Université de Mons
Faculté des Sciences
Institut d'Informatique
Service de Réseaux et Télécommunications

Routeur de bordure 6LoWPAN sous Linux

Directeur : Pr. Bruno QUOITIN

Rapporteurs : Dr. Olivier DELGRANGE
M. Mathieu MICHEL

Projet réalisé par
David HAUWEELE

en vue de l'obtention du grade de
Master en Sciences Informatiques

Année académique 2012-2013

Table des matières

Liste des acronymes

ABI	Application Binary Interface
ALOHA	Additive Link Off Hawaiian Access
APL	ApPLicative layer
APO	APplication Object
APS	APplication Sub layer
ARP	Address Resolution Protocol
ASK	Amplitude Shift Keying
BPSK	Binary Phase Shift Keying
BSD	Berkeley Software Distribution
CCA	Clear Channel Assessment
CFQ	Completely Fair Queuing
CRC	Cyclic Redundancy Code
CPU	Central Processing Unit
CS	Chip Select
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
ED	Energy Detection
EUI	Extended Unique Identifier
FFD	Full-Function Device
FUSE	Filesystem in USErspace
GCC	GNU Compiler Collection
GFSK	Gaussian Frequency Shift Keying
GNU	GNU is Not Unix
GPL	General Public License
GPIO	General Purpose Input Output
GTS	Guaranteed Time Slot
HC	Header Compression
HCWSN	Health Care Wireless Sensor Network
HDMI	High Definition Multimedia Interface
HIRD	Hurd of Interfaces Representing Depth
HURD	Hird of Unix-Replacing Daemons
ISA	International Society of Automation
IID	Interface Identifier

I^2C	Inter Integrated Circuit
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
ISR	Interrupt Service Routine
LoWPAN	Low Power Wireless Area Network
LQI	Link Quality Indicator
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Medium Access Control
MBR	Master Boot Record
MCPS-SAP	MAC Common Part Sublayer Service Access Point
MIPS	Microprocessor without Interlocked Pipeline Stages
MLD	Multicast Listener Discovery
MLME	MAC subLayer Management Entity
MLME-SAP	MAC subLayer Management Entity Service Access Point
MPSK	Multiple Phase Shift Keying
MTU	Maximum Transmission Unit
NA	Neighbor Advertisement
ND	Neighbor Discovery
NFS	Network File System
NH	Next-Hop
NS	Neighbor Solicitation
NWK	NetWorK layer
O-QPSK	Orthogonal Quadrature Phase-Shift Keying
OS	Operating System
PAN	Personal Area Network
PD-SAP	PHY Data Service Access Point
PHY	Physical
PLME-SAP	Physical Layer Management Entity Service Access Point
PMTUD	Path MTU Discovery
PPDU	Physical Protocol Data Unit
POSIX	Portable Operating System Interface for uniX
PRU	Programmable Realtime Unit
RAM	Random Access Memory
RFC	Request For Comments
RFD	Reduced-Function Device
RPL	Routing Protocol for Low power and lossy network
RA	Router Advertisement
RS	Router Solicitation
RTT	Round-Trip Time

SFTP Secure File Transfer Protocol
SLAAC StateLess Address AutoConfiguration
SPI Serial Peripheral Interface
SoC System on Chip
SOHO Small Office, Home Office
SSH Secure SHell
SSHFS Secure SHell File System
TCP Transmission Control Protocol
TDMA Time Division Multiple Access
UART Universal Asynchronous Receiver Transmitter
UDP User Datagram Protocol
U/L Universal/Local
USB Universal Serial Bus
WPAN Wireless Personal Area Network
WSN Wireless Sensor Network
ZDO Zigbee Device Object

Introduction

L'utilisation de réseaux de capteurs sans fil s'intensifie dans des domaines de plus en plus variés. Des standards comme IEEE 802.15.4 et 6LoWPAN facilitent le déploiement de ces réseaux de capteurs. Le standard IEEE 802.15.4 définit la couche physique (PHY) et la couche d'accès au support (MAC) pour les réseaux de capteurs sans fil. Le standard 6LoWPAN permet le transport de paquets IPv6 au dessus de IEEE 802.15.4. En particulier l'utilisation d'un routeur de bordure 6LoWPAN facilite le rattachement de ces réseaux de capteurs sans fil au réseau Internet IPv6.

Ce projet consiste en la création d'un routeur de bordure basé sur une platforme ARM fonctionnant sous Linux. Ce projet se distingue des autres solutions existantes et n'utilise pas une solution en espace utilisateur mais s'appuie sur les couches MAC IEEE 802.15.4 et 6LoWPAN intégrées dans le noyau Linux.

Ce rapport se divise en quatre chapitres. Le premier chapitre introduit le contexte nécessaire à la compréhension du projet. Le second chapitre présente un état de l'art des solutions de routeur de bordure 6LoWPAN. Le troisième chapitre présente l'implémentation du projet. Enfin le quatrième chapitre présente les tests effectués pour valider cette implémentation.

Chapitre 1

Internet des objets

1.1 Internet des objets

L'Internet of Things [43] (IoT) est un concept qui vise à intégrer les objets physiques à Internet via un système d'adressage standard permettant de les identifier de manière unique. Un objet intelligent est typiquement équipé de senseurs qui permettent de collecter de l'information sur l'environnement et d'actuateurs qui permettent de manipuler l'environnement.

La très grande diversité d'applications de ces objets intelligents amène à utiliser de nombreux protocoles différents. Les différences entre ces protocoles posent des problèmes d'interopérabilité lorsque ces objets intelligents sont amenés à coopérer et à communiquer entre eux. Pour pallier à ces problèmes, l'utilisation d'Internet Protocol (IP) normalise la communication et permet aux applications de se reposer sur une panoplie de protocoles déjà existants. En particulier IPv6 [31] et son très grand espace d'adressage identifie de manière unique les objets intelligents dans le réseau et permet ainsi de les intégrer directement à Internet sans dépendre de protocoles spécialisés.

Les réseaux de capteurs sans fil – Wireless Sensor Network [30, 41, 24] (WSN) constituent une sous-classe parmis ces réseaux d'objets intelligents. Ceux-ci sont constitués de capteurs de petite taille, peu couteux, munis d'une interface radio, couplés à un microcontrôleur basse consommation qui permet un traitement limité. Akyildiz et al. [24] explorent des applications rendues possibles par ces réseaux de capteurs dans le domaine militaire, environnemental, médical, domotique et commercial. Plus récemment Egbogah et Fapojuwo [36] recensent différents projets de réseaux de capteurs sans fil dans le domaine des soins de santé – Health Care Wireless Sensor Network (HCWSN).

1.2 IEEE 802.15.4

Le standard IEEE 802.15.4 [2] est un protocole de communication pour des réseaux sans fils personnels à très basse consommation – Low-rate Wireless Personal Area Network (LR-WPAN). En particulier ce standard définit la couche physique (PHY) et de contrôle d'accès au support (MAC). Ce standard a pour objectif de permettre la réalisation de transceivers à très bas

coûts, optimisés pour leur facilité d'installation, une longue autonomie et assurant le transport de données à des débits maximum de 250 kbit/s. Ces caractéristiques rendent son utilisation très populaire dans les systèmes embarqués, en particulier pour les capteurs sans fil.

Ces réseaux de capteur sans fil s'organisent sous la forme d'un réseau WPAN (Wireless Personal Area Network). Ceux-ci forment un sous-ensemble des réseaux personnels – Personal Area Network (PAN) qui assurent la mise en réseau de périphériques et des objets intelligents.

Deux types de noeuds sont définis par le standard : les noeuds entièrement fonctionnels – Full-Function Device (FFD) et les noeuds à fonctionnalités réduites – Reduced-Function Device (RFD). Les FFD sont capables de relayer des messages et peuvent servir de coordinateur au sein du réseau. Les RFD sont des noeuds simples avec des ressources limitées et sont associés à un unique FFD. Dans le réseau WPAN un noeud FFD agit comme coordinateur principal du réseau, le coordinateur PAN.

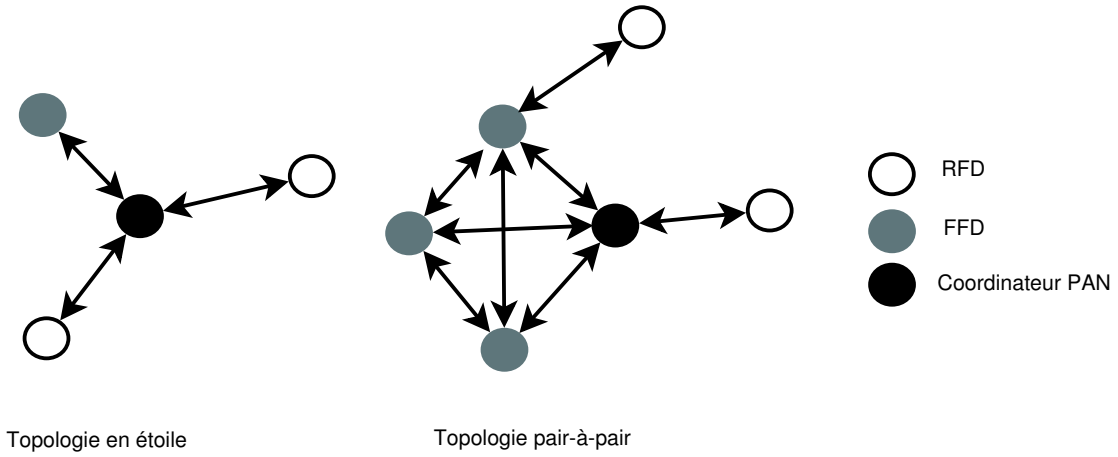


FIGURE 1.1 – Topologie en étoile et pair-à-pair dans un réseau IEEE 802.15.4 [2].

Les noeuds peuvent s'organiser en deux types de topologies différentes illustrées à la Figure 1.1. Dans une topologie en étoile toutes les communications passent par le coordinateur PAN. Dans une topologie pair-à-pair, les noeuds peuvent communiquer avec tous les noeuds FFD à portée radio. Ceux-ci relayent l'information depuis un coordinateur PAN désigné au démarrage du réseau.

1.2.1 Couche physique

La couche physique (PHY) assure la transmission des unités de données de protocole physique – Physical Protocol Data Unit (PPDU) sur le canal radio. Cette couche assure également l'activation et la désactivation du transceiver, la sélection du canal, la mesure de la qualité du lien – Link Quality Indicator (LQI), la détection d'énergie – Energy Detection (ED) et la détection d'utilisation du canal – Clear Channel Assessment (CCA).

La spécification de la couche physique définit en plusieurs bandes de fréquence, techniques de modulation et débits de transmissions supportés. Ces différentes techniques sont résumées dans la Table 1.1. Chaque bande de fréquence est divisée en une série de canaux

Fréquence (MHz)	**Modulation**	**Débit (kbit/s)**	**Canaux**
779 – 787	O-QPSK	250	0 – 7
	MPSK		
868 – 868.6 / 902 – 928	BPSK	20	0 – 10
		40	
	ASK	250	
	O-QPSK	100	
		250	
950 – 956	GFSK	20	0 – 21
	BPSK		
2400 - 2483.5	O-QPSK	250	11 – 26

TABLE 1.1 – Bande de fréquence, technique de modulation et débits supportés.

identifiés par un nombre. Certaines fréquences ne sont pas autorisées dans certaines régions du monde.

1.2.2 Couche MAC

La couche de contrôle d'accès au support (MAC) assure l'association et la désassociation au PAN, le partage du canal avec CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance) ou ALOHA (Additive Link Off Hawaiian Access) [2, Section 4.5.4], la validation des trames, l'acquittement des trames, la gestion des intervalles de temps garantis – Guaranteed Time Slot (GTS), l'envoi des balises de synchronisation (beacon) et la sécurité sur la couche liaison de données.

Le système d'adressage utilisé par la couche MAC repose sur des adresses étendues et des adresses courtes. Les adresses étendues sont dérivées d'un identifiant global – Extended Unique Identifier [11] (EUI-64) et ont une taille de 64 bits. Les adresses courtes sont allouées par le coordinateur PAN lors de l'association et ont une taille de 16 bits.

Le standard permet un mode balisé et un mode non balisé. Dans le mode non balisé, les noeuds transmettent en utilisant CSMA/CA et doivent continuellement écouter le canal pour assurer la réception des trames. Dans le mode balisé des trames balises sont envoyées par un coordinateur et délimitent une supertrame. Ce mode permet aux noeuds de synchroniser leurs réveils avec les balises et donc d'économiser de l'énergie. Cette supertrame est divisée en 16 intervalles de temps égaux durant lesquels les noeuds peuvent transmettre soit en concurrence avec CSMA/CA ou ALOHA, soit seuls avec un intervalle de temps garanti.

Taille (octets)	2	1	0 – 20	0 / 14	0 – n	2
Champ	Contrôle	Séquence	Adresses	Sécurité	Données	CRC

FIGURE 1.2 – Forme générale d'une trame IEEE 802.15.4.

Il existe quatre types différents de trames transmises par la couche MAC :

— La trame de *balise* (BEACON) est utilisée pour délimiter la supertrame dans le mode balisé, signaler les messages en attentes et annoncer le PAN sur le canal.
— La trame d'*acquittement* (ACK) qui confirme la réception d'une trame.
— La trame de *commande* (COMMAND) utilisée pour contrôler le réseau.
— La trame de *données* (DATA) utilisée pour tous les transferts de données.

La forme générale d'une trame est illustrée à la Figure 1.2. La taille maximum des trames est de 127 octets. La taille de l'en-tête est variable et certains champs de l'en-tête peuvent être encodés sous une forme plus compacte grâce au champ **contrôle**. Ce champ contrôle spécifie le type de trame, la compression du champ d'adresses, la demande d'un acquittement et la sécurisation du lien. Dans le cas d'une demande d'acquittement, la réception de la trame est confirmée par une trame d'acquittement contenant la valeur du champ **séquence** du message d'origine. Si l'acquittement n'est pas reçu avant un certain délai ou si le numéro de séquence contenu dans la trame d'acquittement ne correspond pas, la trame d'origine est retransmise. Le champ **adresses** peut contenir les adresses MAC sources, destinations et les identifiants du PAN source et destination. Un champ supplémentaire spécifie les paramètres de sécurité. Finalement, un champ **CRC** (Cyclic Redundancy Code) portant sur l'en-tête et les données permet de controler l'intégrité de la trame. Dans le cas d'une trame de données, le type d'adresses utilisées et les paramètres de sécurité font varier la charge utile de la trame de 88 à 118 octets.

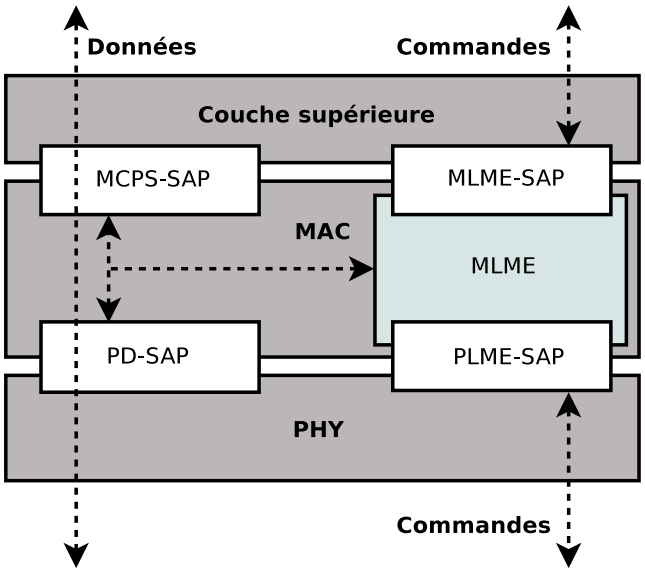


FIGURE 1.3 – Agencement des composants et interfaces de la couche MAC.

1.2.3 Interface de la couche MAC

La Figure 1.3 illustre l'agencement des composants et interfaces de la couche MAC. Celle-ci comprend une entité de gestion appellée MAC subLayer Management Entity (MLME). Cette entité est une abstraction du fonctionnement interne de la couche MAC. Deux services fournissent un ensemble de primitives qui permettent l'interfaçage avec la couche supérieure. Premièrement le MAC subLayer Management Entity Service Access Point (MLME-SAP) assure l'envoi de commandes de gestion vers et depuis le MLME. Deuxièmement le MAC Common Part Sublayer Service Access Point (MCPS-SAP) assure le transport des données. Deux services similaires assurent l'interfaçage de la couche MAC avec la couche PHY. Premièrement le Physical Layer Management Entity Service Access Point (PLME-SAP) qui assure l'envoi de commandes vers et réception depuis la couche physique. Deuxièmement le PHY Data Service Access Point (PD-SAP) assure le transport des données.

1.3 ZigBee

ZigBee [23, 26, 1] est un ensemble de standards spécifié par l'alliance ZigBee [19]. Ces standards définissent, au dessus du standard IEEE 802.15.4, une couche réseau – NetWorK layer (NWK) responsable du routage multisaut et une couche applicative – APpLicative layer (APL) sous forme d'un framework dédié au développement d'applications distribuées sur réseaux de capteurs sans fil. La Figure 1.4 fournit un aperçu de la pile ZigBee.

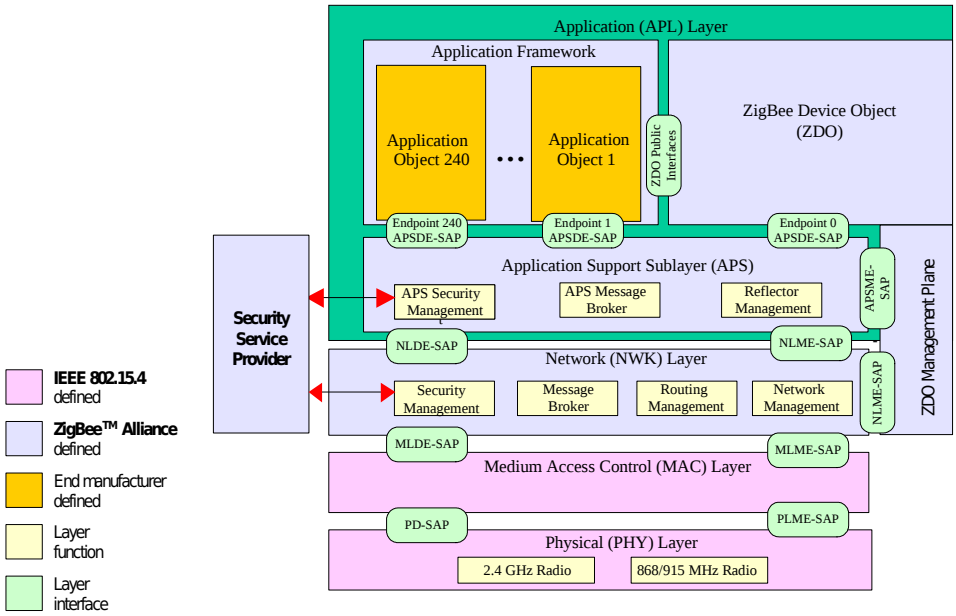


FIGURE 1.4 – Aperçu de l'architecture de la pile ZigBee [1].

La couche APL comprend deux sous-couches. La première est composée des ZigBee Device Objects (ZDO) et de l'Application Framework. La première couche met en place des

mécanismes de contrôle au dessus de la couche réseau. En particulier ces mécanismes permettent la découverte de nouveaux noeuds et la découverte des services offerts par ces noeuds. La seconde couche est composée de l'Application Sub Layer (APS) qui constitue une interface entre les dispositifs de sécurité, la couche NWK et la couche APL. Une application ZigBee est décomposée dans l'Application Framework en Application Objects (APO) qui exploitent les services fournies par le ZDO et agissent comme une application distribuée sur le réseau entier.

ZigBee est un standard populaire dans le domaine des réseaux de capteurs sans fils. En particulier le framework permet le développement rapide d'applications optimisées pour les réseaux de capteur sans fil. Les spécifications du standard sont disponibles gratuitement pour toute utilisation non commerciale. Toutefois l'utilisation de la marque ZigBee, le processus de certification et la participation au développement des spécifications du standard requiert l'adhésion à l'alliance ZigBee pour un côut minimum de 3500$ [20] par an.

1.4 WirelessHART

WirelessHART [53, 12] est un standard de communication sans fil conçu pour des applications industrielles avec des contraintes en temps et sécurité. Il se base sur le standard IEEE 802.15.4 avec une couche MAC modifiée auquel il ajoute les fonctionnalités de saut de canal, l'utilisation de TDMA (Time Division Multiple Access) et la censure de canaux pour éviter les collisions. La censure de canaux [35] évite l'utilisation de certains canaux lorsque des inteférences amoindrissent la qualité des communications sur ceux-ci. Le standard définit également ses propres couches réseau, transport et applicative. Ces mécanismes supplémentaires limitent les interférences et permettent aux réseaux de coexister sur la même bande de fréquence en respectant des contraintes temps réel. Lennvall, Svensson et Hekland [40] ont réalisé une comparaison de l'utilisation de ZigBee et WirelessHART en milieu industriel. Il ressort que contrairement à WirelessHART, ZigBee montre des faiblesses en milieu industriel. En particulier à cause de son manque de robustesse face aux interférences dans des environnements riches en métaux ainsi que le manque de diversité des chemins dans la couche réseau.

1.5 ISA 100.11a

ISA 100.11a est un standard de communication sans fil développé par l'International Society of Automation [13] (ISA) orienté vers des applications industrielles à contraintes de temps et sécurité. Il utilise le standard IEEE 802.15.4 avec une couche MAC modifiée qui supporte TDMA et le saut de canal. Les couches réseau et transport sont basées sur des standards ouverts : 6LoWPAN [44] et UDP. Petersen et Carlsen [47] ont réalisé une comparaison entre WirelessHART et ISA 100.11a. Il en ressort que les deux standards sont capables d'assurer des communications robustes dans des environnements industriels rigoureux. Toutefois des études plus approfondies doivent être menées pour déterminer comment les différences entre les deux standards influencent leur performance en milieu industriel.

1.6 IPv6

L'Internet Protocol version 6 [31] (IPv6) est le successeur de IPv4. Il a été développé dans le but de remédier à certains problèmes de conception dans IPv4. En particulier la facilité d'installation, la gestion de la fragmentation, les champs optionnels, les adresses multicast, la simplification de l'en-tête IP et le problème de pénurie des adresses.

Le problème de pénurie des adresses est un problème anticipé de longue date. En effet les adresses IPv4 sont codées sur un espace de 32 bits ce qui limite le nombre d'adresses disponible pour IP à 4 milliards d'adresses. Plusieurs facteurs comme l'arrivée sur Internet d'appareils mobiles, la démocratisation des connections à large bande et l'arrivée sur Internet de pays comme la Chine et l'Inde ont accéléré cette pénurie. Afin de remédier à ce problème IPv6 utilise des adresses codées sur 128 bits ce qui permet de l'ordre de 10^{38} adresses disponibles.

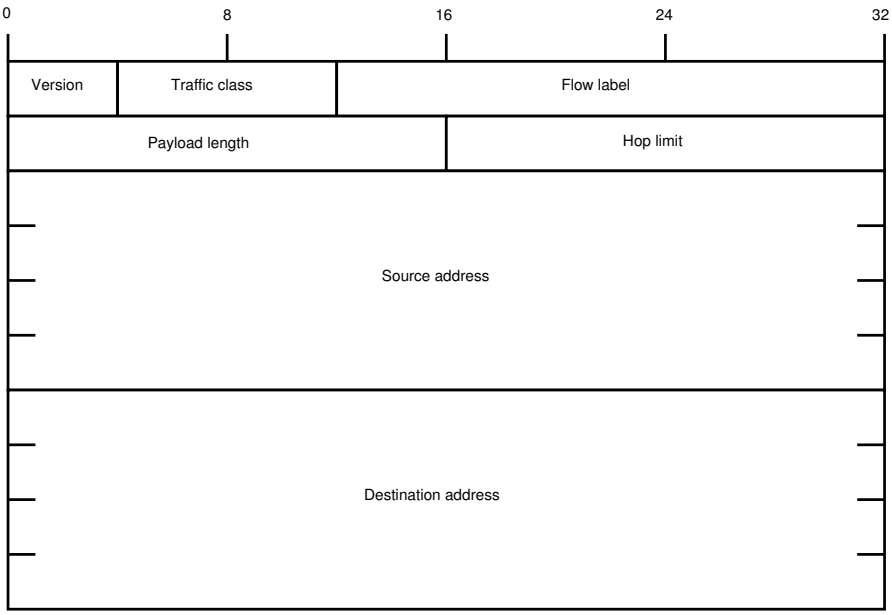


FIGURE 1.5 – En-tête d'un paquet IPv6.

L'en-tête d'un paquet IPv6 est illustrée à la Figure 1.5. Le champ **Traffic class** identifie la classe de service à appliquer au paquet. Le champ **Flow label** permet de marquer les paquets pour des traitements spéciaux par les routeurs. Le champ **Payload length** contient la taille de la charge utile du paquet ainsi que les extensions d'en-tête. Le champ **Next header** identifie l'en-tête qui suit l'en-tête du paquet IPv6. Ce champ peut indiquer la présence d'extensions d'en-tête ou le protocole de la couche supérieure. Le champ **Hop limit** est l'équivalent du champ Time To Live (TTL) de IPv4. La valeur de ce champ est décrémentée à chaque noeud où le paquet est forwardé et le paquet est jeté quand cette valeur tombe à zéro. Les champs **Source address** et **Destination address** spécifient l'adresse de la source et l'adresse de la destination du paquet.

Il est important de noter qu'à la différence de l'en-tête du paquet IPv4, il n'y a plus de champ en rapport avec la fragmentation. En effet, avec IPv4 les routeurs peuvent avoir à fragmenter les paquets lors du passage dans un lien avec une unité de transfert maximale –

Maximum Transmission Unit (MTU) plus faible. Dans le cas d'un routeur IPv6, le paquet n'est pas fragmenté au niveau du routeur mais le paquet est jeté et un message ICMPv6 Packet Too Big est envoyé à la source. La source fragmente alors les paquets en rajoutant une extension d'en-tête de fragmentation. Les paquets sont alors retransmis et le processus est réitéré jusqu'à ce que les paquets puissent passer par tous les routeurs sans nécessité de fragmentation. Ce processus est appelé Path MTU Discovery (PMTUD). De cette manière le travail de fragmentation est délégué à la source ce qui libère les routeurs d'un travail couteux.

Afin de réduire le nombre de fragmentations nécessaires lors du PMTUD. Le standard IPv6 spécifie un MTU minimum de 1280 octets. De cette manière les hôtes peuvent s'assurer qu'il n'y aura pas de fragmentation nécessaire pour autant que la taille du paquet émis soit inférieure ou égale à 1280 octets.

