Réseau sans-fil

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

Table des matières 2

I. Structure du projet 3

I.1 Présentation du projet 3

I.2 Type de nœuds et firmware 3

I.3 Scripts d’automatisation 3

II. Description des scenarios 4

II.1 Configuration choisie 5

II.2 Expérience choisie 5

III. Métriques retenues 5

IV. Résultats 6

V. Analyse des résultats 7

1. Structure du projet

I.1 Présentation du projet

L'objectif de ce projet est de comparer les performances du protocole TSCH et de l’ordonnanceur Orchestra dans un réseau IoT avec différentes configurations. En effet, Time Slotted Channel Hopping (TSCH)[[1]](#footnote-2) est un protocole MAC qui permet de réduire la consommation d'énergie et d'augmenter le débit du réseau et Orchestra[[2]](#footnote-3) est une solution d'ordonnancement autonome pour TSCH qui permet, avec l’utilisation de RPL[[3]](#footnote-4), de réduire la latence et 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 et de leur envoyer des trames de données. Les nœuds type sender ont pour objectif de recevoir les trames de données du coordinateur et de les renvoyer. 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 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.

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 moy enne est affichée dans le terminal (puissance moyenne ou fréquence moyenne d'activité radio), sinon il affiche directement les graphes (plot) ou il les sauvegarde (save).

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.

1. 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 | ✕ | ✓ | ✓ |

Une image contenant texte

Description générée automatiquementPour 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 et la dernière permet de compiler TSCH avec Orchestra comme orchestrateur.

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.

1. 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.

1. Résultats
2. Analyse des résultats

1. https://docs.contiki-ng.org/en/develop/doc/programming/TSCH-and-6TiSCH.html [↑](#footnote-ref-2)
2. https://docs.contiki-ng.org/en/develop/doc/programming/Orchestra.html [↑](#footnote-ref-3)
3. https://docs.contiki-ng.org/en/develop/doc/programming/RPL.html [↑](#footnote-ref-4)