ÉCOLE POLYTECHNIQUE DE L'UNIVERSITÉ DE NANTES
DÉPARTEMENT D'INFORMATIQUE

RAPPORT DE RECHERCHE ET DÉVELOPPEMENT

Projet de Recherche et Développement

Expérimentation MP-OLSR sur des Raspberry Pi

Benjamin MOLLÉ & Denis SOURON

février 2015

encadré par Benoît PARREIN

— Équipe IVC —
INSTITUT DE RECHERCHE EN COMMUNICATIONS ET EN CYBERNÉTIQUE DE NANTES

Projet de Recherche et Développement

Expérimentation MP-OLSR sur des Raspberry Pi

Benjamin MOLLÉ & Denis SOURON

Remerciements

Pour la réalisation de ce projet, nous tenons particulièrement à remercier :

- Benoît Parrein, docteur en informatique et enseignant/chercheur à Polytech Nantes, laboratoire IRCCyN, France.
- Jiazi Yi, docteur en informatique et chercheur à l'École Polytechnique, laboratoire LiX, France.
- Ricardo T. Macedo, membre d'un groupe de recherche NR2 de l'université de Parana, Brésil.
- Michel Nogueira, professeur du département informatique de l'université de Parana, Curitiba, Brésil.
- Jean-Yves Leblin, responsable informatique au LINA, Université de Nantes, France.
- Dimitri Pertin, doctorant, chercheur à l'université de Nantes et à Fizians, France.

Table des matières

1

Introduction

Ce chapitre a pour objectif de présenter le sujet de manière ordonnée. Dans un premier temps nous détaillerons la problématique du sujet d'étude. Après avoir défini les différents aspects du problème nous expliquerons les objectifs poursuivis afin d'y remédier. Puis, nous détaillerons notre exercice au sein de ce sujet, notamment en précisant le travail réalisé d'une part, et notre contribution à la problématique d'autre part. Nous achèverons le premier chapitre en explicitant un plan d'étude pour le projet de recherche et développement.

1.1 Présentation de la problématique

Le protocole de routage MP-OLSR (Multi-Path Optimized Link State Routing Protocol) a été introduit par l'équipe IVC du laboratoire IRCCyN. Il se base sur une extension du protocole OLSR utilisé pour la communication dans les réseaux mobiles.

Ce protocole récemment développé est actuellement en cours de normalisation IETF que l'on peut appeler *Internet-Draft* [YP14]. L'un des principaux intérêt est d'appuyer la requête de normalisation en fournissant une expérimentation du protocole sur un réseau sans fil à grande échelle.

Les simulations déjà effectuées semblent prometteuses [YADP11, p. 10], c'est pourquoi il est nécessaire de les compléter avec des évaluations pratiques afin de d'évaluer l'efficacité du protocole. En effet la normalisation ne peut être définie qu'avec des analyses théoriques et des simulations. Après des tests sur des configurations possédant un nombre réduit de nœuds (moins de 10), il faut également appuyer le propos avec essais dans des réseaux denses (entre 50 et 200 nœuds)

1.2 Objectifs poursuivis

Face à cette problématique, nous devons définir les objectifs à suivre dans le cadre du projet. Le principal but est de proposer un réseau de capteurs dans un environnement régi par le protocole MP-OLSR. Puis, dans un second temps, nous devons effectuer des tests de performance sur cette infrastructure. L'analyse des résultats obtenus viendra compléter l'action de normalisation.

1.3 Travail à réaliser

Dans un premier temps, nous devons étudier le protocole MP-OLSR et plus précisément comprendre et analyser les spécifications de ce protocole. Ainsi nous pourrons l'utiliser et le mettre en œuvre au sein d'une expérimentation ultérieure.

L'étape suivante concerne la prise en charge de l'environnement de travail. Tout d'abord d'un point de vue "logistique", nous devrons interagir avec la plateforme IoT-Lab (réseaux de capteurs à grande échelle) qui sera l'infrastructure utilisée pour nos expérimentations. Puis, ensuite, dans un cadre technique, nous devrons appréhender les *daemons* OLSRd et MP-OLSRd correspondant respectivement aux protocoles OLSR et MP-OLSR. L'implémentation du *daemon* MP-OLSRd se base sur celle de OLSRd, c'est pourquoi ces études vont de pair.

Après avoir pris connaissance de l'environnement et l'organisation autour de la plateforme Senslab (nom générique des plateformes du même type que IoT-Lab), nous pourrons passer à des tests en pratique. D'abord sur des petits ensembles de capteurs (environ une dizaine), nous étendrons ensuite si possible les expérimentations sur des réseaux comprenant 200 à 300 noe uds.

Nous pouvons rencontrer des problèmes de compatibilité, d'interface et de performances lors de l'expérimentation à grande échelle. C'est pourquoi, en cas d'échec, nous nous concentrerons sur une expérimentation réduite, en spécifiant les problèmes rencontrés et détaillant un processus permettant de mener à bien des tests sur un grand réseau de capteurs.

1.4 Contribution

A remplir au fur et à mesure de la progression.

1.5 Plan de l'étude

Résumer le contenu de chaque chapitre. Se baser sur l'introduction de chaque chapitre

I

État de l'art

Dans cette partie nous nous efforcerons à faire le point sur l'état des connaissances sur le protocole de routage MP-OLSR. Nous nous baserons dans un premier temps sur l'article de Jiazi Yi relatant la thèse sur le protocole MP-OLSR [Yi10]. Nous verrons ensuite le fonctionnement d'IoT-lab, le but étant de savoir si la plateforme peut convenir à nos expérimentations ou de préparer les expérimentations voulu pour qu'elle fonctionne sur cette plateforme.

2

Le protocole MP-OLSR

2.1 Présentation

Nous basons principalement notre discours sur MP-OLSR à partir de la thèse de Jiazi Yi qui est primordiale car elle est constitue les fondements du protocole. Pour résumer le protocole dans les lignes suivantes nous nous basons sur un article rédigé avant la thèse.[YADP11]

Après avoir relaté les travaux approchant les idées du protocole MP-OLSR, l'article détaille les principaux travaux autour du protocole en deux parties distinctes.

Premièrement, un fort accent est mis sur l'aspect théorique du protocole de routage. Ce protocole peut être défini comme un protocole de routage multi-chemin hybride, car il combine des fonctions pro-actives et réactives. En effet, ce protocole envoie des requêtes fréquemment afin de connaître la topologie du réseau mais les routes possibles seront calculées au moment de l'envoi des paquets. Les nœuds ont ainsi connaissance de la topologie du réseau. Le calcul des routes lors de l'envoi d'un paquet sont basé sur l'algorithme de Dijkstra adapté au multi-chemin. L'algorithme original est légèrement modifié pour obtenir plusieurs chemins distincts mais possédant potentiellement des nœuds en commun. MP-OLSR intègre également un système de récupération de route pour palier aux problèmes éventuels dus au routage source (utilisé par MP-OLSR). Avant d'envoyer un paquet, un nœud intermédiaire vérifie que le prochain *saut* du point de vue du routage source est dans la table de voisinage. Si le nœud suivant n'est pas disponible alors la route, n'étant plus disponible, est recalculée à partir du nœud concerné. L'algorithme utilisé pour générer les chemins évite les boucles. Cependant, les technologies permettant de garantir la réception des paquets peuvent introduire des boucles dans le réseau, exemple : récupération de route. En comparant les routes initiales calculées par la source et la nouvelle route obtenue après récupération, il est possible de détecter les boucles. Dans le cas où il y a une boucle, une autre route sans boucle est générée à partir de l'algorithme muli-chemin. Et si aucune route n'est disponible, le paquet est détruit.

Dans un second temps, l'article se concentre sur l'application et la mise en pratique du protocole MP-OLSR. Une première expérience est menée sur Qualnet simulator 4.5.1. Son principal

but est de montrer les écarts entre OLSR et MP-OLSR et de caractériser de manière théorique les gains et les pertes apportés par MP-OLSR. Cette simulation est menée avec un grand nombre de nœuds (80) et un nombre de source variable (4 ou 10). Différents critères évaluent les performances de OLSR et MP-OLSR, comme le taux de réception, le délai moyen et la distribution du délai de réception. Ces deux protocoles sont également succinctement comparés au protocole DSR. Pour accompagner le propos, l'article détaille une expérience en situation réelle avec une source, un destinataire et 4 nœunds intermédiaires. Les outils de comparaisons utilisés sont identiques à la phase de simulation. Les différents tests semblent montrer que MP-OLSR offre des gains significatifs par rapport à OLSR.

A la fin de l'article, la conclusion s'attarde sur les performances du protocole MP-OLSR, en précisant des points supplémentaires. Des avantages sont donnés à MP-OLSR en terme de sécurité (Man in the middle difficile à mettre en place), mise à l'échelle et durée de vie du réseau.

2.1.1 Spécifications et fonctions intégrées au protocole

Détection de topologie

Nous avons besoin d'informations sur la topologie du réseau. Pour cela, nous allons passer par plusieurs étapes successives et complémentaires. Ces fonctions ne sont d'autres que la détection de liens, la détection de voisins et la découverte de topologie (procédés déjà présents dans le protocole OLSR). La construction de la topologie nécessite l'utilisation de messages Hello et TC (Définitions et fonctionnement au chapitre 3).

Pour la détection des liens et des voisins, des messages Hello sont échangés afin de définir les n ?uds qui répondent et les liaisons possibles pour la communication de données.

A la réception d'un paquet, le n ?ud le parse. En cas d'échec, le paquet est dropé. Le contenu du message est lu et/ou exécuté si besoin. Enfin, si le message est destiné à être transféré vers un autre n ?ud, le paquet est envoyé vers les autres destinataires.

Calcul de routes

La première route est obtenue avec l'algorithme de Dijkstra. Pour le calcul des autres routes, des fonctions incrémentales sont utilisées à chaque passage. Une première (*fp*) ajoute un poids aux liens utilisées par la route trouvée, la deuxième (*fe*) marque les liens connexes à la route trouvée. Les autres liens sont marqués à 1. Suivant les valeurs données par les deux fonctions les chemins vont avoir tendance à être plus ou moins disjoints. Les schémas 2.1 et 2.2 illustrent le calcul des routes avec des fonctions différentes. Dans la première figure, l'algorithme cherche à séparer les arcs. Dans le deuxième cas, on va essayer d'éloigner les nœuds déjà utilisés dans d'autres routes.

Récupération de route

MP-OLSR utilise le routage à partir de la source et l'un des principaux défauts de ce procédé est la coupure de la route entre deux nœuds. Cependant, le protocole de routage implémente

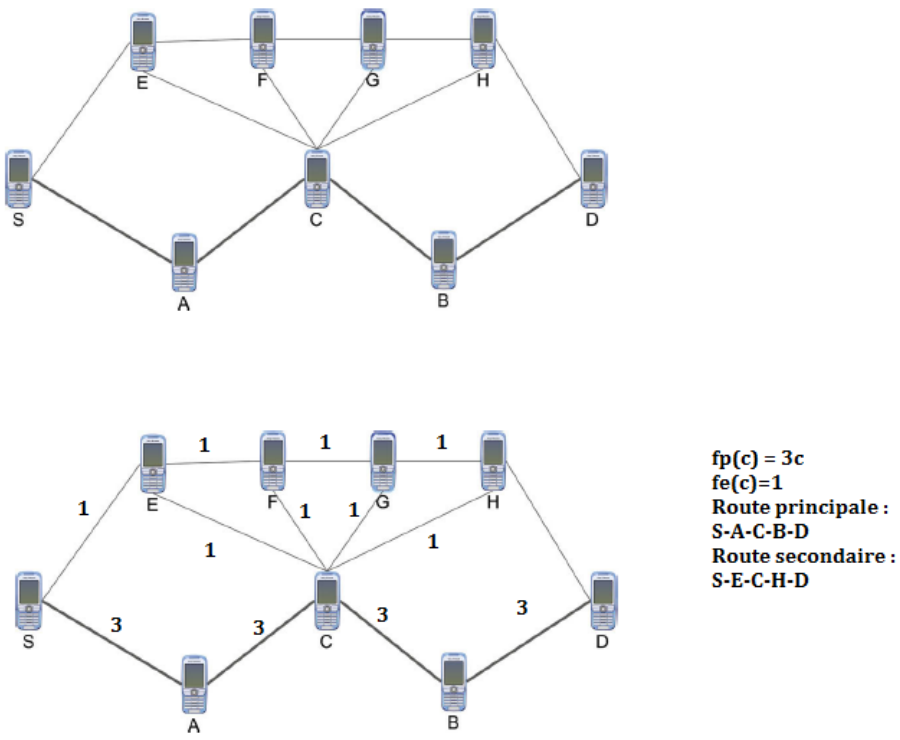


FIGURE 2.1 – Calcul de routes utilisant des liens différents

un système de récupération de chemin qui fonctionne de la manière suivante. Chaque nœud intermédiaire dans la transmission d'un paquet vérifie si le prochain saut est présent dans sa liste de voisins. Si oui, le paquet est naturellement transmis au prochain nœud. Sinon, le nœud concerné va recalculer une route avec l'algorithme de Dijsktra et le paquet sera transmis par cette route une fois définie. Un point important sur cette fonctionnalité est à souligner. Elle utilise les ressources locales sur les nœuds donc peu de délais sont introduits par ce biais et elle permet d'augmenter drastiquement le taux d'acheminement au sein d'un réseau.

Détection de boucle

Comme OLSR, MP-OSLR intègre la fonction LLN (Link Layer Notifications). Ce procédé peut être ajouté aux messages HELLO et TC utilisés habituellement. Le protocole envoie des notifications aux nœuds voisins lors d'évènements spéciaux (comme la perte de lien). L'utilisation de LLN dans un réseau améliore le taux de livraison des paquets pour OLSR et MP-OLSR (voir OLSRv2 testbed et simulation de MP-OLSR sur NS2).

Lors du calcul de routes, on utilise l'algorithme de Dijkstra qui nous garanti d'avoir une route ne comportant pas de boucles. Cependant, en pratique, MP-OLSR intègre des fonctionnalités qui peuvent amener la route d'origine à être modifiée. Avec ces modifications des boucles peuvent être introduites au sein du routage source.

Les quatre schémas 2.3, 2.4, 2.5 et 2.6 décrivent comment des boucles peuvent rapidement

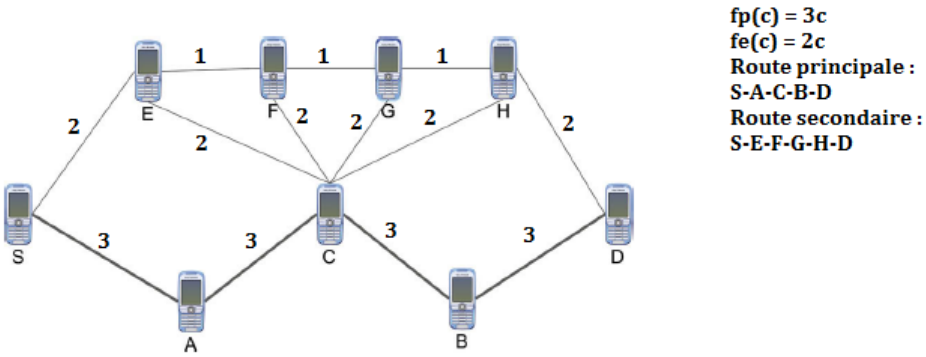


FIGURE 2.2 – Calcul de routes avec peu de nœuds en communs

apparaitre dans une transmission. Le premier schéma indique le contexte d'un envoi de paquet avec A, un nœud intermédiaire, B, un voisin de A et C, la destination des paquets. Le lien entre les nœuds A et C est désactivé car C a été placé hors de portée. La fonction de récupération de route se met alors en place et calcule une nouvelle route comme le montre la deuxième illustration sur la figure 2.4. Cependant, A ne sait pas que la liaison entre B et C est également désactivée. Quand un paquet arrive en B, ce dernier sait que le liaison directe vers B est indisponible. La fonction de récupération de route se met en place sur le nœud B afin de trouver une route vers C, schéma 2.5. Le paquet reçu par B est alors renvoyé vers A créant ainsi une boucle.

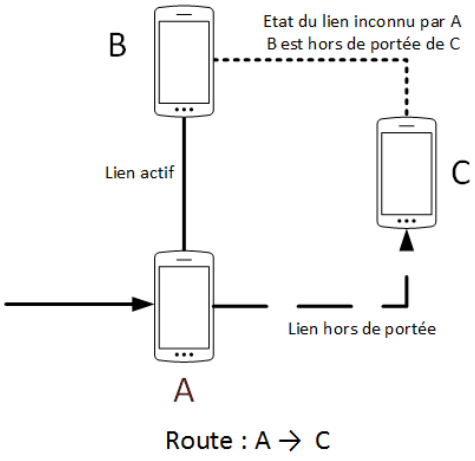


FIGURE 2.3 – Contexte

Ce phénomène est présent car les liens A vers C et B vers C ne sont supprimés que après quelques secondes (6 secondes par défaut dans OLSR). Par conséquent, la boucle n'est pas infinie mais engendre une congestion locale.

Pour palier à ce problème, le protocole MP-OLSR propose une méthode basée sur le routage source. On remarque que les boucles sont créées lors de l'utilisation de la fonction de récupération de route. Le principe de détection de boucle opère juste après avoir obtenu la nouvelle route. Il va comparer les routes, celle d'origine et celle qui a été recalculée car un lien a été perdu, et

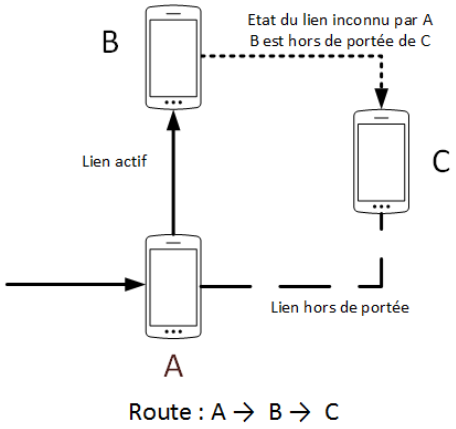


FIGURE 2.4 – Récupération de route sur le nœud A

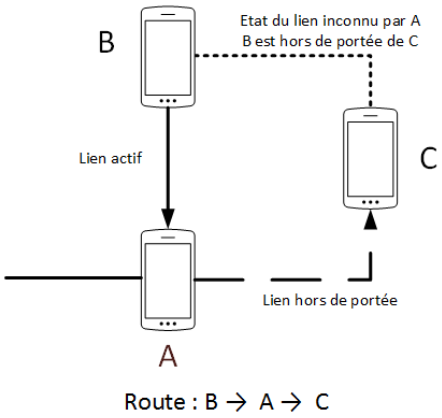


FIGURE 2.5 – Récupération de route sur le nœud B

regarder si des nœuds présents dans la route originelle le sont aussi dans la nouvelle route. Si un noeud a déjà utilisé précédemment, alors il y a des chances qu'une boucle soit introduite. Le protocole va donc chercher d'autres chemins, si aucuns ne sont valides le paquet est effacé.