1.6.1 Adressage

L'utilisation d'adresses de 128 bits constitue le plus important changement dans IPv6. Afin de représenter ces adresses une nouvelle notation est utilisée. La notation d'une adresse IPv6 repose sur 8 valeurs hexadécimales de 16 bits séparées par des doubles points. De plus deux doubles points peuvent être utilisés pour remplacer une longue séquence de zéros. Toutefois ce raccourci de notation ne peut être utilisé qu'une seule fois. La Table 1.2 illustre quelques exemples d'adresses IPv6 dans leur notation courte et dans leur notation longue.

Notation longue	**Notation courte**
`2001:6f8:202:0:0:0:0:1fbe`	`2001:6f8:202::1fbe`
`fe80:0:0:0:219:b9ff:fe4c:6bd7`	`fe80::219:b9ff:fe4c:6bd7`
`0:0:0:0:0:0:0:1`	`::1`
`0:0:0:0:0:0:0:0`	`::`

TABLE 1.2 – Exemples de notation d'adresses IPv6.

Il existe trois grandes classes d'adresses IPv6 déterminées par les premiers bits de l'adresse : les adresses link-local, les adresses globales unicast et les adresses multicast. Les adresses link-local sont utilisées pour les communications qui restent locales au lien, i.e. non routables. Elles sont de la forme `fe80::/10`. Les adresses globales unicast sont globalement routables. Elles sont de la forme `2000::/3` ou `3000::/3`. Les adresses multicast permettent d'identifier un groupe d'interfaces. Elles sont de la forme `ff00::/8`. Les détails de l'architecture des adresses IPv6 est décrit dans le RFC 4291. À noter qu'à la différence de certaines piles IPv4 les interfaces peuvent posséder plusieurs adresses IPv6.

Certaines adresses ont également un sens particulier. L'adresse `::1` est l'adresse loopback équivalente à l'adresse IPv4 127.0.0.1. L'adresse `::` désigne l'adresse *unspecified*. Cette adresse est utilisée lorsque l'adresse source n'est pas connue. Cette adresse n'est associée à aucune interface. L'adresse multicast `ff02::1` désigne le groupe d'interfaces formé de tous

les noeuds accessibles sur le lien. L'adresse multicast `ff02::2` désigne le groupe d'interfaces formé de tous les routeurs accessibles sur le lien.

1.6.2 Découverte des voisins

La découverte des voisins – Neighbor Discovery [45] (ND) est utilisée pour déterminer l'adresse MAC des voisins attachés au lien, découvrir d'autres noeuds sur le lien, découvrir les routeurs, détecter les adresses dupliquées et permettre l'autoconfiguration des adresses. Le protocole se base sur ICMPv6 est utilise cinq messages différents. Premièrement les Router Advertisements (RA) utilisés par les routeurs pour signaler leur présence sur le lien ainsi que des paramètres du réseau tel que le préfixe ou le MTU du lien. Deuxièmement les Router Solicitations (RS) utilisés par les hôtes pour demander au routeurs de signaler leur présence par un message RA. Troisièmement les Neighbor Solicitations (NS) utilisés par les hôtes pour vérifier si une adresse existe, si elle est encore accessible et pour trouver l'adresse MAC d'un voisin. Quatrièmement les Neighbor Advertisements (NA) utilisés pour répondre au message NS. Ces deux derniers types de messages remplacent l'Address Resolution Protocol (ARP) utilisé pour la détection des adresses dupliquées et l'association des adresses IP aux adresses MAC dans IPv4. Et finalement les messages Redirect utilisés par les routeurs pour informer les hôtes qu'un meilleur chemin est disponible pour contacter une adresse. En particulier ce message est utilisé pour signaler aux hôtes que l'adresse est contactable sur le lien sans passer par le routeur. Le protocole ICMPv6 inclu également le Multicast Listener Discovery (MLD) qui remplace le protocole IGMP utilisé pour la gestion des groupes multicast dans IPv4.

1.6.3 Autoconfiguration des adresses sans état

L'autoconfiguration des adresses sans état – Stateless Address Autoconfiguration [50] (SLAAC) est un mécanisme utilisé pour permettre de configurer automatiquement les adresses en se basant sur l'identifiant de l'interface – Interface IDentifier (IID). L'identifiant de l'interface est lui même dérivé d'un identifiant global – Extended Unique Identifier (EUI-64) supposé unique sur le lien. Dans le cas d'un lien Ethernet, l'EUI-64 est dérivé de l'adresse MAC de 48 bits. La valeur FFFE est insérée au milieu de l'adresse pour combler les 16 bits manquant et le bit Universal/Local (U/L) est inversé. Ce bit particulier spécifie si l'adresse est administrée localement, i.e. par l'administrateur du réseau ou globalement. Il est également possible de dériver cet identifiant aléatoirement avec les Privacy Extensions.

L'IID est ensuite combiné avec un préfixe pour former une adresse complète. Cette adresse passe alors par trois états. Premièrement l'adresse se trouve dans l'état **tentative**. Une adresse dans cet état est utilisée uniquement pour détecter si l'adresse est disponible. L'interface envoie un message NS contenant l'adresse. Ce message est envoyé à l'adresse multicast *solicited-node* avec comme adresse source, l'adresse *unspecified*. L'adresse multicast solicited-node permet d'éviter d'utiliser l'adresse broadcast et ainsi de limiter le nombre d'hôte traitant le message. Si l'interface reçoit un message NA en réponse, cela signifie que l'adresse est utilisée. Dans ce cas le processus est réitéré avec un nouveau IID. Dans le cas contraire, l'adresse est unique et elle passe dans l'état **preferred**. Une adresse dans cet état peut être utilisée pour

envoyer et recevoir du trafic pendant une certaine durée appelée *lifetime*. Une fois le lifetime expiré, l'adresse passe dans l'état **deprecated**. Une adresse dans cet état peut encore être utilisée pour les communications déjà en place mais ne peut plus être utilisée pour de nouvelles communications.

Dans un premier temps, l'IID est combiné avec le préfixe link-local `fe80::/10` pour obtenir une adresse avec une portée limitée au lien. Une fois cette adresse obtenue, l'IID est combiné avec le préfixe obtenu depuis les messages RA pour obtenir une adresse avec une portée globale. Si aucun message RA n'a été reçu, l'interface sollicite un routeur avec un message RS. On obtient ainsi une adresse globalement routable sans avoir à conserver d'état à la différence de DHCP. La Figure 1.6 reprend la séquence des opérations nécessaires à l'autoconfiguration des adresses d'une interface.

1.7 6LoWPAN

6LoWPAN [44] est un standard qui permet le transport de paquets IPv6 sur des réseaux de capteurs sans fil à très basse consommation (LR-WPAN). Ce standard définit une sous-couche d'adaptation entre la couche IEEE 802.15.4 MAC et la couche réseau IPv6. La Figure 1.7 compare la pile IP classique avec la pile 6LoWPAN. Les couches transport et application se basent sur des standards déjà existants. Le protocole UDP est toutefois préféré à TCP en raison de sa complexité et de l'overhead qu'il impose sur le réseau.

Un réseaux de capteurs sans fil à basse consommation qui supporte 6LoWPAN est appellé LoWPAN (Low Power Wireless Area Network). Les noeuds d'un LoWPAN partagent le même préfixe IPv6. Ces noeuds peuvent peuvent jouer le rôle de simple hôte ou de routeur. Des routeurs de bordure assurent la connectivité des LoWPAN vers d'autres réseaux. Les LoWPANs peuvent être classés en trois catégories illustrées à la Figure 1.8. Les LoWPANs ad-hoc qui ne possèdent pas d'infrastructure. Les LoWPANs simples connectés à un routeur de bordure. Les LoWPANs étendus qui partagent le même préfixe sur plusieurs routeurs de bordure reliés entre eux par un lien backbone.

Les motivations à la mise en place d'une sous-couche 6LoWPAN dans les réseaux LR-WPAN sont détaillées dans le RFC 4919 [39]. Une partie de cette motivation est la suivante. Le standard IPv6 spécifie un MTU d'au moins 1280 octets. Toutefois la capacité maximum de la charge utile d'une trame IEEE 802.15.4 est de 118 octets. De plus la taille minimum d'un paquet IPv6 est de 40 octets. Si on enlève encore l'en-tête UDP de 8 octets, dans le meilleur des cas seuls 68 octets restent disponibles pour la couche application.

1.7.1 Compression des en-têtes

Un mécanisme de compression des en-têtes IPv6 permet à 6LoWPAN de diminuer l'overhead des couches réseau et transport sur les couches supérieures. Ce mécanisme se base sur les trois observations suivantes :

— Dans le cas d'une adresse autoconfigurée sans état [46] les adresses source et destination IPv6 sont redondantes avec les adresses de la couche MAC.
— Les champs IPv6 flowlabel et traffic class sont souvent à zéro.

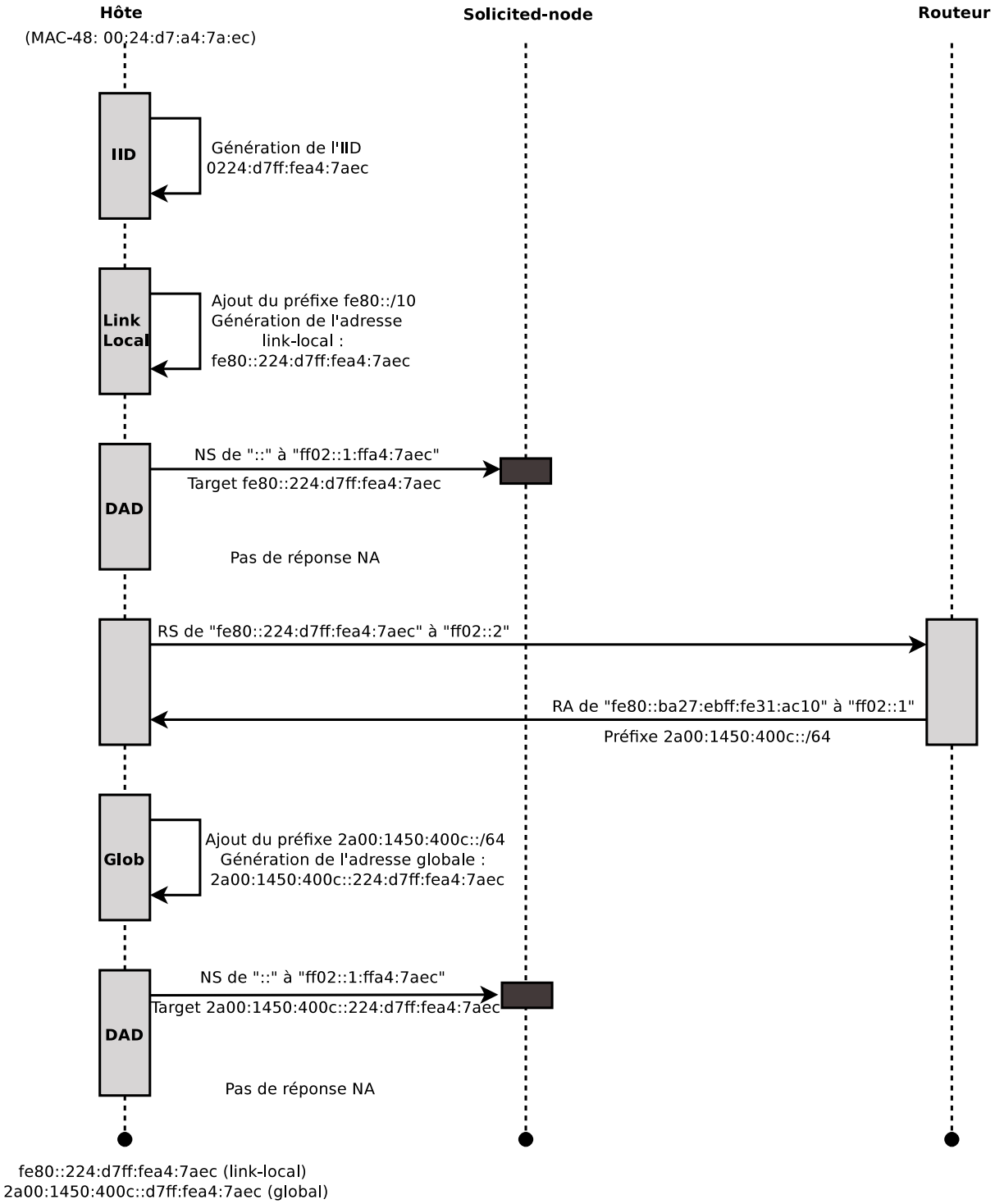


FIGURE 1.6 – Autoconfiguration des adresses sans état.

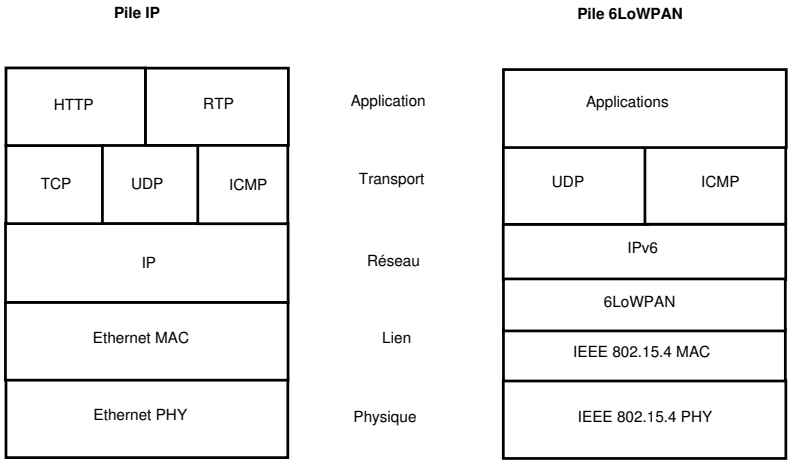


FIGURE 1.7 – Comparaison de la pile IP et 6LoWPAN [51].

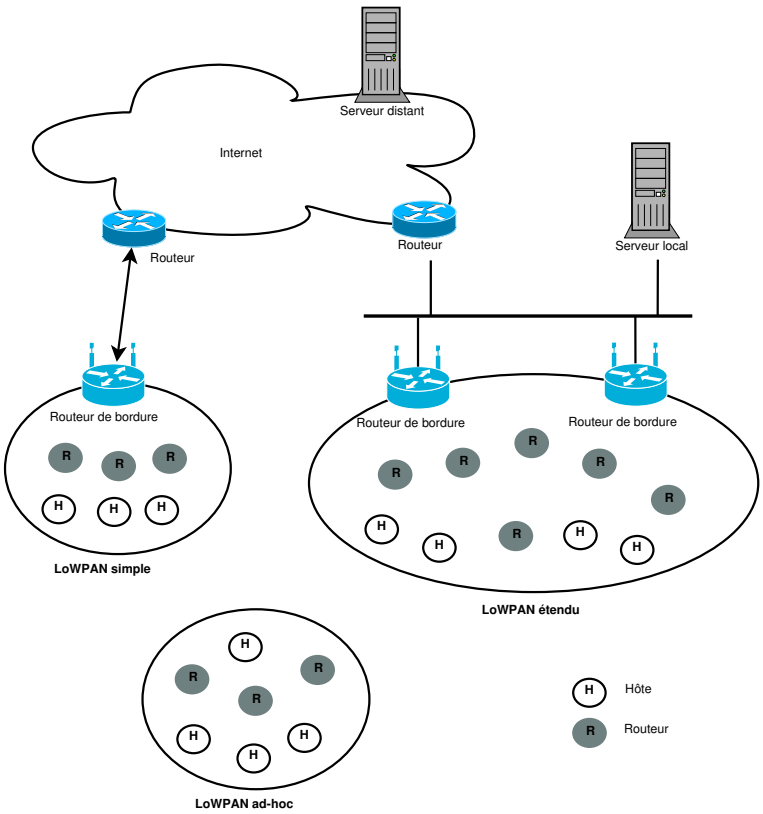


FIGURE 1.8 – Architecture 6LoWPAN [51].

Mode	Préfixe	IID	Taille (bits)
0	non compressé	non compressé	128
1	non compressé	omis et dérivé de la couche MAC	64
2	omis et supposé link-local (`fe80::/64`)	non compressé	64
3	omis et supposé link-local (`fe80::/64`)	omis et dérivé de la couche MAC	0

TABLE 1.3 – Modes de compression des adresses sans état (HC) [51].

Mode	Description	Taille (bits)
0	non compressé	128
1	link-local + IID (64 bits)	64
2	link-local + adresse courte (16 bits)	16
3	link-local + adresse MAC	0
4	reservé	–
5	contexte + IID (64 bits)	64
6	contexte + adresse courte (16 bits)	16
7	contexte	0

TABLE 1.4 – Modes de compression des adresses avec état optionel (IPHC) [51].

— Un nombre limité de ports est utilisé par la couche transport.

Deux méthodes sont proposées par le standard pour effectuer la compression des en-têtes : une méthode sans état (HC) et une méthode avec état optionnel définit dans le RFC 6282 [38] (IPHC).

Dans le mode sans état deux octets sont utilisés pour spécifier la compression. Ces deux octets spécifient les champs IPv6 à omettre, la compression de l'en-tête UDP et la compression des adresses source et destination. Pour chaque adresse, quatre modes de compression peuvent être utilisés. Ces modes sont résumés dans la Table 1.3. Cette table reprend également la taille nécessaire pour encoder une adresse dans chacun des modes. Le mode 0 reprend l'adresse complètement. Le mode 1 reprend le préfixe et dérive l'identifiant depuis la couche MAC. Le mode 2 reprend l'identifiant et utilise le préfixe link-local. Le mode 3 dérive l'identifiant depuis la couche MAC et utilise le préfixe link-local. Le mode 1 est utilisé dans le cas le plus courant. En effet, dans ce cas des adresses globales sont utilisées et le préfixe de 64 bits ne peut pas être omis. Dans le cas d'un datagramme UDP, il est possible de dériver la taille de celui-ci depuis la taille de la trame. Il est également possible de restreindre les ports disponibles de F0B0 à F0BF. Dans ce cas un octet est nécessaire pour décrire le port source et destination du datagramme.

Mode	Source	Destination	Taille (bits)
0	non compressé	non compressé	32
1	non compressé	F0xx	24
2	F0xx	non compressé	24
3	F0Bx	F0Bx	8

TABLE 1.5 – Modes de compression des ports UDP (IPHC) [51].

Le mode de compression avec état permet l'utilisation d'un contexte optionnel partagé pour compresser les adresses globales. Le mécanisme utilisé pour partager ce contexte est défini dans une proposition de mise à jour du RFC 4944, draft-ietf-6lowpan-nd [52]. Les modes de compression et la taille d'adresse résultante sont résumés dans la Table 1.4. Le mode 0 reprend l'adresse complètement. Le mode 1 reprend un identifiant de 64 bits et utilise un préfixe link-local. Le mode 2 reprend un identifiant de 16 bits et utilise un préfixe link-local. Le mode 3 dérive l'identifiant depuis l'adresse MAC et utilise un préfixe link-local. Les modes 5 à 7 se reposent sur un contexte partagé. Le mode 5 reprend un identifiant de 64 bits et utilise un préfixe partagé. Le mode 6 reprend un identifiant de 16 bits et utilise un préfixe partagé. Le mode 7 dérive l'adresse entière depuis le contexte. Dans le cas le plus courant d'adresses globales avec un identifiant de 16 bits, le mode 2 est utilisé et 32 bits sont nécessaires pour stocker les adresses source et destination dans l'en-tête IPv6 lorsque le partage de contexte est disponible.

Dans le cas d'un datagramme UDP il est possible de dériver la taille de celui-ci depuis la taille de la trame. Il est également possible d'omettre le checksum. Dans ce cas, le récepteur calcule celui-ci avant de transmettre le datagramme à la couche supérieure. Les différents modes de compression disponibles pour les ports source et destination sont résumés dans la Table 1.5. Le mode 0 reprend les ports source et destination complètement. Le mode 1 reprend le port source et restreint le port destination de F000 à F0FF. Le mode 2 reprend le port destination et restreint le port source de F000 à F0FF. Le mode 3 restreint le port source et destination de F0B0 à F0BF.

1.7.2 Fragmentation des paquets

Le standard IPv6 assume que la couche lien est capable de transporter un paquet d'au moins 1280 octets sans fragmentation. Toutefois la taille maximum des trames IEEE 802.15.4 MAC est de 127 octets. Lorsqu'un paquet est trop large pour être transféré en une unique trame, le paquet est découpé et un en-tête de fragmentation propre à 6LoWPAN est ajouté à chaque fragment. Cet en-tête contient la taille du paquet non fragmenté, l'offset du fragment dans le paquet et l'identifiant du paquet.

Il est important de noter la différence entre la fragmentation au niveau IPv6 et 6LoWPAN. La fragmentation IPv6 a lieu lorsque la taille du paquet à transmettre dépasse le MTU du lien. Toutefois il existe pour IPv6 une garantie sur le MTU minimal du lien. Dès lors en cas de fragmentation, les fragments IPv6 utilisent le maximum des 1280 octets qui leur sont garantis.

Comme le lien IEEE 802.15.4 ne permet pas de garantir ce MTU, le mécanisme de fragmentation de la couche 6LoWPAN découpe les fragments IPv6 en trames de 127 octets. Les deux mécanismes de fragmentations sont donc utilisés en même temps dans IPv6 et 6LoWPAN.

1.7.3 Découverte des voisins

Les mécanismes de découverte des voisins IPv6 [45] sont nécessaires pour découvrir les noeuds sur le lien, trouver les routeurs, détecter les adresses dupliquées et l'autoconfiguration des adresses. Toutefois ces mécanismes sont trop complexes pour pouvoir être utilisés tels quels dans les réseaux LoWPANs. Une extension du standard 6LoWPAN [52] les simplifie en déléguant la conservation de l'état des adresses aux routeurs de bordure. La mise à jour de cet état est assurée par des messages ICMP d'enregistrement des adresses. À noter qu'il est toujours possible d'utiliser les messages ICMP sans passer par un routeur de bordure.

1.8 Linux

Le noyau Linux [28] est le noyau d'un système d'exploitation de la famille des Unix-Like. Il s'agit d'un noyau monolithique, multiutilisateur, multiprocesseur et préemptif qui supporte les fonctionnalités des systèmes d'exploitation modernes. Il vise à être conforme avec le standard IEEE POSIX. Linux est le plus connu d'une famille de noyaux de système d'exploitation opensource. On peut également citer : FreeBSD, OpenBSD, NetBSD, Minix et GNU Hurd.

Le code source du noyau Linux est disponible dans les Linux Kernel Archives [14]. Il est publié sous une license GNU General Public License (GPL). Le noyau est bien supporté, bien documenté, flexible et compatible avec de nombreuses architectures notamment : i386, x86_64, MIPS et de nombreux SoC ARM ce qui rend son utilisation populaire dans de nombreux systèmes embarqués. Il est également capable d'exploiter de nombreuses couches réseau : Ethernet, 802.11, 802.15, IPv4 et IPv6. Ainsi que de nombreux bus hardware : USB (Universal Serial Bus), UART (Universal Asynchronous Receiver Transmitter), SPI (Serial Peripheral Interface), I^2C (Inter Integrated Circuit). Il permet également la programmation de pins GPIO (General Purpose Input Output).

L'infrastructure de compilation du noyau est très complète et repose essentiellement sur des Makefiles et la GNU Compiler Collection (GCC). Elle dispose d'un support pour la compilation concurrente et la cross-compilation qui permet d'éviter d'avoir à compiler sur l'architecture cible. La configuration du noyau passe à travers un fichier texte simple et des scripts permettent de passer en revue la configuration du noyau à travers une interface graphique.

Le noyau permet l'utilisation de modules pour gagner en flexibilité malgré son caractère monolithique. Un module est un fichier objet dont le code peut être lié au noyau pendant l'exécution. La plupart des pilotes de périphérique et des composants du noyau sont disponibles sous forme de module. L'infrastructure de compilation permet de compiler le module inclu dans le code du noyau (builtin) ou sous la forme d'un fichier objet chargeable à l'éxecution (module).

1.9 Contiki

Contiki [33, 8] est un système d'exploitation léger conçu pour les réseaux de capteurs sans fil. Il est optimisé pour fonctionner sur des microcontrolleurs 8 bits avec moins de 20 kB de RAM. Il est implémenté en C et il est porté sur de nombreux microcontrolleurs, en particulier l'Atmel AVR et le Texas Instrument MSP430. Il supporte des threads légers appelés protothreads pour faciliter la programmation évènementielle. Il fourni une pile IPv4, IPv6, la couche 6LoWPAN et le protocole de routage RPL [55] (Routing Protocol for Low power and lossy network). Contiki inclu également un simulateur (Cooja) qui permet de simuler des réseaux de capteurs sans fil.

Le système d'exploitation permet également de charger et décharger des applications et des services en cours d'exécution. Les applications sont transmises vers les noeuds à travers le réseau ce qui facilite la maintenance. La capacité de ne charger et décharger que certains composants du système évite également d'avoir à transmettre l'image complète du système d'exploitation par le réseau.

A la différence d'autres systèmes d'exploitation complètement évènementiels comme TinyOS, Contiki utilise un noyau évènementiel sur lequel s'appuient des processus implémentés sous forme de protothreads [34]. Les protothreads fournissent des opérations d'attente et de blocage conditionnel qui permettent de simplifier l'écriture de programmes dirigés par les évènements au détriment d'un léger overhead.

1.10 Raspberry Pi

Le Raspberry Pi [15], illustré à la Figure 1.9, est un petit ordinateur bon marché et à très basse consommation (3 à 4 Watts). Il est construit autour d'un System on Chip (SoC) Broadcom BCM 2835 qui contient un processeur ARM1176JZFS d'architecture ARMv6 cadencé à 700MHz avec 256Mo ou 512Mo de RAM et une mémoire physique accessible sur une carte SDHC. Le Raspberry Pi possède de nombreuses entrées-sorties : GPIO, SPI, UART, I^2C. Il possède également des ports USB, une sortie HDMI (High Definition Multimedia Interface), audio et une interface Ethernet 10/100 RJ45.

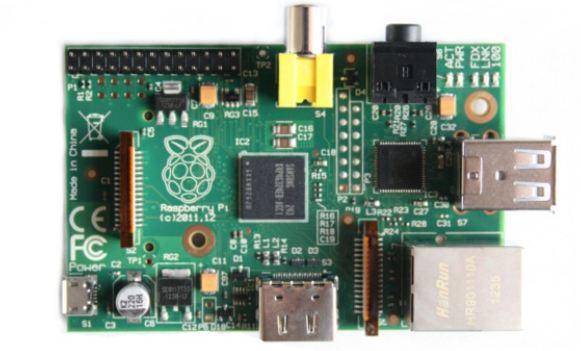


FIGURE 1.9 – Raspberry Pi.

