Compte rendu 5 semaine de stage en mise en place d’un réseaux ad-hoc et service de journalisation

Sommaire

I-Présentation de l’ONERA Toulouse

Introduction

Dans le cadre de mon stage, j’ai intégré le centre de Toulouse de l’ONERA (Office national d’études et de recherches aérospatiales), principal organisme public français de recherche dans le domaine aéronautique, spatial et de défense. Il contribue au développement scientifique et technologique du secteur et accompagne les industriels dans leurs projets d’innovation.

Présentation générale de l’ONERA

Créé en 1946, l’ONERA est un établissement public à caractère industriel et commercial (EPIC) placé sous la tutelle du ministère des Armées. Il mène des recherches amont et appliquées pour la conception et l’amélioration des aéronefs, satellites et systèmes de défense. Ses missions principales sont de développer les connaissances scientifiques, concevoir et exploiter des moyens expérimentaux, accompagner les industriels et former des doctorants et ingénieurs.

Le centre ONERA de Toulouse

Situé sur le campus scientifique de Rangueil, le centre de Toulouse se trouve au cœur d’un environnement marqué par l’industrie aéronautique et spatiale, favorisant les collaborations académiques et industrielles. Les équipes travaillent sur le traitement de l’information et les systèmes embarqués, l’électromagnétisme et les technologies radar, l’optique, la physique de l’environnement spatial et la modélisation multiphysique.

Activités et rôle

L’ONERA Toulouse conduit des recherches fondamentales et appliquées, élabore des modèles théoriques, conçoit des outils numériques et réalise des expérimentations pour améliorer la compréhension et la performance des systèmes aérospatiaux. Le centre participe à des projets nationaux et européens et accueille des doctorants et stagiaires, renforçant le lien entre recherche et formation.

II- Objectif du stage

L'objectif principal de mon stage a été de développer et tester des applications fonctionnant sur des **Raspberry Pi Zero 2 W** au sein d'un réseau mesh ad-hoc. Ce réseau repose sur des nœuds autonomes communiquant entre eux et coordonnés par un **nœud maître** basé sur une Raspberry Pi 3. L'expérimentation visait à étudier à la fois la gestion du trafic et la génération de données au sein du réseau.

Le stage s'est articulé autour de deux applications distinctes mais complémentaires. La première application a été conçue pour **surveiller le réseau**. Elle collecte et affiche des informations sur l'état des nœuds, la disponibilité des liaisons et le trafic passant par chaque nœud. Ces données permettent de visualiser le fonctionnement du réseau en temps réel et d'identifier d'éventuels goulots d'étranglement ou pertes de paquets. Cette application a été développée avec un souci de légèreté et de compatibilité avec les ressources limitées des Raspberry Pi Zero 2 W.

La seconde application a pour rôle de **générer du trafic applicatif** afin de tester le réseau. Elle envoie différents types de données, allant de simples messages texte à des flux vidéo ou des pings entre nœuds. L'objectif est de simuler des scénarios d'utilisation réalistes et de mesurer les performances du réseau en conditions variées. La combinaison des deux applications permet d'analyser la robustesse, la réactivité et l'efficacité du réseau mesh.

Le choix des Raspberry Pi Zero 2 W s'est avéré pertinent pour ce stage, car ces cartes offrent un compromis entre puissance de calcul et faible consommation, tout en permettant de simuler un réseau de nœuds relativement dense. Le nœud maître sur Raspberry Pi 3 centralise la collecte de données et la coordination du réseau, garantissant une expérimentation fiable et reproductible.

Au cours du stage, le développement a inclus la programmation des applications, la configuration des nœuds et la mise en place d'un protocole de communication simple mais efficace. Les tests ont permis de valider le bon fonctionnement du réseau, de mesurer la latence et le débit des communications, et de générer des rapports détaillés sur le comportement du réseau selon différentes charges applicatives.

Cette expérience m'a permis de mettre en pratique des compétences en **programmation embarquée**, en **réseaux ad-hoc** et en **analyse de performance**, tout en travaillant dans un environnement proche des contraintes industrielles. Le stage a également renforcé ma compréhension des défis liés aux systèmes distribués et à la communication entre multiples nœuds dans un réseau dynamique.