2.2 Analyse

Lors de la lecture du document, les résultats de la simulations semblent vraiment impressionnants. En effet, on observe des délais de livraison 4 fois moins longs, des temps d'attente de paquets dans les files d'attente 5 fois plus rapides ou encore une amélioration du taux de succès de transmission jusqu'à 20% supérieur (dans le cas de 10 sources communiquant en simultané).

Mais d'un point de vue de la simulation réelle, celle-ci reste très limitée. Avec seulement 9 nœuds non mobiles, on ne rencontre que peu de cas. Il sera donc très intéressant de refaire une expérimentation avec 2 problématiques dominantes :

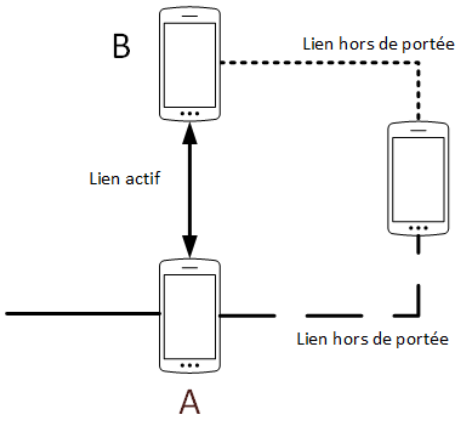


FIGURE 2.6 – Apparition d'une boucle

- la gestion d'un grand nombre de nœuds (ce nombre reste a définir),
- la gestion des routes dans le cas où les nœuds sont très mobiles avec beaucoup d'interférences possibles ce qui impliqueraient de recalculer de nouvelles routes régulièrement.

2.2.1 Intérêts de la proposition

Cette proposition est la base de notre projet. En effet elle fait partie des derniers travaux sur MP-OLSR, qui à de plus été réalisée à Polytech Nantes par Jiazi Yi avec Benoît Parrein. Notre objectif est donc de partir de celle-ci pour élaborer une expérimentation complémentaire et peut-être les améliorations nécessaires. Hormis cela, la proposition reste complète d'un point de vue théorique. Il ne nous reste plus qu'à nous pencher sur la réalisation d'une ou plusieurs expérimentations pour assoir les travaux de Jiazi et les compléter.

3

Modèle de domaine

Dans cette section, nous allons présenter tout les concepts clé autour de notre projet. En effet MP-OLSR est l'amélioration de d'un protocole existant déjà OSLR (qui lui même a plusieurs versions). MP-OLSR vient aussi par l'optimisation de l'algorithme de Dijkstra comme nous avons pu le voir dans la section suivante.

3.1 Définitions

OLSR (Optimized Link State Routing Protocol) OLSR est un protocole de routage pour les réseaux ad hoc et mobile. Il fonctionne comme un protocole proactif, des informations de topologie avec d'autres nœuds du réseau sont régulièrement échangées. Pour cela on utilise 2 types de messages :

- "HELLO", permet à un nœud de diffuser des informations sur son voisinage et de l'état de ses liens avec eux.

- "TC"(Topology control), permet la construction des tables de routage car ce message permet l'envoi de la liste de ses voisins ayant choisi le nœud comme MPR(nœuds choisis qui expédient des messages de diffusion pendant le processus d'inondation).

OLSR est une variante de LSR (Link State Routing), son optimisation vient de l'utilisation de relai multipoints (MPR). Ces derniers sont les seuls à déclarer leurs liens et sont sélectionnés par les autres nœuds de manière à ce que ceux-ci puissent atteindre n'importe qui en deux sauts. Les principales fonctionnalités d'OLSR sont donc la découverte des voisins et la diffusion de la topologie. Ceci permet de réduire la taille du paquet de contrôle. : au lieu de tous les liens, OLSR déclare qu'une partie des liens avec ses voisins qui sont ses sélecteurs de relais multipoints. De plus, on minimise l'inondation de la circulation en utilisant uniquement les nœuds sélectionnés. Seuls les MPR d'un nœuds peuvent retransmettre ses messages diffusés.

Le protocole n'a pas besoin d'entité centrale grâce à son fonctionnement distribué. De plus, il ne nécessite pas de transmission fiable pour les messages de contrôles qui sont envoyé periodiquement.

Quelques précision sur la détection des voisins. Deux nœuds sont considérer comme voisin quand il y a une connexion bidirectionnel et direct entre les deux. Pour cela, chaque nœuds envoie périodiquement un message "HELLO" à tous les voisins à 1 saut.

Un message "HELLO" contient le type de lien(symétrique, asymétrique ou perdu), la volonté du nœud de devenir MPR, des informations sur les voisins :

- les voisins entendus mais pour lesquels une communication bi-directionnelle n'a pu être établie
- les voisins avec qui le nœud a pu établir une liaison bi-directionnelle
- Les nœuds désignés comme MPR par le nœud originaire du message HELLO

Voici le datagramme d'un paquet "HELLO" (extrait de Wikipedia) :

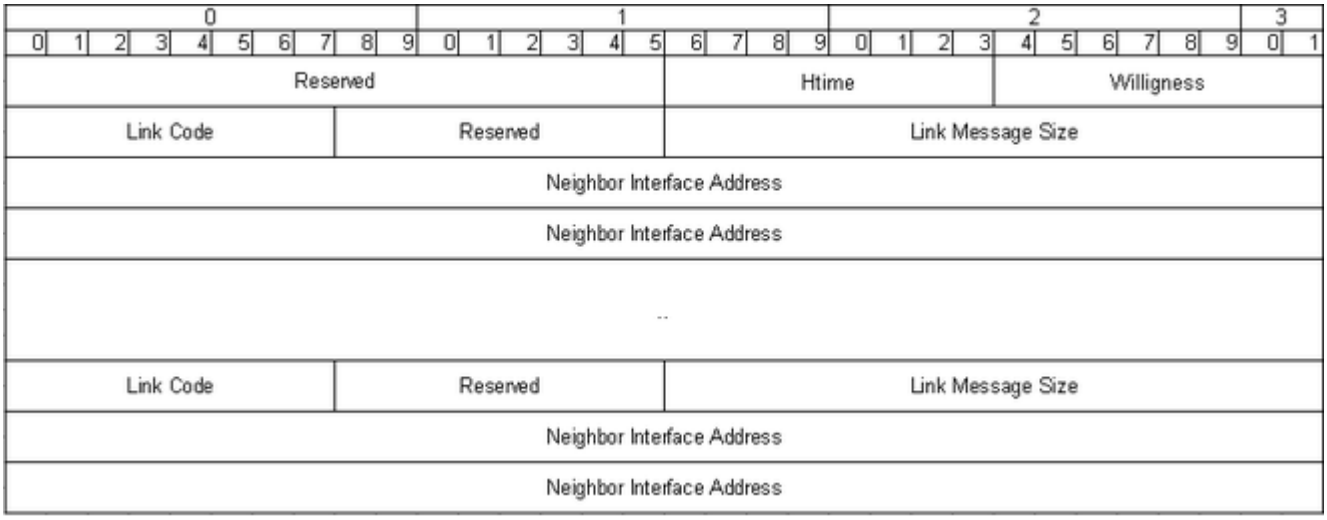


FIGURE 3.1 – Datagramme HELLO

- "Reserved" : doit contenir "0000000000000000"
- "Htime" : Intervalle d'émission des messages HELLO
- "Willigness" : permet de forcer le passage d'un nœud en MPR
- "Link Code" : Code identifiant le type de lien (pas d'information, symétrique, asymétrique, etc.) entre l'expéditeur et les interfaces listées

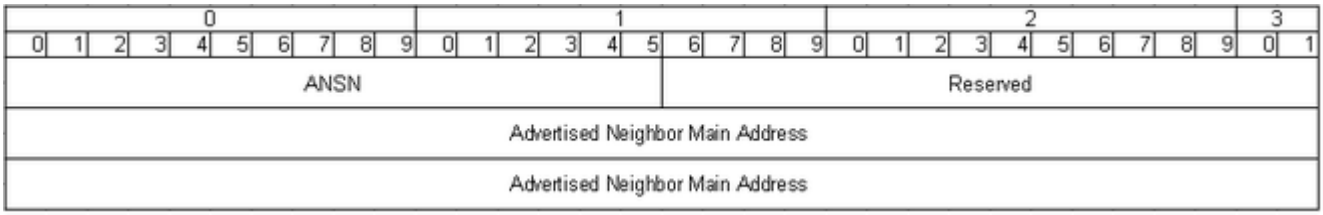


FIGURE 3.2 – Datagramme HELLO

Les nœuds utilisent aussi un second type de message : le message “TC” (Topology Control). Ils sont utilisés pour construire la table de routage. Seuls les MPR envoient ces messages en broadcast ce qui leur permet de périodiquement transmettre la liste de leurs voisins qui les ont choisi comme MPR. Voici le datagramme d’un paquet “TC” (extrait Wikipedia) :

Enfin, regardons un peu le fonctionnement des nœuds MPR. Le but des relais multipoints est de minimiser l’inondation de paquets de diffusion dans le réseau en réduisant les retransmissions en double vers un même nœuds. OLSR repose sur la sélection des relais multipoints, et calcule ses routes vers toutes les destinations connues à travers les nœuds. Les MPR sont choisis comme des nœuds intermédiaires dans le chemin. Le calcul d’itinéraire ce fait en utilisant un algorithme de plus court chemin. [Wik14]

4

Environnement d'expérimentation

Dans ce chapitre, nous détaillerons d'abord les différentes solutions possibles pouvant constituer l'environnement de travail pour nos expériences. Nous nous attacherons particulièrement à préciser les points forts et les faiblesses de chaque possibilité. Dans un second temps, nous décrirons plus en détails le fonctionnement et l'utilisation de la solution choisie.

4.1 Analyse des choix disponibles

4.1.1 Simulation via Qualnet ou Network Simulator 2 / 3

Avantages :

L'utilisation d'un outil de simulation permet de s'abstraire du matériel. Ainsi, on évite les problèmes de compatibilité entre les nœuds.

De plus, la configuration d'une simulation est plus simple et plus rapide. En effet, celle-ci est applicable en parallèle sur tous les nœuds. La topologie est également facile à mettre en place car les nœuds peuvent être déplacés immédiatement et à volonté.

Inconvénients :

Comme énoncé juste précédemment, la simulation est indépendante du matériel. On ne peut donc pas s'assurer du fonctionnement sur une ou plusieurs machines réelles.

La simulation restant assez proche de la théorie, les résultats ne correspondent pas avec ceux obtenus lors d'un test en situation réelle.

4.1.2 Réseaux de Raspberry Pi

Avantages :

Les Raspberry Pi utilisent un système d'exploitation Debian optimisé pour le matériel intégré. Ce système d'exploitation, basé sur un noyau Linux standard, est largement compatible avec les programmes développés sur Linux.

Nous avons également accès à plus de 200 équipements de ce type. Ce qui, pour notre expérimentation, est suffisant dans le cadre de test à grande échelle (plus de 100 nœuds).

Enfin, les machines sont aussi présentes dans nos locaux. Nous pouvons directement agir sur elles : placements, paramétrages.

Inconvénients :

Les Raspberry Pi sont à disposition des étudiants pour des expériences. Dès lors nous ne pouvons garantir la disponibilité d'un nombre suffisant de machines pour nos tests.

Les capteurs utilisés dans les infrastructures de réseaux sans-fil sont généralement dotés d'une puissance de calcul nettement plus faible (par rapport aux Raspberry Pi). Les résultats obtenus sur Raspberry Pi ne permettent pas d'affirmer le fonction sur un réseau de capteurs. En effet, ces derniers peuvent voir leurs performances réduites de part les temps de calcul (détermination des routes ou détection de boucle par exemple).

4.1.3 Plateforme de Testbed réseaux sans-fil

Avantages :

De nombreux capteurs et périphériques sont disponibles et ceux-ci permettent de savoir si le protocole est applicable sur un réseau de capteurs.

De plus, certaines plateformes (comme IoT-Lab) proposent de louer gratuitement des capteurs pour une durée variable (définie par l'utilisateur).

Inconvénients :

N'ayant pas d'accès direct aux capteurs, la topologie est plus difficile à mettre en place.

4.2 Étude de la solution retenue : la plateforme FIT IoT-LAB (Testbed)

4.2.1 Présentation

IoT-LAB est une partie de la plateforme d'expérimentation de FIT (Future Internet of Things). FIT est composé de 4 grandes parties :

- **NOC** (Network Operations Center) : a pour but d'unifier les bancs d'essai comme une resources partagée unique
- **CorteXLab** est un environnements qui permet de développer et designer plus efficacement des nœuds radios.
- **IoT-LAB** permet de modéliser physiquement un monde en mouvement de nœuds radio.
- **OneLab** permet la mise en oeuvre d'une combinaison d'environnements filaires et sans-fils.

IoT-LAB est donc une banc d'essai scientifique nous permettant un contrôle total sur le réseau des nœuds et un accès direct à ceux-ci.

4.2.2 Accès

IoT-Lab est organisé autour de 6 grands sites répartis en France. Au total, ce sont plus de 2700 nœuds qui sont accessibles à Grenoble, Lille, Paris, Rennes, Rocquencourt et Strasbourg.

La plateforme permet de nombreux accès vers les différents intervenants. Tout d'abord, l'interface web offre des outils puissants de monitoring. L'utilisateur peut, à tout moment, consulter les états des nœuds et interrompre / reprendre l'exécution des systèmes d'exploitation des capteurs.

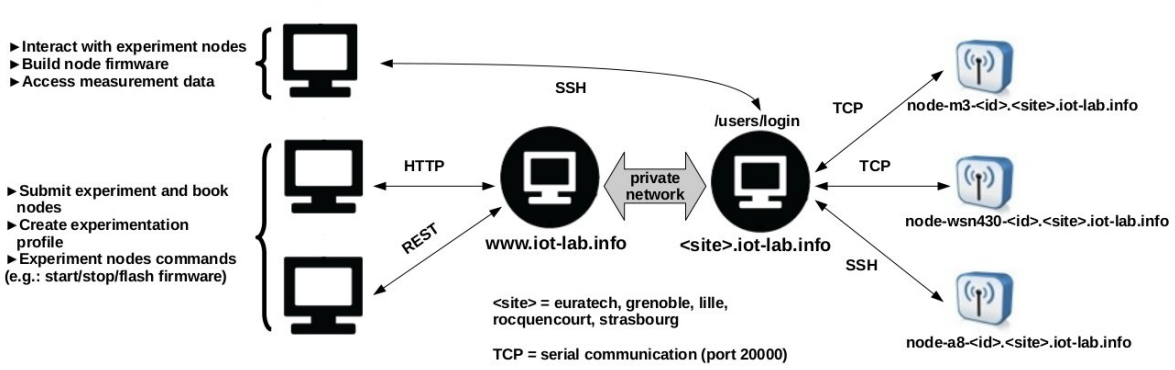


FIGURE 4.1 – Connexions à la plateforme IoT-Lab

4.2.3 Capteurs disponibles

Dans la plateforme IoT-Lab sont proposés 2 types de capteurs. Ces nœuds peuvent dialoguer à l'aide d'antenne radio dont tous sont pourvus.

Une première partie des capteurs est appelée « Open Node ». De part sa nomination, on comprend que cela concerne des nœuds sur lesquels l'utilisateur à un accès complet et direct. En

effet, la personne chargée de mener des expérimentations (par exemple) peut agir directement sur les nœuds, en ajoutant un autre système d'exploitation ou en modifiant les logiciels utilisés par ces nœuds.

Les autres capteurs sont un peu plus complexes car ils regroupent deux fonctions majeures supplémentaires, qui viennent compléter les « Open Node ». Ils sont alors appelés « Host Node » s'ils intègrent ces éléments complémentaires. Le premier composant sert de passerelle vers le capteur « Open Node » en offrant une interface qui permet de diriger et surveiller les nœuds. Le second élément est dénommé « Control Node », et comme son nom l'indique, il a pour objectif d'interagir avec le capteur. En utilisant la passerelle, il peut ainsi mesurer certaines informations et changer l'alimentation (batterie ou prise secteur) pendant les phases de test.

Dans la plateforme IoT-Lab nous retrouvons des WSN430 « Open Node » basés sur des microprocesseurs intégrés MSP430 fabriqués par Texas Instruments. Les autres nœuds, « Host Node », sont construits autour de « M3 Open Node » et « A8 Open Node ». Ces deux capteurs sont plus puissants que les WSN430 car ils sont équipés de microprocesseurs ARM, respectivement M3 et A8, bénéficiant d'instructions 32-bit et de fréquences accrues par rapport aux capteurs précédemment décrits (9 fois supérieures pour les puces M3, 75 fois pour les modèles dotés de microprocesseurs A8)

Pour ce projet nous aurons besoin de modifier les nœuds afin d'y implémenter des protocoles de communications par exemple. Il nous est nécessaire de savoir si les changements que nous sommes amener à faire sont possibles. C'est pourquoi nous allons détailler ci-dessous les capacités des capteurs et leurs fonctionnalités.