Les systèmes d'exploitation Linux, FreeBSD et Risc OS sont supportés sur le Raspberry Pi. Dans le cas de Linux, seules certaines distributions sont supportées. La distribution recommandée appelée Raspbian est basée sur Debian Testing (Wheezy). Cette distribution est disponible sous deux versions respectant des Application Binary Interfaces (ABI) différentes. La première version, *armel*, utilise la convention d'appels softfp pour le passage des arguments flottants. Dans cette convention, les arguments flottants sont placés sur des registres entiers. La deuxième version, *armhf*, utilise la convention d'appels hard-float. Dans cette convention, les arguments flottants sont placés sur des registres flottants dédiés. La version armel est nécessaire pour l'utilisation de la JVM d'Oracle qui ne supporte pas l'ABI hard-float sur ARM. À noter que la distribution officielle de Debian pour ARMv6 est disponible uniquement dans la version armel.

Le Raspberry Pi a de nombreuses applications. Dans la distribution Raspbian de base, il est livré avec l'environnement de bureau LXDE et peut être utilisé comme ordinateur fixe. Il peut également être utilisé sans l'environnement graphique en utilisant Secure SHell (SSH) sur le port Ethernet ou une connexion série. Il peut aussi être utilisé comme serveur personnel pour autant que la charge ne soit pas trop importante.

Toutefois un point faible du Raspberry Pi est l'utilisation d'une carte SD pour la mémoire physique. En effet, ces cartes ont généralement un débit lecture/écriture de l'ordre de 20 Mo/s maximum ce qui peut poser des problèmes pour les applications qui utilisent beaucoup les I/O.

1.11 BeagleBone

Le BeagleBone [6], illustré à la Figure 1.10, est une plateforme de développement similaire au Raspberry Pi. Il s'articule autour d'un SoC Sitara AM335X qui contient un processeur ARM Cortex-A8 d'architecture ARMv7 cadencé à 720MHz avec 256Mo ou 512Mo de RAM et une mémoire physique accessible sur une carte micro-SD. Il dispose de nombreuses entrées-sorties : GPIO, SPI, UART, I^2C et CAN. Deux Programmable Realtime Unit (PRU) permettent de gérer plus finement les entrées-sorties. Il possède également un port USB, une sortie HDMI et une interface Ethernet 10/100 RJ45.

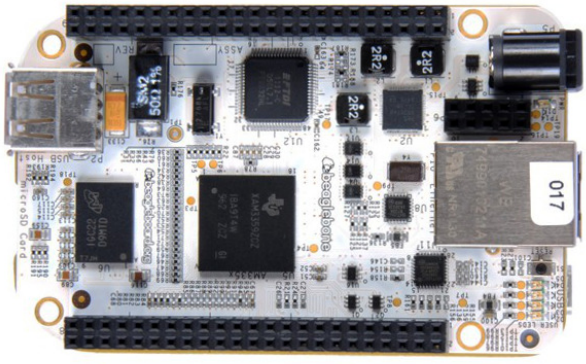


FIGURE 1.10 – BeagleBone [4].

Le BeagleBone possède un nombre d'entrées-sorties plus élevé que le Raspberry Pi. L'utilisation d'un processeur d'architecture ARMv7 lui permet d'exploiter une gamme plus large de distributions du système d'exploitation Linux. En particulier il permet d'utiliser la version officielle armhf de Debian, Ubuntu ou Fedora. Il permet également d'utiliser d'autres systèmes d'exploitations basés sur Linux dont Android 4.0. Toutefois celui-ci reste plus coûteux que le Raspberry Pi et son utilisation est moins répandue.

Chapitre 2

État de l'art

2.1 Linux-ZigBee

Le projet Linux-ZigBee [37], démarré en 2009, implémente le support pour IEEE 802.15.4 dans le noyau Linux. Contrairement au nom du projet, le support pour le reste de la pile ZigBee n'est pas inclu dans le noyau à cause d'une incompatibilité entre la license de la ZigBee Alliance et la license GPL. Le projet comprend également un ensemble d'outils en espace utilisateur pour manipuler la couche MAC et mettre en place le coordinateur PAN. L'ensemble du projet est encore expérimental et de nombreux bogues subsistent.

La pile supporte pour les périphériques un mode SoftMAC et HardMAC. Les périphériques HardMAC implémentent le MLME (pour rappel voir la Section 1.2.3). Dans ce cas les commandes MAC peuvent être directement enregistrées dans la pile. Les périphériques SoftMAC sont plus proches de la couche PHY. Dans ce cas, le MLME est implémenté dans le noyau par le sous système mac802154. L'architecture permet également de rajouter d'autres sous systèmes MLME comme S-MAC [56].

A ce jour, l'implémentation du MLME pour les périphériques SoftMAC dans le noyau Linux n'est pas conforme au standard IEEE 802.15.4 et certaines fonctionnalités comme CSMA/CA, les ACK et la sécurité ne sont pas implémentées. Les outils en espace utilisateurs, notamment pour la configuration de la couche MAC ne sont pas utilisables dans la version 0.3. Des patchs existent dans les logiciels suivants pour reconnaître le standard IEEE 802.15.4 sous Linux : libpcap, tcpdump et wireshark.

La plupart des transceivers sont connectés via un bus SPI. Les transceivers IEEE 802.15.4 supportés comprennent : Atmel AT86rf230/231, MC13192, MC13122x [25], Analog Device ADF7242, Texas Instrument CC2420 et Microchip MRF24J40. Toutefois ces pilotes ne sont pas complets et nécessitent des correctifs et des améliorations.

Une implémentation de 6LoWPAN dérivée du code de Contiki est également disponible dans le noyau. Toutefois l'implémentation n'est pas complète et de nombreuses fonctionalités ne sont pas implémentées. À ce jour il n'est pas encore possible d'effectuer des communications avec la couche 6LoWPAN.

2.2 Raspberry PI

Le noyau utilisé dans la distribution est maintenu sur GitHub [16]. À ce jour, la branche stable 3.2.27 du noyau ne comprend pas la pile IEEE 802.15.4. Il existe également deux branches 3.6 et 3.8 en développement qui comprennent la pile IEEE 802.15.4 et certains drivers.

Robinson [48] travaille sur le driver du transceiver de Texas Instrument CC2520 dans le but selon ses propres mots : *"The eventual goal is to get it running a full 6LoWPAN network stack and act as a wireless mesh network router, bridging low-power sensor networks of mesh-networking motes with traditional IP networks."* [49]

2.3 Passerelle 6LoWPAN

2.3.1 Grinch

Grinch [42] est un routeur de bordure 6LoWPAN basé sur un module Zigbit, un processeur Atmega 1281 et un transceiver Atmel AT86rf230. Le routeur de bordure utilise Contiki auquel l'auteur a ajouté le support pour les interfaces multiples et utilise une interface Ethernet pour le lier au reste du réseau.

2.3.2 NanoRouter

Une autre solution propriétaire embarquée, NanoRouter, développée par Sensinode [17] permet d'exploiter une platforme ARM 9 comme routeur de bordure sous Linux. Le logiciel NanoRouter supporte 6LoWPAN, 6loWPAN-ND, le protocole de routage RPL, ICMPv6 et les protocoles de transport UDP et TCP. Cette solution est utilisé dans le routeur de bordure fourni dans le kit CC-6LoWPAN [18] de Texas Instrument. Il est également possible d'utiliser cette solution sur une carte BeagleBone.

2.3.3 Arch Rock PhyNet Router

Une autre solution propriétaire embarquée développée par la compagnie [1] Arch Rock [5] utilise TinyOS dans un router de type Small Office, Home Office (SOHO) illustré à la Figure 2.1.

2.3.4 JenNet-IP Border-Router

La solution de routeur de bordure développée par Jennic [3] se base sur une distribution Linux OpenWRT et un transceiver JenNet-IP JN5148. Un démon en espace utilisateur, 6LowPANd, utilise une connexion série et un tunnel TUN/TAP pour faire le pont entre le noyau et le transceiver. Cette solution est illustrée à la Figure 2.2.

1. Cisco a fait l'acquisition de Arch Rock en 2010.

FIGURE 2.1 – Arch Rock PhyNet Router [5].

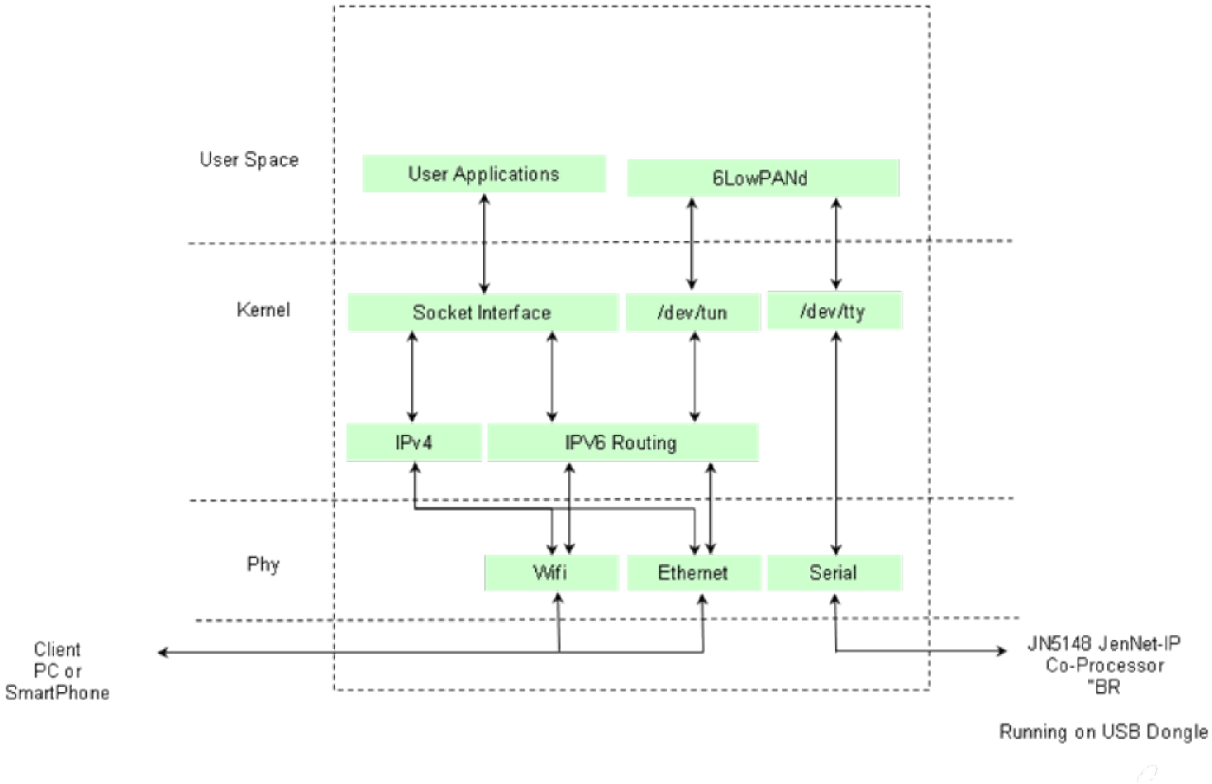


FIGURE 2.2 – Architecture de JenNet-IP Border-Router [3].

2.3.5 Redwire Econotag

Dans le cadre d'un stage au CETIC (Centre d'Excellence en Technologies de l'Information et de la Communication), Maxime Denis [32] utilise un prototype de routeur de bordure développé par des membres du CETIC. Ce routeur de bordure s'articule autour d'un capteur Redwire Econotag sous Contiki et utilise un système appelé Packet Filter pour exploiter les deux interfaces Ethernet et IEEE 802.15.4.

2.3.6 Autres solutions envisageables

La plupart des plateformes possédant un port Ethernet et un bus SPI sont des solutions envisageables pour un routeur de bordure à condition toutefois qu'elles supportent l'installation d'un système tel que Contiki ou Linux récent. Les plateformes SOHO tel que Carambola [7] pourraient profiter d'une solution sous Linux implémenté dans OpenWRT.

Il est important de noter que la plupart des personnes impliquées dans le développement de la couche IEEE 802.15.4 sous Linux ont pour objectif la réalisation d'un routeur de bordure 6LoWPAN. Les plateformes les plus couramment utilisées sont les Raspberry Pi et les BeagleBone. Dans d'autres cas le développement est effectué entièrement avec des interfaces de simulation. Certains développeurs ont démarré des projets [27, 29] qui visent à implémenter le protocole de routage RPL. Toutefois ces implémentations en sont encore au stade de démarrage du projet et ne sont pas utilisables.

Chapitre 3

Implémentation

Ce chapitre reprend les différentes étapes de l'implémentation du projet. Dans sa première partie, ce chapitre reprend les étapes nécessaires à la mise en place des outils utilisés pour le développement. En particulier le Raspberry Pi, le noyau Linux et le projet Linux-ZigBee. Les techniques utilisées pour faciliter le développement sont également reprises. Dans sa deuxième partie, ce chapitre reprend le développement de certains outils développés pour faciliter le débogage et la maintenance du projet. Finalement dans sa troisième partie, ce chapitre reprend le développement et le débogage du projet Linux-ZigBee ainsi que le développement de fonctionnalités supplémentaires.

3.1 Image Raspbian

La distribution Raspbian recommandée pour les Raspberry Pi est disponible sous forme d'une image à installer sur la carte SD. Toutefois cette distribution est orienté vers des utilisateurs novices et comprend dans l'image de base de nombreux outils et services qui ne sont pas nécessaire à la création d'une passerelle 6LoWPAN. De plus la plupart de ces services sont lancés par défaut au démarrage du Raspberry Pi ce qui consomme de la RAM inutilement. Cette section décrit les diverses optimisations qui ont été réalisées afin d'optimiser l'utilisation du Raspberry Pi comme un routeur de bordure.

La méthode de backup utilisée pour le Raspberry Pi consiste à effectuer régulièrement des copies des cartes SD utilisées pour le développement. Toutefois un grand nombre de logiciels sont installés par défaut dans la distribution Raspbian. Afin d'économiser sur la taille des images sauvegardés et la RAM, un nettoyage des paquets est effectué avec *aptitude*. La suppression d'un paquet entraine la suppression de tous les paquets possédant une dépendance vers celui-ci. Dans certains cas la suppression d'un paquet entraine la suppression du système dans son entièreté à cause de problèmes de dépendances. Pour éviter cela la suppression s'effectue par étape successives et certains paquets sont enlevés manuellement lorsqu'un problème de dépendance survient. Après ce nettoyage 817 Mo sont utilisés sur la racine contre 1,4 Go à l'origine et 15 Mo de RAM [1] utilisé au démarrage contre 27 Mo à l'origine.

1. Il s'agit de la mémoire RAM active.

L'utilisation de certains logiciels propres au gestionnaire de paquets de Debian permettent d'économiser encore plus de place sur la partition racine. Le logiciel *localepurge* est un script permettant d'économiser l'espace disque en évitant l'installation de fichiers de localisation inutiles. Ce script permet de choisir les localisations à conserver sur le système. Il se branche sur le système de gestion des paquets et supprime les fichiers de localisations superflus lors de l'installation ou mise-à-jour des paquets. Le paquet *deborphan* cherche les paquets qui n'ont aucune dépendance depuis un autre paquet. Ces paquets sont appelés paquets orphelins. En particulier le logiciel est utilisé ici pour trouver et supprimer les librairies orphelines. Finalement la commande suivante effectue une purge des fichiers de configuration des paquets désinstallés :

```
dpkg --get-selections | grep install | xargs sudo dpkg -P
```

D'autre part, certaines fonctionnalités ne sont pas activées par défaut sur l'image d'origine. En particulier le fichier `/etc/modprobe.d/ipv6.conf` désactive le support de l'IPv6 par défaut. Il est nécessaire de supprimer ce fichier pour réactiver l'IPv6. Il est aussi nécessaire de supprimer d'autres fonctionnalités qui peuvent poser un risque de sécurité. En particulier un compte utilisateur par défaut est associé à l'image de la distribution. Le nom de ce compte et son mot de passe sont communs à toutes les images Raspbian. Comme il est possible que les plateformes soient connectées directement à Internet par IPv6. Il est nécessaire de supprimer ce compte pour éviter les intrusions par ssh. Il est également nécessaire de désactiver l'autentification de l'utilisateur root par ssh dans le fichier `/etc/ssh/sshd_config`.

3.2 Compilation du noyau

Le projet utilise la version 3.2.27 du noyau officiel du Raspberry Pi. Les sources de ce noyau sont disponibles sur Git avec l'URL suivante :

```
git://github.com/raspberrypi/linux.git
```

Il est important de noter que les sources de ce noyau sont différentes de celles disponibles pour les noyaux officiels Linux, i.e. disponibles sur les Linux Kernel Archives. En particulier le noyau du Raspberry Pi rajoute une implémentation spécifique du SoC BCM2835 et d'autres modifications propres au Raspberry Pi. Cette section décrit les étapes nécessaires à la compilation du noyau pour le Raspberry Pi.

La compilation du noyau est une tâche fréquente dans le projet. Dans la plupart des cas seuls certains modules doivent être recompilés mais des mises à jours plus importantes depuis le noyau officiel obligent la recompilation du noyau dans son entièreté. Lorsque c'est le cas, la méthode de compilation la plus simple consisterait à compiler directement le noyau sur la cible (ARM). Toutefois étant donné les ressources disponibles et les débits entrées sorties de la carte SD, la compilation du noyau avec cette méthode peut prendre jusqu'à 10 heures. Dans les sections qui vont suivre, le terme *hôte* désigne la machine sur laquelle est effectuée la compilation. Le terme *cible* désigne l'architecture pour laquelle les sources sont compilées.

On remédie à ce problème en utilisant le mécanisme de cross-compilation [9] offert par l'infrastructure de compilation du noyau. Il est nécessaire pour cela d'installer

CPU	4 x Intel(R) Core(TM) i5 CPU M 560 @ 2.67GHz
RAM	8 Go
DISK	Samsung SSD 840-PRO 256GB
OS	Debian GNU/Linux 3.6.10-gawen-x201-1 (x86_64)

TABLE 3.1 – Caractéristiques de la machine utilisée pour la cross compilation.

la toolchain *arm-gnueabi* ainsi qu'un compilateur GCC qui supporte la cible ARM. Ces derniers sont disponibles sous la forme de paquets Debian dans le repository Emdebian [10]. L'installation des paquets nécessaires est reprise dans l'Annexe A.1. Une fois les paquets installés, il est possible de compiler depuis une autre architecture avec les commandes suivantes :

```
make ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- oldconfig
make ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- menuconfig
make -j 8 ARCH=arm CROSS_COMPILE=/usr/bin/arm-linux-gnueabi- -k
```

La première ligne met à jour la configuration depuis la configuration précédente du noyau. Si de nouvelles options sont présentes, un message demande à l'utilisateur s'il faut les inclure dans le noyau. La seconde ligne lance l'interface de configuration du noyau pour l'architecture ARM. Ce script permet de générer un fichier *.config* qui spécifie les options et paramètres du noyau ainsi que les modules à compiler. La troisième ligne lance la compilation du noyau. L'argument `-j 8` spécifie le nombre de processus à utiliser en paralèlle pour la compilation.

Les caractéristiques de la machine utilisée sont reprises dans la Table 3.1. Le temps de compilation d'un noyau complet avec la cross-compilation est alors de 7 minutes et 30 secondes. Toutefois dans de nombreux cas seuls certains modules sont modifiés. En particulier pendant les phases de débogage il est fréquent d'avoir à compiler le même fichier plusieurs dizaines de fois sur la même journée. Dans ce cas le Makefile compile uniquement les modules dont les fichiers sources ont été modifiés. La compilation prend alors une dizaine de secondes.

3.3 Compilation des outils en espace utilisateur

Les outils en espace utilisateur *lowpan-tools* sont nécessaires pour manipuler la pile IEEE 802.15.4 du noyau Linux. Ces outils sont disponibles dans les paquets de la distribution Raspbian. Toutefois il est nécessaire d'utiliser la dernière version des outils pour suivre le développement du projet. Cette section décrit les étapes nécessaires à la compilation de ces outils pour le Raspberry Pi. La dernière version des outils est disponible sur Git avec l'URL suivante :

```
git://linux-zigbee.git.sourceforge.net/gitroot/
      linux-zigbee/linux-zigbee
```

Ces outils dépendent également d'une autre librairie *libnl3*. Cette librairie permet l'utilisations des sockets netlinks qui interfaçent les outils lowpan-tools à la pile IEEE 802.15.4 du noyau. Afin de s'assurer que cette librairie soit à jour également, elle est compilée depuis les sources du paquet dans la distribution Debian unstable. La procédure pour récupérer les sources est décrite dans l'Annexe A.2.

D'une manière similaire à la compilation du noyau, il est possible d'économiser le temps de compilation en exploitant la cross-compilation. Toutefois l'infrastructure de compilation se base ici sur un script de configuration autoconf. Ce script permet de configurer le Makefile et les sources depuis les fonctionnalités présentes sur la machine. Comme à la section précédente, la cross-compilation nécessite l'installation de la toolchain *arm-gnueabi* ainsi qu'un compilateur GCC qui supporte la cible ARM.

Le script `/usr/share/misc/config.guess` permet de trouver le nom complet de l'architecture cible. Ce script doit être exécuté sur le Raspberry Pi. Dans notre cas l'architecture cible s'appelle `armv6l-unknown-linux-gnueabihf`. Une fois l'architecture cible obtenue, il est possible de compiler et installer la librairie libnl3 avec les commandes suivantes :

```
CC=arm-linux-gnueabi-gcc ./configure
  --host=armv6l-unknown-linux-gnueabihf \
  --prefix=/usr/arm-linux-gnueabi
make
make install
```

L'argument `-prefix` de la première ligne spécifie que la librairie doit être installée dans la toolchain ARM. Cette première ligne configure le Makefile et le code pour l'architecture cible et les options spécifiées. La seconde ligne lance la compilation. Finalement la troisième ligne installe la librairie. Le script de configuration des outils lowpan-tools vérifie sur la machine hôte la présence des librairies dont il dépend. En particulier le script cherche l'emplacement de la librairie libnl3 compilée précédemment. Il faut préciser l'emplacement de cette librairie avec les variables d'environnements suivantes :

```
export NL_CFLAGS="-I/usr/arm-linux-gnueabi/include/libnl3"
export NL_LIBS="/usr/arm-linux-gnueabi/lib/libnl-3.so
                /usr/arm-linux-gnueabi/lib/libnl-genl-3.so"
```

À la différence de la librairie libnl3, on installe pas les outils lowpan-tools dans la toolchain. En effet, ils seront installés sur la cible uniquement. Une fois les variables d'environnement configurées, on lance la compilation avec les commandes suivantes :

```
CC=arm-linux-gnueabi-gcc ./configure
  --host=armv6l-unknown-linux-gnueabihf
make -j 8
```

3.4 Mise en place des outils et du noyau

Une fois les outils et le noyau compilés les fichiers résultants se trouvent sur la machine hôte. Ces fichiers doivent être installés sur la machine cible. Toutefois un simple transf des fichiers n'est pas suffisant. En effet, plusieurs problèmes nécessitent une intervention manuelle après le transfère des fichiers. Premièrement certains programmes effectuent des opérations complémentaires après avoir copiés leurs fichiers. Par exemple l'installation de librairies nécessite la création de liens symboliques supplémentaires. Deuxièmement il est fréquent que l'emplacement des répertoires d'installation sur la machine cible ne soient pas les mêmes que sur la machine hôte. De ce fait l'arborescence des fichiers ne peut s'extraire correctement sur la machine cible et les fichiers doivent être copiés manuellement. Cette section décrit la solution mise en oeuvre pour éviter ces problèmes.

La solution utilisée consiste à effectuer le processus d'installation directement sur la machine cible. Il est possible d'utiliser une archive complète du programme à installer. Toutefois cela nécessite le double du temps pour compresser l'archive et ensuite la décompresser. De plus les fichiers sources non utilisés pour l'installation devraient être transférés également. Pour éviter cela on utilise un système de fichier partagé à travers le réseau. Plusieurs solutions existent pour réaliser cela sous Linux. Elles sont décrites dans les paragraphes suivant.

Premièrement le Network File System (NFS) permet de monter un système de fichier distant. On accède aux fichiers à travers un point de montage sur la machine cliente. Cela permet aux programmes sur la machine cliente d'accéder de manière transparente aux fichiers sans se soucier de leur caractère distant. Il est nécessaire de spécifier les systèmes de fichiers à exporter et leur politique d'accès sur la machine distante. Cette solution est implémentée directement dans le noyau. Toutefois elle peut poser des problèmes de sécurité si le client n'est pas correctement authentifié. De plus cette solution est très flexible et dans certains cas plus difficile à configurer.

Deuxièmement le Common Internet File System (CIFS) permet d'accéder à des ressources distantes. En particulier, elle permet l'accès à des systèmes de fichiers distant. Cette solution est généralement utilisée pour le partage de fichiers avec des machines sous Windows. Les implémentations possèdent généralement un mécanisme pour authentifier les utilisateurs. Sous Linux cette solution nécessite des démons et outils supplémentaires en espace utilisateur. Il est également possible de monter les fichiers de manière transparente. Cette solution nécessite également de spécifier les ressources à exporter et leur politique d'accès sur la machine distante.

Troisièmement le Secure SHell File System (SSHFS) se base sur le Secure File Transfer Protocol (SFTP) pour le transfert des fichiers. Le SSH File Transfer Protocol est une extension du protocole SSH qui permet le transfert de fichiers sécurisés sur le réseau. Ce protocole nécessite un serveur SSH qui supporte le protocole SFTP ainsi qu'un programme client. Le SSHFS permet d'abstraire le rôle de ce programme client à travers un système de fichiers en espace utilisateur. Le système de fichiers est alors monté de manière transparente similairement aux solutions précédentes. L'authentification est directement gérée à travers le protocole SSH. De plus il n'est pas nécessaire de spécifier les systèmes de fichiers à partager. En effet, le client peut accéder à tous les fichiers auquels il a accès selon les droits de son utilisateur sur la machine distante. Toutefois

étant donné le caractère sécurisé des transferts cette solution est moins performante que les solutions précédentes.