III-Outils et choix matériels

Pour réaliser mon expérimentation sur un réseau mesh ad-hoc, j'ai choisi d'utiliser des **Raspberry Pi Zero W2 (voir annexe 1)** comme nœuds du réseau et une **Raspberry Pi 3 (voir annexe 2)** comme nœud maître. Ce choix a été motivé par plusieurs critères : accessibilité, flexibilité et faible consommation énergétique. Les Raspberry Pi offrent une interface simple pour développer et tester des protocoles de communication, tout en restant proches des contraintes matérielles que l'on pourrait rencontrer sur des systèmes embarqués réels.

D'autres solutions matérielles, comme les cartes Arduino ou les modules ESP32, ont été envisagées. Cependant, un ESP32 ne peut pas faire fonctionner nativement un véritable réseau ad hoc 802.11 IBSS car sa pile Wi-Fi, fournie par ESP-IDF, ne prend en charge que les modes station et softAP et n'expose pas le contrôle MAC de bas niveau nécessaire au routage ad hoc conforme aux normes. Cependant, il peut toujours former des réseaux dynamiques multi-sauts en utilisant des cadres Espressif comme ESP-MESH ou des communications entre pairs sans connexion via ESP-NOW. Dans ces configurations, les développeurs peuvent mettre en œuvre une logique de routage ou de transfert personnalisée au niveau de l'application ou de la couche maillée, ce qui permet une communication multi-nœuds à autorégénération, mais avec des limites de ressources plus strictes et moins de souplesse de protocole que les véritables mises en œuvre de MANET IBSS. Pour avoir accès à ces fonctions, nous avons besoin d'un environnement Linux.

L'utilisation de Raspberry Pi a permis de disposer d'un environnement complet, capable d'exécuter le logiciel de gestion du réseau et de simuler des scénarios réalistes, tout en restant modulaire pour l'ajout ou le retrait de nœuds. Ainsi, ce choix permet de justifier l'expérimentation sur un réseau fonctionnel et reproductible, tout en offrant un cadre proche des applications industrielles pour lesquelles la scalabilité et la robustesse des communications sont des facteurs clés.

IV-Travaux réalisé

A-Choix des OS

Le choix du système d'exploitation constitue une étape clé pour le développement et le déploiement des applications sur un réseau de Raspberry Pi. Dans le cadre de mon stage, j'ai étudié plusieurs distributions afin de sélectionner celle qui offrirait la meilleure stabilité, compatibilité matérielle et facilité de configuration pour les nœuds du réseau et le nœud maître.

Plusieurs options ont été évaluées. La distribution **Trixie** a été envisagée en raison de ses fonctionnalités modernes et de son support pour les dernières versions des logiciels. Cependant, elle a été écartée car il s'agit d'une licence récente, encore peu utilisée dans les communautés techniques. La documentation disponible reste limitée, ce qui aurait pu poser des difficultés pour l'installation, la configuration et le débogage des applications développées pour le stage. **Non retenu.**

La distribution **Ubuntu Server** a également été considérée. Elle bénéficie d'une large base d'utilisateurs et d'un suivi actif. Toutefois, son utilisation sur les Raspberry Pi a été rendue compliquée par des problèmes liés à l'outil Imager, entraînant des erreurs d'installation et des incompatibilités avec certains périphériques. Ces difficultés auraient allongé le temps de mise en place et introduit des risques pour la stabilité des expérimentations. **Non retenu.**

Finalement, le choix s'est porté sur la distribution **Debian Bookworm**, adaptée aux Raspberry Pi et relativement stable. Bien que cette licence soit plus ancienne, elle présente plusieurs avantages décisifs : la documentation est abondante, les outils de paramétrage sont simples à utiliser et la compatibilité avec le matériel Raspberry Pi est éprouvée. Cette distribution permet d'installer rapidement les bibliothèques nécessaires et de déployer les scripts expérimentaux sans rencontrer de blocages techniques majeurs. **Solution retenue !**

L'utilisation de Debian Bookworm a également facilité l'intégration avec le nœud maître basé sur une Raspberry Pi 3, garantissant une uniformité dans l'environnement logiciel et une meilleure gestion des communications au sein du réseau mesh. Ce choix contribue à assurer la reproductibilité des tests, la fiabilité des mesures de trafic et la simplicité des mises à jour logicielles au cours du stage.

