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

I. Structure du projet 3

I.1 Présentation du projet 3

I.2 Firmware des nœuds 3

I.3 Scripts d’automatisation 3

II. Description des scenarios 4

III. Métriques retenues 5

IV. Résultats 6

V. Analyse des résultats 8

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. Un fichier de configuration nommé project-conf.h permet de modifier la configuration de TSCH et Orchestra, c'est ce que nous avons principalement utilisé pour chaque expérience. 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.

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 surveiller la consommation d'énergie des nœuds. Il prend en paramètre le nom de l'expérience, la durée de l'expérience, le nombre de nœuds, la métrique à observer (puissance électrique ou activité radio) et le type de protocole MAC utilisé (CSMA ou TSCH). Il attend que l'expérience de se termine et récupère les données de consommation du coordinateur. Il crée ensuite des graphiques à partir de ces données à l'aide d'un script python qui sont sauvegardés dans un répertoire nommée plot.

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.

netcat\_filter.sh : ce script permet d'analyser le trafic réseau. Il prend en paramètre le chemin relatif d'un fichier. Il analyse ce fichier pour en extraire le nombre de paquets perdus et la latence du réseau. Les résultats sont affichés dans le terminal.

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.

1. Description des scenarios
2. Métriques retenues
3. Résultats
4. 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)