La solution retenue ici utilise le SSHFS. En effet, cette solution est la plus rapide à mettre en place. La plupart des machines sous Linux possèdent déjà un serveur SSH installé. De plus par défaut la configuration permet déjà aux utilisateurs présent sur le système de se connecter et de monter les systèmes de fichiers. Les seuls éléments supplémentaires à installer sont sshfs et le Filesytem in USErspace (FUSE). En particulier FUSE permet de créer des systèmes de fichiers en espace utilisateur. Cette solution s'articule autour d'un module spécifique dans le noyau et une librairie pour l'espace utilisateur. On utilise souvent celle-ci pour le prototyping de système de fichiers. On l'utilise également lorsque le système de fichiers fait appel à des éléments qui ne peuvent pas ou difficilement être intégrés au noyau. En particulier cela permet d'exploiter les nombreuses fonctionnalités disponibles uniquement dans des librairies en espace utilisateur. Dans le cas de SSHFS elle permet d'interfacer directement le client SFTP.

Une fois les outils mis en place, il est possible de monter le système de fichier avec la commande suivante :

```
sshfs user@hostname:remote-path local-path
```

Cette commande prend en argument le nom d'utilisateur **user**, le nom de l'hôte distant **hostname**, le chemin sur l'hôte distant **remote-path** et le point de montage sur l'hôte local **local-path**. En cas de problème de connection le système de fichiers n'est pas coupé. Il est alors nécessaire de démonter le système de fichiers manuellement avec la commande *umount*.

Une fois le système de fichiers monté, les outils peuvent être installés. Pour cela on se place dans le répertoire racine de l'outil sur la machine cible et on exécute la commande d'installation. Dans le cas des outils lowpan-tools la commande `make install` suffit à installer le programme. Dans le cas du noyau deux commandes sont nécessaires pour finir l'installation. La première consiste à copier le noyau dans la partition du bootloader du Raspberry Pi. Pour cela il faut copier le fichier `arch/arm/boot/Image` depuis le noyau vers le fichier `/boot/kernel.img`. La deuxième commande consiste à installer les modules sur la partition racine. Pour cela il suffit d'exécuter la commande `make modules_install` à la racine du noyau. Cette commande peut prendre du temps car elle copie tous les modules depuis la machine distante vers la carte SD du Raspberry Pi.

Il est possible que l'installation du noyau ne fonctionne pas. Par exemple car le noyau a été compilé avec des options incompatibles ou car le Raspberry Pi n'a pas été redémarré correctement et le noyau n'a pas été correctement écrit sur la partition du bootloader. Dans ce cas il n'est plus possible de démarrer sur la carte SD. Toutefois il n'est pas nécessaire de remettre en place un backup pour remédier au problème. En effet, il suffit de brancher la carte SD sur une autre machine et de réinstaller le noyau manuellement depuis celle-ci. En cas de problème d'incompatibilité des options du noyau, il est également prudent d'effectuer une sauvegarde de l'image du noyau avant d'en installer un nouveau.

3.5 Optimisation du noyau

La configuration du noyau de base de Raspberry Pi est optimisée pour une utilisation orientée environnement de bureau. Il est possible d'utiliser directement cette configuration pour mettre en place la pile IEEE 802.15.4 et 6LoWPAN. Toutefois de nombreux modules non nécessaires sont installés avec chaque version du noyau. Ces modules supplémentaires augmentent le temps nécessaire à l'installation d'un nouveau noyau et occupent inutilement de la place sur la partition racine. En particulier la taille de l'ensemble des modules avec la configuration d'origine est de 286 MB. La taille de l'ensemble des modules après nettoyage de la configuration est de 7 MB. De plus le fait de travailler avec un noyau minimaliste permet d'éviter les problèmes liés aux sous systèmes inutilisés. Cela permet également de détecter plus facilement des problèmes de dépendance entre les sous-systèmes. Enfin cette optimisation diminue l'overhead en général ce qui rend le noyau mieux adapté pour une utilisation de type routeur à basse consommation. Cette section résume les différentes options modifiées dans la configuration du noyau. Toutefois la liste des options présentées dans cette section n'est pas exhaustive. La liste des options built-in supprimées du noyau d'origine est reprise dans l'Annexe B.2. De plus la liste des modules supprimés n'est pas reprise dans cette section. Celle-ci est composée en majeur partie de divers pilotes de périphériques. Cette liste est reprise dans l'Annexe B.1. Au total 845 options, modules et built-in compris, ont été supprimées du noyau.

La première option supprimée concerne les BSD process accounting. Cette option permet de mesurer et d'enregistrer la consommation des ressources par les différents processus. Un programme en espace utilisateur peut utiliser ces informations pour effectuer des tâches administratives et définir des quotas d'utilisation par utilisateur. Des options similaires permettant de regrouper des statistiques sur l'utilisation des ressources par processus sont également supprimées, notamment les options concernant les control groups. Ces options permettent de limiter l'utilisation des ressources en groupes de processus. Elles sont particulièrement utiles pour les systèmes orientés environnement de bureau lorsqu'elles sont utilisées pour optimiser l'ordonnanceur avec l'option auto-group. Une autre option supprimée concerne le support d'audit. Cette fonctionnalité est utilisée par certains sous-systèmes comme SELinux. Toutefois elle n'est pas utilisée par la couche Linux-ZigBee. Les options de profiling spécifiques au noyau sont également supprimées. En effet le profiling est réalisé ici d'une manière plus simple avec des compteurs entre différents points précis dans l'implémentation.

La configuration par défaut de Raspberry Pi concerne un noyau avec de fortes contraintes sur le temps. En particulier de nombreux points de préemptions sont rajoutées via diverses options dans la configuration. Comme le noyau n'est pas utilisé dans le cadre d'un environnement de bureau et qu'il n'a de traitements particuliers à effectuer, ces points de préemption supplémentaires sont inutiles. Premièrement l'option de préemption des Read-Copy-Update (RCU) est supprimée. Le RCU est une technique de synchronisation particulière utilisée dans le noyau. Cette option utilise une implémentation optimisée pour des systèmes temps réel. Le modèle de préemption du noyau est également changé d'un modèle qui optimise la latence du noyau à un modèle privilégiant le débit de traitement. Cela permet de tester le fonctionnement de la pile IEEE 802.15.4 dans ce modèle de préemption spécifique.

L'ordonnanceur d'entrées-sorties Completely Fair Queuing (CFQ) est remplacé par l'ordonnanceur deadline. En effet, CFQ suppose qu'il est nécessaire de réordonner les requêtes pour optimiser les accès au disque dur. Toutefois il n'est pas nécessaire d'effectuer ce réordonnancement dans le cadre d'entrées-sorties sur la carte SD. Il est possible de ne pas utiliser d'ordonnanceur et de regrouper les entrées-sorties dans une simple file FIFO. Toutefois cela peut causer des problèmes de famine lorsque de nombreuses entrées-sorties sont exécutées en parallèle. Au contraire l'ordonnanceur deadline offre un temps de service garantie pour les requêtes d'entrées-sorties. Le support pour les quotas d'utilisation du disque est également supprimé.

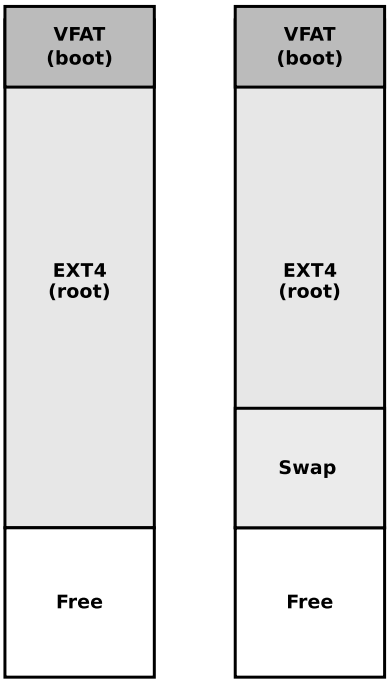
D'autres options diverses sont également supprimées. Premièrement les options liées à la mise en veille. En effet, le Raspberry Pi ne supporte pas la mise en veille. Les sous-systèmes 802.11 (WiFi), 802.15.1 (Bluetooth) et 802.16 (WiMAX) sont également supprimés. La gestion de certains périphériques d'entrées comme les souris, les joysticks et les écrans tactiles sont également supprimés. Le support pour la carte son, les périphériques d'entrées vidéos et les périphériques de sortie vidéo sont également supprimés. Finalement le support pour des systèmes de fichiers non utilisés est également supprimé. En particulier le support pour les systèmes de fichiers NFS et CIFS discutés à la section précédente sont supprimés. De plus certaines options sont également ajoutées comme l'option permettant la génération des coredump. Les coredump permettent d'enregistrer l'état de la mémoire d'un programme lorsque celui termine avec un signal d'abortion. En particulier cela facilite le débogage des programmes qui n'étaient pas a priori lancés dans le contexte d'un débogueur.

3.6 Outils de backup du Raspberry Pi

La méthode généralement utilisée pour effectuer les backups du Raspberry Pi consiste a effectuer une sauvegarde complète de la carte SD. Cette méthode possède l'avantage d'être facile à utiliser. En effet, il suffit d'utiliser les commandes suivantes pour respectivement sauvegarder ou restaurer la carte SD :

```
dd if=/dev/sdb of=sd-card.img
dd if=sd-card.img of=/dev/sdb
```

Toutefois cette méthode pose problème lorsque les cartes sont très larges. En effet cette méthode ne prend pas en compte l'existence des systèmes de fichiers sur la carte. Dès lors, chaque image sauvegardé est aussi large que la carte utilisée. Il est possible de réduire cette taille au détriment du temps de sauvegarde en compressant les images avec *bzip2*, *gzip* ou *xz*. Cette compression atteint de bons résultats lorsque la plupart des blocs de la carte n'ont pas encore été écrits et contiennent donc la même valeur. Toutefois à mesure que l'écriture se poursuit, les blocs de la cartes sont remplis avec des données. Dès lors, même s'ils ne sont plus référencés par le système de fichiers, ils rendent la tâche de l'algorithme de compression plus difficile. Il est important de noter également que si la sauvegarde constitue en une lecture complète de la carte. La restauration nécessite l'écriture de l'image sur l'entièreté de la carte. Comme tous les blocs sont réécrit au moins une fois par restauration cela limite la durée de vie des cartes. Un autre problème consiste en la restauration sur des cartes de taille différente. Lorsque



(a) Sans swap (b) Avec swap

FIGURE 3.1 – Agencement des partitions sur la carte.

la restauration a lieu sur une carte de taille supérieure à l'image cela ne pose pas de problèmes. Toutefois la restauration n'est pas possible lorsque la taille de la carte cible est inférieure à la taille de l'image. Cette section décrit la solution mise en oeuvre pour faciliter les backups et limiter ces problèmes sur le Rapsberry Pi.

La méthode utilisée pour résoudre ce problème consiste à prendre en compte l'existence des partitions sur la carte. Au lieu de copier chaque bloc de la carte, les partitions sont montées et leur contenu est sauvegardé. Cela permet de ne copier que les données associées au système de fichier ce qui limite la taille de la carte résultante. De plus cela offre plus de flexibilité au système de sauvegarde. En effet, il est alors possible de restaurer une sauvegarde sur une carte de taille différente. Toutefois cette méthode rend la procédure de backup plus complexe. En effet, il est nécessaire d'utiliser un script à part entière pour effectuer les sauvegardes et les restaurations. De plus il est nécessaire de prendre compte la possible existance de partitions supplémentaires pour la mémoire *swap*. Les paragraphes suivants discutent de l'implémentation de ces scripts.

La Figure 3.1 représente l'agencement des différentes partitions sur la carte du Raspberry Pi. Deux cas de figure sont généralement possibles. Dans les deux cas on retrouve la partition du bootloader. Cette partition contient un système de fichier *vfat*. Celui-ci contient les fichiers nécessaires au démarrage du Raspberry Pi. Par la suite on appellera cette partition, la partition *boot*. On retrouve également une partition qui contient un système de fichier *ext4*. Cette partition constitue le système de fichier racine du système d'exploitation se trouvant sur le Raspberry Pi. Par la suite on appellera cette partition, la partition *root*. On note également dans les deux cas un espace libre à la fin des partitions. Le fait de prendre en compte cette espace libre permet au script

d'ajuster de manière transparente la taille de la partition *root* à la taille de la carte. Plus particulièrement la Figure 3.1a représente le cas le plus courant où aucune partition n'est utilisée pour la mémoire swap. Dans ce cas un fichier dans la partition racine la remplace. La Figure 3.1b représente le cas où une partition est utilisée pour la mémoire swap. Cette solution apporte de meilleures performances dans le cas où la mémoire swap est utilisée. En effet, on évite alors l'overhead causé par le système de fichiers. Toutefois il faut noter que l'utilisation de la mémoire swap est déconseillée pour les systèmes embarqués. Dès lors, l'utilisation d'une partition à part entière revient souvent à gaspiller de la place sur la carte.

Le script de sauvegarde est repris dans l'Annexe C.1. La méthode utilisée par ce script consiste à archiver les données de l'ensemble des partitions dans un tarball. Pour cela l'ensemble des partitions de la carte est monté dans un répertoire temporaire. Les partitions de type swap sont ignorées. Les partitions sont montées de telle manière que le nom du répertoire de montage corresponde au numéro de la partition sur la carte. Les 512 premiers Octets de la carte qui constituent le Master Boot Record (MBR) et l'état de la table des partitions sont également sauvegardés dans ce répertoire. Ensuite la commande *tar* est utilisée pour créer un tarball qui préserve les attributs des fichiers. Cette archive est ensuite compressée avec la commande *bzip2*. Il est important de noter que cette archive n'est pas créé dans le répertoire `/tmp`. En effet, il est courant pour des raisons de performance de monter ce répertoire sur un système de fichiers tmpfs se trouvant dans la RAM. En conséquence l'espace disponible pour les fichiers temporaires s'en trouve réduite. D'autre part la taille de l'archive non compressée peut facilement dépasser plusieurs gigaoctets. L'archive est donc créée avec un nom temporaire dans le répertoire où se trouvent les images sauvegardées. Par mesure de sécurité un hash *SHA* est réalisé sur l'archive obtenue pour tester son intégrité a posteriori. Le script démonte alors les partitions. Finalement il efface les fichiers temporaires et renomme l'archive compressée avec le timestamp de la sauvegarde.

Le script de restauration est repris dans l'Annexe C.2. La méthode utilisée par le script de restauration est plus complexe que le script de sauvegarde. En effet, il est nécessaire en plus des fichiers de restaurer correctement les partitions. Tout d'abord l'archive est décompressée avec la commande *bunzip2*. Ensuite le script se charge de restaurer la table des partitions. Cette étape revient simplement à écrire le MBR sur la carte. Toutefois il est possible que la table des partitions se refère à une carte de taille différente. Dans ce cas il est nécessaire de réajuster la taille de la partition root. Le script suppose que cette partition est toujours en deuxième position. Ensuite il est nécessaire de formater à nouveau les partitions de table. Le script lit alors la table nouvellement créée et lance l'outil de formatage adéquat selon le type de la partition. Trois types de partitions différentes sont reconnues : les partitions vfat, ext4 et swap. Une fois le formatage terminé, le script monte les partitions de données dans un répertoire temporaire. De manière similaire au script de sauvegarde, les partitions sont montées de telle manière que le nom du répertoire de montage corresponde au numéro de la partition sur la carte. Le tarball est alors extrait dans ce répertoire temporaire et les fichiers sont associés à leur partition respective. On émet ensuite la commande *sync* pour s'assurer que l'écriture sur les partitions est effective. Le script démonte finalement les partitions et efface les fichiers temporaires.

Date	**Taille**
28 janvier 2013	$333,2$ MiB
1 mars 2013	$390,1$ MiB
6 mai 2013	$320,8$ MiB
19 mai 2013	$211,5$ MiB
Moyenne	$313,9$ MiB

TABLE 3.2 – Taille des images de sauvegarde de la carte.

Algorithme	**Taille**	**Temps (compression)**	**Temps (décompression)**
–	$958,4$ MiB	–	–
gzip	$364,0$ MiB	$1m57s$	$21s$
		$44s$	$10s$
bzip2	$333,2$ MiB	$4m48s$	$1m49s$
		$1m19s$	$35s$
xz	$265,0$ MiB	$18m33s$	$48s$

TABLE 3.3 – Performance des algorithmes de compression.

Méthode	**Taille**	**Temps (backup)**	**Temps (restore)**
dd	$7,8$ GiB	$43m32s$	$1h38m34s$
backup/restore.sh	$211,5$ MiB	$2m18s$	$1m42s$

TABLE 3.4 – Comparaison des méthodes de backup.

La Table 3.2 reprend la taille de quatre archives obtenues avec ce système de backup. On constate que la taille des archives décroit fortement vers la fin du projet. Cela est dû à l'optimisation du noyau décrit à la Section 3.5. La taille moyenne d'une archive avec ce système est de $313,9$ MiB. Il est possible d'utiliser différents algorithmes pour compresser les images sauvegardées. La Table 3.3 reprend la taille obtenue après compression, le temps de compression et de décompression pour différents algorithmes. La première ligne spécifie la taille de l'archive d'origine. Dans le cas des algorithmes gzip et bzip2, une version optimisée pour les processeurs multicoeurs est également utilisée. Les caractéristiques de la machine utilisée pour ces tests sont reprises à la Table 3.1. On constate que le meilleur résultat de compression est obtenu avec l'algorithme xz. Toutefois même si le temps de décompression reste raisonnable, le temps de compression est nettement supérieur aux autres algorithmes. C'est ce temps de compression en particulier qu'on cherche à minimiser. En effet, il est plus fréquent de sauvegarder les images que de les restaurer. Les algorithmes gzip et bzip2 offrent des résultats de compression du même ordre de grandeur. L'algorithme gzip offre les temps de compression plus rapide. Toutefois la version parallélisée de bzip2 offre des temps sensiblement similaires pour un meilleur ratio de compression. Si on tient compte de la faible fréquence des sauvegardes, le ratio de compression prime sur le temps de compression. C'est donc l'algorithme bzip2 qui est utilisé.

La Table 3.4 compare les deux systèmes de backup. Cette table reprend pour chaque méthode la taille de l'image résultante, le temps de sauvegarde et le temps de restauration. La première méthode utilise une sauvegarde complète de la carte avec la commande dd. La seconde utilise les scripts backup et restore. Dans les deux cas les images résultantes sont compressées avec l'algorithme bzip2. Les deux tests sont réalisés avec une carte de 16 Go. Le contenu de la carte est identique au bloc prêt pour chaque test. Les caractéristiques de la machine utilisée pour ces tests sont reprises à la Table 3.1. On constate que la méthode de backup avec les scripts est beaucoup plus performante aussi bien en terme de temps de sauvegarde, temps de restauration et espace occupé par l'image. Cela permet de faciliter le développement. En effet, les sauvegardes sont effectuées fréquemment dans le projet. Il n'est pas rare que les alimentations soient débranchés incorrectement et que des données soient perdues sur la carte. Dans ce cas de figure les opérations de sauvegarde et de restauration doivent être rapides. De plus malgré le fait que ces scripts soient adaptés au Raspberry Pi, il est facile de les adapter pour d'autres types de plateformes où l'agencement des partitions reste simple.

3.7 Outils de débogage pour les réseaux de capteurs sans fil

Lors de la manipulation de réseaux IEEE 802.15.4 il est utile de disposer d'outils permettant de capturer et de manipuler le trafic. Ces outils sont particulièrement utiles lorsque le sous-système IEEE 802.15.4 et les couches supérieures sont encore au stade expérimental. Certains outils [21] permettent déjà de capturer le trafic sous forme d'un fichier *PCAP*. Ce format est utilisé par de nombreux programmes pour stocker le résultat de captures du réseau. Les programmes Wireshark et Tcpdump en particulier utilisent ce format. Toutefois cette méthode présente quelques limitations. Premièrement il est difficile d'observer directement l'état des trames sans passer par un programme externe pour visualiser le fichier PCAP résultant. Deuxièmement ces outils ne permettent pas d'injecter du trafic dans le réseau. Finalement ces outils ne permettent pas de manipuler les trames capturées de manière à en extraire les différentes composantes. Ces deux derniers points permettent d'effectuer une série de capture-replay pour simuler par exemple la présence d'un noeud sur le réseau. Cette section décrit l'outil WSN-Tools [22] développé pour répondre à ces limitations.

Ce projet comprend un ensemble d'outils écrits en C pour les réseaux IEEE 802.15.4. Ces outils permettent de manipuler des trames MAC 802.15.4, de mettre en place un sniffer, d'injecter et de rejouer du trafic directement depuis la ligne de commande. Ces outils agissent comme des clients pour des transceivers connectés par UART au moyen d'un protocole spécifique. Ce protocole permet d'adapter facilement les clients à de nouveaux transceivers. Il se base sur une unité de transmission appelé message. Chaque message comporte un byte d'information suivi de 0 à 127 Octets de données. Ce byte d'information spécifie le type de message et sa taille. Un message peut être soit un message de trame ou un message de contrôle. Les messages de trame sont utilisés pour transmettre une trame depuis et vers le transceiver. L'entièreté des données suivant le byte d'information son utilisés pour représenter la trame. Les messages de contrôle sont utilisés pour transmettre des commandes et des réponses depuis et vers le transceiver. En particulier ces messages permettent de configurer le transceiver, de transmettre des messages textuels et de signaler les succès ou les erreurs.

L'outil *WSN-Sniffer-CLI* est un client pour un transceiver utilisé comme sniffer. Le programme est capable d'enregistrer la capture dans un fichier PCAP. Il peut également analyser les trames MAC 802.15.4 d'une manière autonome et les afficher directement à l'utilisateur. L'outil *WSN-Injector-CLI* permet d'injecter des trames sur le réseau. Il permet également de décomposer, construire et désassembler des trames IEEE 802.15.4. Il peut sélectionner et remplacer différentes parties de la trame comme l'en-tête ou la charge utile. Il peut également modifier les éléments internes à l'en-tête de la trame MAC dont le type de trame, les adresses sources et destinations et le numéro de séquence. La commande *WSN-Ping-CLI* permet de vérifier le type de firmware présent sur le transceiver. Cette commande permet aussi de tester le protocole et l'envoi de messages sur l'UART. Finalement l'outil *PCAP-Selector* permet d'extraire rapidement une trame d'un fichier PCAP pour la manipuler ou la réinjecter.

Ces outils ont été utilisés pour déboguer la couche MAC IEEE 802.15.4 et la couche d'adaptation 6LoWPAN du noyau. En particulier ils ont permis de simuler des noeuds 6LoWPAN pour résoudre des problèmes avec l'implémentation de Linux-

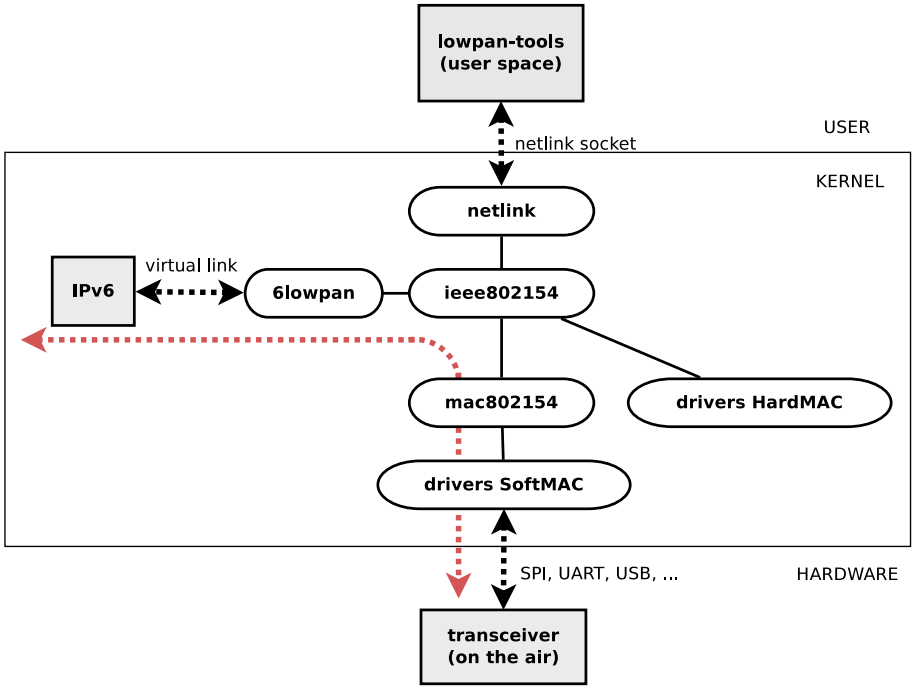


FIGURE 3.2 – Composition du projet Linux-ZigBee dans le noyau.

ZigBee. Le Chapitre 4 décrit cette simulation en détails. Il est important de noter également que ces outils sont publiés sous la license libre GPLv3. Ils ont également été utilisés par d'autres personnes travaillant sur les réseaux de capteurs sans fils.

3.8 Rétroportage de la couche IEEE 802.15.4

La couche IEEE 802.15.4 de Linux-ZigBee n'est pas disponible dans le noyau 3.2.27 utilisé sur l'image Raspbian. Il est donc nécessaire de rétroporter les changements depuis la branche de développement de Linux-ZigBee. Le projet Linux-ZigBee se base sur la branche de développement *net-next* du noyau Linux. Cette branche de développement regroupe les changements qui ont trait aux sous-systèmes orientés réseau du noyau Linux. Les patchs acceptés par les mainteneurs sont proposés sur cette branche avant d'être acceptés dans le noyau. Le rétroportage est réalisé directement depuis le système de contrôle de version Git afin de conserver l'historique entre les différentes branches. Il faut noter également que le rétroportage des fichiers seuls ne suffit pas et des fixes sont souvent nécessaires pour adapter les sources aux changements dans les sous-systèmes du noyau Linux.

Le projet Linux ZigBee constitue un sous-système du noyau. Ce sous-système peut lui même se décomposer en plusieurs parties. Premièrement la partie *ieee802154* permet la gestion des interfaces. Elle s'occupe également des interaction avec la couche utilisateur. Deuxièmment la partie *mac802154* implémente le MLME pour les périphériques SoftMAC. Troisièmement la partie *6lowpan* permet la mise en place de l'interface d'adaptation d'IPv6 vers 6LoWPAN. Et finalement les *drivers* qui contient les pilotes pour les transceiver IEEE 802.15.4. La Figure 3.2 illustre l'agencement de ces différents composants dans le noyau. On remarque également la présence d'un nouveau compo-

sant *netlink*. Celui-ci est un sous composant de ieee802154. Il s'occupe spécifiquement de l'interaction avec la couche utilisateur via des sockets netlink. Les sockets netlink constituent une interface de communication entre entre l'espace utilisateur et le noyau.