En résumé, Debian Bookworm a été retenu pour sa stabilité, sa compatibilité matérielle, sa documentation complète et sa facilité de configuration, répondant ainsi aux besoins spécifiques du projet. Ce choix s'inscrit dans une logique de minimisation des risques techniques et de maximisation de l'efficacité lors de la mise en œuvre des applications sur les nœuds du réseau.

B-Construction d’un script

1-Script OLSR (gamma.sh)

Dans cette section nous allons décrire la structure des codes rédigés sous un format de sommaire.

1. Lancement
2. Téléchargement olsr
3. Log
4. Mise en place de service

Avant le lancement des étapes

Mise en place d’un système d'alerte en cas de mauvais lancement du script,
Mise en place des journaux et système intéractif de log.

Étape [1/18]

Mise en place du fichier qui permet de traverser le proxy,
Mise en place de la date,
Installation des apt.

Étape [2/18]

Désactivation de service.

Étape [3/18]

Activation de service.

Étape [4/18]

Mise en place du serveur de synchronisation de date et d’heure .

Étape [5/18]

Installation et décompression OLSR

Étape [6/18]

Tentative de configuration OLSR

Étape [7/18]

Exclusion de Wlan0

Étape [8/18]

Passage de Wlan0 en IBSS

Étape [9/18]

Passage en IBSS et log

Étape [10/18]

Script de vérification et contrainte IBSS

Étape [11/18]

Ping broadcast

Étape [12/18]

Forçage du lancement de OLSR

Étape [13/18]

Check OLSR ok au boot

Étape [14/18]

Script .json situation de teste

Étape [15/18]

Script .json maître

Étape [16/18]

Script .json 600s

Étape [17/18]

Service de reset de l’ip après le boot

Étape [18/18]

Log et fin du script

Mise en service

La mise en service du script se déroule en **3 étapes** bien délimitées. La première partie sert de gardien afin de vérifier que vous soyez bien en sudo et que vous avez bien pensé à joindre le numéro qui qualifie l'ip du script.
Il y a ensuite la création d'un fichier de log dans **/var/log/logmaj.txt** qui enregistre toutes les info qui seront remontées dans le script et ensuite, il y a l'ajout d'un **entête** pour désigner le début du script.

Étape [1/18]

Mise en place d'une condition qui **supprime** le fichier **95proxy** si il existe déjà, puis, il le **copie** dans le fichier **/etc/apt/apt.conf.d/** ensuite **mise à jour manuel de la date** afin de ne pas avoir d'apt qui rate, puis installation de tous les **apt nécessaires** à la mise en place du script.

Étape [2/18]

Désactivation du service **NetworkManager** et du service **wpa_supplicant** qui empêche le passage en mode IBSS

Étape [3/18]

Log du lancement de **systemd-networkd** et **lancement** de ce service.

Étape [4/18]

Mise en place de deux boucles, une première, qui va s'occuper d'identifier le nœud **maître** (10.0.0.1) et installation d'un **serveur chrony** dessus afin de distribuer l'heure et la date à toutes les machines clients qui viennent chercher ces informations directement dessus. Le deuxième algorithme vient chercher la **date** sur toutes les ip du réseaux 10.0.0.x/16 sur le nœud maitre (le 10.0.0.1) afin de mettre toutes les machine au là.

Étape [5/18]

Déplacement de **OONF** et don des droit superuser sur le dossier,
Mise en place de la **liste cmake**,
Lancement des installations,
Ajout d'une boucle de vérification à la fin afin de certifier que les **librairies** sont bien installées.

Étape [6/18]

Tentative de **configuration de olsr,** cependant, il y as un problème majeur, lors la configuration, quand je souhaite demander à OLSR d'utiliser des

paramètres de personnalisation, il cherché à piocher le nom de l'interface ou il souhaite modifier les paramètres de manière **dynamique** et, il échoue à chaque fois, cela rendait donc les **modifications impossible** et donc sont utilisation par la même occasion.