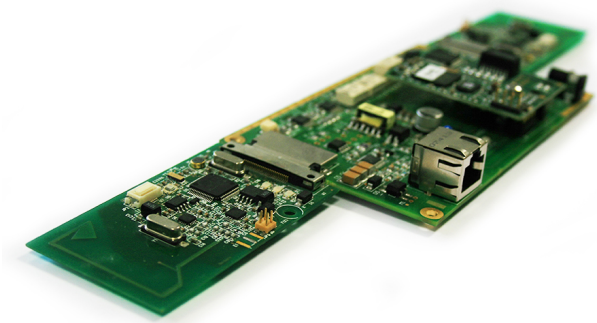


FIGURE 4.2 – Photo d'un capteur WSN430

WSN430

- **Micro-contrôleur :** Texas Instruments MSP430 8 MHz / 16-bit / 10kB RAM
- **Antenne Radio :** 868 MHz 6LoWPAN ou 2.4 GHz ZigBee (compatible avec la norme IEEE 802.15.4 sur les LR WPAN, Low Rate Wireless Personnal Area Network)

- **Système d'exploitation :** FreeRTOS, Contiki, Riot, TinyOS, OpenWSN
- **Stockage :** 1 MB (Stockage externe ROM)

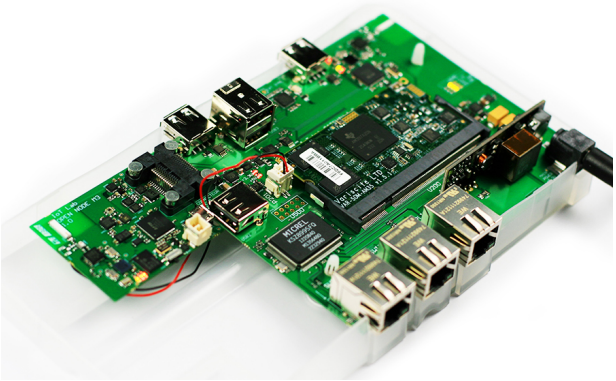


FIGURE 4.3 – Photo d'un capteur M3

M3

- **Micro-contrôleur :** ARM Cortex M3 72 MHz / 32-bit / 64kB RAM
- **Antenne Radio :** ATMEL AT68RF231 2.4 GHz (compatible IEEE 802.15.4, ZigBee, RF4CE et autres)
- **Système d'exploitation :** FreeRTOS, Contiki, Riot, OpenWSN
- **Stockage :** 16 MB (Stockage externe ROM)

FIGURE 4.4 – Photo d'un capteur A8

A8

- **Micro-contrôleur :** ARM Cortex A8 600 MHz / 32-bit / 256MB RAM
- **Antenne Radio :** ATMEL AT68RF231 2.4 GHz (compatible IEEE 802.15.4, ZigBee, RF4CE et autres) sur le nœud M3 couplé
- **Système d'exploitation :** Linux (Noyau 3.9.6)
- **Compléments :**M3 Open Node en relation directe, prises Ethernet et USB

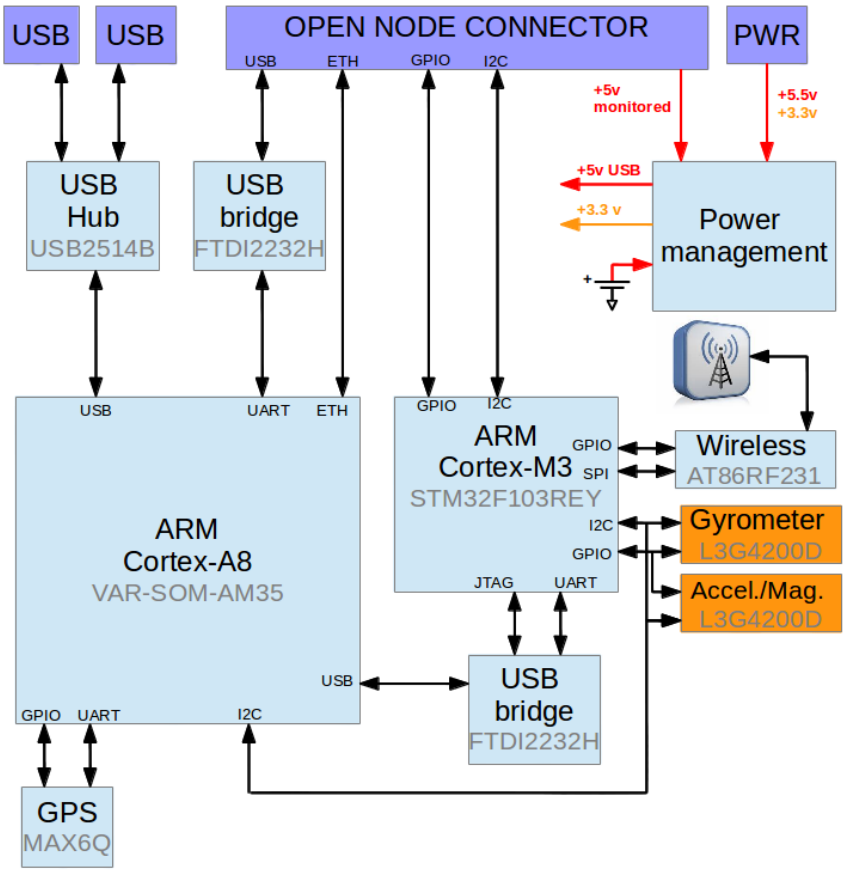


FIGURE 4.5 – Interaction A8 - M3

4.2.4 Intervenants

Le matériel proposé par la plateforme IoT-Lab permet la recherche dans plusieurs domaines de pointes. Et c'est pourquoi la plateforme a beaucoup d'interactions avec l'Inria, les Instituts Mines-Télécom et le CNRS. L'infrastructure est financée par le Ministère de l'Éducation nationale, de l'Enseignement supérieur et de la Recherche.

4.2.5 Premier test : Dialogue entre 2 capteurs

Début de l'expérience :

Avant la première expérience sur la plateforme IoT-Lab, il est conseillé de configurer un accès `ssh` vers les serveurs IoT-Lab. Pour cela, l'utilisateur génère une clé RSA, dont il conserve la clé privée pour se connecter. Puis il est possible de renseigner une clé publique sur les profils IoT-Lab. Ainsi, l'utilisateur peut, en configurant l'accès ssh avec la clé privée obtenue, accéder directement aux serveurs IoT-Lab en ne renseignant que le login.

Etape 1 :

L'interface Web de la plateforme permet de louer, pendant une durée définie, un ou plusieurs nœuds sur les différents sites localisés en France. Pour notre exemple, nous réservons 2 nœuds de type A8 sur le site de Grenoble pendant 30 minutes (cf Figure 4.6). Par la suite, il est également possible de définir des associations entre les nœuds. Nous n'en aurons pas la nécessité pour cette expérience.

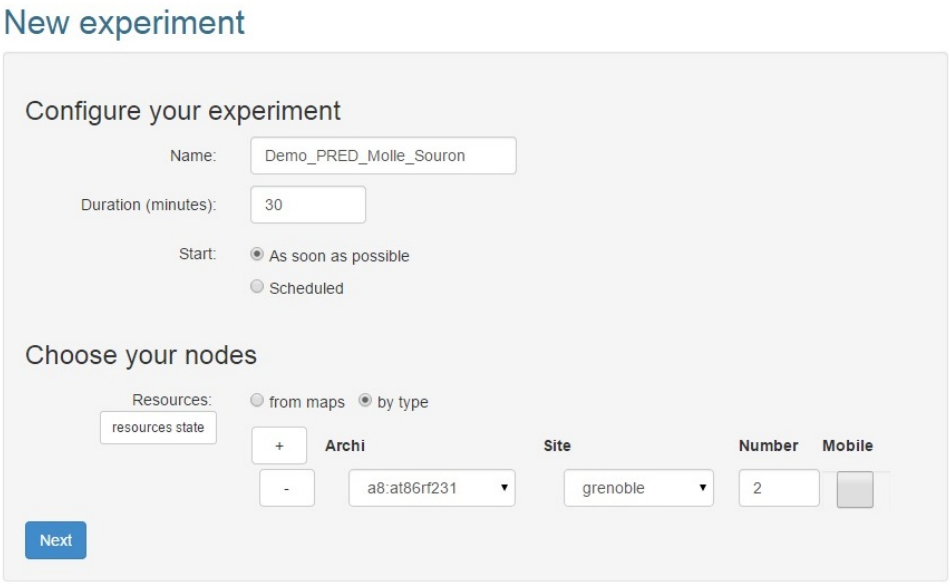


FIGURE 4.6 – Réservation des nœuds

Etape 2 :

Après avoir réserver les nœuds pour une expérience, nous devons nous connecter sur le site où nous avons loué les capteurs, Grenoble dans notre cas. Puis nous récupérons un programme minimaliste fourni par IoT-Lab pour les tests

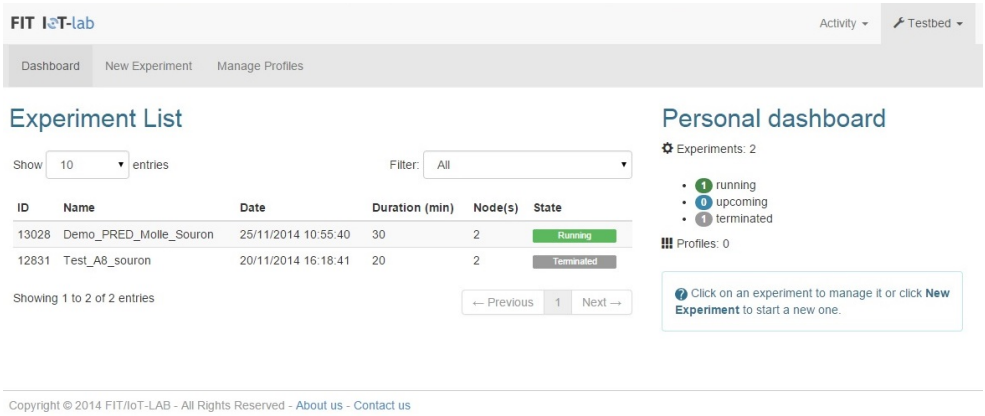


FIGURE 4.7 – Tableau de bord IoT-Lab

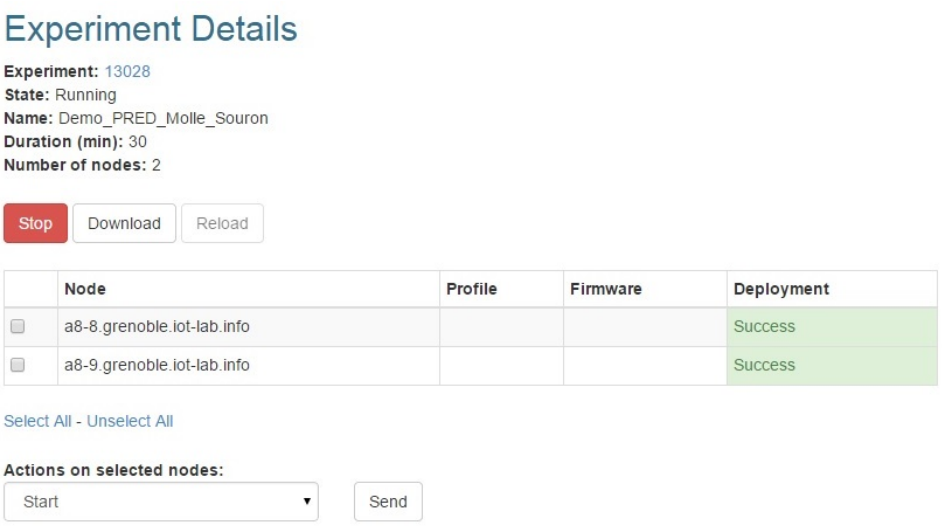


FIGURE 4.8 – Gestion des nœuds par interface Web

Etape 3 :

Pour la suite, nous nous connectons en super-utilisateur sur un nœud A8 (nœud n°8). Nous pouvons également tester la connexion entre les 2 nœuds réservés pour notre expérience. Par exemple, nous pouvons faire des requêtes PING vers l'autre nœud (cf Figure 4.9)

Etape 4 :

Les nœuds semblent être en état de fonctionnement, nous pouvons alors remplacer le firmware actuel par celui que nous avons récupéré précédemment.

Etape 5 :

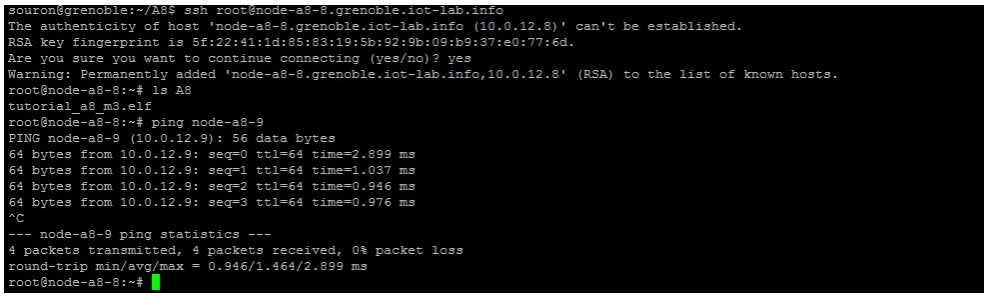


FIGURE 4.9 – Connexion aux nœuds A8

Lancement d'un script en python permettant d'envoyer 2 types de paquets de longueurs différentes

- "Hello World ! : 0" (Taille 16 octets)
- "Big Hello World ! : 0 0123456789012345678901234567890123456789012345678" (Taille 60 octets)

Nous répétons les étapes 3 à 5 pour l'autre nœud.

Etape 6 :

Nous pouvons observer sur la figure 4.10 les échanges de paquets commandés par l'utilisateur à travers le script python.

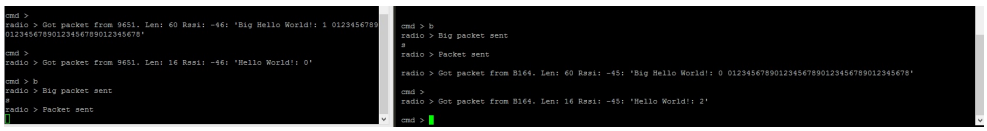


FIGURE 4.10 – Test de communication entre les 2 nœuds réservés et paramétrés

Fin de l'expérience :

A la fin de l'expérimentation, le tableau de bord est mis à jour et le test est archivé (cf Figure 4.11).

4.2.6 Première compilation

Pour l'expérience précédente, nous avons utilisé un programme (au format Executable and Linkable Format) qui nous était fourni par le tutoriel. Hors nous allons surement devoir faire notre propre programme pour notre projet. C'est pourquoi nous nous sommes renseignés sur la manière dont IoT-Lab fonctionne.

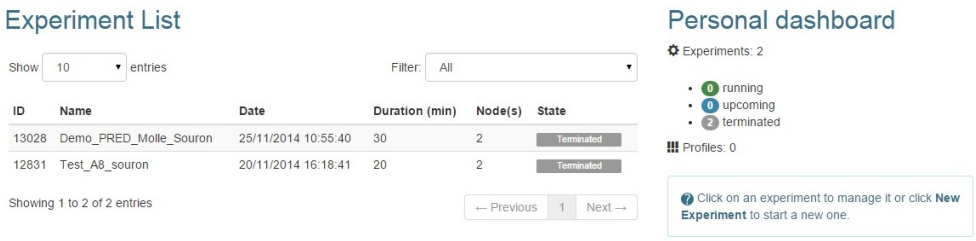


FIGURE 4.11 – Etat du tableau de bord à la fin des tests

Grâce à un tutoriel du site, nous avons pu faire ceci.

Voici la procédure de compilation que nous avons exécuté sur notre compte du site de Grenoble. Pour des raisons de développement, il est a priori possible de dupliquer cette environnement sur une machine Linux. Tout d'abord nous devons mettre en place l'environnement de travail :

```
ssh <login>@grenoble.iot-lab.info (site = grenoble, euratech,
   rocquencourt, strasbourg)
<login>@grenoble~$ git clone https://github.com/iot-lab/iot-lab.git
Cloning into iot-lab...
...
<login>@grenoble~$ cd iot-lab
<login>@grenoble:~/iot-lab$ make

  Welcome to the IoT-LAB development environment setup.

  targets:
        setup-wsn430
        setup-openlab
        setup-contiki

        pull
```

Ensuite on prépare la cible et on compile, ici pour les nœuds A8 :

```
<login>@grenoble:~/iot-lab$ make setup-openlab
git clone https://github.com/iot-lab/openlab.git parts/openlab
Cloning into parts/openlab...
...
less parts/openlab/README-IoT-LAB.md
<login>@grenoble:~/iot-lab$ cd parts/openlab
<login>@grenoble:~/iot-lab/parts/openlab$ mkdir build.a8-m3 ; cd build
   .a8-m3/
<login>@grenoble:~/iot-lab/parts/openlab/build.a8-m3$ cmake .. -
```

```
   DPLATFORM=iotlab-a8-m3
<login>@grenoble:~/iot-lab/parts/openlab/build.a8-m3$ cd ..
```

Comme on peut le voir dans les lignes de commandes précédentes, on utilise cmake comme moteur de production. Et enfin pour compiler et générer le fichier binaire .elf on exécute :

```
<login>@grenoble:~/iot-lab/parts/openlab$ cd build.m3
<login>@grenoble:~/iot-lab/parts/openlab/build.m3$ make tutorial_m3
<login>@grenoble:~/iot-lab/parts/openlab/build.m3$ ls bin/tutorial_m3.
   elf
bin/tutorial_m3.elf
```

Nous avons donc recompilé le fichier que nous avions utilisé et exécuté sur les nœuds A8 réservés lors de l'expérimentation précédente.

Mais ce qui nous intéresse le plus ici ce sont les code sources de cette application que l'on peut trouver dans le dossier : `iot-labpartsopenlabappliiotlab_exampletutorial`

On y retrouve 3 fichiers :

- `README.md`
- `main.c` le code source du programme
- `CMakeLists.txt` utilisé par cmake

Vous pouvez retrouver le code de main.c en annexe A page 50.