On peut voir également la dépendence qui existe entre les différents composants. Ceux-ci s'articulent autour du composant ieee802154. Les composants mac802154 et 6lowpan par exemple permettent la création d'une interface et enregistre les opérations du MLME auprès du composant ieee802154. Dans le cas de 6lowpan les opérations du MLME sont reprises depuis l'interface d'origine du lien virtuel déjà enregistrée dans le composant ieee802154. Cette figure illustre également le chemin suivit par un paquet IPv6 vers ou depuis le transceiver. On peut voir que le paquet transite à travers les différents composants du sous-système. Ces transitions restent bien délimitées dans le noyau à la différence des autres solutions décrites dans le Chapitre 2 qui utilisent un composant en espace utilisateur. Le cheminement des paquets à travers les différents composants est décrit avec plus de détails à la Section 3.11.

Le rétroportage consiste à récupérer les commits portant sur le sous-système Linux-ZigBee dans la branche net-next et les intégrer au sommet de la branche 3.2.27 du noyau du Raspberry Pi. Cela permet de continuer à recevoir les fixes du noyau du Raspberry Pi ainsi que les fixes et nouvelles fonctionnalités intégrées à la branche net-next. Toutefois il subsiste deux problèmes avec cette méthode. Premièrement il existe de nombreuses différences au niveau des API depuis la version 3.2.27 à la branche de développement net-next du noyau. Ces problèmes empêchent la compilation. Il est alors nécessaire d'adapter ces changements d'API. Ces changements sont relativement simples. En effet, dans la plupart des cas ils consistent simplement à modifier le nom de certaines fonctions et structures, adapter les prototypes de fonctions ou adapter le code à un emploi différent de la syntaxe. Il faut noter toutefois que ces changements même s'ils sont simples restent très nombreux.

Un autre problème relatif aux changements entre les deux branches du noyau se pose lorsqu'un sous-système utilisé dans la branche de développement n'existe pas encore dans la version 3.2.27. Dans ce cas deux solutions sont alors possible. La première consiste à porter le sous-système manquant également dans la version 3.2.27. Toutefois cette solution peut également introduire d'autres changements au niveau de l'API. La seconde solution consiste à modifier légèrement le code pour éviter d'avoir à utiliser ce sous-système. Ce problème généralement est plus complexe à résoudre car il nécessite une compréhension plus fine des différents sous-systèmes impliqués. Toutefois il est moins fréquent et ne s'est produit qu'une seule fois lors de rétroportage.

Le fait de travailler dans un noyau beaucoup plus vieux que la branche de développement pose également un problème lorsque des fixes sont réalisés dans l'implémentation. Dans ce cas il est nécessaire de porter les patchs depuis la version 3.2.27 vers la branche net-next utilisée pour le développement. Par mesure de prudence on vérifie toujours que les patchs s'appliquent correctement sur la branche net-next et que celle-ci compile toujours. Toutefois ces contraintes accentuent considérablement le temps requis pour la réalisation des patchs.

Une fois le rétroportage terminé on peut activer la couche MAC IEEE 802.15.4, la sous-couche 6LoWPAN ainsi que les drivers dans le noyau. Pour cela il faut ajouter les options suivantes à la configuration :

— `CONFIG_IEEE802154`
— `CONFIG_IEEE802154_6LOWPAN`
— `CONFIG_MAC802154`
— `CONFIG_IEEE802154_DRIVERS`
— `CONFIG_IEEE802154_FAKEHARD`
— `CONFIG_IEEE802154_FAKELB`
— `CONFIG_IEEE802154_MRF24J40`

Ces options sont compilées sous forme de module afin de faciliter le développement. En effet, cela permet de recharger uniquement le module concerné lorsqu'un module est modifié. Des options en builtin obligent à recompiler et à réinstaller le noyau.

3.9 Installation du transceiver

Une fois le noyau capable de gérer la couche MAC IEEE 802.15.4 et la sous-couche 6LoWPAN, il est nécessaire d'installer le transceiver. Toutefois il n'est pas possible d'utiliser ici un mécanisme d'autodétection pour charger le driver adéquat au démarrage du noyau. Il est donc nécessaire de procéder à des changements dans le code de l'architecture propre au SoC BCM 2835. Cette section reprend la procédure d'installation du transceiver ainsi que les changements à effectuer au noyau.

Le transceiver utilisé ici est un MRF24J40MA de Microchip. Il s'agit d'un transceiver IEEE 802.15.4 qui intègre la couche MAC et PHY. Toutefois ce transceiver n'intègre pas le MLME. Il est donc interfacé par le composant mac802154. Le montage du circuit est repris à l'Annexe D. Le transceiver est connecté au Raspberry Pi via un bus Serial Peripheral Interface (SPI). Celui-ci permet l'échange synchrone de données entre deux périphériques. Deux pins MISO et MOSI permettent le transfert bit par bit avec le transceiver. Un signal d'horloge de 10 MHz est généré par le Raspberry Pi pour synchroniser ces transferts. Finalement un pin Chip Select (CS) est utilisé par le Raspberry Pi pour signaler un transfert au transceiver. Le transceiver possède également un pin d'interruption pour signaler des évènements. Ces interruptions déclenchent à leur tour un transfert SPI de la part de Raspberry Pi pour lire un registre contenant le status de l'interruption.

La configuration du bus SPI dans le noyen repose sur deux types de pilotes particulier : les controller drivers et les protocol drivers. Les controller drivers gèrent le caractère bas niveau du transfert sur le bus SPI. En particulier ils s'occupent de configurer les registres propres au SoC. Les protocol drivers s'occupent uniquement des messages qui transitent sur le bus SPI. En d'autre termes ils s'occupent du protocole associé au bus SPI. La configuration par défaut du SoC BCM 2835 du Raspberry Pi utilise les protocol drivers SPI en espace utilisateur (spidev). Celui-ci est souvent utilisé sur le Raspberry Pi pour manipuler le bus SPI depuis des scripts en Python. Cela permet de simplifier le développement d'applications nécessitant l'utilisation du SPI.

La configuration d'un nouveau périphérique, en l'occurence un transceiver MRF24J40 nécessite la configuration du protocol driver pour le chip select associé dans le fichier de configuration du SoC (`arch/arm/mach-bcm2708/bcm2708.c`). Pour cela il est nécessaire de modifier la structure `spi_board_info` du protocol drivers associé au chip select 0 dans le tableau `bcm2708_spi_devices`. La Table 3.5 reprend la valeur

.modalias	“mrf24j40”
.max_speed_hz	10000000
.bus_num	0
.chip_select	0
.mode	SPI_MODE_0

TABLE 3.5 – Configuration du protocol drivers pour le MRF24J40.

des membres de cette structure. Le noyau utilise la valeur de la chaîne `modalias` pour charger le module associé à ce protocol drivers. Le membre `max_speed_hz` spécifie la fréquence de l'horloge pour le SPI. Le membre `bus_num` spécifie le SPI master parent du protocol drivers. Le membre `chip_select` spécifie le pin à utiliser pour signaler les transferts SPI. Finalement le membre `mode` spécifie le mode de transfert à utiliser. Ce mode spécifie la polarité et la phase à utiliser pour synchroniser les transferts sur le signal de l'horloge. Le membre `irq` de cette structure permet de configurer l'interruption associé au périphérique géré par le protocol drivers. Cette interruption est associée ici au pin GPIO 25. Ce GPIO est lui même relié au pin d'interuption du transceiver.

3.10 Débogage du MRF24J40

Le noyau lance à chaque démarrage le protocol drivers associé au transceiver. Le pilote du MRF24J40 dépend lui même du sous-système ieee802154 auquel il s'associe. Si on charge également le module 6lowpan, cette configuration devrait suffire à effectuer des transferts à partir du transceiver. Toutefois les tests ont montrés qu'il subsistait encore des problèmes au niveau du pilote. En effet après des transferts intensifs il est fréquent que les pilotes se bloquent et ne répondent plus aux interruptions des transceivers. Cette section décrit les problèmes rencontrés avec le pilote et les fixes créés pour résoudre ces problèmes.

Le transceiver maintient la ligne d'interruption à un niveau haut aussi longtemps qu'aucune interruption n'est déclenchée. Lorsqu'une interruption est déclenchée, la ligne passe à un niveau bas. Ce changement d'un niveau haut vers bas est détecté par Linux qui signale l'interruption à la fonction Interrupt Service Routine (ISR) du pilote. Cette fonction désactive alors la ligne d'interruption et traite l'évènement dans un thread séparé pour éviter de bloquer l'ISR avec les transferts SPI. Dès que le registre de status d'interruption (INTSTAT) est lu sur le MRF24J40, la ligne est remise à un niveau haut par le transceiver. Le pilote réactive la ligne et d'autres interruptions peuvent être capturées.

Toutefois lorsqu'un ping intensif est réalisé entre les transceivers, la ligne d'interruption se bloque sur un niveau bas après quelques secondes. Ce problème est dû à une race condition au niveau de la désactivation de la ligne d'interruption. La Figure 3.3 illustre l'évolution du niveau de la ligne d'interruption lorsqu'une race condition se produit. La ligne d'interruption commence à un niveau haut. Au temps **A**, une interruption

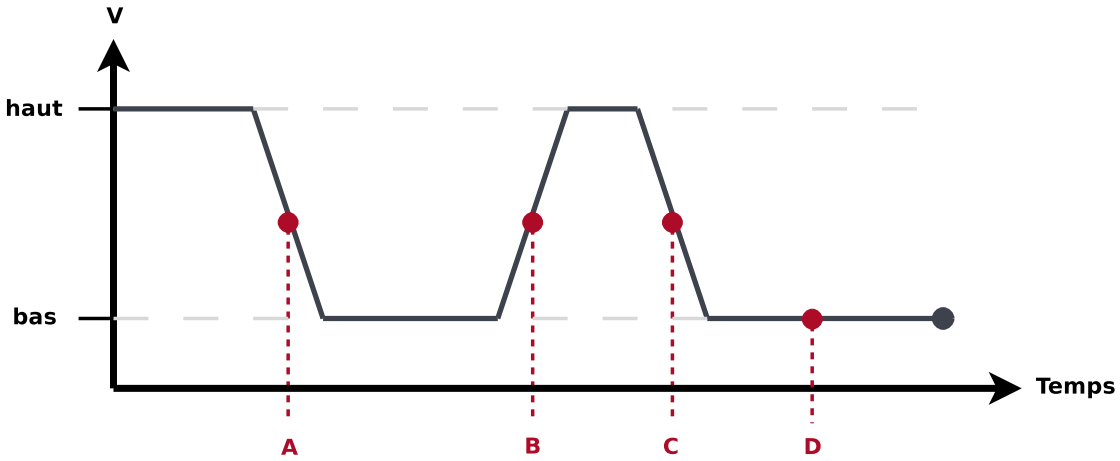


FIGURE 3.3 – Niveau de la ligne d'interruption dans le cas d'une race condition.

est déclenchée. Le transceiver met le niveau de la ligne à bas. Ce changement de niveau est détecté par Linux. Celui-ci lance l'ISR qui désactive immédiatement la ligne d'interruption. Le travail est ensuite transféré vers un autre thread pour éviter de bloquer dans un contexte non interruptible. Au temps **B**, le registre INTSTAT est lu. Le pilote effectue l'opération adéquate selon le type d'interruption. Le transceiver quant à lui replace le niveau de la ligne à haut. Peu après au temps **C**, une nouvelle interruption est déclenchée. Toutefois comme la ligne d'interruption est toujours désactivée, cette interruption n'est pas capturée par le noyau. Finalement au temps **D**, le traitement de la première interruption est terminé. Le thread réactive alors la ligne d'interruption. Cependant la ligne d'interruption reste à un niveau bas jusqu'à ce que le registre INTSTAT soit lu. Comme aucun changement de niveau ne déclenche à nouveau l'ISR, le registre n'est jamais lu et la ligne d'interruption reste indéfiniment au niveau bas.

La solution utilisée consiste à ne pas désactiver la ligne d'interruption dans l'ISR. Cela ne porte pas à conséquence car l'ISR est exécuté dans un contexte non interruptible. Le travail transféré au thread annexe quant à lui est sérialisé à travers une FIFO. Après discussion sur la mailing list, une autre solution proposée consiste à remplacer les interruptions *edge-triggered* par des interruptions *level-triggered*. Avec ces dernières le noyau signale les interruptions tant que le niveau de la ligne reste bas. Dans ce cas la désactivation de la ligne d'interruption permet de sérialiser le traitement de celles-ci. Qui plus est le fait de manquer une interruption ne bloque plus le pilote. En effet, le noyau signale alors l'interruption juqu'à ce qu'elle soit traitée et que le niveau de la ligne remonte à haut. Toutefois cette solution pose deux problèmes. Premièrement les interruptions level-triggered ne sont pas gérées pour les GPIO dans l'architecture du SoC BCM2835 intégrée au noyau du Raspberry Pi. Deuxièmement Linus Torvalds recommande fortement de ne pas utiliser les interruptions level-triggered :

> *"If you have a driver that requires level-triggered interrupts, then your driver is arguably buggy. NAPI or no NAPI, doesn't matter. Edge-triggered interrupts is a fact of life, and deciding that you don't like them is not an excuse for saying "they should not work".*" [54]

Ce bogue en particulier met en évidence un autre problème. La désactivation de la ligne d'interruption ne devrait provoquer de race condition. En effet, les interruptions

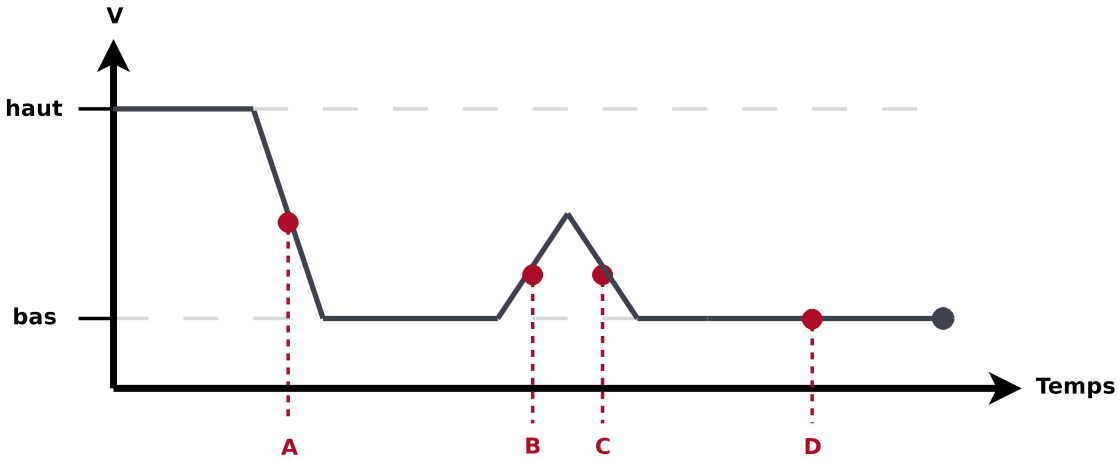


FIGURE 3.4 – Niveau de la ligne d'interruption dans le cas d'une race condition.

manquées lorsque la ligne est désactivée sont normalement rejouées par le noyau dès que la ligne est réactivée. Toutefois ce n'est pas le cas sur le Raspberry Pi ce qui conduit à des interruptions manquées. Ce phénomène est à l'origine d'une incompréhension sur la mailing list. En effet, ce problème n'apparait pas avec d'autres architectures. Dès lors, le bogue est difficilement reproductible. Toutefois un autre problème beaucoup plus rare peut amener à perdre une interruption. La Figure 3.4 illustre l'évolution du niveau de la ligne d'interruption pour ce problème. Comme précédemment la ligne d'interruption commence à un niveau haut. Au temps **A**, une interruption est déclenchée. Le transceiver met le niveau de la ligne à bas. Ce changement de niveau est détecté par Linux. Celui-ci lance l'ISR similairement au cas précédent. Au temps **B**, le registre INTSTAT est lu. Le transceiver replace le niveau de la ligne à haut. Au temps **C** et avant que la ligne d'interruption n'ait pu remonter totalement à haut, une autre interruption est déclenchée. Le transceiver remet le niveau de la ligne à bas. Toutefois la descente n'est pas assez marquée pour être détectée. Dès lors, l'interruption est manquée et au temps **D**, la ligne reste définitivement au niveau bas.

Finalement un dernier problème peut encore se produire lorsqu'une trame est reçue alors qu'une autre est en cours d'envoi. Lorsque ce cas se présente il arrive que le transceiver se bloque et n'émette plus d'interruption. Pour remédier à cela on utilise une section critique qui protège l'envoi et la réception. Cette solution nécessite également une seconde file de travail pour éviter un possible deadlock. Une autre solution proposée après discussion sur la mailing list utilise les *threaded interrupts*. Celles-ci permettent de se passer de la section critique et de la seconde file d'attente. Toutefois elle utilise également les interruptions *level-triggered* qui ne sont pas supportées sur le Rasperry Pi.

Une fois ces solutions mises en place, il est possible d'utiliser le transceiver sans problème pendant des heures et cela avec toute sorte de trafic. Toutefois l'utilisation d'une section critique pour protéger l'envoi et la réception de trames provoque de nombreuses pertes de paquets. Le Chapitre 4 illustre en détails ce problème et ses implications.

3.11 Débogage de 6LoWPAN

Des tests ont mis en évidence deux problèmes au niveau de la couche 6LoWPAN. En effet lors du passage à une nouvelle version du noyau, le trafic utilisant les adresses link-local a cessé de fonctionner. Ce problème affecte la compression IPHC et la décompression des adresses link-local basées sur les adresses MAC. (pour rappel voir la Section 1.7.1). Lorsque celui-ci fut résolu un second problème est apparu relatif cette fois à la fragmentation. Cette section commence tout d'abord par décrire le fonctionnement de la couche 6LoWPAN dans le sous-système Linux-ZigBee. Ensuite elle cible le problème de compression dans le code et décrit la solution.

La Figure 3.5 représente le cheminement d'un paquet IPv6 vers et depuis un lien IEEE 802.15.4. Elle met en particulier l'accent sur la transformation d'un paquet IPv6 en paquet 6LoWPAN et inversément. La gestion de la fragmentation est omise par souci de simplicité. Les noeuds sont associés à des fonctions dans les composants 6lowpan et mac802154. Il est important de noter que cette figure ne représente pas exactement un graphe d'appel des fonctions mais d'une représentation des différentes étapes dans le cheminement des paquets. La transmission de paquets IPv6 vers une interface IEEE 802.15.4 a lieu à travers un lien virtuel. Ce lien compresse les paquets IPv6 en paquets 6LoWPAN avant de les transférer sur l'interface IEEE 802.15.4 (*wpan0*). Inversément ce lien décompresse les paquets 6LoWPAN en paquets IPv6 avant de les faire sortir du lien virtuel (*lowpan0*).

Dans le cas de paquets IPv6, le cheminement a lieu depuis l'interface lowpan0 vers l'interface wpan0. Tout d'abord, La fonction `lowpan_header_create` transforme l'en-tête IPv6 en en-tête 6LoWPAN. Cette transformation utilise le mode de compression IPHC pour diminuer la taille de l'en-tête résultante. En particulier elle utilise les adresses MAC source et destination pour choisir au mieux le mode d'adressage du paquet 6LoWPAN. Par exemple dans le cas d'une adresse link-local basée sur l'adresse MAC, celle-ci est totalement omise de l'en-tête. Elle appelle ensuite la fonction `dev_hard_header` pour créer l'en-tête MAC. Celle-ci appelle la fonction de création de l'en-tête MAC associée à l'interface wpan0. Dans ce cas il s'agit de la fonction `mac802154_header_create` du composant mac802154. Dans le cas d'un paquet 6LoWPAN, le cheminement a lieu depuis l'interface wpan0 vers l'interface lowpan0. Tout d'abord la fonction `mac802154_header_parse` extrait les adresses MAC source et destination de l'en-tête MAC. Ensuite la fonction `lowpan_rcv` analyse l'en-tête 6LoWPAN. Si l'en-tête indique que le paquet n'est pas compressé alors celui-ci est transmis directement à l'interface lowpan0. Dans le cas contraire si le paquet est compressé avec le mode IPHC ou consitue un fragment alors il est transmis à la fonction `lowpan_process_data`. Cette fonction transforme l'en-tête 6LoWPAN en l'en-tête IPv6. Le paquet est alors transmis à la fonction `lowpan_skb_deliver` qui le transmet à son tour à l'interface lowpan0.

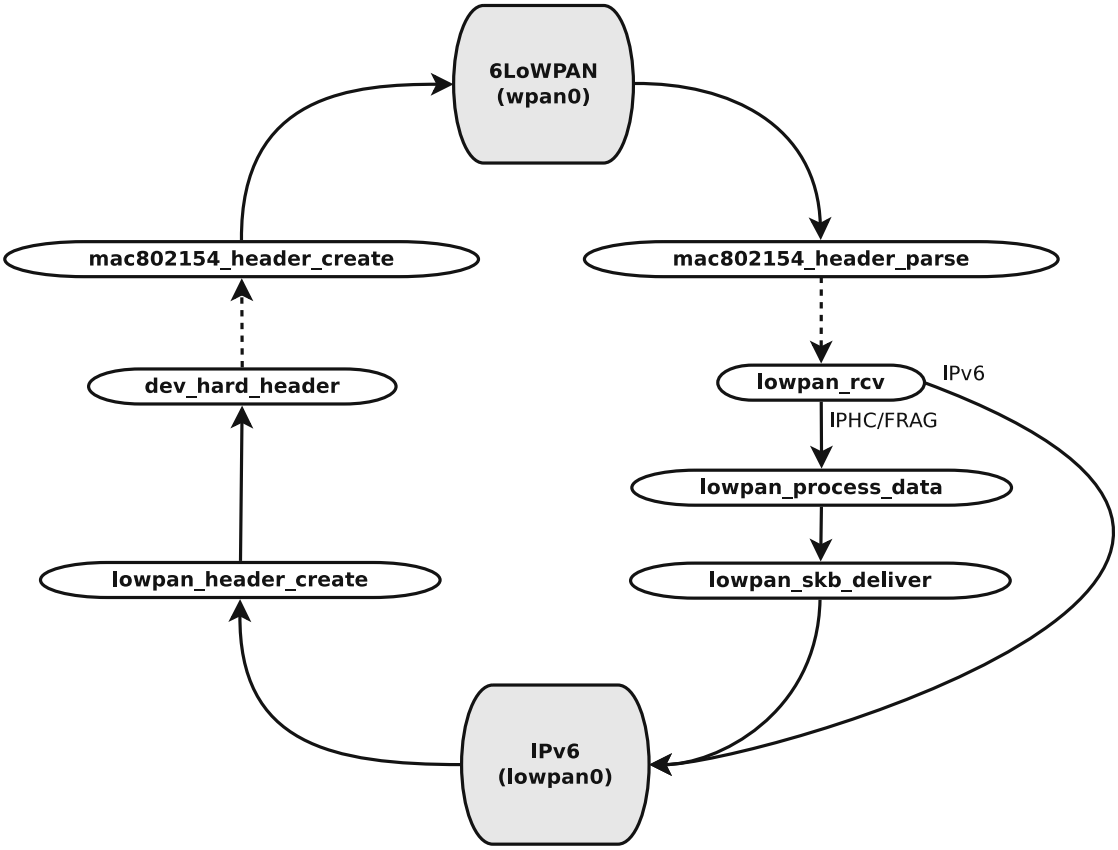


FIGURE 3.5 – Transmission et réception via 6LoWPAN.

Le premier problème survient dans le cas de la décompression d'un paquet 6LoWPAN en paquet IPv6. En particulier dans `lowpan_uncompress_addr` où l'adresse IP est reconstituée depuis l'adresse MAC. Dans le code d'origine l'adresse MAC n'est pas transmise correctement à cette fonction. Dès lors, la décompression d'adresses compressée depuis l'adresse MAC ne fonctionne pas. Toutefois ce problème n'apparait pas dans la version 3.2.27 du noyau. En effet, une erreur au niveau de la gestion des adresses MAC empêche la fonction de compression des adresses IPv6 vers 6LoWPAN de rattacher les adresses MAC aux adresses link-local. Dès lors, la compression basée sur les adresses MAC est toujours évitée.

Le second problème apparait lorsque des paquets fragmentés sont envoyés via l'adresse link-local sur le nouveau noyau. Dans ce cas les adresses source et destination des paquets IPv6 réassemblés depuis leur fragment 6LoWPAN sont incorrectes. Pour décrire l'origine de ce problème il est nécessaire d'expliquer le fonctionnement de la fragmentation dans le composant 6lowpan. Lorsqu'un nouveau premier fragment 6LoWPAN est reçu, un buffer est créé pour le paquet réassemblé. A mesure que les fragments arrivent, leur contenu est réassemblé dans ce buffer. Une fois le dernier fragment reçu, le buffer du paquet réassemblé est substitué à celui du dernier fragment. La fonction continue alors son processus de transformation du paquet 6LoWPAN en paquet IPv6. C'est alors le paquet réassemblé qui est utilisé. De plus dans le cas d'une adresse compressée depuis l'adresse MAC, l'adresse MAC est extraite d'un élément particulier du buffer, le *control block*. Cet élément est utilisé pour enregistrer des informations par paquet. Dans ce cas il est utilisé pour enregistrer des informations relatives à la couche MAC. Toutefois lors de la création du buffer pour le paquet réassemblé, le control block n'est pas copié dans le code d'origine. Dès lors, l'adresse MAC extraite de ce control block n'est pas correcte et par conséquent l'adresse IP reconstituée n'est pas correcte non plus.

3.12 Implémentation supplémentaires

Des éléments supplémentaires ont été intégrés au pilote du MRF24J40. Certains de ces éléments sont utilisés uniquement pour faciliter les tests. D'autres en revanche ont été codés en vue d'une future intégration dans le code officiel. Cette version spéciale du pilote utilisée uniquement dans ce projet est appelé le *pilote d'audit*. C'est cette version qui est incluse dans les patchs à appliquer au noyau ainsi que dans la dernière version du noyau 3.8 patché. Il est possible d'activer ou de désactiver les fonctionnalités via les options du module.

La Table 3.6 reprend la liste des options disponibles pour le pilote d'audit. La première permet d'activer le mode turbo associé au transceiver MRF24J40. Ce mode permet des débit théoriques jusqu'à 625 Kbps. La seconde permet de forcer la désactivation des ACK request. Cette option est utile pour tester l'impact des ACK le trafic entre les transceivers. La troisième permet de désactiver la ligne d'interuption. Cette option est utile pour déboguer le problème relatif aux interruptions décrit dans la Section 3.10. La quatrième option permet de désactiver la protection de la transmission et de la réception. Cette option permet de tester l'impact du fixe décrit dans la Section 3.10 sur le trafic. Finalement la cinquième et dernière option permet de récolter des statis-

Option	Description
turbo	Mode turbo (625 Kbps)
noack	Désactive les ACK request
dirq	Désactive la ligne d'interruption
notxrx	Désactive la protection TXRX
irq_stats	Récolte des statistiques sur le traitement des IRQ

TABLE 3.6 – Options disponibles pour le pilote d'audit.

tiques sur le temps passé à traiter chaque IRQ. En particulier cette option permet de mesurer la contribution au RTT moyen du traitement des paquets dans le pilote.