Étape [7/18]

Dans cette étape, on essaye de faire **sortir Wlan0** des griffes du méchant système qui essaye continuellement de nous remettre Wlan0 en interface de fonctionnement simple. C'est à dire qu'il cherche à sortir Wlan0 du **mode IBSS** que l'on essaye de lui imposer.

Étape [8/18]

On tente de mettre les **paramètres d'exclusion** des services problématique en appliquant directement dans le fichier qui s'occupe de gérer la configuration réseaux ces paramètres.

Étape [9/18]

Tentative de remise en place du **mode IBSS** (car en général la première fois cela avait sauté) ajoute de **log à chaque étape** afin de prendre le recule nécessaire et d'identifier les étapes qui pose problème afin d'aligner l'étape suivante (Je conseil de garder cette étape simplement pour éviter que l'interface ne soit plus en mode IBSS au redémarrage)

Étape [10/18]

Mise en place d'un autre fichier qui explique au système que l'interface Wlan0 est maintenant en IBSS et affiche l'état du daemon directement depuis la zone d'exécution après sont redémarrage.

Étape [11/18]

Cette étape sert à mettre en place un script qui vas se lancer au démarrage et envoyer des **requêtes ICMP** en broadcast un peu n'importe ou sur le réseaux **10.0.0.x** afin de créer la liste des routes disponibles dans le réseaux et de pouvoir les afficher avec **ip neigh show dev wlan0**.

Étape [12/18]

Tentative de configuration d'un service de **OLSR** (ça marche pas) et tentative de **démarrage du daemon** (c'est toujour un échec)

Étape [13/18]

Script qui lance une boucle qui va **chercher pendant 300s** si OLSR retourne Active quand on fait un **systemctl status OLSRv2.service**. Si le service ne tourne pas alors le **script crash**. Il y a ensuite une vérification que le service **Telnet** et le service de **Fowarding** tourne sinon crash du script.

Étape [14/18]

Mise en place d'un script un peu différent des autres mais qui en finalité ressemble énormément à celui de l'**étape 16** juste pour le nœud maître.

Étape [15/18]

Mise en place d'un script particulier, il fonctionne uniquement sur le **noeud maître** et permet à celui-ci de se connecter en **ssh** sur toute les machines du réseaux 10.0.0.x et exécute la commande qui permet de la lancé le **script clients**, permet en finalité de ce connecter en ssh sur les machines sur lequel elle as lancé le scripte client et de partir **récupérer le script** et d'en faire une **copie** sur lui.

Étape [16/18]

Script client qui lance une séquence durant laquelle elle envoie des r**equêtes ICMP (ping)** dans le **résaux 10.0.0.0** et récolte les rendus de chaque ping dans des tableaux **.json**, ceci et fait pendant **600s** à partir de départ de l'ordre donnée.

Étape [17/18]

Mise en place d'un dernier script qui **remplace toute les informations** d'ip mise sur **Wlan0** pour qu' il appartienne bien au **réseaux 10.0.0.x** .

Étape [18/18]

Dernière étape, mise en Log de fin de script et de redémarrage pour mettre en place tous les services.

C- Mise en place de topologie teste

Mise en place de topologie afin de tester dans la nature les cartes et la robustesse du script.

Topologie 1 : Bureau

Mise en place de cette topologie **(Voir annexe 1)** afin de prendre des valeurs de référence et vérifier que les puces soient bien logées dans les fichiers .json .

Topologie 2 : Sous sol

Mise en place de cette topologie **(Voir annexe 2)** afin de visualiser une formation en bus et la solidité des routes.

Topologie 3 : BATMAN-adv

Mise en place de cette topologie **(Voir annexe 3)** afin de visualiser une formation en ligne droite avec une cible mouvante (Entourer en noir) qui ce déplace dans le périmètre des autre puce, afin de tester la souplesse des routes.