On va donc pouvoir utiliser les informations sur l'architecture des fichiers et ce code pour pouvoir développer notre propre firmware afin de faire les tests sur MP-OLSR. [IL14]

4.2.7 Topologie

Un autre point important d'IoT-LAB pour notre projet. La topologie des nœuds. Pour cela nous avons utilisé un programme fourni par IoT-LAB pour connaître à un instant donné la position de tous les nœuds d'un site.

Sur la figure 4.12 on peut voir la disposition des nœuds au Mardi 25 Novembre 2014 à Grenoble (1 point = 1 nœud). Ils sont dans l'ensemble répartis sur 3 étages et sur 2 couloirs parallèles reliés par un petit couloir. Ensuite, la figure 4.13 fait référence à un plan en vu de dessus.

A la vue de la disposition des nœuds, nous pensons que nous allons pouvoir faire des tests intéressants sur MP-OLSR d'un point de vue de la quantité de nœuds. Le problème que nous évoquions au sujet de la gestion de la perte de route et/ou de nœuds quand à elle va être plus compliquée. Il faudrait voir si avec le changement d'étage, ou déployer des nœuds conçus pour interférer lors d'une expérimentation pourra nous apporter ce que nous voulons.

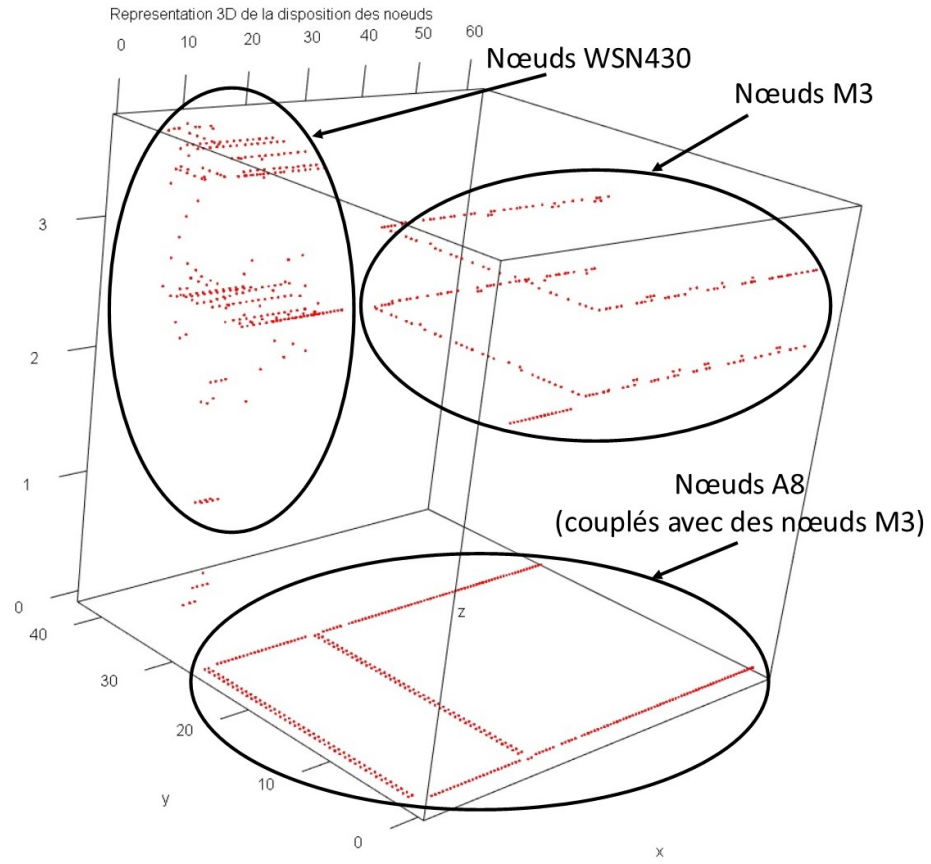


FIGURE 4.12 – Répartition spatiale des nœuds de Grenoble (1 point = 1 nœud)

4.2.8 Compatibilité MP-OLSR et IoT-LAB

Sur la plateforme IoT-LAB, les nœuds A8 et M3 ont un fonctionnement très lié. En effet, les deux types de nœuds ont un fonctionnement couplé, chaque nœud A8 est lié (par un lien USB à 500Kpbs) à un nœud M3. Le nœud A8 à pour système d'exploitation un Linux (noyau 3.9.6) et le nœud M3 utilise FreeRTOS. Ainsi, lorsque l'on veut développer un programme, la méthode de compilation que nous avons évoqué en partie 4.2.6 permet de mettre a jour le programme sur le nœud M3 qui communique en UDP ou TCP avec le nœud A8.

On peut donc supposer que pour le déploiement de MP-OLSR sur ce genre de nœud hybride il faut développer sur le nœud M3 le système d'émission de paquet (les paquets HELLO, TC . . .) et sur le nœud A8 on déploie MP-OLSR (probablement la même version que celle qu'a utilisé Jiazi YI pour sa thèse). Il faudra surement modifier le fonctionnement de MP-OLSR de manière à ce qu'au lieu d'envoyer lui même le paquet, le nœud A8 doit envoyer un message au nœud M3 pour qu'il envoie le message voulu par son antenne radio.

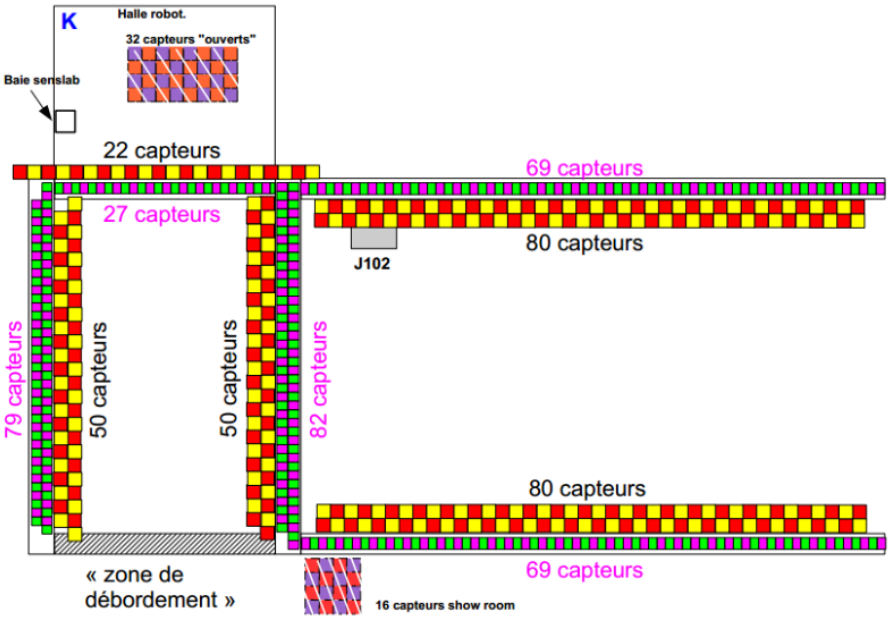


FIGURE 4.13 – Plan vu du dessus des nœuds de Grenoble

5

Propositions

Dans ce court chapitre, nous expliquons nos choix pour la suite du projet à l'aide des éléments dont nous disposons. Nous avons pu, grâce un état de l'art, déterminer les solutions pour répondre à la problématique. Nous revenons tout d'abord sur la solution envisagée à l'origine. Puis nous détaillons la solution que nous avons retenue ainsi que les raisons confortant notre choix.

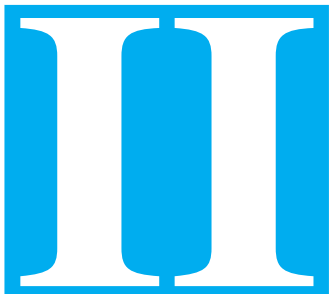
5.1 Plateforme Iot-LAB

Comme nous l'avons expliqué dans la partie précédente, les capteurs disponibles sur la plateforme IoT-Lab utilise des systèmes d'exploitations adaptés à leur fonctionnement. Le comportement de ces capteurs est déterminé par un firmware présent dans les espaces mémoires. Ces firmwares ne peuvent pas être installés, ils doivent être copiés directement vers la mémoire. De ce fait, les nouveaux programmes doivent compilés sur une autre machine avant de transférer le code compilé sur les capteurs. L'ajout de l'implémentation de MP-OLSR semble difficile par cette méthode. En effet, dans le même programme, celui qui constitue le firmware, nous devons intégrer l'implémentation de MP-OLSR en plus des éléments principaux nécessaires au fonctionnement des capteurs (envoyer des paquets HELLO et TC, recevoir et lire ces paquets, etc...). La solution n'est pas envisageable dans l'immédiat car elle nécessite une adaptation complète de l'implémentation de MP-OLSR, ce qui s'éloigne de l'objectif du projet et peut s'avérer très long. Nous devons donc nous tourner vers une autre solution que nous avons abordé précédemment, un réseau de Raspberry Pi.

5.2 Réseau de Raspberry Pi

Dans l'état de l'art, nous nous sommes rapidement passé sur les intérêts à utiliser un réseau de Raspberry Pi pour notre projet. Pourtant, c'est la solution que nous retenons pour mener des expérimentations. Ces équipements peuvent être doté de Raspbian, une distribution Linux très

proche de Debian optimisée pour les Raspberry Pi. L'implémentation de MP-OLSR utilisée pour la thèse de Jiazi Yi est donc en théorie compatible avec ce type de matériel, sachant que les précédentes expérimentations ont été effectuées sur des Netbooks. Cependant, ces équipements ne sont pas parfaitement adaptés à l'utilisation de MP-OLSR. Les Raspberry Pi ne sont pas munis d'antenne radio (Wifi, Bluetooth, ...), ce qui contraint les expérimentations à l'utilisation de l'Ethernet. Par conséquent, les résultats obtenus ne reflètent pas intégralement les apports possibles via l'utilisation de MP-OLSR.



Réalisations

6

Portage

Pour l'exécution du projet, nous avons fait une phase importante de portage du programme MP-OLSR fourni.

En effet, le programme a été réalisé en 2010, il a donc été réalisé sur la version 2.6.x du noyau linux. Hors celui-ci a reçu beaucoup de modification depuis, surtout lors du passage à la version 3 du noyau. Cela rend le programme MP-OLSR obsolète pour les noyaux modernes. De plus, comme vu dans le chapitre précédent, nous avons décidé de travailler sur des Raspberry Pi, dont le noyau le plus ancien supporté est le 3.2.0. On ne peut donc pas passer à une version antérieur de manière stable sur un Raspberry Pi. La solution la plus viable et la plus utile est donc la mise à jour du programme.

Ce chapitre expliquera notre phase de portage vers un Raspberry Pi.

6.1 Test d'olsrd 0.5.6-r2 sur Raspberry

L'implémentation de MP-OLSR est un programme basé sur OLSRd. On utilise OLSRd dans lequel on va inclure un plugin afin de faire les changements nécessaire pour passer du fonctionnement classique d'OLSR vers celui de MP-OLSR. Il nous a donc fallu vérifier tout d'abord le bon fonctionnement d'OLSRd sur Raspberry Pi. La version 0.5.6-r2 est fonctionnelle malgré son ancienneté, et nous l'avons utilisé pour la suite. En effet, la mise à jour de MP-OLSR fut longue, c'est pour cela que nous n'avons pas utilisé de version plus récente mais ceci serait une amélioration très intéressante dans le futur. On pourrait passer d'OLSRv1 à OLSRv2 qui sera une version (une fois la certification par l'IETF faite) qui corrigera les défauts de la première version (meilleur gestion d'IPv4/IPv6, réduction de la complexité, ...) pour base de MP-OLSR.

6.2 Première compilation sur Raspberry

Raspbian, version de décembre 2014 Dans un premier temps, nous avons essayé de faire une compilation/installation directement pour voir où nous aurons des problèmes. Tout d'abord, MP-OLSR nécessite la librairie Boost v1.38, mais celle-ci mets beaucoup de temps à compiler sur Raspberry Pi (entre 6 et 8 heures). Il nous a donc fallu beaucoup de temps pour l'installation de Boost sachant qu'elle n'a pas fonctionné du premier coup car il nous a fallu ajouter quelques librairies qui ne sont pas présente sur Raspbian de base (exemple python-dev ou libbz2-dev). Toutes les phases de l'installation de Boost sont détaillés dans le fichier d'installation de MP-OLSR en annexe. Là aussi, nous utilisons une ancienne version de Boost. Nous avons essayé de faire la mise à jour de MP-OLSR afin d'utiliser la dernière version de Boost (version 1.57). Mais cela impliquerait beaucoup de modification que nous avons pas eu le temps de mettre en place.

Ensuite nous avons tenté une compilation de toutes les parties du programmes MP-OLSR. C'est ici que nous avons les premiers problèmes bloquant pour la compilation. Les fichiers étant écrit pour un noyau 2.6.21. Il n'était plus compatible avec les noyaux récents. Il fallait mettre à jour MP-OLSR ou essayer le rétrograder la version du noyau du Raspberry Pi. Comme expliqué plus haut, la dernière proposition est très compliquée voire impossible. Nous avons réussit à identifier quelques points d'erreur de compilation. Par exemple, nous utilisons les en-têtes IP pour le routage dans MP-OLSR, hors ceux-ci ne sont plus accessibles de la même manière entre la version 2.6 et une actuel du noyau.

La définition en 2.6 était la suivante :

```
struct sk_buff {

        ...

        union
{
        struct iphdr    *iph;
        struct ipv6hdr  *ipv6h;
        struct arphdr   *arph;
        struct ipxhdr   *ipxh;
        unsigned char   *raw;
} nh;
...
}
```

Hors la définition dans les noyaux actuels est la suivante :

```
struct sk_buff {

        ...

        __u16                   inner_transport_header;
__u16                   inner_network_header;
```

```
__u16                   inner_mac_header;

__be16                  protocol;
__u16                   transport_header;
__u16                   network_header;
__u16                   mac_header;

__u32                   headers_end[0];
sk_buff_data_t          tail;
sk_buff_data_t          end;
unsigned char           *head,

...
}
```

L'utilisation de cette variable a donc totalement changé. Et nous découvrirons même plus tard que l'utilisation de cette variable ne se fait plus en direct comme anciennement mais via une fonction. En effet, son ancienne utilisation était de la forme suivante :

Exemple dans `mpolsr_testbed/src_iptables/ser_module/skb_tools.c` :

```
iph = skb->nh.iph
```

Et maintenant les nouveaux noyaux nous devons utiliser la fonction suivante :

```
iph = ip_hdr(skb);
```

On ne passe plus par la structure de skb qui est un structure sk_buff. Mais on passe par une fonction qui nous retourne le résultat voulu, ici un header IP.

Distribution Backtrack En parallèle, on a aussi essayé de faire les mêmes tests sur une version Backtrack de Linux pour Raspberry Pi (Kali Linux). Nous avons eu les mêmes soucis ce qui nous a permis de confirmer que les problèmes de compilation était bien dû au noyau utilisé et non pas à la distribution. Nous avons voulu vérifié car nous savions que Jiazi Yi avait fait son expérimentation sur un version Backtrack en 2010.

Pour conclure sur cette phase, nous savions qu'il fallait absolument mettre à jour la version de MP-OLSR que nous savions car elle était totalement incompatible avec les systèmes Linux actuels.

6.3 Compilation sur CentOS 5.9 avec un noyau 2.6.21.5

Nous avons aussi voulu vérifier le bon fonctionnement de MP-OLSR sur un ancien noyau. C'est pourquoi nous avons suivi l'article "Multipath OLSRd (SEREADMO)" [1]. Dans cette article, l'installation de MP-OLSR est expliquée pour la distribution CentOS(version 5.9). Là aussi l'installation du noyau 2.6.21.5 suivi de l'installation de Boost 1.38 peut-être assez longue en

[1] http://blog.asiantuntijakaveri.fi/2013/02/multipath-olsrd-sereadmo.html

fonction des ressources offertes par le matériel. Nous l'avons fait une machine virtuel. En fin de compte, nous avons réussi à avoir 2 machines virtuel avec CentOS qui faisaient fonctionner MP-OLSR entres-elles.

6.4 Mise à jour de MP-OLSR

Après tous les tests précédents, nous pouvions donc être certain de la nécessitée de mettre à jour le programme que nous avions de MP-OLSR pour les noyaux modernes. Pour cela, Jaizi Yi nous a mis en contact avec une équipe brésilienne qui avait a priori déjà effectué une mise à jour du programme en 2012.

De cette équipe, nous avons pu discuter avec Michel Nogueira (professeur du département informatique de l'université de Parana) et Ricardo T. Macedo (membre d'un groupe de recherche NR2 de l'université de Parana). Ainsi, ils nous ont fourni un document expliquant comment mettre à jour le programme de MP-OLSR pour Ubuntu 12.04 qui utilisait, à la date de la rédaction du rapport, le noyau 3.2.x. Vous trouverez ce document en annexe B page 55.

Ainsi, avec ce document nous avons réussi a faire la mise à jour du programme de MP-OLSR pour Raspberry Pi. Tout d'abord, nous avons mis en place un Raspberry Pi avec un noyau 3.2.0. Ensuite, nous avons fait tous les changements expliqués par Ricardo dans son document.

C'est ainsi que nous avons aujourd'hui 2 Raspberry Pi qui peuvent faire fonctionner MP-OLSR. Nous avons aussi généré une nouvelle archive de fichiers source pour MP-OLSR avec les modifications que nous avons apporter et aussi un document d'installation le plus complet possible que vous trouverez en annexe C page 64.

6.5 Continuer le portage

En passant beaucoup de temps sur le portage du code vers un noyau plus récent, des pistes importantes de continuation se sont révélées.

En effet, les Raspberry Pi sont particulièrement lent pour la compilation de gros programme comme Boost. C'est pourquoi réfléchir l'exo-compilation pourrait être un moyen d'accélérer le développement sur ce matériel. Sinon comme nous l'évoquions dans les paragraphes précédents, la mise à jour n'est pas encore complète. Même si d'après nos tests MP-OLSR fonctionne sur des noyaux récents. Il reste deux pistes d'amélioration majeures :

- Mettre à jour pour utiliser la dernière version de Boost,
- Mettre à jour pour utiliser la dernière version de OLSRd ou même prévoir l'utilisation d'OLSRv2 qui est en cours de certification.