3.13 Installation du noyau 3.8

Lors des derniers tests de validation entre les transceivers, le noyau générait des erreurs non récupérables (*kernel panic*) pour un test relatif à la fragmentation. Ces erreurs sont dûes à un problème dans la gestion mémoire du noyau 3.2.27. Afin de poursuivre les tests, l'unique solution envisageable consiste en l'utilisation d'un noyau plus récent dans lequel ce problème serait corrigé. En particulier on utilise ici le noyau 3.8. Toutefois cette migration nécessite de réitérer le processus de rétroportage décrit à la Section 3.8. Cependant étant donné le fait que le noyau cible est plus récent, le rétroportage fut d'autant plus facile. En effet, la plupart des éléments du sous-système Linux-ZigBee sont déjà présents dans le noyau 3.8. De plus les API et les différents sous-systèmes utilisés correspondent. Dans le chapitre qui suit la plupart des tests de validations ont été effectués avec les deux noyaux (3.2.27 et 3.8). Uniquement les tests posant problème avec le noyau 3.2.27 ont été testé avec le noyau 3.8 seul. Lorsque c'est le cas cela est précisé dans l'énoncé du test.

Chapitre 4

Validation

Ce chapitre reprend les différents tests réalisés pour valider l'implémentation. En particulier ce chapitre comprend les tests suivant :

— Simulation de ping
— Ping
— UDP et TCP
— Fragmentation
— Mesure de débit
— Benchmark du pilote
— Static routing
— Configuration du préfixe avec radvd
— Interaction avec Contiki

Ces tests ont été effectués avec les noyaux 3.2.27 et 3.8. Le résultat des mesures provient généralement du noyau 3.2.27. Lorsqu'une différence significative des résultats par rapport au noyau 3.8 est observée, cette différence est signalé dans le test. Cependant aucune différence significative n'a été observée parmis tous les tests. C'est pourquoi le cas n'est jamais soulevé. Dans le cas du test pour le static routing et la mesure de débit seul le noyau 3.8 a été utilisé. En effet, comme discuté à la Section 3.13 du chapitre précédent, une erreur dans la gestion mémoire du noyau 3.2.27 provoque des kernel panic dans certains cas et empêche la réalisation du test.

4.1 Configuration

Cette section préliminaire reprend les commandes nécessaires pour monter une interface 6LoWPAN avec Linux-ZigBee et le MRF24J40. Pour faciliter cette tâche, un script effectue ces opérations automatiquement. Ce script est repris dans l'Annexe E.1. Les commandes sont exécutées ici avec l'utilisateur root. Lorsque cela est utile le résultat de la commande est également montré.

La premiere commande consiste à lister les interfaces physiques disponibles. Cette commande montre les interfaces physiques qui peuvent servir pour créer des interfaces IEEE 802.15.4 :

```
# iz listphy
wpan-phy0 IEEE 802.15.4 PHY object
```

```
    page: 0  channel: n/a
    channels on page 0: 11 12 13 14 15 16 17 18 19 20 21 \
                        22 23 24 25 26
```

Ensuite on peut rajouter une interface MAC IEEE 802.15.4 à une interface PHY spécifique. Cette commande va créer l'interface MAC associée et retourner son nom. Dans le cas suivant l'interface MAC est associée à *wpan-phy0* et l'interface MAC crée se nomme *wpan0* :

```
# iz add wpan-phy0
Registered new device ('wpan0') on phy wpan-phy0
```

On configure ensuite l'adresse MAC longue de l'interface. La longueur de cette adresse est de 64 bits. Dans les sections suivantes on abrège souvent cette adresse longue en omettant les `00` intermédiaires comme dans la notation des adresses IPv6. Par exemple l'adresse longue `a0:00:00:00:00:00:00:01` est abrégée `a0::1`. On peut ensuite vérifier que cette adresse est bien configurée avec la commande `ip link show`. Dans ce cas on constate que l'adresse est bien configurée. On constate aussi que le MTU vaut 127 Octets. Toutefois il faut garder à l'esprit que ce MTU représente la taille maximal de la trame sans les en-têtes :

```
# ip link set wpan0 address a0:00:00:00:00:00:00:01
# ip link show wpan0
3: wpan0: <BROADCAST,NOARP> mtu 127 qdisc noop state DOWN  \
           mode DEFAULT qlen 300
    link/ieee802.15.4 a0:00:00:00:00:00:00:01 brd ff:ff:ff \
                                              :ff:ff:ff:ff:ff
```

On peut ensuite monter l'interface MAC. À noter que pour l'instant la couche réseau n'est pas encore disponible :

```
ifconfig wpan0 up
```

Ensuite on peut configurer les options spécifiques à l'interface MAC. À savoir l'identifiant du PAN en hexadécimal (0777), l'adresse courte (b001) et le canal à utiliser (11) :

```
iz set wpan0 777 b001 11
```

Il faut maintenant procéder à la mise en place de l'interface la couche IP. Comme expliqué à la Section 3.11 du chapitre précédent, la mise en place de la couche 6LoWPAN passe à travers la création d'un lien virtuel. Ce lien est découpé en deux partie. Premièrement une interface lowpan capable de recevoir et d'émettre des paquets IPv6.

Dans notre cas cette interface se nomme *lowpan0*. Deuxièmement l'interface MAC IEEE 802.15.4 capable de recevoir et d'émettre des paquets 6LoWPAN. Dans notre cas cette interface se nomme *wpan0*. La première commande crée ce lien virtuel et lui associe le type *lowpan* :

```
ip link add link wpan0 name lowpan0 type lowpan
```

On configure ensuite l'adresse MAC de l'interface lowpan0. Cette adresse est utilisée pour la compression des adresses IPv6 dans l'en-tête 6LoWPAN. Il faut donc que cette adresse soit identique à celle utilisée pour l'interface MAC IEEE 802.15.4 :

```
ip link set lowpan0 address a0:00:00:00:00:00:00:01
```

On peut également vérifier l'état de l'interface avec la commande `ip link show`. Dans ce cas on observe que le MTU vaut maintenant 1281 Octets soit le minimum nécessaire pour l'IPv6 :

```
5: wpan0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 127 qdisc \
          pfifo_fast state UNKNOWN mode DEFAULT qlen 300
    link/ieee802.15.4 a0:00:00:00:00:00:00:01 brd ff: \
                                    ff:ff:ff:ff:ff:ff:ff:ff
```

On peut maintenant lancer l'interface 6LoWPAN et la couche réseau :

```
ip link set lowpan0 up
```

Afin d'observer et de capturer le trafic, on lance également le sniffer avec la commande suivante. Cette commande configure le sniffer pour le firmware se trouvant sur le port série *ttyUSB0* à une vitesse de 115200 bauds et configure le transceiver pour écouter sur le canal 11. Les options `-A` et `-P` permettent d'afficher le contenu des trames. Si la commande s'interface correctement au firmware, le nom de celui-ci est affiché. Le sniffer attend ensuite les trames. Dans le cas où la commande ne s'interface pas correctement, un message d'erreur est affiché après quelques secondes.

```
$ wsn-sniffer-cli -C 11 -A -P -b 115200 -p capture.pcap \
                  /dev/ttyUSB0
Sniffer Firmware v0.3
Waiting [\]
```

Une fois l'interface lancée, on observe les paquets ICMPv6 relatifs à l'autoconfiguration sans état de l'adresse link-local (pour rappel voir la Section 1.6.3).

4.2 Simulation de ping

Ce test consiste à simuler un ping vers un noeud sur le Raspberry Pi. La Figure 4.1 illustre la disposition des noeuds. Le Raspberry Pi possède l'adresse longue `a0::01` et l'adresse courte `b001`. Un autre noeud est simulé par injection de paquet. Ce noeud possède l'adresse longue `a0::02` et l'adresse courte `b002`. Les adresses IP sont link-local et basées sur les adresses MAC. Finalement un troisième noeud est utilisé pour capturer et observer le trafic. Pour effectuer ce test on dispose des payload 6LoWPAN pour des paquets associés à un ping entre deux hôtes. Ils sont disponibles dans le repository de l'outil wsn-tools décrit à la Section 3.7. À noter également que les ACK requests sont désactivés pour ce test.

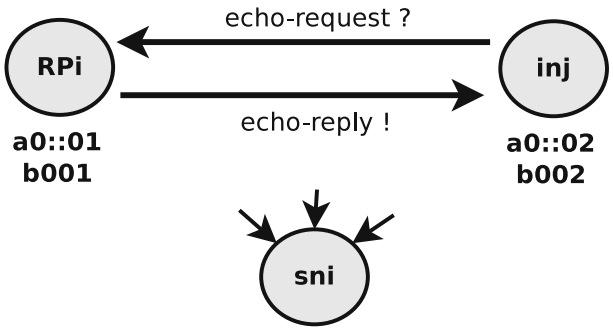


FIGURE 4.1 – Test de simulation d'un ping par injection.

Nom	Description
`ns.pkt`	Neighbor Solicitation
`na.pkt`	Neighbor Advertisement
`echo-request.pkt`	PING Echo-request
`echo-reply.pkt`	PING Echo-reply

TABLE 4.1 – Payload de paquets 6LoWPAN pour les tests d'injection.

La Table 4.1 reprend la liste des différents paquets disponibles pour l'injection. Ces paquets ont pour source l'adresse de l'injecteur et destination l'adresse du Raspberry Pi. Ceux-ci ont été obtenus par injection-replay successif d'un ping vers et depuis un noeud 6LoWPAN. La commande utilisée pour injecter les paquets est la suivante :

```
wsn-injector-cli --saddr 777-a0:00:00:00:00:00:00:02     \
                 --daddr 777-a0:00:00:00:00:00:00:01     \
                 --enable-pan-comp -C 11 -p <packet>.pkt \
                 -D -b 115200 /dev/ttyUSB1
```

La Figure 4.2 reprend le résultat observé. Les numéros de séquence des trames MAC IEEE 802.15.4 sont reprises dans la marge. Premièrement on voit le paquet echo-request initial injecté sur le réseau avec pour numéro de séquence 0. Ensuite le noeud `a0::01` répond par un paquet NS. En effet, il ne connaît pas encore l'adresse MAC associée à l'injecteur. Cette trame possède le numéro de séquence 106. Ce même paquet est envoyé quatre fois de suite car les ACK sont désactivés. Comme l'injection est effectuée manuellement, la réponse n'est pas assez rapide et un timeout se produit. La couche 6LoWPAN réinjecte le paquet avec cette fois le numéro de séquence 107. L'injecteur envoie alors un paquet NA en réponse. Ce paquet possède le numéro de séquence 1. Finalement le noeud `a0::01` connaît maintenant l'adresse MAC de l'injecteur. Il répond alors au ping par un paquet echo-reply.

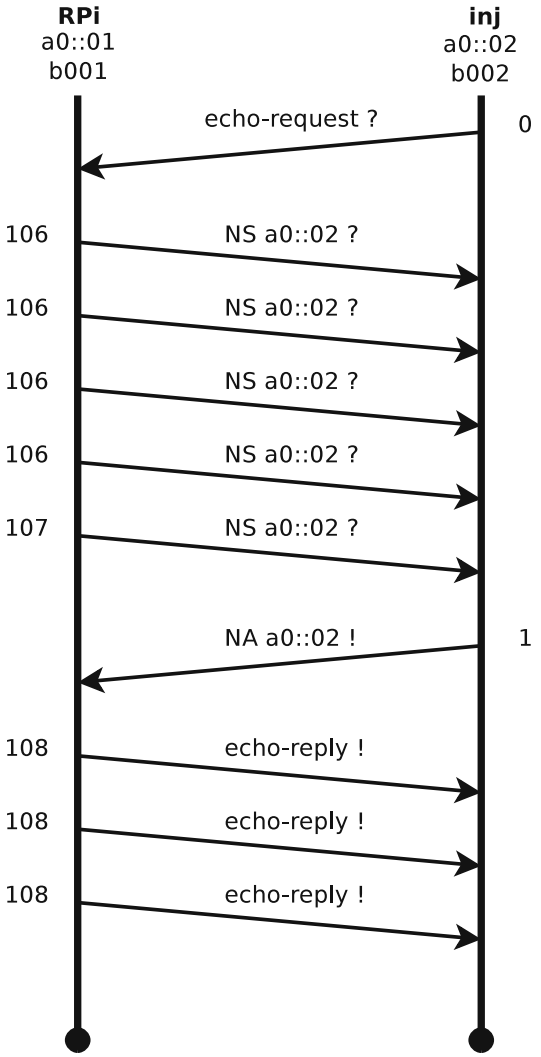


FIGURE 4.2 – Séquence des opérations pour le ping simulé.

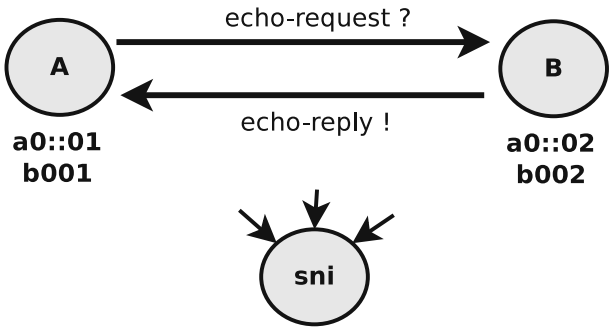


FIGURE 4.3 – Test d'un ping entre deux noeuds Raspberry Pi.

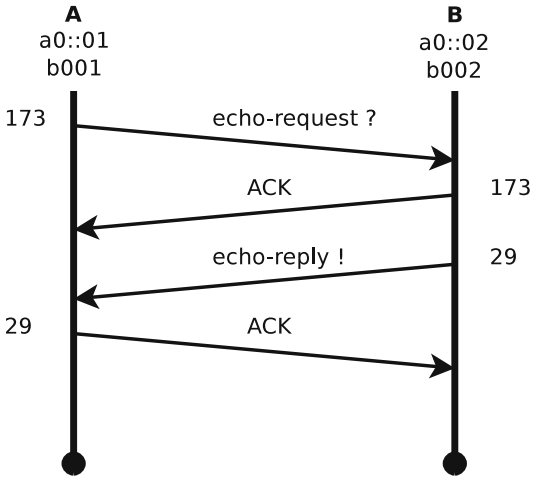


FIGURE 4.4 – Séquence des opérations pour le ping simple.

4.3 Ping

Ce test consiste à effectuer un ping entre deux noeuds Raspberry Pi. La Figure 4.3 illustre la disposition des noeuds. Cette disposition est similaire à la disposition représentée à la Figure 4.1 utilisée dans la Section 4.2. Toutefois les pings effectués ici ne sont pas simulés. Le noeud *A* possède l'adresse longue `a0::01` et l'adresse courte `b001`. Le noeud *B* possède l'adresse longue `a0::02` et l'adresse courte `b002`. Les adresses IP sont link-local et basées sur les adresses MAC. Finalement un troisième noeud est utilisé pour capturer et observer le trafic. Les ACK requests sont activés pour ce test. Les pings sont réalisés depuis le noeud A vers le noeud B.

La première partie du test réalise un ping simple entre les deux noeuds. Dans ces tests les échanges de paquets NS/NA ont déjà eu lieu. Dès lors, les deux noeuds connaissent l'adresse MAC de leur voisin. La Figure 4.4 reprend le résultat observé. Premièrement on voit le paquet echo-request initial envoyé par le noeud A. Cette trame possède le numéro de séquence 173. Un ACK est alors émis par le noeud B pour cette trame. Ensuite le noeud B envoie la réponse echo-reply. Cette trame possède le numéro de séquence 29. Finalement un ACK est émis par le noeud A pour cette trame. Les résultats de la commande ping sont repris ici :

```
$ ping6 -c 4 fe80::a200:0:0:2%lowpan0
```

```
PING fe80::a200:0:0:2%lowpan0(fe80::a200:0:0:2) 56 data bytes
64 bytes from fe80::a200:0:0:2: icmp_seq=1 ttl=64 time=15.3 ms
64 bytes from fe80::a200:0:0:2: icmp_seq=2 ttl=64 time=14.6 ms
64 bytes from fe80::a200:0:0:2: icmp_seq=3 ttl=64 time=15.2 ms
64 bytes from fe80::a200:0:0:2: icmp_seq=4 ttl=64 time=15.2 ms

--- fe80::a200:0:0:2%lowpan0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 14.632/15.134/15.383/0.306 m
```

On constate que le Rount-Trip Time (RTT) moyen est de $15,1$ ms. On constate également qu'il n'y a pas de perte de paquets. Toutefois ce test est trop simple pour illustrer les pings correctement.

La seconde partie du test réalise un ping sur l'adresse multicast `ff02::1`. Dans ce test les ACK requests sont désactivés. Ce test a également été réalisé avec les ACK activés et les résultats sont similaires. Les résultats de la commande ping sont repris ici :

```
$ ping6 ff02::1%lowpan0
PING ff02::1%lowpan0(ff02::1) 56 data bytes
64 bytes from fe80::a200:0:0:1: icmp_seq=1 ttl=64 time=0.181 ms
64 bytes from fe80::a200:0:0:2: icmp_seq=1 ttl=64 time=28.9 ms \
                                                        (DUP!)
64 bytes from fe80::a200:0:0:1: icmp_seq=2 ttl=64 time=0.168 ms
64 bytes from fe80::a200:0:0:2: icmp_seq=2 ttl=64 time=28.8 ms \
                                                        (DUP!)

--- ff02::1%lowpan0 ping statistics ---
2 packets transmitted, 2 received, +2 duplicates, \
                       0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.168/14.512/28.908/14.327 ms
```

On constate ici que les paquets sont convenablement reçu. Lorsque le noeud A envoie son paquet il reçoit une réponse de sa propre interface. Cette réponse est la première traitée par la commande ping avec un RTT de $0,181$ ms. Ensuite la réponse est envoyé par le noeud B avec un RTT de $22,9$ ms. Dans le trafic capturé on observe également que l'adresse MAC destination du paquet echo-request est l'adresse broadcast `FFFF`. En effet, le réseau IEEE 802.15.4 ne gère par les adresses multicast. Dès lors, lorsqu'un paquet 6LoWPAN est envoyé sur une adresse multicast, la trame correspondante est envoyée sur l'adresse broadcast.

La troisième partie du test réalise un ping adaptif avec l'option `-A`. Cette option permet d'adapter l'intervalle entre les paquets au RTT. De cette manière pas plus d'un echo-request se trouve à la fois sur le lien. Toutefois afin d'utiliser des intervalles plus petit que 200 ms, la commande ping doit être exécutée avec les droits de l'utilisateur root. Le test a été réalisé sur une période de 3 minutes. Les résultats de la commande sont repris ici :

```
# ping6 -A fe80::a200:0:0:2%lowpan0
```

```
(...)
--- fe80::a200:0:0:2%lowpan0 ping statistics ---
6334 packets transmitted, 6137 received, +202 duplicates, \
                          3% packet loss, time 148290ms
rtt min/avg/max/mdev = 12.486/26.883/309.950/38.795 ms,   \
                       pipe 9, ipg/ewma 23.415/16.184 m
```

On constate ici que des paquets sont perdus. Toutefois le pourcentage de perte reste limité. Ce test donne une meilleur aperçu des statistiques relatives au RTT. En effet, celles-ci portent sur un nombre beaucoup plus grand de pings. Le RTT minimum est de $12,486$ ms ce qui correspond au RTT trouvé avec le ping simple. Le RTT moyen est de $26,883$ ms. Cette valeur est un peu plus élevée. Cela est dû au fait que des paquets sont perdus dans ce test. On remarque également que le RTT maximum est de $309,950$ ms. En effet, lorsque des paquets sont perdus, les retransmissions qui s'en suivent augmentent le RTT.

La quatrième et dernière partie du test réalise un ping flood avec l'option `-f`. Cette option permet d'envoyer les paquets sans intervalle entre les echo-request. De cette manière le canal est inondé d'echo-request. Cela permet de tester les limites du lien. Cela permet également de détecter les race conditions décrites à la Section 3.10 du chapitre précédent. Comme au test précédent la commande ping doit être exécutée avec les droits de l'utilisateur root. Les résultats de la commande sont repris ici :

```
# ping6 -f fe80::a200:0:0:2%lowpan0
 PING fe80::a200:0:0:2%lowpan0(fe80::a200:0:0:2) 56 data bytes
(...)
--- fe80::a200:0:0:2%lowpan0 ping statistics ---
2015 packets transmitted, 1134 received, +274 duplicates,   \
                          43% packet loss, time 60670ms
rtt min/avg/max/mdev = 148.335/590.989/3221.908/798.291 ms, \
                       pipe 103, ipg/ewma 30.124/2379.669 m
```

On constate ici que presque la moitié des paquets sont perdus. Cela est dû à la section critique utilisée pour protéger la transmission et la réception des trames dans le pilote du MRF24J40. On constate également que les statistiques relatives au RTT sont beaucoup plus élevées que dans le test précédent et même le RTT minimum dépasse les 100 ms. En effet, comme le canal est innondé, toutes les trames nécessitent des retransmissions qui augmentent considérablement le RTT.

4.4 UDP et TCP

On cherche ici à tester la couche transport. Dans ce test des paquets IPv6 sont transmis entre deux Raspberry Pi. Dans la première partie de ce test, les paquets contiennent un datagramme UDP. Dans la seconde partie de ce test, les paquets contiennent un segment TCP. La disposition des noeuds est représentée à la Figure 4.5. Elle est similaire à celle présentée à la Figure 4.3. Le port destination 1234 est utilisé pour les transferts UDP et TCP. Les ACK requests sont activés pour ce test. Les transferts sont

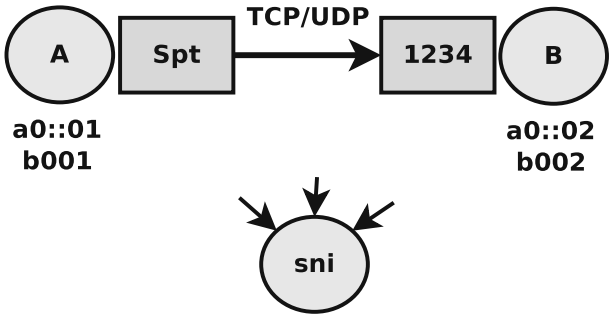


FIGURE 4.5 – Test d'un transfert UDP/TCP entre deux noeuds Raspberry Pi.

réalisés depuis le noeud A vers le noeud B. Dans les deux cas le payload envoyé est "HELLOWORLD".

La première partie de ce test réalise un transfert UDP du noeud A vers le noeud B. Pour réaliser cela on utilise la commande *netcat6*. Cette commande permet de lire et d'écrire des données à travers le réseau en utilisant des connections TCP ou UDP. Deux cas de figure sont testés. Dans le premier le port UDP 1234 est ouvert en écoute sur le noeud B. Dans le second cas le port UDP 1234 est fermé sur le noeud B. Les commandes utilisées sont reprises ici :

```
nc6 -u -l -p 1234
nc6 -u fe80::a200:0:0:2%lowpan0 1234
```

La première commande ouvre le port 1234 en écoute sur le noeud B. La seconde commande envoie un datagramme UDP au noeud B en utilisant le port 1234. Dans le cas où le serveur est activé sur le noeud B, on observe la réception du datagramme sur le noeud B. Un ACK confirme également la réception de la trame. Au total 2 trames sont envoyées pour transmettre le message. Dans le cas où le serveur n'est pas activé sur le noeud B, celui-ci répond avec un paquet ICMPv6 Destination Unreachable/Port unreachable. Deux ACK confirment également la réception du datagramme UDP et du paquet ICMP.

La seconde partie de ce test réalise un transfert TCP du noeud A vers le noeud B. Comme au cas précédent, deux cas de figure sont testés. Dans le premier le port TCP 1234 est ouvert en écoute sur le noeud B. Dans le second cas le port TCP 1234 est fermé sur le noeud B. Les commandes utilisées sont reprises ici :

```
nc6 -l -p 1234
nc6 fe80::a200:0:0:1%lowpan0 1234
```

La première commande ouvre le port 1234 en écoute sur le noeud B et accepte les nouvelles connections. La seconde commande établit une connection TCP vers le noeud B et envoie un segment TCP. Dans le cas où le serveur est activé sur le noeud B. On observe premièrement l'établissement de la connection (SYN/SYNACK). On observe ensuite l'envoi du segment et son ACK en réponse. Finalement on observe la terminaison de la connection (FIN/ACK). Pour chaque trame envoyé, un ACK est reçu. Au total 16 trames sont envoyées pour transmettre le message. Dans le cas où le serveur n'est pas activé sur le noeud B, celui-ci répond au paquet TCP SYN avec un paquet

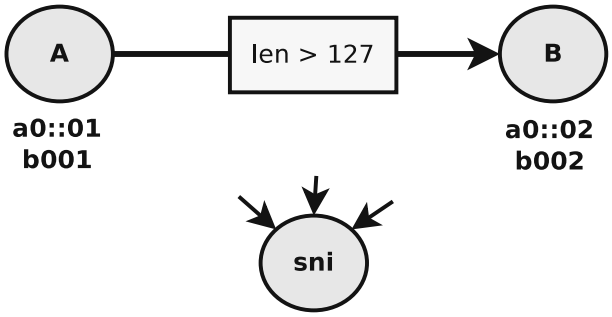


FIGURE 4.6 – Test de la fragmentation entre deux noeuds Raspberry Pi.

TCP RST/ACK ce qui termine la connection. Pour chaque trame un ACK est également reçu.

On note ici que l'établissement de connections TCP augmente considérablement le nombre de trames transmises. En particulier lorsque les ACK requests sont activés au niveau de la couche MAC.

4.5 Fragmentation

Ce test consiste à effectuer un trafic entre deux noeuds Raspberry Pi de manière à déclencher le mécanisme de fragmentation de la couche 6LoWPAN. Pour cela des paquets IPv6 sont envoyés avec une taille telle que la version 6LoWPAN du paquet ne puisse pas tenir dans une unique trame. Dans la seconde partie du test on mesure l'incidence de la fragmentation 6LoWPAN sur le RTT. La disposition des noeuds est représentée à la Figure 4.6. Elle est similaire à celle présentée à la Figure 4.3. Le port destination 1234 est utilisé pour les transferts UDP. Les ACK requests sont activés pour ce test. Les transferts sont réalisés depuis le noeud A vers le noeud B.

