabilité.

V.4 Analyse du pourcentage de réussite selon l'ordonnanceur

Nous pouvons observer que le pourcentage de réussite au niveau MAC est à l'avantage de TSCH (avec 6tisch), pourtant Orchestra promet une amélioration de la rapidité. Cependant, Orchestra contient de nombreux paramètres de configuration, et nous avons fait le choix de prendre la configuration par défaut qui n'est peut-être pas la plus adaptée dans notre cas de figure. Nous aurions pu par exemple adapter la taille des slotframe.

De plus nous avons remarqué que pour des expériences plus longues que 10 min (plus de 2 heures), Orchestra devient plus performant que 6TiSCH.
Par exemple, nous avons réalisé deux expériences de 2 heures avec 4 nœuds, une avec TSCH et 6TiSCH et l'autre avec TSCH et Orchestra. Nous avons obtenu les résultats suivants :

	TSCH + 6TiSCH	TSCH + Orchestra
Pourcentage de réussite applicatif	99.27 %	99.46 %
Délai de communication applicatif	20.8s	11.0s
Pourcentage de réussite via les logs de TSCH	70.8 %	98.1 %

VI. Conclusion

Nous avons donc pu observer les promesses de TSCH vis-à-vis de CSMA en mode non-beacon, c'est-à-dire, réduire la consommation énergétique et augmenter la fiabilité en réduisant la perte de paquets. Pour ce qui est d'Orchestra, nous n'avons pas pu observer les promesses faites vis-à-vis de 6TiSCH. Cependant, dans un second temps, nous avons pu les observer sur une expérience de 2 heures, où cette fois-ci Orchestra était plus performant que 6TiSCH tout en ayant une consommation énergétique similaire. Nous en concluons donc que pour aboutir à de meilleures analyses et comparaison entre ordonnanceur, il faut réaliser de longues expériences.

Nous avons été limités sur certains aspects de ce projet, notamment à propos de l'observation des états de chaque nœud (endormi, écoute, transmet, idle). Nous n'avons malheureusement pas trouvé de moyen d'observer quand est-ce que les nœuds étaient endormis ou en mode idle.