7

Expérimentations et résultats

7.1 Plateforme d'expérimentation au LINA

Afin de mener des expériences sur MP-OLSR, nous avons besoin d'avoir beaucoup de nœuds inter-connectés. Une plateforme contenant 20 Raspberry Pi est accessible au LINA, Laboratoire d'Informatique de Nantes Atlantique, laboratoire rattaché à l'Université de Nantes et à l'Ecole des Mines de Nantes. Cette plateforme est à disponibilité des chercheurs mais également pour les projets des étudiants lorsque cela est possible. Nous avons alors montrés notre intérêt envers cette plateforme pour mener nos expérimentations. Nous avons pris contact avec Jean-Yves Leblin, Responsable adjoint du service informatique du LINA et responsable de la plateforme Rasperry Pi, pour soumettre notre besoin envers une plateforme de ce type. Nous avons pris un rendez-vous afin de nous acquitter des formalités administratives et techniques. C'est ainsi qu'un accès à la plateforme via les différents locaux de l'université a été défini, nous permettant alors d'intervenir directement depuis le site de la Chantrerie.

L'intérêt d'une telle plateforme est bien entendu sa configuration. En effet, toute la partie système des équipements est identique car dupliquée depuis un système vers tous les autres. Seuls les réglages précis associés aux machines et différents sur chaque poste reste à faire, ce qui réduit fortement le temps de paramétrage de la plateforme tout en réduisant les risques d'erreurs humaines.

7.2 Schémas de topologies réseaux

Sur cette plateforme expérimentale, nous pouvons mener nos expériences réseaux. Dans cette partie nous allons détailler deux topologies pour tester le protocole MP-OLSR. Après la mise en place de ces deux topologies sur la plateforme, nous serons en capacité de récolter des données et faire des mesures sur l'efficacité de MP-OLSR.

7.2.1 Topologie à 4 nœuds : Tests minimalistes

Dans un premier temps, nous voulons voir les effets de MP-OLSR sur une architecture simple. Nous pourrons ainsi comprendre ce que apporter MP-OLSR sur un réseau avec peu de nœuds. L'organisation est présentée sur la Figure 7.1.

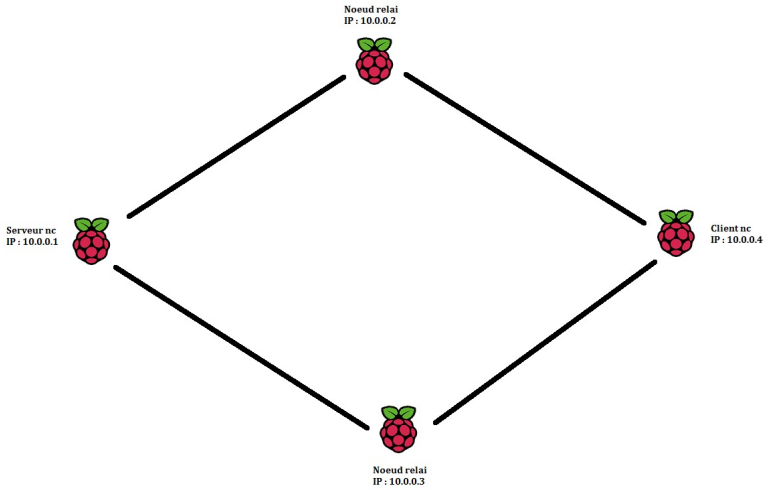


FIGURE 7.1 – Topologie à 4 nœuds

7.2.2 Topologie à 6 nœuds : Tests complémentaires

Après avoir testé rapidement MP-OLSR, nous voulons complexifier légèrement la topologie. Cette dernière reste cependant proche de la précédente afin de pouvoir les comparer. La topologie est disponible sur la Figure 7.2.

7.2.3 Topologies avancées

Nous manquons de temps pour aborder cette partie dans notre projet, qui est pourtant très intéressante et l'aboutissement de notre travail. L'objectif ici est d'atteindre des topologies avec 20, 50 ou 200 nœuds inter-connectés. Nous n'abandonnons pas l'idée pour autant. En effet, il est probable que l'expérimentation avec 20 nœuds sur la plateforme du LINA soit reconduite pour un projet réseaux de 4^{e} année. Auquel cas nous soutiendrions le groupe d'étudiants responsable de ce projet pendant leur 4^{e} année.

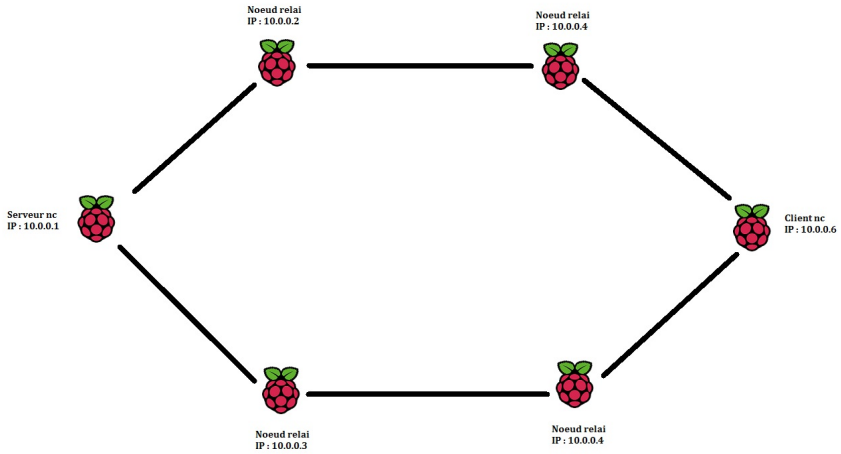


FIGURE 7.2 – Topologie à 6 nœuds

7.3 Données et Mesures pour MP-OLSR

Dans cette section, nous expliquons pourquoi nous choisissons certaines données vis-à-vis d'autres. Pour notre analyse, nous retenons que trois indicateurs qui sont très régulièrement utilisés : le pourcentage de paquets perdus, le délai et la gigue.

7.3.1 Le pourcentage de paquets perdus (% Packet Loss)

Même si la notion de paquet perdu est inévitable dans tout réseau, elle a encore plus de sens dans le cadre de MP-OLSR. Revenons rapidement à la définition de MP-OLSR. Les deux lettres MP correspondent à "Multi-Path" qui veut dire multi-chemin en anglais. MP-OLSR est donc une application multi-chemin du protocole OLSR. L'un des objectifs de MP-OLSR et de l'utilisation de plusieurs chemins est d'augmenter le taux de livraison des paquets en ajoutant des chemins différents pour la transmission. Dans nos expérimentations la mesure des paquets perdus va nous permettre de connaître l'intérêt et surtout l'efficacité de MP-OLSR par rapport à un autre protocole dans un réseau.

Remarque : Nos expériences se basent sur des connections ethernet. C'est pourquoi nous nous attendons à avoir un nombre très faible de paquets perdus contrairement aux liaisons sans fil.

7.3.2 Le délai de transmission des paquets (End-to-end delay)

Ce critère est couramment utilisé aujourd'hui dans l'estimation de la qualité d'une connexion internet. Dans le cadre de MP-OLSR, cette notion est importante car elle permet de prendre

en compte les temps de calcul liés au protocole dans le temps d'acheminement des paquets depuis une source vers une destination. L'intérêt est de comparer le temps perdu par des calculs plus complexes par rapport aux gains apportés via les fonctionnalités mises en place, comme la récupération de route, la détection de boucle ou plus simplement l'utilisation de plusieurs routes (cf Chapitre 2 : Le protocole MP-OLSR).

Remarque : Nos communications filaires ne devraient pas excéder 5 à 10 millisecondes de délai entre la source et la destinations. Dans le cas d'un réseau sans-fil, nous avons un délai de l'ordre de la centaine de millisecondes et pouvant même atteindre la seconde.

7.3.3 La gigue (Jitter)

La gigue correspond à la différence de temps de transmission dans un flux de paquets. La mesure de la gigue est très rare sur un réseau filaire car celle-ci est généralement inexistante ou suffisamment faible pour être ignorée grâce à la stabilité des réseaux câblés. Dans le cas des réseaux sans fil, cette notion est omniprésente car le temps de transmission entre deux nœuds est en constante évolution à cause de la mobilité des nœuds, des conditions climatiques, etc... Les principales conséquences de la gigue concernent la réception des paquets car ceux-ci peuvent être désordonnés. Une des solutions consiste à utiliser un buffer mais ce dernier va créer une latence dans la transmission, ce que nous voulons éviter (cf. paragraphe précédent).

MP-OLSR est particulièrement destiné aux réseaux sans-fil, c'est pourquoi la gigue est une mesure importante. Nous nous attendons à avoir une gigue réduite avec l'usage de plusieurs chemins. Nos expériences sur réseau filaire ne reflètent pas entièrement les conditions d'utilisation sur un réseau sans fil qui peut introduire beaucoup de gigue.

7.4 Résultats et Analyse

Malheureusement nous n'avons pas pu mener d'expériences avant la fin de notre projet. Cependant, nous n'abandonnons pas cette partie très importante. Bien au contraire, la phase d'expérimentation va être transmise à un groupe d'étudiants en 4^{e}année dans le cadre de projets réseaux. Nous serons alors présents pour soutenir et apporter nos enseignements pour mener à bien ces expériences.

8

Conclusion

Dans un premier temps nous avons étudié le protocole réseau constituant le centre de notre projet de recherche et développement, MP-OLSR. Nous avons commencé par comprendre les concepts qui ont mené à son élaboration. Puis nous nous sommes intéressés à la position d'un tel protocole dans les réseaux actuels. C'est pourquoi nous avons fait une analyse des environnements d'expérimentations permettant de mettre en œuvre MP-OLSR. De cette analyse, nous avons tiré deux cas pratiques dans lequel nous pouvons tester MP-OLSR : Une plateforme de tests dotée de nombreux capteurs, et un plateforme comprenant des Raspberry Pi. La première solution est intéressante mais elle nécessite une nouvelle implémentation de MP-OLSR afin que celle-ci soit adaptée aux capteurs, ce qui ne fait pas partie des objectifs du projet. Nous nous sommes alors tournés vers les Raspberry Pi, utilisant un système Linux comme l'exige l'implémentation fournie.

Dans la partie réalisation, nous nous sommes donc concentrés sur l'installation de MP-OLSR sur un Raspberry Pi. Nous avons pu installer une version sur Raspberry Pi avec l'aide de Ricardo T. Macedo qui a également travaillé sur MP-OLSR. Cette étape a été longue car plusieurs problèmes se sont succédés. On peut notamment citer quelques-uns comme l'écart d'*âge* entre l'implémentation de MP-OLSR (2009-2010) et la commercialisation des Raspberry Pi (2012), les changements de la gestion réseau dans le noyau Linux au travers des années. Ces problèmes induisent alors des conflits au sein des installations de certains programmes comme MP-OSLR. Dans le peu de temps restant, nous avons cherché à construire des expérimentations sur des topologies. Nous n'avons que les premières lignes de ce travail. Cependant, nous avons essayé de recueillir les éléments importants pour les expérimentations et les analyses des résultats obtenus (cf Chapitre 7).

De ce projet nous pouvons tirer plusieurs enseignements. D'une part, celui-ci s'attèle à la recherche donc à sujet récent et de pointe. Nous sommes parmi les premiers à travailler sur le sujet et nous accordons beaucoup plus d'importance à notre travail dans ces conditions. Nous

nous sentons plus impliqués et plus utiles vis à vis du domaine où nous travaillons. Une autre dimension qui n'est jamais assez représentée dans les rapports concerne les échanges créés dans le contexte du projet. Dans notre cas, nous avons pu dialoguer avec Jiazi Yi qui a écrit la thèse MP-OLSR, des chercheurs originaires du Brésil, et d'autres personnes qui ont participé à MP-OLSR. Nous avons pu également rencontrer certaines de ces personnes afin de discuter de leurs travaux ainsi que notre implication pour MP-OLSR. La communication entre tous ces acteurs a eu un impact essentiel sur ce projet.

Bibliographie

[IL14] IoT-LAB. Get and compile a m3 firmware code. *https://www.iot-lab.info*, 2014. [1]. 32

[MMSN14] R. Macedo, R. Melo, A. Santos, and M. Nogueira. Experimental performance comparison of single-path and multipath routing in vanets. In *Global Information Infrastructure and Networking Symposium (GIIS), 2014*, pages 1–6, Sept 2014.

[QPV14] Pham tran anh Quang, Kandaraj Piamrat, and Cesar Viho. Qoe-aware routing for video streaming over vanets. In *IEEE 80th Vehicular Technology Conference : VTC2014-Fall*, Vancouver, Canada, September 2014.

[Wik14] Wikipedia. Optimized link state routing protocol. *http://fr.wikipedia.org*, 2014. [2]. 21

[YADP11] Jaizi Yi, Asmaa Admane, Sylvain David, and Benoît Parrein. Multipath optimized link state routing for mobile ad hoc networks. *Ad Hoc Networks*, 9, 2011. [3]. 9, 13

[YCH+09] Jiazi Yi, Eddy Cizeron, Salima Hamma, Benoît Parrein, and Pascal Lesage. Implementation of multipath and multiple description coding in olsr. *arXiv preprint arXiv :0902.4781*, 2009.

[Yi10] Jiazi Yi. *Protocole de routage à chemins multiples pour les réseaux ad hoc*. PhD thesis, Université de Nantes, November 2010. [4]. 12

[YP14] Jiazi Yi and Benoit Parrein. Multi-path extension for the optimized link state routing protocol version 2 (olsrv2). Internet-Draft draft-ietf-manet-olsrv2-multipath-02, IETF Secretariat, October 2014. http://www.ietf.org/internet-drafts/draft-ietf-manet-olsrv2-multipath-02.txt. 9

[1] https://www.iot-lab.info/tutorials/get-compile-a-m3-firmware-code/
[2] http://fr.wikipedia.org/wiki/Optimized_link_state_routing_protocol
[3] http://jiaziyi.com/documents/MP_OLSR.pdf
[4] http://www.jiaziyi.com/documents/MP-OLSR_thesis.pdf

Table des figures

Code Source

A.1 tutorial IoT-LAB : main.c

main.c :

```
#include <platform.h>
#include <stdint.h>
#include <stdlib.h>
#include <printf.h>
#include <string.h>

#include "phy.h"
#include "soft_timer.h"
#include "event.h"

#ifdef IOTLAB_M3
#include "lps331ap.h"
#include "isl29020.h"
#endif
#include "mac_csma.h"
#include "phy.h"

// choose channel in [11-26]
#define CHANNEL 11
#define RADIO_POWER PHY_POWER_0dBm

#define ADDR_BROADCAST 0xFFFF

// UART callback function
static void char_rx(handler_arg_t arg, uint8_t c);
static void handle_cmd(handler_arg_t arg);

// timer alarm function
static void alarm(handler_arg_t arg);
static soft_timer_t tx_timer;
#define BLINK_PERIOD soft_timer_s_to_ticks(1)

/* Global variables */
// print help every second
volatile int8_t print_help  = 1;
volatile int8_t leds_active = 1;
```

```
/**
 * Sensors
 */
#ifdef IOTLAB_M3
static void temperature_sensor()
{
    int16_t value;
    lps331ap_read_temp(&value);
    printf("Chip␣temperature␣measure:␣%f\n", 42.5 + value / 480.0);
}

static void light_sensor()
{
    float value = isl29020_read_sample();
    printf("Luminosity␣measure:␣%f␣lux\n", value);
}

static void pressure_sensor()
{
    uint32_t value;
    lps331ap_read_pres(&value);
    printf("Pressure␣measure:␣%f␣mabar\n", value / 4096.0);
}
#endif


/*
 * Radio config
 */
static void send_packet()
{
    uint16_t ret;
    static uint8_t num = 0;

    static char packet[PHY_MAX_TX_LENGTH - 4];  // 4 for mac layer
    uint16_t length;
    // max pkt length <= max(cc2420, cc1101)
    snprintf(packet, sizeof(packet), "Hello␣World!:␣%u", num++);
    length = 1 + strlen(packet);

    ret = mac_csma_data_send(ADDR_BROADCAST, (uint8_t *)packet, length);

    printf("\nradio␣>␣");
    if (ret != 0)
        printf("Packet␣sent\n");
    else
        printf("Packet␣sent␣failed\n");
}

static void send_big_packet()
{
    uint16_t ret;
    static uint8_t num = 0;

    static char packet[PHY_MAX_TX_LENGTH - 4];  // 4 for mac layer
    static char pluspack[40]="01234567890123456789012345678901234567890\0";
    uint16_t length;

    snprintf(packet, sizeof(packet), "Big␣Hello␣World!:␣%u␣%s",num++, pluspack);
    length = 1 + strlen(packet);

    ret = mac_csma_data_send(ADDR_BROADCAST, (uint8_t *)packet, length);

    printf("\nradio␣>␣");
    if (ret != 0)
```

```
        printf("Big␣packet␣sent\n");
    else
        printf("Big␣packet␣sent␣failed\n");
}


/* Reception of a radio message */
void mac_csma_data_received(uint16_t src_addr,
        const uint8_t *data, uint8_t length, int8_t rssi, uint8_t lqi)
{
    // disable help message after receiving one packet
    print_help = 0;

    printf("\nradio␣>␣");
    printf("Got␣packet␣from␣%x.␣Len:␣%u␣Rssi:␣%d:␣'%s'\n",
            src_addr, length, rssi, (const char*)data);
    handle_cmd((handler_arg_t) '\n');
}

/* Leds action */
static void leds_action()
{
  printf("\nleds␣>␣");
  if (leds_active) {
    // The alarm timer looses the hand
    leds_active = 0;
    // Switch off the LEDs
    leds_off(LED_0 | LED_1 | LED_2);
    printf("off\n");
  } else {
    // The alarm timer takes the hand
    leds_active = 1;
    printf("blinking\n");
  }

}

/*
 * HELP
 */
static void print_usage()
{
    printf("\n\nIoT-LAB␣Simple␣Demo␣program\n");
    printf("Type␣command\n");
    printf("\th:\tprint␣this␣help\n");
#ifdef IOTLAB_M3
    printf("\tt:\ttemperature␣measure\n");
    printf("\tl:\tluminosity␣measure\n");
    printf("\tp:\tpressure␣measure\n");
#endif
    printf("\ts:\tsend␣a␣radio␣packet\n");
    printf("\tb:\tsend␣a␣big␣radio␣packet\n");
    printf("\te:\ttoggle␣leds␣blinking\n");
    if (print_help)
        printf("\n␣Type␣Enter␣to␣stop␣printing␣this␣help\n");
}

static void hardware_init()
{
    // Openlab platform init
    platform_init();
    event_init();
    soft_timer_init();

    // Switch off the LEDs
```

```
    leds_off(LED_0 | LED_1 | LED_2);

    // Uart initialisation
    uart_set_rx_handler(uart_print, char_rx, NULL);

#ifdef IOTLAB_M3
    // ISL29020 light sensor initialisation
    isl29020_prepare(ISL29020_LIGHT__AMBIENT, ISL29020_RESOLUTION__16bit,
            ISL29020_RANGE__16000lux);
    isl29020_sample_continuous();

    // LPS331AP pressure sensor initialisation
    lps331ap_powerdown();
    lps331ap_set_datarate(LPS331AP_P_12_5HZ_T_12_5HZ);
#endif

    // Init csma Radio mac layer
    mac_csma_init(CHANNEL, RADIO_POWER);

    // Initialize a openlab timer
    soft_timer_set_handler(&tx_timer, alarm, NULL);
    soft_timer_start(&tx_timer, BLINK_PERIOD, 1);

}

static void handle_cmd(handler_arg_t arg)
{
    switch ((char) (uint32_t) arg) {
#ifdef IOTLAB_M3
        case 't':
            temperature_sensor();
            break;
        case 'l':
            light_sensor();
            break;
        case 'p':
            pressure_sensor();
            break;
#endif
        case 's':
            send_packet();
            break;
        case 'b':
            send_big_packet();
            break;
        case 'e':
            leds_action();
            break;
        case '\n':
            printf("\ncmd > ");
            break;
        case 'h':
        default:
            print_usage();
            break;
    }
}

int main()
{
    hardware_init();
    platform_run();
    return 0;
}
```

```
/* Reception of a char on UART and store it in 'cmd' */
static void char_rx(handler_arg_t arg, uint8_t c) {
    // disable help message after receiving char
    print_help = 0;
    event_post_from_isr(EVENT_QUEUE_APPLI, handle_cmd,
            (handler_arg_t)(uint32_t) c);
}

static void alarm(handler_arg_t arg) {
     if (leds_active)
       leds_toggle(LED_0 | LED_1 | LED_2);

    /* Print help before getting first real \n */
    if (print_help) {
        event_post(EVENT_QUEUE_APPLI, handle_cmd, (handler_arg_t) 'h');
        event_post(EVENT_QUEUE_APPLI, handle_cmd, (handler_arg_t) '\n');
    }

}
```