Pour réaliser la première partie du test un datagramme UDP avec un payload de 5 Ko est envoyé vers le noeud B au port 1234. Le datagramme réassemblé est ensuite extrait du noeud B en écoute sur ce même port. Finalement la valeur extraite est comparée avec le payload d'origine. Les commandes utilisées sont reprises ici :

```
dd if=/dev/urandom of=large-udp.pkt bs=1 count=5000
nc6 -u -l -p 1234
nc6 -u fe80::a200:0:0:2%lowpan0 1234 < large-udp.pkt
```

La première commande crée un payload de 5 Ko. La seconde commande ouvre le port 1234 en écoute sur le noeud B. Finalement la troisième commande envoie le datagramme UDP vers le noeud B en utilisant le port 1234. La Figure 4.7 illustre le résultat obtenu. Dans la capture on observe un ensemble de trames IEEE 802.15.4. Chacune de ces trames de 127 Octets contient un fragment 6LoWPAN. Ce même fragment 6LoWPAN contient 88 Octets de données. Une fois réassemblés, ces fragments constituent un paquet IPv6 de 1280 Octets (en-têtes comprises). Cela correspond au MTU de l'interface lowpan0. Ces paquets IPv6 sont eux mêmes des fragments. Une fois réassemblés, ces fragments constituent le datagramme UDP de 5 Ko. Pour chaque trame

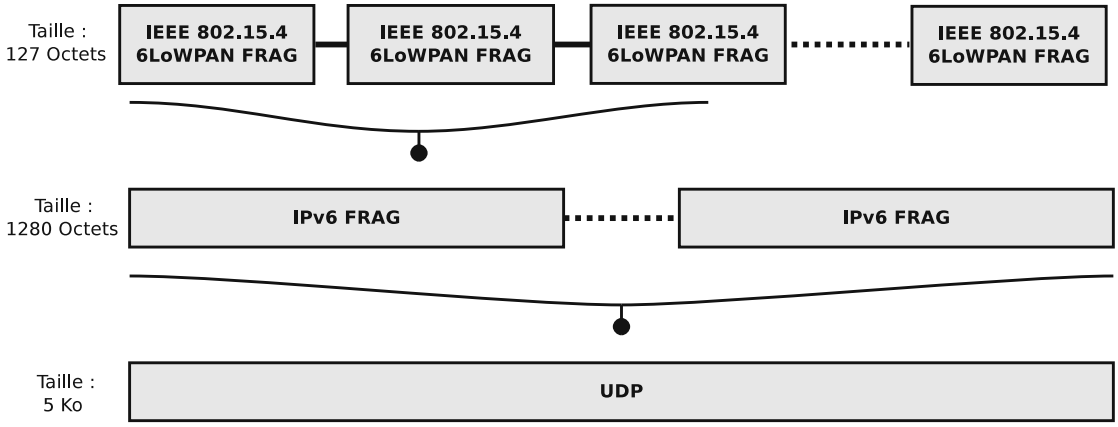


FIGURE 4.7 – Résultat de la fragmentation d'un datagramme UDP de 5 Ko.

IEEE 802.15.4 transmise, un ACK est transmis également. Le nombre d'ACK nécessaires pour un datagramme fragmenté est donc très important.

Pour réaliser la seconde partie du test on utilise l'argument `-s` de la commande ping pour envoyer des echo-request de taille variable. La taille est les RTT observés sont reportés dans la Table 4.2. La Figure 4.8 illustre ces données. On constate que le RTT varie linéairement avec la taille des fragments. C'est effectivement ce à quoi on s'attend au vu des composants impliqués dans le mécanisme de réassemblage. En effet ces composants consistent à copier les fragments reçus dans le paquet réassemblé. Dès lors, ils ont tous une complexité linéaire. Au contraire si une variation superlinéaire était observée, cela indiquerait un problème d'overhead au niveau d'un des composants.

4.6 Mesure de débit

Les tests suivants mesurent le débit en fonction de divers paramètres. En raison d'un problème avec la gestion mémoire du noyau 3.2.27, ce test est réalisé uniquement avec le noyau 3.8. On utilise le programme *iperf* pour réaliser ces tests. Ce programme permet de mesurer des débits à travers le réseau. En particulier il permet de réaliser ces tests en utilisant une connection TCP ou des datagrammes UDP. Dans leur première partie les tests font varier divers paramètres du pilote d'audit. En particulier l'utilisation ou non d'ACK, de la protection TXRX et finalement du mode turbo. Dans leur seconde partie les tests mesurent l'impact de la fragmentation sur le débit. La disposition des noeuds est similaire à la Figure 4.5. Toutefois dans ce cas le port destination utilisé est le port par défaut du programme iperf 5001.

La première partie de ce test mesure la bande passante en fonction de divers paramètres. Dans ce test, la commande iperf réalise un transfert de datagrammes UDP de taille n Octets pendant une durée de t secondes. Le débit mesuré est alors reporté, ainsi que les pertes et le *jitter*. Le jitter est une mesure de la variabilité de latence dans la transmission des datagrammes. Il représente la déviation moyenne par rapport à la latence moyenne. Les commandes utilisées pour le test sont reprises ici :

```
iperf -Vu -s
iperf -Vu -c fe80::a200:0:0:2%lowpan0 -t <t> -b 1M -l <n>
```

Taille	**RTT (ms)**
10	13, 9
100	27, 2
200	44, 2
400	78, 0
1000	184
2000	381
3000	551
4000	751
5000	945
6000	1129
6500	1215
7000	1312
8000	1500
9000	1694
10000	1890
15000	2815
20000	3748

TABLE 4.2 – RTT en fonction de la taille du paquet.

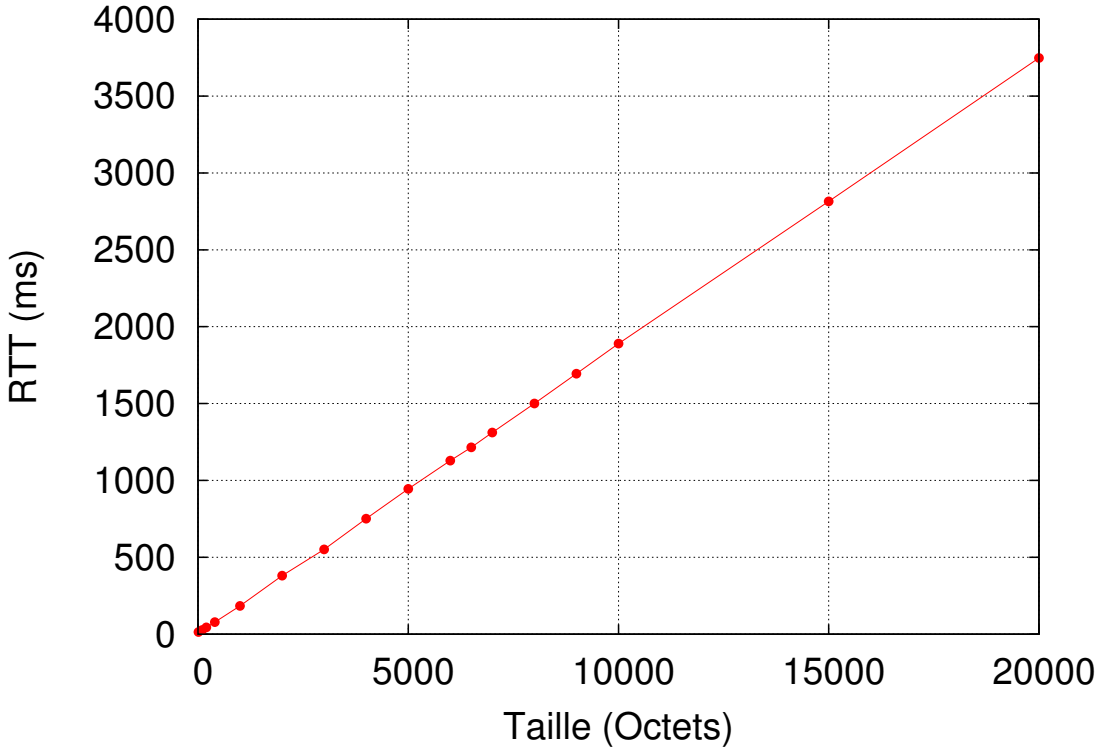


FIGURE 4.8 – RTT en fonction de la taille du paquet.

La première commande lance le serveur en écoute sur le port UDP 5001 sur le noeud B. La seconde commande lance le client vers le noeud A. En particulier elle mesure le débit pendant un temps <t> avec des datagrammes de taille <n>. Pour ce premier test, le temps <t> vaut 60 s et la taille <n> vaut 70 Octets. Lorsque la protection TXRX est désactivée, le temps <t> est réduit à 5 s pour éviter que le transceiver ne se bloque. La Table 4.3 reprend les résultats pour différentes combinaisons des paramètres du pilote d'audit. Premièrement on constate que dans tous les cas le débit est bien loin des 250 Kbps théoriques et ce même dans le cas du mode turbo. Deuxièmement on observe que l'utilisation ou non d'ACK requests a une influence considérable sur le débit. Toutefois le fait d'enlever les ACK requests augmente les pertes de paquets. En effet, dans ce cas les paquets ne sont pas retransmis et donc ils sont plus facilement perdus. Troisièmement on observe que la présence ou non de la protection TXRX n'influence pas le débit. Toutefois le fait d'enlever cette protection diminue la perte de paquets. On constate également que le jitter reste constant avec les ACK requests. Le jitter diminue légèrement sans les ACK requests. Dans le cas du mode turbo, on observe une hausse considérable du débit. Cette hausse s'accentue encore lorsque les ACK requests sont désactivés. On constate également que le jitter diminue avec le mode turbo. Toutefois cela peut être dû au fait que la variance du RTT diminue lorsque le débit augmente. En effet, étant donné que le temps du transfert est constant, une augmentation du débit signifie une augmentation du nombre de paquets. Dès lors, la moyenne de la déviation est calculée sur un échantillon plus large. Cependant on constate que le jitter diminue fortement lorsque les ACK requests ne sont pas utilisées.

La seconde partie de ce test consiste à mesurer l'influence de la fragmentation sur

ACK	TXRX	TURBO	Débit (Kbps)	Jitter (ms)	Perte
1	1	0	$77,1$	$55,567$	$0,071\%$
1	0	0	$77,4$	$56,493$	0%
0	1	0	$84,5$	$50,899$	$0,11\%$
1	1	1	$113,4$	$38,470$	$0,025\%$
0	1	1	122	$7,231$	$1,2\%$

TABLE 4.3 – Variation du débit, de jitter et de la perte en fonction des ACK requests, de la protection TXRX et du mode urbo.

Taille (Octets)	Débit (Kbps)	Jitter (ms)	Perte
100	$3,99$	$32,435$	$\approx 100\%$
500	$3,41$	$104,678$	$\approx 100\%$
1000	$3,17$	$127,598$	$\approx 100\%$

TABLE 4.4 – Variation du débit, de jitter et de la perte en fonction de la taille des datagrammes.

le débit, le jitter et la perde. Ce test est similaire au précédent à cela prêt que le paramètre <n> choisie de manière à déclencher le mécanisme de fragmentation 6LoWPAN. Le mécanisme de fragmentation d'IPv6 en revanche n'est pas déclenché car la taille totale du datagramme UDP (en-têtes comprises) reste inférieur au MTU du lien. Dans ce test les ACK requests et la protection TXRX sont activés. La Table 4.4 reprend les résultats pour différentes tailles de paquets. Premièrement on constate que la fragmentation diminue très fortement le débit. On constate également que le taux de perte augmente et atteint approximativement 100%. Et finalement on constate que la fragmentation augmente également le jitter.

Le même test est effectué avec des tailles de datagrammes variant de 90 à 100 Octets pour mettre en évidence le déclenchement du mécanisme de fragmentation. La taille et les débits observés sont reportés dans la Table 4.5. La Figure 4.9 illustre ces données. On constate que le débit diminue fortement lorsque la taille du datagramme passe de 95 à 96 Octets. En effet, si on additionne la taille de l'en-tête MAC (21), la taille de l'en-tête 6LoWPAN et UDP (9) ainsi que la taille du FCS (2) on obtient 32 Octets. On a donc 95 Octets de disponible pour que le payload du datagramme UDP tienne dans une trame IEEE 802.15.4.

4.7 Benchmark du pilote

L'option *irq_stats* du pilote d'audit permet de récolter diverses statistiques sur le temps passé à traiter chaque IRQ. En particulier ces statistiques nous informent sur le temps de traitement nécessaire pour traiter chaque trame. Ces statistiques sont impri-

Taille	**Débit (Kbps)**
90	55.7
91	54.3
92	56.6
93	58.4
94	57.5
95	57.4
96	9.12
97	6.62
98	5.48
99	4.44
100	3.74

TABLE 4.5 – Débit en fonction de la taille du paquet.

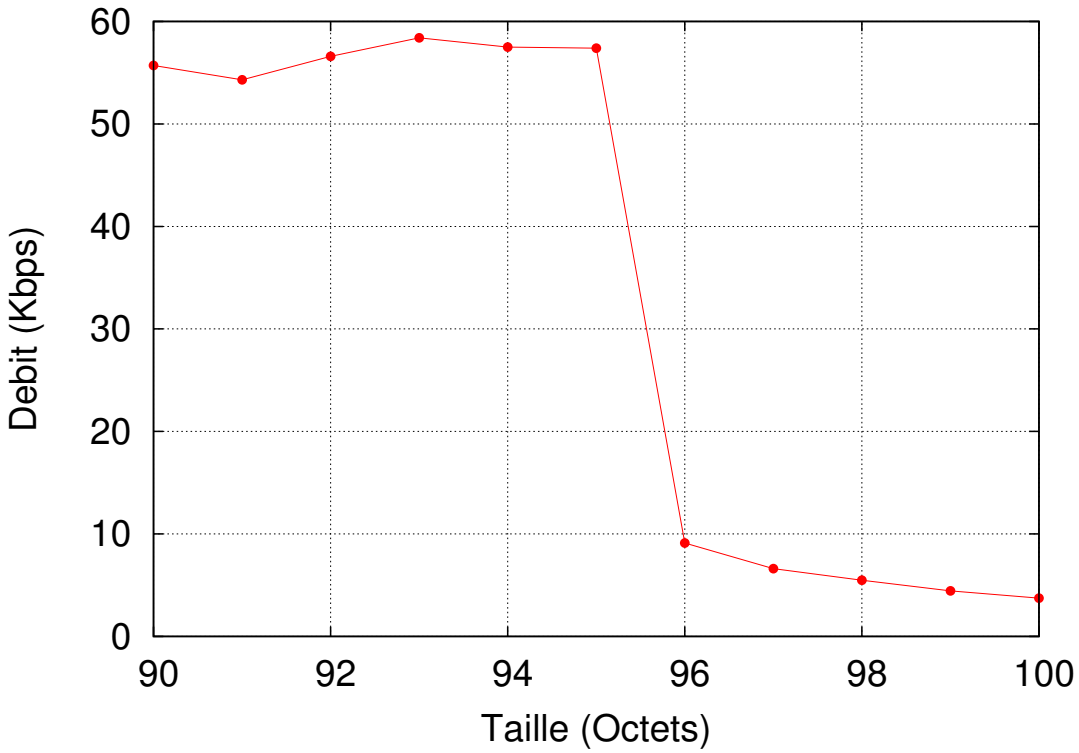


FIGURE 4.9 – Débit en fonction de la taille du paquet.

Type	Nombre IRQ	Moyenne
Envoi	8734	132 μs
Réception	9151	113 μs

TABLE 4.6 – Temps moyen par IRQ.

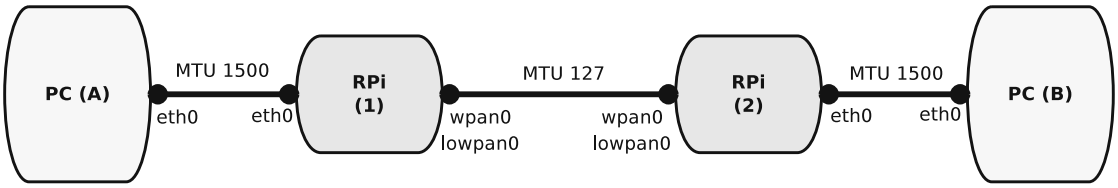


FIGURE 4.10 – Test avec une route statique.

mées dans le log dès que le module est enlevé du noyau. Le test est réalisé avec un flood d'echo-request pour augmenter la pression sur l'ISR. Le noeud A est exclusivement dédié à la transmission. Le noeud B quant à lui est exclusivement dédié à la réception. De cette manière on peut distinguer dans l'ISR le traitement de trames à envoyer du traitement des trames reçues. La Table 4.6 reprend le nombre d'IRQ et le temps moyen passé par IRQ pour le noeud d'envoi et le noeud de réception. On constate que le nombre d'IRQ est plus faible pour l'envoi que pour la réception. Cette différence est dûe aux pertes de trames sur le canal. On constate également que l'ISR passe plus de temps à envoyer une trame qu'à traiter sa réception. Toutefois aussi bien pour l'envoi que pour la réception, le temps par IRQ est du même ordre de grandeur. De plus la contribution de ce délai au RTT moyen est négligeable.

4.8 Static routing

Ce test consiste à mettre en place une route statique passant par un lien IEEE 802.15.4 formé des Raspberry Pi et des transceivers MRF24J40. En raison d'un problème avec la gestion mémoire du noyau 3.2.27, ce test est réalisé uniquement avec le noyau 3.8. La Figure 4.10 reprend la disposition des différents noeuds impliqués dans cette route. Les deux noeuds *A* et *B* constituent la source et la destination dans le test. Ces noeuds sont des PC de bureau équipés d'une interface Ethernet 10/100. Les noeuds *1* et *2* sont constitués des deux Raspberry Pi. La configuration des liens est la suivante :

— Le noeud A est connecté au noeud 1 via un lien Ethernet de MTU 1500
— Le noeud 1 est connecté au noeud 2 via un lien IEEE 802.15.4 de MTU 127
— Le noeud 2 est connecté au noeud B via un lien Ethernet de MTU 1500

Il est importer de noter que même si le lien entre les noeuds 1 et 2 possède un MTU de 127 Octets, le MTU apparent est de 1280 Octets grâce à la couche 6LoWPAN. Le préfixe utilisé pour les adresses IPv6 est `2001:0000::/29`. Pour chaque interface seuls les 16 derniers bits de l'adresse sont utilisés. La Table 4.7 reprend l'adresse associée à chaque interface. Pour rappel la Section 4.1 explique comment configurer ces interfaces. Toutefois la configuration ne prend pas en compte les adresses globales utilisées ici. Il est donc nécessaire de les rajouter manuellement pour chaque interface

Noeud	Interface	Adresse (16 bits)
A	eth0	`000A`
1	eth0	`A001`
1	lowpan0	`2001`
2	lowpan0	`1002`
2	eth0	`B002`
B	eth0	`000B`

TABLE 4.7 – Adresse des interfaces.

avec la commande *ip*. Par exemple pour le noeud A et son interface eth0, il faut utiliser la commande suivante :

```
ip address add 2001::a/128 dev eth0
```

Ensuite il est nécessaire de configurer la route statiquement en modifiant la table de routage sur chaque noeud de manière à ce que le trafic puisse transiter de A vers B et de B vers A. Comme les adresses sont configurées manuellement il est également nécessaire pour chaque interface d'ajouter la route vers son Next-Hop (NH). Les adresses utilisées dans les routes sont complètement spécifiées de manière à éviter les interférences avec d'autres routes IPv6 déjà en place. On utilise la commande *ip* pour configurer la route. Par exemple pour le noeud 1, forwarder les paquets pour l'adresse `2001::b` via le NH `2001::a001` sur l'interface eth0 :

```
ip -6 route add 2001::a001/128 dev eth0
ip -6 route add 2001::b/128 via 2001::a001 dev eth0
```

Il est également nécessaire d'activer le forwarding sur chaque noeud avec la commande suivante :

```
echo "1" > /proc/sys/net/ipv6/conf/all/forwarding
```

Il est maintenant possible d'envoyer des paquets IPv6 du noeud A vers le noeud B et du noeud B vers le noeud A en passant par le lien IEEE 802.15.4. En particulier quatre tests ont été réalisés pour tester le forwarding. Ces tests et leur résultats sont présentés dans les paragraphes suivants :

Destination	**Next-Hop**	**Interface**
`2001::b`	`2001::a001`	eth0
`2001::a001`	–	eth0

TABLE 4.8 – Table de routage du noeud A.

Destination	**Next-Hop**	**Interface**
`2001::a`	`2001::b002`	eth0
`2001::b002`	–	eth0

TABLE 4.9 – Table de routage du noeud B.

Destination	**Next-Hop**	**Interface**
`2001::b`	`2001::1002`	lowpan0
`2001::1002`	–	lowpan0
`2001::a`	–	eth0

TABLE 4.10 – Table de routage du noeud 1.

Destination	**Next-Hop**	**Interface**
`2001::a`	`2001::2001`	lowpan0
`2001::2001`	–	lowpan0
`2001::b`	–	eth0

TABLE 4.11 – Table de routage du noeud 2.

Le premier test consiste en un ping de 4 echo-request du noeud A vers le noeud B. Les echo-request et leur réponse echo-reply sont bien transmis à travers la route. Le résultat de la commande est le suivant :

```
ping6 -c 4 2001::b

PING 2001::b(2001::b) 56 data bytes
64 bytes from 2001::b: icmp_seq=1 ttl=62 time=20.9 ms
64 bytes from 2001::b: icmp_seq=2 ttl=62 time=19.1 ms
64 bytes from 2001::b: icmp_seq=3 ttl=62 time=20.0 ms
64 bytes from 2001::b: icmp_seq=4 ttl=62 time=17.6 ms

--- 2001::b ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 17.663/19.455/20.981/1.236 m
```

Le second test consiste en un ping adaptif du noeud A vers le noeud B pendant une durée de 5 minutes. Ici aussi les paquets sont bien transmis à travers la route. Certains paquets sont perdus, toutefois le taux de perte reste limité à 1%. Le résultat de la commande est le suivant :

```
ping -A 2001::b

(...)
--- 2001::b ping statistics ---
12573 packets transmitted, 12413 received, +138 duplicates, \
                           1% packet loss, time 294057ms
rtt min/avg/max/mdev = 17.114/24.666/335.350/26.402 ms,     \
                           pipe 9, ipg/ewma 23.389/20.407 m
```

Le troisième test consiste en l'envoi d'un ping d'une taille de 5 Ko de manière à déclencher le mécanisme de fragmentation sur le lien IEEE 802.15.4. Ici aussi les paquets sont bien transmis et le mécanisme de fragmentation réassemble les paquets correctement. Le résultat de la commande est le suivant :

```
ping6 -c 4 -s 5000 2001::b

PING 2001::b(2001::b) 5000 data bytes
5008 bytes from 2001::b: icmp_seq=1 ttl=62 time=967 ms
5008 bytes from 2001::b: icmp_seq=2 ttl=62 time=961 ms
5008 bytes from 2001::b: icmp_seq=3 ttl=62 time=949 ms
5008 bytes from 2001::b: icmp_seq=4 ttl=62 time=961 ms

--- 2001::b ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 949.610/960.158/967.896/6.727 ms
```

Finalement le quatrième test consiste en l'envoi d'un datagramme UDP depuis le noeud A vers le noeud B. Ici aussi le datagramme est transmis à travers la route.

4.9 Configuration du préfixe avec radvd

Le programme *radvd* est un router advertisement daemon pour IPv6. En particulier il a pour rôle d'avertir les hôtes de l'adresse du routeur et du préfixe à utiliser pour les adresses globales autoconfigurées (pour rappel voir la Section 1.6.3). Il envoie régulièrement un paquet RA contenant ces informations. Il écoute également pour des paquets RS lorsqu'une nouvelle interface cherche à configurer ses adresses. Dans ce test le démon radvd est utilisé pour configurer les adresses globales avec le préfixe `2001:2f6:132c::/64` sur le canal 11 dans le PAN `0777`. La configuration du démon est la suivante :

radvd.conf

```
interface lowpan0
{
  AdvSendAdvert on;
  AdvHomeAgentFlag off;

  prefix 2001:2f6:132c::/64
  {
    AdvOnLink on;
    AdvAutonomous on;
    AdvRouterAddr on;
  };
};
```

Une fois le démon lancé, les adresses sont correctement configurées au premier message RA émis par le démon sur le lien IEEE 802.15.4. Les deux noeuds sont alors capables de se contacter via leur adresse globale. De plus une route par défaut est également configurée vers le noeud qui exécute radvd. Si le forwarding est activé sur ce noeud et que les paquets IPv6 provenant d'Internet lui sont forwardé, il est alors possible de contacter un noeud sur Internet via l'interface IEEE 802.15.4. Il est également possible de contacter un noeud dans le PAN depuis Internet.