D-Commande utile

Pour voir l'état des routes: *ip neigh show dev wlan0*
Pour voir l'état d'un service: *systemctl status olsrv2.service*
Pour lancer le service de journalisation pendant 10 min: *sudo mesh-run-icmp*
Pour voir les commandes utilisable dans cette version de Olsr: */usr/sbin/olsrd2_dynamic --schema*
Uniquement pour batman : *mesh-info*

E-Lien utile

Petit résumé des sources utilisés lors de mes recherches ou, utile pour la suite,
Site de doc de BATMAN : https://www.open-mesh.org/projects/batman-adv/wiki/Wiki
Site de doc OLSRv2 : https://openwrt.org/docs/guide-user/start
Site de téléchargement: https://www.raspberrypi.com/software/operating-systems/
Site de recherche le plus utilisé : https://chatgpt.com/
Site de recherche le plus pertinent : https://claude.ai
Site installation OONF 'stable' 0.15.1 :
https://github.com/OLSR/OONF/archive/refs/tags/v0.15.1.tar.gz

V- Expérience de teste

A-(OLSR)Bureau

Pour effectuer ce test, j’ai simplement installé le script gamma.sh sur toutes les puces que j’avais à ma disposition.
J’ai ensuite lancé le script sudo mesh-run-icmp
Je laisse le script tourné puis je regarde que tous les fichiers sont bien copiés sur mon nœud maître. Une fois cela effectué je jette un coup d'œil au résultat et dans les faits, tous les nœuds on répond et on tous pu communiquer les un avec les autres.

Interprétation :
Passage d’un cap dans l’évolution de mon stage, passage du développement au test.

B-(OLSR)Sous sol

Cette topologie a été pensée par mon génialissime superviseur de stage, Hugo, cette topologie permet de tester un bus à plusieurs sorties soit un topologie qui n’est pas anodine car non réalisable avec une couche physique différente. Pour tester cette configuration nous avons donc mis les Raspberry dans la configuration de l'annexe 2 puis avant essayer de lancer le script .json qui as pour but d’envoyer des requests ICMP ainsi que de créer et maintenir les routes entres les noeud, cependant, lors de cette expérience, nous avons pu voir que les pings n’arrivé pas à N+1 (noeud + 1 saut), il fallait une connexion direct pour faire transiter des information

Interprétation :
Problème de configuration OLSR. Cette topologie à permis de mettre en évidence que le script possède des failles de LVL 99.

C-BATMAN-adv (Sous sol)

Pour cette expérience nous avons repris la topologie sous sol afin de pouvoir tester la robustesse des routes, ceci fait nous avons pu voir que ce script était belle et bien fonctionné mais non adapté nécessairement pour faire transiter des information sur le réseau car une fois que la mise en place fut terminé, nous avons pu voir que la latence entre les noeud devient rapidement importante et ne supporter pas un trafic important.

Interprétation :
Réussite au niveau de la mise en place du système natif BATMAN-adv de bookworm. Cependant, problème non réglé au niveau des logs, car l'affichage du nombre de saut effectué par les packet n’est pas affiché.

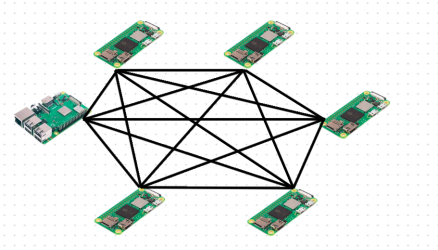
D- BATMAN-adv (Mix)

Pour cette topologie, c'est une mise en place primaire qui sert simplement à faire des tests rapide à l’abri de la pluie.

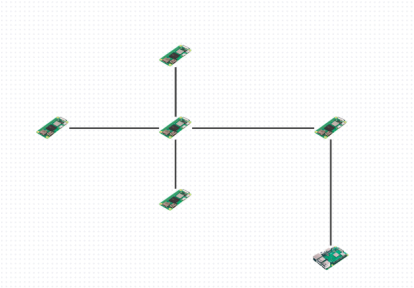
Interprétation :
Mise en place de cette topologie pour mettre en place des tests comme le correctif du .json.

VI-Annexe

Annexe 1 : Topologie Bureau



Annexe 2 : Topologie Sous sol



Annexe 3 : Topologie Mix