Mise à jour de MP-OLSR pour Ubuntu 12.04

December 1th, 2012

Configuring MP-OLSR on Ubuntu 12.04

NR2 - Federal University of Paraná

Authors:

Ricardo Tombesi Macedo
Michele Nogueira

1. Installing packages

Dependences:

```
# apt-get install bison flex unrar automake g++
```

Download and compile boost 1.38

```
$ mkdir -p /opt/boost
$ cd /opt/boost
$ wget http://sourceforge.net/projects/boost/files/boost/1.38.0/boost_1_38_0.tar.gz
$ tar xvzf boost_1_38_0.tar.gz
$ cd boost_1_38_0
$ ./configure
$ make
$ make install
$ echo "/usr/local/lib/" >>/etc/ld.so.conf
$ ldconfig
```

libnfnetlink

```
$ mkdir -p /opt/libnfnetlink
$ cd /opt/libnfnetlink
$ wget http://ftp.netfilter.org/pub/libnfnetlink/libnfnetlink-0.0.41.tar.bz2
$ tar xvjf libnfnetlink-0.0.41.tar.bz2
$ cd libnfnetlink-0.0.41
$ ./configure
$ make
$ make install
```

libnetfilter_queue 0.0.17

```
$ mkdir -p /opt/libnetfilter_queue
$ cd /opt/libnetfilter_queue
$ wget http://ftp.netfilter.org/pub/libnetfilter_queue/libnetfilter_queue-0.0.17.tar.bz2
$ tar xvjf libnetfilter_queue-0.0.17.tar.bz2
$ cd libnetfilter_queue-0.0.17
$ PKG_CONFIG_PATH=/usr/local/lib/pkgconfig ./configure
$ make
$ make install
```

Download and unpack mpolsr

```
$ wget http://mocha3004.mochahost.com/~jiaziyi/documents/SEREADMO/temp.rar
$ rar x temp.rar
```

2. Adapting SK_buff Structure to new Kernels

```
$ cd mpolsr_testbed/src_iptables/ser_module
```

In File sereadmo-module.c modify:

```
#include <linux/netfilter_ipv4.h> to #include "/usr/include/linux/netfilter\_ipv4.h"
```

In library netfilter_ipv4.h (/usr/include/linux/netfilter_ipv4.h) add

```
#include <limits.h>
```

File: skb_tools.c. In lines 46 / 73 / 116 / 176 / 245 / 296 / 349 change skb->nh.iph ou (skb)->nh,iph; to ip_hdr(skb);

File: hook_local_out.c. Change lines:

```
37: #include <linux/netfilter_ipv4.h> to #include "/usr/include/linux/netfilter_ipv4.h"
54 and 178: (*skb)->nh.iph, to ip_hdr(*skb)
95: skb->nh.iph, to ip_hdr(skb)
145: skb_new->nh.iph, to ip_hdr(skb_new)
161: change skb_new->dst->dev to skb_dst(skb_new)->dev
```

Add the line below after the bloc "include":

```
extern int ip_route_me_harder(struct sk_buff *skb, unsigned addr_type);
```

In file: hook_local_in.c. Add the line below after the bloc "include":

```
extern int ip_route_me_harder(struct sk_buff *skb, unsigned addr_type);
```

Line 39 change **#include <linux/netfilter_ipv4.h>** to **#include "/usr/include/linux/-netfilter_ipv4.h"**

In lines 90 / 107 / 119, change **skb->nh.iph** to **ip_hdr(skb)**

In line 153 change **skb_new->dst->dev** to **skb_dst(skb_new)->dev**

Line 170 change **(*skb)->nh.iph** to **ip_hdr(*skb)**

File: hook_pre_routing.c

Lines 64 / 159 skb->nh.iph, change to ip_hdr(skb)

Line 113 skb_new->nh.iph, change to ip_hdr(skb_new)

Line 220 (*skb)->nh.iph, change to ip_hdr(*skb)

Change **#include <linux/netfilter_ipv4.h>** to **#include "/usr/include/linux/netfilter_ipv4.h"**

Add the line below after the bloc "include":

```
    extern int ip_route_me_harder(struct sk_buff *skb, unsigned addr_type);
```

Line 129, change **skb_new->dst->dev** to **skb_dst(skb_new)->dev**

File: ser_device.c

Line 36: Change: **static rwlock_t g_ser_device_lock = RW_LOCK_UNLOCKED;** to **static rwlock_t g_ser_device_lock = ___RW_LOCK_UNLOCKED(g_ser_device_lock);**

Line 240: Change **if(unregister_chrdev(g_major_dec_num, DEVICE_NAME) < 0)** to: **(unregister_chrdev(g_major_dec_num, DEVICE_NAME);**

Line 241: Comment all this line.

File ser_path_tc.c

Line 39: Change **rwlock_t g_ser_tc_lock = RW_LOCK_UNLOCKED;** to **rwlock_t g_ser_tc_lock = ___RW_LOCK_UNLOCKED(g_ser_tc_lock);**

```
$ make
$ cd ../../../
```

2.1 Instaling ser_routing module

```
$ cd mpolsr_testbed/src_iptables/ser_routing
$./configure
```

Editing files from folder src:

In files:

hook_local_in.cpp hook_local_out.cpp hook_pre_routing.cpp ser_routing.cpp

Case there are the libraries:

```
#include <linux/netfilter.h>        /* for NF_ACCEPT */
#include <linux/netfilter_ipv4.h> // for NF_IP_xxx
#include <libnetfilter_queue/libnetfilter_queue.h>
```

Move to between the braces

```
extern "C" {

}
```

Edit the file

ser_hooks.h

After:

```
#ifndef _SER_HOOKS_
```

Add:

```
extern "C" {
#include "/usr/local/include/libnetfilter_queue/libnetfilter_queue.h"
}
```

File: ser_routing/src/configfile.cpp

Add the libraries stdio.h, stdlib.h and string.h

File: ser_routing/src/hook_local_in.cpp

Add the beginning of this file:

```
#include "/usr/include/netinet/in.h"
```

OBS.: add the beginning of this file!
Add the library string.h
File: ser_routing/src/hook_pre_routing.cpp
Add the library string.h
File: ser_routing/src/ippacket.cpp
Add the library stdio.h
File: ser_routing/src/mpolsrpacket.cpp
Add the library string.h
File: ser_routing/src/ser_routing.cpp
Add the beginning of this file:

```
#include "/usr/include/netinet/in.h"
```

Making links to old versions:

```
$ cd /usr/local/lib/
# ln -s libboost_date_time-gcc44-mt.so libboost_date_time-gcc41-mt.so
# ln -s libboost_serialization-gcc44-mt.so libboost_serialization-gcc41-mt.so
# ln -s libboost_system-gcc44-mt.so libboost_system-gcc41-mt.so
# ln -s libboost_thread-gcc44-mt.so libboost_thread-gcc41-mt.so
```

Go back to folder ser_routing.

```
$ make
$ sudo make install
```

Before test, run:

```
$ export LD_LIBRARY_PATH=/usr/local/lib/:\$LD_LIBRARY_PATH
```

2.2 Installing ser_valider_dev module

```
$ cd ../ser_valider_dev
$ cd mpolsr_testbed/src_iptables/ser_valider_dev/
$ make
```

2.3 Installing ser_valider_tc module

```
$ cd ~
$ cp mpolsr_testbed/src_iptables/ser_routing/src/common.* mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path.h mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path.c mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_def.h mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_djk.o mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_tc.c mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_test.c mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_tc.o mpolsr_testbed/src_iptables/ser_valider_tc/
$ cp mpolsr_testbed/src_iptables/ser_routing/src/ser_path_djk.c mpolsr_testbed/src_iptables/ser_valider_tc/
$ cd mpolsr_testbed/src_iptables/ser_valider_tc/
```

Changes in ser_valider_tc.c file

Comment:

Functions test4 and test5 inserting /* in line 133 and */ in line 301. Comment also the lines that call these functions: lines 317, 320, 321.

3. Running

Compile mpolsrd

```
cd /opt/mpolsr/mpolsr_testbed/src_olsr/olsrd-0.5.6-r2/
chmod a+x gcc-warnings ld-warnings
make
make libs
```

Compile sereadmo plugin

```
cd lib
rm -rf old_sereadmo
cd sereadmo
make clean
ln -s /usr/local/include/boost-1_38/boost/ /usr/local/include/boost
./install_plugin.sh
```

Install mpolsrd and plugins

```
cd ../..
make install_all
```

Configure mpolsrd

```
sed -i.bak /etc/olsrd.conf \
    -e's/"XXX" "YYY"/"wlan0"/g' \
    -e's/ClearScreen     yes/ClearScreen     no/g'
cat <<'__EOF__'>>/etc/olsrd.conf
LoadPlugin "olsrd_sereadmo.so.0.1"
{
}
__EOF__
```

ser_routing

```
ln -s /usr/local/lib/libboost_date_time-gcc46-mt.so /usr/local/lib/libboost_date_time-gcc41-mt.so
ln -s /usr/local/lib/libboost_serialization-gcc46-mt.so /usr/local/lib/libboost_serialization-gcc41-mt.so
ln -s /usr/local/lib/libboost_system-gcc46-mt.so /usr/local/lib/libboost_system-gcc41-mt.so
ln -s /usr/local/lib/libboost_thread-gcc46-mt.so /usr/local/lib/libboost_thread-gcc41-mt.so

make clean
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing
chmod a+x configure
./configure
make
```

Config file for ser_routing (same defaults as in source)

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing/src
cat <<'__EOF__'>ser_routing.conf
udp_port=104
DebugLevel_nfqueue=10
DebugLevel_olsrevent=10
DebugLevel_dijkstra=10
Dijkstra_nb_path=3
Dijkstra_coef_fe=2
Dijkstra_coef_fp=2
__EOF__
```

Configure mesh network interfaces eth1, eth2 and eth3 Reboot

4. Running

Launch mpolsrd

```
# olsrd -nofork
```

Launch ser_routing

```
$ cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing/src
$ ./ser_routing
```

Test connection (server)

```
$ nc -v -u -l 0.0.0.0 104
```

Test connection (client) 10.99.2.2 is our mesh interface IP and 10.99.2.1 is destination via mesh

```
$ echo foo | nc -v -u -s 10.99.2.2 10.99.2.1 104

.. FAILS with following on netcat client side dmesg

Feb  9 15:12:02 olsr-2 kernel: LOCAL OUT : receive sk_buf (protocol# : 17)
Feb  9 15:12:02 olsr-2 kernel: LOCAL OUT : UDP port : 104)
Feb  9 15:12:02 olsr-2 kernel: LOCAL OUT : process datagram
Feb  9 15:12:02 olsr-2 kernel: find_datagram_path : failed to find src node (202630A) in TC data
Feb  9 15:12:02 olsr-2 kernel: LOCAL OUT : failed to find path from 202630A to 102630A
Feb  9 15:12:02 olsr-2 kernel: LOCAL OUT : end - DROP
```

5. Possibles Errors

Case you receive the error message:

```
ser_routing: error while loading shared libraries:
libboost_date_time-gcc41-mt-1_38.so.1.38.0: cannot open shared object file:
No such file or directory
```

Run:

```
# export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
```

And try again

Installation de MP-OLSR sur Raspberry Pi (avec Raspbian)

C.1 Changement de noyau

Avant d'installer MP-OLSR sur un Raspberry Pi, nous devons charger un noyau 3.2 car le code n'est entièrement à jour pour les derniers noyaux. La première étape consiste à mettre le système à jour et récupérer un noyau 3.2 pour Raspberry Pi dans les dépôts. Nous aurons également besoin des `headers` pour les phases de compilation.

```
apt-get update
apt-get upgrade
rpi-update

apt-get install linux-headers-3.2.0-4-* linux-image-3.2.0-4-rpi
```

Ensuite, le noyau noyau doit être défini pour le démarrage. Un Raspberry Pi ne possède pas de GRUB et le système qui le remplace est nettement plus simple. En effet, le Raspberry Pi démarre sur une image du noyau spécifique, correspondant au fichier `kernel.img` dans le dossier `/boot`. De plus, l'image récupérée depuis les dépôts est directement disponible dans le même dossier. La procédure à suivre est donc la suivante :

```
cd /boot
cp kernel.img kernel_backup.img
cp vmlinuz-3.2.0-4-rpi kernel.img

reboot
```

La commande `uname -a` permet de vérifier que le changement de noyau s'est bien passé. Il est possible que le Raspberry ne redémarre pas après ce changement. Dans ce cas il est possible d'accéder à la partition contenant le dossier `/boot` car celle-ci est en format `FAT`.

C.2 Récupération des fichiers

Copier le fichier `olsrd.conf` dans le dossier `/etc/` . Le fichier de configuration doit être accessible via `/etc/olsrd.conf`.

Extraire l'archive `mpolsr.zip` dans le dossier `/opt/`.

C.3 Installation des pré-requis

C.3.1 Paquets disponibles dans les dépôts

```
apt-get install python-dev libbz2-dev bison flex automake g++
```

C.3.2 Installation de Bzip2

```
wget http://www.bzip.org/1.0.6/bzip2-1.0.6.tar.gz
tar zxvf bzip2-1.0.6.tar.gz
cd bzip2-1.0.6
make
make install
```

C.3.3 Compilation et installation de Boost 1.38

Cette partie est très couteuse en temps. Si l'installation est prévue sur Raspberry Pi l'exo-compilation est préférable. Dans le cas contraire, cette étape demandera entre 6 et 8 heures de calcul.

```
mkdir -p /opt/boost
cd /opt/boost
wget http://sourceforge.net/projects/boost/files/boost/1.38.0/
   boost_1_38_0.tar.gz
tar xvzf boost_1_38_0.tar.gz
cd boost_1_38_0
./configure
make
make install
echo "/usr/local/lib/" >>/etc/ld.so.conf
ldconfig
```

C.3.4 Compilation et installation de Libnfnetlink

```
mkdir -p /opt/libnfnetlink
cd /opt/libnfnetlink
```

```
wget http://ftp.netfilter.org/pub/libnfnetlink/libnfnetlink-0.0.41.tar
   .bz2
tar xvjf libnfnetlink-0.0.41.tar.bz2
cd libnfnetlink-0.0.41
./configure
make
make install
```

C.3.5 Compilation et installation de Libnetfilter_queue

```
mkdir -p /opt/libnetfilter_queue
cd /opt/libnetfilter_queue
wget http://ftp.netfilter.org/pub/libnetfilter_queue/
   libnetfilter_queue-0.0.17.tar.bz2
tar xvjf libnetfilter_queue-0.0.17.tar.bz2
cd libnetfilter_queue-0.0.17
PKG_CONFIG_PATH=/usr/local/lib/pkgconfig ./configure
make
make install
```

C.4 Compilation des modules

C.4.1 Modification du fichier `netfilter_ipv4.h`

Modifier le fichier "`/usr/include/linux/netfilter_ipv4.h`"

Remplacer à la ligne 13 : `#include <limits.h>` par `#include <linux/limits.h>`.

C.4.2 Compilation de ser_module

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_module
make
```

C.4.3 Restauration du fichier `netfilter_ipv4.h`

Modifier le fichier "`/usr/include/linux/netfilter_ipv4.h`".

Remplacer à la ligne 13 : `#include <linux/limits.h>` par `#include <limits.h>`.

C.4.4 Liens vers d'autres librairies

Pour la suite de la compilation nous avons besoin d'adapter les librairies installées avec celles requises. Elles sont heureusement compatibles, c'est pourquoi nous créons des liens vers des versions plus anciennes pointant sur les nouvelles versions (Attention : la version est à vérifier dans le dossier `/usr/local/lib`, dans l'exemple suivant la version présente est la version

46. Il est possible de rencontrer la version 44 par exemple, auquel le code suivant sera à adapter en conséquence).

```
cd /usr/local/lib/
ln -s libboost_date_time-gcc46-mt.so libboost_date_time-gcc41-mt.so
ln -s libboost_serialization-gcc46-mt.so \
    libboost_serialization-gcc41-mt.so
ln -s libboost_system-gcc46-mt.so libboost_system-gcc41-mt.so
ln -s libboost_thread-gcc46-mt.so libboost_thread-gcc41-mt.so
```

C.4.5 Compilation de ser_routing

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing
./configure
make
sudo make install
```

C.4.6 Compilation de ser_valider_dev

```
export LD_LIBRARY_PATH=/usr/local/lib/:\$LD_LIBRARY_PATH
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_valider_dev/
make
```

C.4.7 Compilation de OLSRd

```
cd /opt/mpolsr/mpolsr_testbed/src_olsr/olsrd-0.5.6-r2/
chmod a+x gcc-warnings ld-warnings
make
make libs
```

C.4.8 Compilation du plugin pour OLSRd

```
cd /opt/mpolsr/mpolsr_testbed/src_olsr/olsrd-0.5.6-r2/lib/sereadmo
make clean
ln -s /usr/local/include/boost-1_38/boost/ /usr/local/include/boost
./install_plugin-1.sh
```

C.5 Installation de MP-OLSRd

C.5.1 Installation de OLSRd

```
cd /opt/mpolsr/mpolsr_testbed/src_olsr/olsrd-0.5.6-r2/
make install_all
```

Attention ! Le fichier `/etc/olsrd.conf` ne doit pas être écrasé.

C.5.2 Installation de ser_routing

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing
chmod a+x configure
./configure
make
```

C.6 Configuration

La configuration de MP-OLSR est très simple. Il suffit de paramétrer correctement les interfaces utilisées dans le fichier `/etc/olsrd.conf`.

C.7 Exécution

Dans un premier terminal :

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_module
./inst_module.csh
olsrd -nofork
```

Puis dans un second terminal :

```
cd /opt/mpolsr/mpolsr_testbed/src_iptables/ser_routing/src
./ser_routing
```

Cross compilation

Comme nous en parlions dans notre phase d'expérimentation, la compilation du programme, notamment de la librairie Boost, peut mettre beaucoup de temps. C'est pourquoi l'usage de la cross-compilation sur Raspberry Pi pourrait être une solution. N'ayant pas eu le temps de faire de recherche à ce sujet. Nous souhaitons tout de même fournir un début de travail sur le sujet. C'est pourquoi vous trouverez dans la citation suivante le travail d'Olivier Blin et Quentin Lebourgeois effectué en 2013-2014 sur des Raspberry Pi pour la *"mise en place d'un système de fichier distribué sur une architecture Cloud privé"*. Bien qu'incomplet, ils ont commencé à travailler sur le sujet et ceci peut-être un bon point de départ.

Cross compilation

Pour mettre en place RozoFS sur Raspberry PI la première fois, nous avons procédé avec une compilation du programme directement sur le Raspberry. Cette étape durant plus de 45 minutes, nous allons chercher à l'accélérer grâce au système de la cross compilation. Cet outil ne nous est pas immédiatement nécessaire car, ayant déjà compilé sur le Raspberry Pi, nous disposons déjà des packages pour installer RozoFS sur tous les Raspberry. Cependant, dans le cas d'une mise à jour de RozoFS, il faut être en mesure de pourvoir le recompiler rapidement pour cette plate-forme. La mise en place de cette cross-compilation peut être assez lourde la première fois, mais il n'est plus nécessaire d'y toucher une fois configurée. La première étape va donc consister à mettre en place l'outil de cross compilation. Nous allons proposer deux compilateurs différents. Un pour sa facilité de mise en place et le deuxième qui est très utilisé et à jour.

Cross compilateur fourni pour le Raspberry

Sur le dépot Github du Raspberry Pi (https://github.com/raspberrypi), on peut retrouver des outils de compilation. Nous allons utiliser le dépôt
https://github.com/raspberrypi/tools.

Il faut donc cloner le dépot
`git clone https://github.com/raspberrypi/tools.git`
Nous allons utiliser le compilateur dans le dossier `tools/arm-bcm2708/gcc-linaro-arm-linux-g`
Le dossier `bin` *contient tous les exécutables pour permettre de compiler de code pour Raspberry Pi.*

Outil crosstool-ng

Cet outil est très utilisé pour faire de la cross compilation. Il va falloir le configurer pour la plate-forme cible puis compiler le compilateur qui va produire le code pour le Raspberry. Cette solution est très intéressante car le compilateur obtenu aura les dernières versions des bibliothèques qu'il intègre. L'absence de mise à jour depuis 1 an sur le dépôt Github du Raspberry nous a pousser à faire cette démarche pour proposer une alternative et plus de souplesse. Le défaut de cette méthode est la que la compilation du compilateur en lui-même peut être beaucoup plus complexe. Nous avons été confronté à de nombreuses erreurs sur certaines versions de bibliothèques utilisés par le compilateur qui nous a pris beaucoup de temps. Le but va être de présenter le chemin que nous avons pris pour résoudre les problèmes et obtenir un compilateur pour Raspberry Pi.

Installation de crosstool-ng

Nous avons utilisé deux plate-forme hôte : Debian 7.2 x86 et Xubuntu x64. Les deux plate-formes ont présenté les mêmes erreurs sur le processus de compilation. Nous allons procéder à la mise en place de l'outil crosstool-ng :

- *Télécharger la dernière version de crosstool-ng (1.19.0) sur leur site :* `http://crosstool-ng.org/download/crosstool-ng/` *;*
- *Extraire le fichier :* `tar xvf` *;*
- *Lancer le script de configuration (voir les dépendances ci-dessous) :* `./configure --prefix=/opt/crosstool`*. Le dossier -prefix donne la localisation de l'installation sur le système ;*
- *Compiler :* `make` *;*
- *Installer :* `make install`*.*

Configuration de crosstool-ng

Il va maintenant falloir configurer cet outil pour créer le compilateur qui va compiler du code à destination d'un Raspberry Pi.

Les manipulations suivantes doivent être faites avec un utilisateur du système (pas de root).

Créez un nouveau dossier puis lancez une nouvelle configuration : `ct-ng menuconfig`
Ensuite, modifiez les paramètres comme ci-dessous :

- *Paths and misc options*
- *Activer 'Try features marked as EXPERIMENTAL'*
- *Target options*
- *Target Architecture sur ARM*
- *Endianness sur 32-bit*
- *Floating point sur hardware (FPU)*
- *Operating system*
- *Target OS sur Linux*
- *C compiler*
- *Activer 'Show Linaro versions'*
- *gccversion sur linaro-4.7-2013.06-1*
- *Désactiver 'Link libstdc++ statically into gcc binary'*
- *Désactiver 'GRAPHITE loop optimisations'*
- *Désactiver 'GRAPHITE loop optimisations'*
- *C-library*
- *eglibc version sur 2_15*

La première chose à faire avant d'aller plus loin est d'installer subversion qui est nécessaire pour récupérer la bibliothèque eglibc (sinon il y a une erreur de compilation).

Les principales problèmes ont été rencontrés sur certaines version de bibliothèques qui apportaient erreurs ou incompatibilités. Par exemple, les version d'eglibc 2_16 et 2_17 ne fonctionnait pas sur notre configuration. Les deux désactivations dans C compiler sont aussi indispensable, puisqu'elles permettent de retirer des bibliothèques qui ne sont plus compilables avec les dernières versions de gcc (à partir de 4.6).

On peut maintenant lancer la compilation (environ 1H) : `ct-ng build`

Un nouveau dossier est créé dans votre home (/x-tools) qui contient le compilateur : `/x-tools/arm-unknown-linux-gnueabi/bin`

Vous pouvez modifier les versions du compilateur ou des bibliothèques pour prendre des versions plus récentes. Cependant, cela peut provoquer des erreurs lors de la compilation, il faudra donc passer un certain temps pour régler les problèmes. "

Planification

		Début	Durée	Fin	Avanc.
1	**Prise en main du sujet**	10/10/14	8	17/10/14	0%
1.1	Analyse	10/10/14	4	13/10/14	0%
1.2	Recherches	14/10/14	1	14/10/14	0%
1.3	Définition des objectifs	15/10/14	3	17/10/14	0%
2	**Étude du protocole MIP-DLSR**	18/10/14	17	03/11/14	0%
2.1	Lecture et analyse de la thèse MIP-DLSR	18/10/14	13	30/10/14	0%
2.2	Rédaction	31/10/14	4	03/11/14	0%
3	**Étude de la plateforme IoT-Lab**	04/11/14	20	28/11/14	0%
3.1	Analyse et premiers tests	04/11/14	20	28/11/14	0%
4	**Étude de la solution Raspberry Pi**	24/11/14	12	06/12/14	0%
4.1	Analyse et premiers tests	24/11/14	12	06/12/14	0%
5	**Portage**	06/12/14	14	19/12/14	0%
5.1	Portage et installation sur la plateforme IoT-Lab	06/12/14	14	19/12/14	0%
6	**Vacances**	20/12/14	14	02/01/15	100%
6.1		20/12/14	14	02/01/15	100%
7	**Expérimentations**	03/01/15	21	23/01/15	0%
7.1	Définition des scénarios	03/01/15	7	09/01/15	0%
7.2	Expérimentations	10/01/15	14	23/01/15	0%
8	**Rédaction de la thèse**	24/01/15	14	06/02/15	0%
8.1	Rédaction des résultats, analyses et conclusions	24/01/15	14	06/02/15	0%
9	**Gestion de projet**	10/10/14	120	06/02/15	0%
9.1	Réunions	10/10/14	120	06/02/15	0%
9.2	Fiches hebdomadaires	10/10/14	120	06/02/15	0%
9.3	Rapport	10/10/14	120	06/02/15	0%
9.4	Soutenances	01/12/14	123	06/02/15	0%

FIGURE E.1 – Diagramme de Gantt prévisionnel

FIGURE E.2 – Diagramme de Gantt final

Fiches de suivi

Fiche de suivi de la semaine 1
du 04 octobre 2014 au 10 octobre 2014

Temps de travail de Benjamin MOLLÉ: 2 h 30 m
Temps de travail de Denis SOURON: 2 h 30 m
Travail effectué.

- Prise en main du sujet
  - Analyse du sujet
  - Réunion avec B. Parrein.
  - Inscription sur IoT-Lab http://www.iot-lab.info

Échanges avec le commanditaire.

- Objectif du projet : Expérimentation / Mesurer les performances d'un protocole réseau afin de consolider le draft

Planification pour la semaine prochaine.

- Lecture : draft IETF, thèse Jiazy Yi (Etat de l'art sur les réseaux Adhoc)

Fiche de suivi de la semaine 2
du 11 octobre 2014 au 17 octobre 2014

Temps de travail de Benjamin MOLLÉ: 10 h 00 m
Temps de travail de Denis SOURON: 10 h 00 m

Cette semaine, nous nous sommes surtout penchés sur les documents fournis la semaine dernière par B. Parrein. Notre objectif étant de se familiariser avec la technologie MP-OLSR.

Grâce à cette lecture, nous avons extrait du vocabulaire technique autour de cette technologie que nous souhaitons étudier la semaine prochaine tout en commençant la rédaction du livrable.
Travail effectué.

- Lecture Article de Jaizi Yi
- Identification du vocabulaire à étudier

Planification pour la semaine prochaine.

- Recherche sur le vocabulaire clé trouvé
- Début de rédaction

Fiche de suivi de la semaine 3
du 18 octobre 2014 au 24 octobre 2014

Temps de travail de Benjamin MOLLÉ: 8 h 00 m
Temps de travail de Denis SOURON: 7 h 00 m

On a continué les lectures des documents fourni en plus des deux articles reçu dans la semaine par B. Parrein.

Pour information, nous avons tous les deux été accepté sur le site de IoT-Lab. Nous pouvons commencer a étudier ceci à la rentrée.
Travail effectué.

- Lecture Article (IEEE et VANETs)
- Début de rédaction
- Continuer la bibliographie

Planification pour la semaine prochaine.

- Continuer les recherches
- Étude IoT-Lab

Fiche de suivi de la semaine 4
du 25 octobre 2014 au 31 octobre 2014

Temps de travail de Benjamin MOLLÉ: 0 h 00 m
Temps de travail de Denis SOURON: 2 h 00 m
Estimation de retard : 1 semaine
Raison : Stage pour Benjamin + peu de travail effectué en semaine 44.

Travail effectué.

- Rédaction livrable

Travail non effectué.

- Fiche de lecture
- Rédaction de la bibliographie

Planification pour la semaine prochaine.

- Rédaction du livrable (Introduction, État de l'art)
- Documentation sur OLSR
- Étude IoT-Lab

Fiche de suivi de la semaine 5
du 1 novembre 2014 au 7 novembre 2014

Temps de travail de Benjamin MOLLÉ: 8 h 00 m
Temps de travail de Denis SOURON: 8 h 00 m

Travail effectué.

- Analyse des algorithmes de fonctionnement de MP-OLSR (extension de Dijkstra pour le réseau : chemins distincts, les sommets peuvent être utilisés plusieurs fois)
- Rédaction du livrable (Introduction)

Échanges avec le commanditaire.

- Échange par mail avec Jiazi Yi : Description du sujet, organisation du projet, intégration au sein du Draft MP-OLSR.

Planification pour la semaine prochaine.

- Rédaction du livrable (État de l'art)
- Détailler les fiches de lecture (trop succinctes)

Fiche de suivi de la semaine 6
du 8 novembre 2014 au 14 novembre 2014

Temps de travail de Benjamin MOLLÉ: 8 h 00 m
Temps de travail de Denis SOURON: 9 h 00 m

Cette semaine a surtout été concentrée autour de la rédaction de l'état de l'art. Principalement tourné autour de la thèse de Jiazi Yi. Nous comptons compléter l'état de l'art avec les quelques articles fourni et trouvé entre temps.

L'objectif à venir est l'expression des premières expérimentations possibles et d'étudier le fonctionnement d'IoT-Lab pour celle-ci.

Travail effectué.

- Réaction de l'étude d'une proposition pour l'état de l'art.

Planification pour la semaine prochaine.

- IoT-Lab, faire une réunion pour comprendre le fonctionnement
- Faire les premières propositions d'expérimentation

Fiche de suivi de la semaine 7
du 15 novembre 2014 au 21 novembre 2014

Temps de travail de Benjamin MOLLÉ: 8 h 00 m
Temps de travail de Denis SOURON: 8 h 00 m
Travail effectué.

- Réaction de l'état de l'art.
- Réunion avec Benoît Parrein pour faire le point sur le projet :
  - Pour le rapport intermédiaire
  - Pour la soutenance
  - Pour la suite du projet (spécifications, expérimentations menées, résultats attendus, ...)
- Premiers tests sur la plateforme IoT-Lab : échange de paquets Hello entre 2 capteurs (Réservation de 2 capteurs, configuration et mise en application avec un programme simple)

Planification pour la semaine prochaine.

- Finir l'état de l'art (Urgent)
- Echanges avec Benoît Parrein sur la conformité du rapport intermédiaire.
- Continuer les expérimentations pour comprendre comment faire fonctionner notre propre programme sur les nœuds. Trouver une topologie des nœuds (peut-être la demander).

Fiche de suivi de la semaine 8
du 22 novembre 2014 au 28 novembre 2014

Temps de travail de Benjamin MOLLÉ: 25 h 00 m
Temps de travail de Denis SOURON: 25 h 00 m

Cette semaine, nous nous sommes concentré particulièrement sur l'étude du fonctionne IoT-LAB afin de savoir si la plateforme allait pouvoir nous convenir. Il se trouve que les capteurs disponibles peuvent probablement faire fonctionner le protocole MP-OLSR mais il faudra pour cela adapter ce qui a déjà été développé pour l'adapter à l'architecture des capteurs A8/M3 fourni par IoT-LAB. L'utilisation d'un Testbed de Raspberry Pi semble être aussi une solution efficace. Le choix sera à faire la semaine prochaine avec Benoît Parrein.

Notre objectif le plus urgent est la préparation de la soutenance qui aura lieu le Mardi 2 Décembre à 8H00. Ensuite, nous espérons pouvoir choisir la plateforme de travail pour commencer la phase de développement.

Travail effectué.

- Réaction de l'état de l'art.
- Mise au point sur le rapport avec Benoît Parrein (Éclaircir certaines parties, corrections, ajouts d'informations, suppression de sections non pertinentes...)
- Étude IoT-LAB et autre solution envisageable.

Planification pour la semaine prochaine.

- Présenter l'intégralité de la première phase par le biais d'une soutenance.
- Choisir la plateforme de développement.
- Commencer la phase de développement.

Fiche de suivi de la semaine 9
du 29 novembre 2014 au 5 décembre 2014

Temps de travail de Benjamin MOLLÉ: 13 h 00 m
Temps de travail de Denis SOURON: 13 h 00 m

Cette semaine, notre travail a fortement été orienté autour de la soutenance intermédiaire, avec la préparation de celle-ci d'une part, en considérant les retours d'autre part. Nous avons pris connaissance des points à compléter dans le rapport, par exemple le fonctionnement de l'algorithme de Dijkstra utilisé par MP-OLSR.

Travail effectué.

- Préparer la soutenance intermédiaire (effectuée mardi 2 décembre)
- Réunion avec Benoît Parrein (Jeudi 4), suite du projet :
  - Étudier la version de OLSRd supportée actuellement par MP-OLSRd
  - Le protocole sera testé sur Raspberry Pi à travers interface ethernet.
  - Analyse du code MP-OLSRd
- Test sur OLSRd (0.5.6)
  - compilation et exécution sur linux (sur plusieurs postes en salle C007)

    => les machines dialoguent entre elles : échange de messages 'HELLO' et 'TC' (il manque la configuration des iptables pour permettre aux applications d'utiliser olsr)
  - compilation et exécution sur Raspberry Pi (olsrd semble fonctionnel sur un Raspberry Pi. Il manque cependant un test sur plusieurs raspberry pi pour savoir s'ils peuvent dialoguer avec olsrd)
- Demande de rendez-vous avec le LINA pour Jeudi 11 (en attente)

Planification pour la semaine prochaine.

- Compilation et exécution de MP-OLSRd (sur Linux et Raspberry Pi)
- Analyser le fonctionnement de OLSRd et MP-OLSRd avec iptables
- Scénarios de test (+Gantt)

Fiche de suivi de la semaine 10
du 6 décembre 2014 au 12 décembre 2014

Temps de travail de Benjamin MOLLÉ: 10 h 00 m
Temps de travail de Denis SOURON: 10 h 00 m

Nous sommes actuellement en plein étude du fonctionnement du programme MP-OLSR. Nous avons entamé l'installation sur des Raspberry Pi. Certains modules passent la compilation et semblent s'exécuter normalement comme la partie serveur et client de l'application. Malheureusement, nous rencontrons des erreurs à la compilation d'autres parties, qui concernent les règles iptables relatives à la gestion du routage avec MP-OLSR. Nous allons tester très prochainement (pendant la semaine 51) la compilation sur un autre système d'exploitation, backtrack,

qui est une distribution orientée sur le test de sécurité réseau et celle utilisée lors des expérimentations menées dans la thèse de Jiazi Yi.

Travail effectué.

- Analyse de l'implémentation de MP-OSLR (en cours)
- Compilation et exécution de MP-OSLR sur Rasberry Pi (en cours)
  - Compilation et lancement réussis pour certains parties
  - Échec pour les autres parties (Erreur de compilation en cours d'investigation), solution possible : utiliser un autre système d'exploitation

Travail non effectué.

- Réunion au LINA jeudi 11 (Indisponibilité)

Planification pour la semaine prochaine.

- Rendez-vous jeudi 18 décembre au LINA pour l'accès à la plateforme de Raspberry Pi
- Analyse du code MP-OLSR
- Compiler et exécuter tous les modules
- Scénarios de test (+ Gantt)

Fiche de suivi de la semaine 11
du 13 décembre 2014 au 19 décembre 2014

Temps de travail de Benjamin MOLLÉ: 24 h 00 m
Temps de travail de Denis SOURON: 20 h 00 m

Cette semaine, nous avons tourné nos efforts vers la compilation et l'exécution du programme implémentant MP-OLSR. Nous avons réussi après de nombreux essais sur différents noyaux et distributions Linux à compiler et exécuter le programme. Pour cela nous avons utilisé une distribution CentOS avec un noyau Linux non actuel (2.6.21). Malgré nos efforts, nous ne sommes pas encore parvenu à compiler sur un système récent. Cependant, l'intégration de ce programme au sein des nouveaux noyaux peut également faire partie du sujet.

Travail effectué.

- Compilation et première exécution sur CentOS 5.9 avec un noyau 2.6.21
- Mercredi 17 : Rendez-vous avec Jean-Yves Leblin au LINA pour l'accès aux Raspberry Pi.

Travail non effectué.

- Compilation sur des Raspberry Pi : L'installation de certaines librairies (Boost par exemple) est très longue sur ces équipements (au moins 4 heures)
- Compilation sur des noyaux récents : fichiers systèmes liés au réseau ont été modifiés lors du passage à la 3.x. Une partie du code est obsolète et renvoi des erreurs sur des fichiers noyaux.

Planification pour la semaine prochaine.

- Scénarios de test(+ Gantt)

Fiche de suivi de la semaine 12
du 20 décembre 2014 au 26 décembre 2014

Temps de travail de Benjamin MOLLÉ: 0 h 00 m
Temps de travail de Denis SOURON: 0 h 00 m

Stage pour Benjamin. Vacances.

Fiche de suivi de la semaine 13
du 27 décembre 2014 au 2 janvier 2014

Temps de travail de Benjamin MOLLÉ: 4 h 00 m
Temps de travail de Denis SOURON: 0 h 00 m

Vacances.

Tentative de mise à jour de MP-OLSR. Problème pour mise à jour, beaucoup de fichiers du noyau sont différent. Les variables utilisées ont changées. Nous espérons avoir rapidement un document promis par des personnes que Jiazi a contacté.

Travail effectué.

- Début de mise à jour de MP-OLSR.

Planification pour la semaine prochaine.

- Continuer mise à jour du programme.
- Gantt

Fiche de suivi de la semaine 14
du 3 janvier 2014 au 9 janvier 2014

Temps de travail de Benjamin MOLLÉ: 13 h 00 m
Temps de travail de Denis SOURON: 15 h 00 m

Cette semaine, lors d'une réunion Jeudi, nous avons pu définir les objectifs et les dates pour le restant du projet. D'autre part, nous avons prévu de ne pas mettre à jour le logiciel pour les noyaux modernes. L'objectif est de créer un nœud Raspberry Pi avec le noyau Linux 2.6.21.5 afin de faire fonctionner le programme d'origine pour la fin de la semaine prochaine. La principale entrave à cette migration est le décalage chronologique entre le noyau (2.6.x dont les mises à jour se sont arrêtées courant 2011 pour passer à la version 3.x) et le matériel (datant de 2012 avec une architecture ARM et le premier Debian implémenté utilise un noyau 3.x) Ensuite, nous déploierons la configuration sur la grappe de Raspberry Pi du LINA la semaine suivante. Nous avons reçu la documentation pour pouvoir utiliser la plateforme. Nous avons vérifié que nous pouvions y accéder depuis Polytech.

Enfin sur les deux dernières semaine de Janvier. Nous ferrons les expérimentations et leurs analyse.

Travail effectué.

- Planning pour la fin de projet.
- Début de création d'un nœud Raspberry Pi avec le noyau requis

– Besoin de ressources compatibles avec une architecture ARM

Planification pour la semaine prochaine.

- Continuer la création du Raspberry Pi.

Fiche de suivi de la semaine 15
du 10 janvier 2015 au 16 janvier 2015

Temps de travail de Benjamin MOLLÉ: 12 h 00 m
Temps de travail de Denis SOURON: 11 h 00 m

Nous avons passé la majorité du temps de cette semaine à essayer de changer le noyau d'un Raspberry Pi comme nous l'avons planifié la semaine dernière. Nous n'avons pas abouti en suivant cette piste. Cependant, l'organisation de la fin du projet a été revu car nous avons de nouveaux éléments. En effet, une équipe travaillant sur mp-olsr nous a transmis des documents sur l'utilisation de l'implémentation de mp-olsr sur un noyau plus récent (3.2). Nous sommes plus confiants concernant le fonctionnement du programme car cette version de noyau est compatible avec le Raspberry Pi.

Travail effectué.

- Changement de noyau Linux sur des nœuds Raspberry Pi (migration 3.18 vers 2.6) : échec, le matériel ne semble pas être adapté et la compilation se conclue toujours par des erreurs.

Planification pour la semaine prochaine.

- Installer et tester mp-olsr sur un Raspberry Pi doté d'un noyau Linux 3.2
- Installer mp-olsr sur le noyau Linux 3.18 actuel et fournir un manuel d'installation à jour si ce processus abouti.

Fiche de suivi de la semaine 16
du 17 janvier 2015 au 23 janvier 2015

Temps de travail de Benjamin MOLLÉ: 19 h 00 m
Temps de travail de Denis SOURON: 22 h 00 m

Durant cette semaine nous nous sommes concentrés sur la compilation et l'installation de mpolsr sur des Raspberry Pi à l'aide des documents qui nous ont été fournis par une équipe qui a travaillé sur un noyau récent. Nous avons pu procéder aux mêmes changements et installer le logiciel. L'installation des pré-requis ainsi que celle du logiciel sont très couteuses en temps. L'exo-compilation est une solution à considérer sérieusement sur Raspberry Pi.

Travail effectué.

- Installation de mpolsr sur Raspberry Pi
  - Changement vers un noyau 3.2 (3.2.0-4-rpi)
  - Installation des pré-requis (Boost 1.38, Libnfnetlink et Libnetfilter_queue)
  - Configuration de mpolsr (mise à jour des fichiers du programme, modification de fichiers systèmes, etc...)
  - Installation des différents modules (iptables, olsrd, ...)
- Exécution de mpolsr sur Raspberry Pi
  - Echec : Lors du lancement du démon olsrd, le chargement du plugin sereadmo créé pour mpolsr se conclut par une erreur.

Planification pour la semaine prochaine.

- Analyser et corriger le plugin sereadmo qui entrave l'exécution de mpolsr.

Fiche de suivi de la semaine 17
du 24 janvier 2015 au 30 janvier 2015

Temps de travail de Benjamin MOLLÉ: 30 h 00 m
Temps de travail de Denis SOURON: 27 h 00 m

Cette semaine était réservée au projet, ce qui explique le nombre d'heure et le travail accompli. Avec l'aide de Ricardo Macedo, ayant travaillé sur l'installation de mpolsr sur Ubuntu 12.04, nous avons pu arrivé à installer sur des Raspberry Pi. Ricardo avait réécrit à la main un script pour compléter la compilation, de notre côté ce script ou un fichier équivalent était absent du code source. Après avoir installer mpolsr sur un deuxième Raspberry Pi, nous avons pu tester rapidement le fonctionnement mpolsr. Dans la deuxième partie de la semaine, nous avons fait un point sur l'avancement du projet en abordant également les phases de rendu et de soutenance. Nous aurons l'occasion pendant la semaine prochaine de rencontrer Ricardo Macedo et Jiazi Yi dans un atelier réservé : avec présentation en anglais, démonstration et échanges.

Travail effectué.

- Compilation du module sereadmo.
- Installation de mpolsr sur Raspberry Pi (noyau 3.2).
- Test d'exécution sur 2 Rasperry Pi connectés en direct :
  - Les 2 nœuds se voient et dialoguent entre eux.
  - Les 2 nœuds mettent fréquemment à jour l'état du lien entre eux.
  - Les 2 nœuds détectent qu'ils sont voisins directs.
  - Les 2 nœuds définissent la liaison comme symétrique.
  - Test de transmission avec NetCat configuré en client sur une machine et serveur sur l'autre.
- Envoi des fichiers sources et de la procédure d'installation sur la plateforme d'expérimentation (au LINA).
- Rédaction du livrable.
- Mise au point avec Benoît Parrein sur le projet.

Planification pour la semaine prochaine.

- Envoyer une première version du rapport pour mardi 3 février.
- Préparer le workshop du vendredi 6 février
  - Présentation orale en anglais
  - Présence d'intervenants ayant créé, contribué ou mis en place mpolsr.
- Mener une expérimentation avec 4 à 6 nœuds sur la plateforme du LINA
- Mesurer les résultats obtenus : Packet Loss / Delay / Jitter

Fiche de suivi de la semaine 18
du 31 janvier 2015 au 6 février 2015

Temps de travail de Benjamin MOLLÉ: 15 h 00 m
Temps de travail de Denis SOURON: 15 h 00 m

Durant cette semaine nous n'avons pas eu beaucoup d'occasions pour avancer la réalisation. Malheureusement, la mise en place de MP-OLSR sur la plateforme du LINA est un peu longue et nous manquons de temps. Nous avons une date de rendu à la fin de cette semaine, c'est pourquoi nous nous sommes concentré sur la rédaction du rapport et d'un document pour MP-OLSR. A l'occasion du Mojette Day, nous avons eu la visite de Jiazi et Ricardo. Nous avons pu échanger sur leurs travaux concernant MP-OLSR, et nous avons fait une présentation orale de notre projet. Puis nous avons enchainé avec une courte démonstration. Cette journée vient appuyer un élément important dans notre projet qui est la communication et les interactions avec les personnes venant de plusieurs origines.

Travail effectué.

- Rédaction du rapport
- Workshop avec Benoît, Jiazi et Ricardo
  - Présentation
  - Démonstration

Travail non effectué.

- Pas d'expérimentation au LINA (Plateforme non déployée)

Planification pour la semaine prochaine.

- Soutenance de fin de projet
- Document d'installation de MP-OLSR à traduire en anglais

Le tableau F.1 récapitule le taux d'avancement du projet. Rappelons que le temps de travail théorique *minimal* correspond au temps indiqué sur la maquette pédagogique auquel on ajoute un strict minimum de 20 % correspondant au travail personnel hors emploi du temps. La partie « haute » de la fourchette correspond à 50 % de temps supplémentaire au titre du travail personnel.

	Temps prévu		Benjamin MOLLÉ			Denis SOURON		
	bas	haut	hebdo.	Σ	%	hebdo.	Σ	%
Semaine	h : m	h : m	h : m	h : m		h : m	h : m	
1	10 : 00	12 : 30	2 : 30	2 : 30	25 (20)	2 : 30	2 : 30	25 (20)
2	20 : 00	25 : 00	10 : 00	12 : 30	62 (50)	10 : 00	12 : 30	62 (50)
3	30 : 00	37 : 30	8 : 00	20 : 30	68 (54)	7 : 00	19 : 30	65 (52)
4	40 : 00	50 : 00	0 : 00	20 : 30	51 (41)	2 : 00	21 : 30	53 (43)
5	50 : 00	62 : 30	8 : 00	28 : 30	57 (45)	8 : 00	29 : 30	59 (47)
6	60 : 00	75 : 00	8 : 00	36 : 30	60 (48)	9 : 00	38 : 30	64 (51)
7	70 : 00	87 : 30	8 : 00	44 : 30	63 (50)	8 : 00	46 : 30	66 (53)
8	80 : 00	100 : 00	25 : 00	69 : 30	86 (69)	25 : 00	71 : 30	89 (71)
9	90 : 00	112 : 30	13 : 00	82 : 30	91 (73)	13 : 00	84 : 30	93 (75)
10	100 : 00	125 : 00	10 : 00	92 : 30	92 (74)	10 : 00	94 : 30	94 (75)
11	110 : 00	137 : 30	24 : 00	116 : 30	105 (84)	20 : 00	114 : 30	104 (83)
12	120 : 00	150 : 00	0 : 00	116 : 30	97 (77)	0 : 00	114 : 30	95 (76)
13	130 : 00	162 : 30	4 : 00	120 : 30	92 (74)	0 : 00	114 : 30	88 (70)
14	140 : 00	175 : 00	13 : 00	133 : 30	95 (76)	15 : 00	129 : 30	92 (74)
15	150 : 00	187 : 30	12 : 00	145 : 30	97 (77)	11 : 00	140 : 30	93 (74)
16	160 : 00	200 : 00	19 : 00	164 : 30	102 (82)	22 : 00	162 : 30	101 (81)
17	170 : 00	212 : 30	30 : 00	194 : 30	114 (91)	27 : 00	189 : 30	111 (89)
18	180 : 00	225 : 00	15 : 00	209 : 30	116 (93)	15 : 00	204 : 30	113 (90)

TABLE F.1 – Avancement du projet par rapport au temps de travail théorique minimal (respectivement haut)

Auto-contrôle et auto-évaluation

Grille d'évaluation du Projet de recherche (et développement)

	Prénom	Nom	Année
Binome 1	Benjamin	Mollé	2014-2015
Binome 2	Denis	Souron	**Date**
			08-janv
Jury 1			
Jury 2			
Jury 3			
Jury 4			

Notation	
A	Maîtrise dans l'application du savoir-faire re
B	Application du savoir-faire requis
C	Insuffisances, lacunes à corriger dans l'appl
D	Insuffisances flagrantes, inacceptables, voir

				A	B	C	D	Remarque / Note / Commentaire
Rapport	Organisation	Plan	Equilibre	x				
			Cohérence	x				
		Fluidité	Introductions (partielles)	x				
			Transitions	x				
			Conclusions (partielles)		x			
		Tableaux, figures	Numérotés	x				
			Légendés	x				
			Référencés (non "en ligne")		x			
	Rédaction	Orthographe	Coquilles	x				
			Fautes évitables	x				
			Franglais, jargon		x			
		Rédaction	Aisée		x			
			Absence de plagiat !	x				
	Bibliographie	Références	Suffisantes (nombre, intérêt)	x				
			Pérennes		x			
			Complètes (auteurs, pages..)	x				
			Conséquentes (volume)			x		
		Références dans le texte		x				

Proposition de note haute	17,66
Proposition de note basse	12,66

Proposition de note du jury	**14**

Projection	Organisation	Plan		x				
		Liaisons		x				
		Numérotation		x				
	Contenu	Informatif		x				
		Concis		x				
		Clair		x				
		Orthographe		x				
		Illustrations		x				
Oral	Présentation	Aisance		x				
		Tenue		x				
		Articulation, compréhension		x				
	Durée	Respect		x				
		Temps de parole équilibré		x				
	Réponses	Pertinence		x				
		Argumentation		x				

Proposition de note haute	19,26
Proposition de note basse	14,44

Proposition de note du jury	**16**

Travail	Etude	Bibliographie	Adéquate	x				
			Suffisante		x			
		Cahier des charges	Clair			x		
			Formalisé			x		
		Hypothèses envisagées	Nombre		x			
			Pertinence	x				
			Analyse *a priori*		x			
		Validation	Tableau comparatif			x		
			Choix argumenté(s)	x				
			Faisabilité	x				
	Complexité	Temps consacré		x				
		Résultats obtenus			x			
		Difficulté	Intrinsèque	x				
			Vis-à-vis du binôme		x			

	Annexes	Fiches d'avancement	Régularité	x					
			Détaillées	x					
		Gantt	Prévisionnel et justifications				x		
			Effectifs, erreurs, corrections				x		

Proposition note haute	15,45
Proposition note basse	10,45

Proposition de note du jury	**13,5**

PROPOSITION DE NOTE (I) **14,5**