En particulier le test effectué ici est illustré à la Figure 4.11. Les noeuds A et B sont connectés entre eux via un lien IEEE 802.15.4. En particulier l'interface Ethernet du noeud A est désactivée et on y accède par une connection série. Dès lors, la seule connectivité dont il dispose passe par son transceiver IEEE 802.15.4. Le démon radvd est lancé sur le noeud B et avertit l'adresse du routeur et le préfixe `2a02:578:f28::/64`. L'adresse autoconfigurée du noeud B est `2a02:578:f28::a200:0:0:2`. Le noeud B est connecté par Ethernet à la passerelle gw. Celle-ci est connectée à l'Internet IPv6 via un tunnel AYIYA chez SixXS. Cette passerelle forward les paquets pour le préfixe `2a02:578:f28::/64` vers le noeud B. Sur Internet l'hôte D possède une adresse IPv6 `2001:6f8:202:549::2`. Un ping est alors réalisé depuis le noeud B vers l'hôte D. Le résultat de la commande est le suivant :

```
ping6 -c 4 2001:6f8:202:549::2
PING 2001:6f8:202:549::2(2001:6f8:202:549::2) 56 data bytes
64 bytes from 2001:6f8:202:549::2: icmp_seq=1 ttl=62 time=61.6 ms
64 bytes from 2001:6f8:202:549::2: icmp_seq=2 ttl=62 time=61.8 ms
64 bytes from 2001:6f8:202:549::2: icmp_seq=3 ttl=62 time=60.3 ms
```

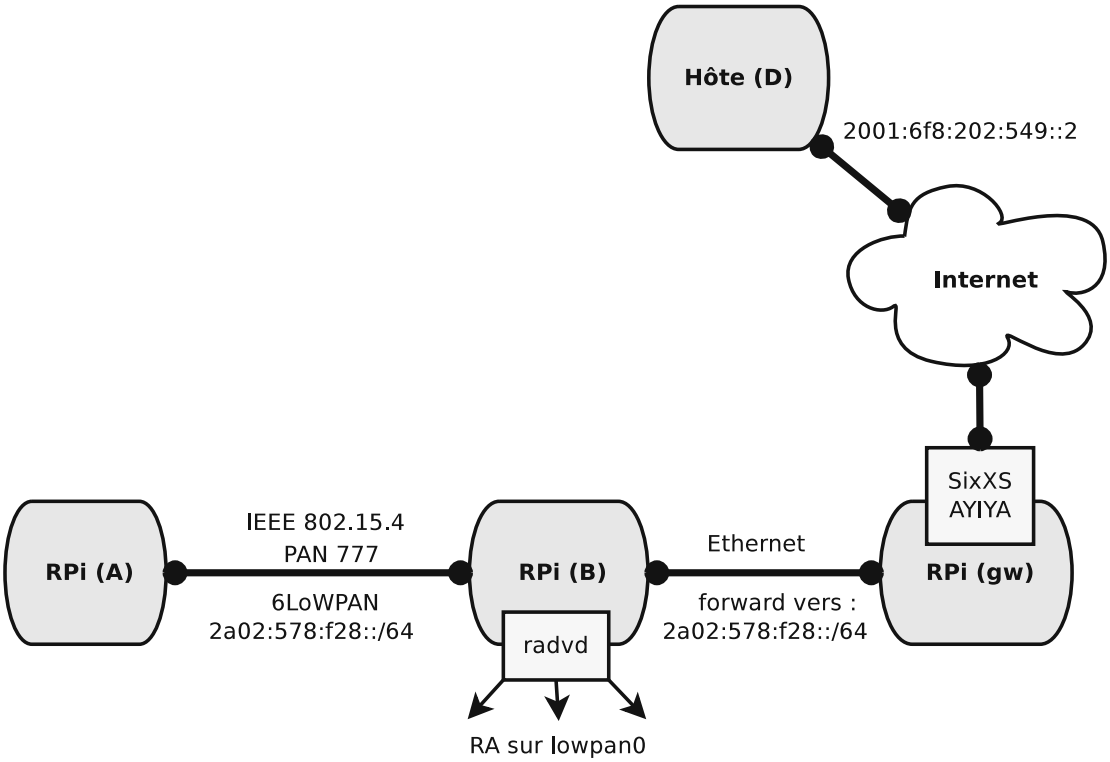


FIGURE 4.11 – Forwading depuis et vers Internet.

```
64 bytes from 2001:6f8:202:549::2: icmp_seq=4 ttl=62 time=60.8 ms

--- 2001:6f8:202:549::2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 60.342/61.179/61.815/0.600 ms
```

Un ping est également réalisé depuis l'hôte D vers le noeud B. Le résultat de la commande est le suivant :

```
ping6 -c 4 2a02:578:f28::a200:0:0:2
PING 2a02:578:f28::a200:0:0:2(2a02:578:f28:0:a200::2) 56 data bytes
64 bytes from 2a02:578:f28:0:a200::2: icmp_seq=1 ttl=57 time=62.1 ms
64 bytes from 2a02:578:f28:0:a200::2: icmp_seq=2 ttl=57 time=60.2 ms
64 bytes from 2a02:578:f28:0:a200::2: icmp_seq=3 ttl=57 time=59.5 ms
64 bytes from 2a02:578:f28:0:a200::2: icmp_seq=4 ttl=57 time=60.3 ms

--- 2a02:578:f28::a200:0:0:2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 59.549/60.558/62.136/0.973 ms
```

On constate dans les deux cas que le ping passe correctement à travers le lien IEEE 802.15.4 et Internet.

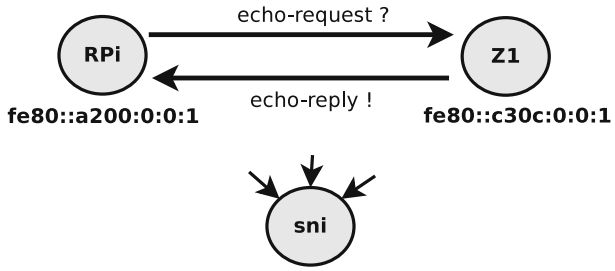


FIGURE 4.12 – Test d'un ping entre un noeud Raspberry Pi et un noeud Z1 sous Contiki.

4.10 Interaction avec Contiki

Dans ce dernier test on fait interagir le sous-système Linux-ZigBee avec un noeud Zolertia Z1 tournant sous Contiki. La Figure 4.12 illustre la disposition des noeuds. L'adresse du Raspberry Pi est `fe80::a200:0:0:1`. L'adresse du Z1 est `fe80::c30c:0:0:1`. Le Z1 est capable de répondre aux echo-request sur son adresse link-local. L'identifiant du PAN est `ABCD` et le canal 26 est utilisé. Trois tests sont réalisés depuis le Raspberry Pi vers le Z1 :

— Un ping simple
— Un ping adaptif
— Un ping flood

Dans le premier test un ping simple est réalisé vers le Z1. Celui-ci répond bien aux echo-request. Le résultat de la commande est le suivant :

```
ping6 -c 4 fe80::c30c:0:0:1%lowpan0
PING fe80::c30c:0:0:1%lowpan0(fe80::c30c:0:0:1) 56 data bytes
64 bytes from fe80::c30c:0:0:1: icmp_seq=1 ttl=64 time=27.2 ms
64 bytes from fe80::c30c:0:0:1: icmp_seq=2 ttl=64 time=27.7 ms
64 bytes from fe80::c30c:0:0:1: icmp_seq=3 ttl=64 time=27.1 ms
64 bytes from fe80::c30c:0:0:1: icmp_seq=4 ttl=64 time=26.1 ms

--- fe80::c30c:0:0:1%lowpan0 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 26.181/27.086/27.799/0.591 ms
```

Dans le second test un ping adaptif est réalisé vers le Z1. Ici également le Z1 répond bien aux echo-request. On remarque également que le taux de perte reste très faible. Le résultat de la commande est le suivant :

```
ping6 -A fe80::c30c:0:0:1%lowpan0
(...)
--- fe80::c30c:0:0:1%lowpan0 ping statistics ---
7740 packets transmitted, 7733 received, +1 duplicates, \
                          0% packet loss, time 220735ms
rtt min/avg/max/mdev = 25.759/28.348/647.853/17.176 ms, \
                          pipe 10, ipg/ewma 28.522/27.186 ms
```

Finalement dans le troisième test un ping flood est réalisé vers le Z1. Dans ce cas le RTT moyen et le taux de perte deviennent très important. En effet, le RTT moyen est alors de $413,476$ ms et le taux de perte de 97%. Cela s'explique par le fait que le Z1 tournant sous Contiki ne dispose de mémoire suffisante pour mettre les paquets en attente comme c'est le cas avec Linux. Le résultat de la commande est le suivant :

```
ping6 -f fe80::c30c:0:0:1%lowpan0
PING fe80::c30c:0:0:1%lowpan0(fe80::c30c:0:0:1) 56 data bytes
(...)
--- fe80::c30c:0:0:1%lowpan0 ping statistics ---
2685 packets transmitted, 72 received, 97% packet loss,      \
                         time 41337ms
rtt min/avg/max/mdev = 413.476/1867.209/3120.873/610.269 ms, \
                         pipe 201, ipg/ewma 15.401/1831.926 ms
```

Conclusion

De nombreuses solutions permettent déjà de configurer un routeur de bordure avec Contiki ou sous Linux. Toutefois les solutions connues à ce jour sous Linux utilisent des tunnels TUN/TAP et des démons en espace utilisateur. Il n'y a pas encore à ce jour de solution qui utilise directement la couche MAC IEEE 802.15.4 et la couche d'adaptation 6LoWPAN du noyau Linux. Cela est dû au fait que ces éléments sont encore très expérimental. En effet, peu de tests ont mis en évidence les fonctionnalités, performances et limitations de ce sous système. Dans ce projet nous avons mis en place ce sous-système sur une plateforme ARM.

Dans un premier temps nous avons mis en place une plateforme ARM Raspberry Pi tournant sous Linux pour acceuillir ce sous-système. En particulier il a été nécessaire de nettoyer l'image de la distribution Linux recommandée. Cette partie présente la technique de cross compilation qui facilite le développement pour ce type architecture embarquée. Elle présente également un outil développé pour faciliter la création de backup sur cette plateforme. Enfin elle propose une optimisation de la configuration du noyau Linux pour une utilisation comme routeur de bordure 6LoWPAN.

Afin de faciliter le débogage des réseaux de capteur sans fils un outil spécifique a été développé. Cet outil a été utilisé tout le long du projet pour faciliter le débogage et la mise en place du sous-système IEEE 802.15.4. Ensuite un rétroportage de la branche de développement du sous système a été réalisé afin de porter celui-ci sur le Raspberry Pi. Finalement le noyau a été modifié afin de supporter un pilote pour le transceiver Microchip MRF24J40. Ces modifications ont mises en évidence plusieurs problèmes avec la plateforme Raspberry Pi utilisée. En effet, il est apparu que certaines solutions ne pouvaient pas être appliquées à cette plateforme car l'architecture dans le noyau ne le permet pas. De plus cette architecture ne fait pas partie du noyau Linux officiel. En conséquence le code est moins testé, contient plus de bogues et moins de contributeurs. D'autre part le fait qu'il ne soit pas possible d'installer une distribution Debian native hard-float peut être vu comme un facteur limitant. Il serait intéressant et plus facile d'utiliser une plateforme similaire comme le BeagleBone.

Une fois le système mis en place nous avons fixé certains problèmes présent dans le pilote du transceiver MRF24J40 ainsi que dans la sous-couche d'adaptation 6LoWPAN. On discute également dans cette partie du rapport de problèmes qui n'ont pas encore été résolus. Comme par exemple les race conditions au niveau du pilote du transceiver. Des fonctionnalités supplémentaires ont également été ajoutées au pilote du transceiver. Ces patchs ont été publiés sur la mailing list du projet en vue d'une possible intégration. Il faut noter que la proposition de patchs sur la mailing list constitue un travail considérable. En effet, les patchs proposés doivent continuellement être retravaillés, discutés et motivés.

Finalement nous avons procédé à divers tests dans le but de valider le sous système IEEE 802.15.4 et la sous-couche d'adaptation 6LoWPAN. Ces tests ont permis de valider les outils développés dans la première partie de ce projet. De plus ils ont mis en évidence les capacités et performances du sous-système et ce dans différents cas de figure. Ces tests ont également mis en évidence la capacité du sous-système à connecter des noeuds 6LoWPAN à Internet ainsi que leur compatibilité avec des noeuds tournant sur d'autres systèmes d'exploitation.

Bibliographie

[1] ZigBee specification. *ZigBee document 053474r17*, 2007.

[2] IEEE Standard for Local and metropolitan area networks - Part 15.4 : Low-Rate Wireless Personal Area Networks (LR-WPANs). *IEEE Std. 802.15.4™- 2011*, 2011.

[3] JenNet-IP border-router. JN-AN-1110 (Application Note), 2011.

[4] Adafruit. `http://www.adafruit.com`, 2012.

[5] Arch Rock. `http://www.archrock.com`, 2012.

[6] BeagleBone. `http://beagleboard.org/Products/BeagleBone`, 2012.

[7] Carambola. `http://shop.8devices.com/wifi4things/carambola`, 2012.

[8] Contiki. `http://www.contiki-os.org`, 2012.

[9] Debian ARM cross-compile. `http://wiki.micromint.com/index.php/Debian_ARM_Cross-compile`, 2012.

[10] Emdebian. `http://www.emdebian.org`, 2012.

[11] Guidelines for 64-bit global identifier (EUI-64). `http://standards.ieee.org/db/oui/tutorials/EUI64.html`, 2012.

[12] HART communication. `http://www.hartcomm.org`, 2012.

[13] International Society of Automation. `http://www.isa.org`, 2012.

[14] Linux kernel. `http://www.kernel.org`, 2012.

[15] Raspberry Pi. `http://www.raspberrypi.org`, 2012.

[16] Raspberry Pi Linux kernel. `http://github.com/raspberrypi/linux`, 2012.

[17] Sensinode. `http://www.sensinode.com`, 2012.

[18] Texas Instrument CC-6LoWPAN. `http://processors.wiki.ti.com/index.php/CC-6LoWPAN`, 2012.

[19] ZigBee alliance. `http://www.zigbee.org`, 2012.

[20] ZigBee membership. `http://www.zigbee.org/Join/MemberBenefits.aspx`, 2012.

[21] OpenSniffer. `http://openwsn.berkeley.edu/wiki/OpenSniffer`, 2013.

[22] WSN-Tools. `http://www.hauweele.net/~gawen/wsn-tools.html`, 2013.

[23] S. Ahamed. The role of ZigBee technology in future data communication system. *Journal of Theoretical and Applied Information Technology*, 5(2) :129, 2009.

[24] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks : a survey. *Computer networks*, 38(4) :393–422, 2002.

[25] M. Alvira. Using the Freescale MC1322x series ARM7 processor with integrated 802.15.4. `http://mc1322x.devl.org`, 2012.

[26] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. Hu. Wireless sensor networks : A survey on the state of the art and the 802.15.4 and ZigBee standards. *Computer communications*, 30(7) :1655–1695, 2007.

[27] M. Becker. A RPL implementation for Linux derived from the Contiki RPL implementation. `http://github.com/markushx/librpl`, 2012.

[28] D. Bovet and M. Cesati. *Understanding the Linux kernel*. O'Reilly Media, Inc, third edition, 2005.

[29] T. Cheneau. A linux-based implementation of the Routing Protocol for Low-Power and Lossy Networks (RPL). `http://github.com/tcheneau/simpleRPL`, 2012.

[30] C. Chong and S. Kumar. Sensor networks : evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8) :1247–1256, 2003.

[31] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), Dec. 1998. Updated by RFCs 5095, 5722, 5871, 6437, 6564.

[32] M. Denis. Rapport de stage CETIC : Prototype de border router Redwire Econotag sous Contiki. 2012.

[33] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pages 455–462. IEEE, 2004.

[34] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads : simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 29–42. ACM, 2006.

[35] C. T. Ee, S. Shenker, and B. C. W. Hong. Interference avoidance in wireless multihop networks. In *poster session presented at the 2nd Annu. IEEE Commun. Soc. Conf. Sensor and Ad Hoc Commun. and Networks*, 2005.

[36] E. Egbogah and A. Fapojuwo. A survey of system architecture requirements for health care-based wireless sensor networks. *Sensors*, 11(5) :4875–4898, 2011.

[37] D. Eremin-Solenikov. Linux ZigBee. `http://sourceforge.net/apps/trac/linux-zigbee`, 2012.

[38] J. Hui and P. Thubert. Compression format for IPv6 datagrams over IEEE 802.15.4 based networks. RFC 6282 (Draft Standard), Sept. 2011.

[39] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs) : overview, assumptions, problem statement, and goals. RFC 4919 (Informational), Aug. 2007.

[40] T. Lennvall, S. Svensson, and F. Hekland. A comparison of WirelessHART and ZigBee for industrial applications. In *Factory Communication Systems, 2008. WFCS 2008. IEEE International Workshop on*, pages 85–88. IEEE, 2008.

[41] F. Lewis. Wireless sensor networks. *Smart Environments : Technologies, Protocols, and Applications*, pages 11–46, 2004.

[42] Maciek. Grinch - simple 6lowpan RPL border router. `http://sixpinetrees.blogspot.be/2011/06/grinch-simple-6lowpan-rpl-border-router.html`, June 2011.

[43] F. Mattern and C. Floerkemeier. From the internet of computers to the internet of things. In K. Sachs, I. Petrov, and P. Guerrero, editors, *From Active Data Management to Event-Based Systems and More*, volume 6462 of *Lecture Notes in Computer Science*, pages 242–259. Springer Berlin Heidelberg, 2010.

[44] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 packets over IEEE 802.15.4 networks. RFC 4944 (Draft Standard), 2007.

[45] T. Narten, W. Simpson, E. Nordmark, and H. Soliman. Neighbor discovery for IP version 6 (IPv6). RFC 4861 (Draft Standard), Sept. 2007. Updated by RFC 5942.

[46] T. Narten, S. Thomson, and T. Jinmei. IPv6 stateless address autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[47] S. Petersen and S. Carlsen. WirelessHART versus ISA 100.11a : The format war hits the factory floor. *Industrial Electronics Magazine, IEEE*, 5(4) :23–34, 2011.

[48] A. Robinson. CC2520 802.15.4 radio in Linux. `http://github.com/ab500/linux-cc2520-driver`, Dec. 2012.

[49] A. Robinson. Raspberry Pi as an 802.15.4 basestation. `http://www.raspberrypi.org/phpBB3/viewtopic.php?t=26913&p=241754`, Dec. 2012.

[50] T. J. S. Thomson, T. Narten. IPv6 stateless address autoconfiguration. RFC 4862 (Draft Standard), Sept. 2007.

[51] Z. Shelby and C. Bormann. *6LoWPAN : the wireless embedded internet*, volume 43. Wiley, 2011.

[52] Z. Shelby, S. Chakrabarti, and E. Nordmark. Neighbor discovery optimization for low power and lossy networks (6LoWPAN). Internet-Draft "draft-ietf-6lowpan-nd-21.txt", Aug. 2012. (Work in progress).

[53] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, and M. Nixon. WirelessHART : Applying wireless technology in real-time industrial process control. In *Real-Time and Embedded Technology and Applications Symposium, 2008. RTAS'08. IEEE*, pages 377–386. IEEE, 2008.

[54] L. Torvalds. Better level triggered IRQ management needed. `http://yarchive.net/comp/linux/edge_triggered_interrupts.html`, 2006.

[55] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL : IPv6 routing protocol for low power and lossy networks. RFC 6550 (Draft Standard), Mar. 2012.

[56] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1567–1576. IEEE, 2002.

Annexe A

Compilation

Cette annexe contient les commandes nécessaires à l'installation des outils de cross-compilation et les sources de certaines librairies. Elle suppose une distribution basée sur Debian disposant de l'outil de gestion des paquets *apt-get*. Dans les commandes décrites ci-dessous le caractère '#' signifie que la commande doit être exécutée avec les droits *root*. À l'inverse le caractère '$' signifie que la commande peut être exécutée avec les droits d'un simple utilisateur.

A.1 Installation de la toolchain arm-gnueabi et GCC

```
# echo "deb http://www.emdebian.org/debian testing main" >> \
     /etc/apt/sources.list
# apt-get update
# apt-get install emdebian-archive-keyring
# apt-get install linux-libc-dev-armel-cross libc6-armel-cross \
        libc6-dev-armel-cross binutils-arm-linux-gnueabi     \
        gcc-4.3-arm-linux-gnueabi g++-4.3-arm-linux-gnueabi  \
        uboot-mkimage
```

A.2 Récupération des sources de libnl3

```
# echo "deb-src http://ftp.fr.debian.org/debian/ unstable " \
     "main contrib non-free" >> /etc/apt/sources.list
$ apt-get source libnl-3-200
```

Annexe B

Liste des options supprimées

Cette annexe reprend la liste des options supprimées du noyau officiel 3.2.27 du Raspberry Pi. Cette liste est constituée de deux parties. Premièrement la liste des modules supprimés qui comprend essentiellement des pilotes de périphériques. Deuxièmement la liste des options built-in supprimées qui comprend essentiellement des sous-systèmes et des fonctionnalités propres au noyau.

B.1 Modules supprimés

```
OPROFILE                NETFILTER_XT_SET        NETFILTER_XT_TARGET_AUDIT
IP_SET                  IP_SET_BITMAP_IP        IP_SET_BITMAP_IPMAC
IP_SET_BITMAP_PORT      IP_SET_HASH_IP          IP_SET_HASH_IPPORT
IP_SET_HASH_IPPORTIP    IP_SET_HASH_IPPORTNET   IP_SET_HASH_NET
IP_SET_HASH_NETPORT     IP_SET_HASH_NETIFACE    IP_SET_LIST_SET
NET_SCH_CBQ             NET_SCH_HTB             NET_SCH_HFSC
NET_SCH_PRIO            NET_SCH_MULTIQ          NET_SCH_RED
NET_SCH_SFB             NET_SCH_SFQ             NET_SCH_TEQL
NET_SCH_TBF             NET_SCH_GRED            NET_SCH_DSMARK
NET_SCH_NETEM           NET_SCH_DRR             NET_SCH_MQPRIO
NET_SCH_CHOKE           NET_SCH_QFQ             NET_CLS_BASIC
NET_CLS_TCINDEX         NET_CLS_ROUTE4          NET_CLS_FW
NET_CLS_U32             NET_CLS_RSVP            NET_CLS_RSVP6
NET_CLS_FLOW            NET_CLS_CGROUP          NET_EMATCH_CMP
NET_EMATCH_NBYTE        NET_EMATCH_U32          NET_EMATCH_META
NET_EMATCH_TEXT         NET_ACT_POLICE          NET_ACT_GACT
NET_ACT_MIRRED          NET_ACT_IPT             NET_ACT_NAT
NET_ACT_PEDIT           NET_ACT_SIMP            NET_ACT_SKBEDIT
NET_ACT_CSUM            AX25                    NETROM
ROSE                    MKISS                   6PACK
BPQETHER                BAYCOM_SER_FDX          BAYCOM_SER_HDX
YAM                     IRDA                    IRLAN
IRNET                   IRCOMM                  IRTTY_SIR
KINGSUN_DONGLE          KSDAZZLE_DONGLE         KS959_DONGLE
USB_IRDA                SIGMATEL_FIR            MCS_FIR
BT                      BT_RFCOMM               BT_BNEP
BT_HIDP                 BT_HCIBTUSB             BT_HCIBCM203X
BT_HCIBPA10X            BT_HCIBFUSB             BT_HCIVHCI
BT_MRVL                 BT_MRVL_SDIO            BT_ATH3K
CFG80211                LIB80211                LIB80211_CRYPT_WEP
```

```
LIB80211_CRYPT_CCMP     LIB80211_CRYPT_TKIP     MAC80211
WIMAX                   NET_9P                  NFC
NFC_PN533               IWMC3200TOP             SCSI_WAIT_SCAN
BLK_DEV_MD              MD_RAID1                MD_RAID456
BLK_DEV_DM              DM_CRYPT                DM_SNAPSHOT
DM_MIRROR               DM_RAID                 DM_LOG_USERSPACE
DM_ZERO                 DM_DELAY                PPP
PPP_BSDCOMP             PPP_DEFLATE             PPP_MPPE
PPPOE                   PPP_ASYNC               PPP_SYNC_TTY
SLIP                    SLHC                    USB_CATC
USB_KAWETH              USB_PEGASUS             USB_RTL8150
USB_NET_CDCETHER        USB_NET_CDC_EEM         USB_NET_DM9601
USB_NET_GL620A          USB_NET_NET1080         USB_NET_PLUSB
USB_NET_MCS7830         USB_NET_RNDIS_HOST      USB_NET_CDC_SUBSET
USB_NET_CX82310_ETH     USB_NET_KALMIA          USB_NET_INT51X1
USB_IPHETH              USB_SIERRA_NET          USB_VL600
LIBERTAS_THINFIRM       LIBERTAS_THINFIRM_USB   AT76C50X_USB
USB_ZD1201              USB_NET_RNDIS_WLAN      RTL8187
MAC80211_HWSIM          ATH_COMMON              ATH9K_HW
ATH9K_COMMON            ATH9K                   ATH9K_HTC
CARL9170                B43                     B43LEGACY
HOSTAP                  IWM                     LIBERTAS
LIBERTAS_USB            LIBERTAS_SDIO           P54_COMMON
P54_USB                 RT2X00                  RT2500USB
RT73USB                 RT2800USB               RT2800_LIB
RT2X00_LIB_USB          RT2X00_LIB              WL1251
WL12XX_MENU             ZD1211RW                MWIFIEX
MWIFIEX_SDIO            RTL8192CU               WIMAX_I2400M
WIMAX_I2400M_USB        INPUT_FF_MEMLESS        INPUT_POLLDEV
INPUT_JOYDEV            INPUT_EVDEV             INPUT_AD714X
INPUT_AD714X_I2C        INPUT_AD714X_SPI        INPUT_ATI_REMOTE2
INPUT_KEYSPAN_REMOTE    INPUT_POWERMATE         INPUT_YEALINK
INPUT_CM109             INPUT_UINPUT            INPUT_GPIO_ROTARY_ENCODER
INPUT_ADXL34X           INPUT_ADXL34X_I2C       INPUT_ADXL34X_SPI
INPUT_CMA3000           SERIO                   SERIO_SERPORT
SERIO_RAW               GAMEPORT                GAMEPORT_NS558
GAMEPORT_L4             POWER_SUPPLY            SSB
MEDIA_SUPPORT           VIDEO_DEV               VIDEO_V4L2_COMMON
DVB_CORE                VIDEO_MEDIA             RC_CORE
LIRC                    RC_MAP                  IR_NEC_DECODER
IR_RC5_DECODER          IR_RC6_DECODER          IR_JVC_DECODER
IR_SONY_DECODER         IR_RC5_SZ_DECODER       IR_MCE_KBD_DECODER
IR_LIRC_CODEC           RC_ATI_REMOTE           IR_IMON
IR_MCEUSB               IR_REDRAT3              IR_STREAMZAP
RC_LOOPBACK             MEDIA_TUNER             MEDIA_TUNER_SIMPLE
MEDIA_TUNER_TDA8290     MEDIA_TUNER_TDA827X     MEDIA_TUNER_TDA18271
MEDIA_TUNER_TDA9887     MEDIA_TUNER_TEA5761     MEDIA_TUNER_TEA5767
MEDIA_TUNER_MT20XX      MEDIA_TUNER_MT2060      MEDIA_TUNER_MT2266
MEDIA_TUNER_MT2131      MEDIA_TUNER_QT1010      MEDIA_TUNER_XC2028
MEDIA_TUNER_XC5000      MEDIA_TUNER_XC4000      MEDIA_TUNER_MXL5005S
MEDIA_TUNER_MXL5007T    MEDIA_TUNER_MC44S803    MEDIA_TUNER_MAX2165
MEDIA_TUNER_TDA18218    MEDIA_TUNER_TDA18212    VIDEO_V4L2
VIDEOBUF_GEN            VIDEOBUF_VMALLOC        VIDEOBUF_DVB
VIDEO_TVEEPROM          VIDEO_TUNER             VIDEOBUF2_CORE
VIDEOBUF2_MEMOPS        VIDEOBUF2_VMALLOC       VIDEO_IR_I2C
VIDEO_MSP3400           VIDEO_CS53L32A          VIDEO_WM8775
```

```
VIDEO_SAA711X           VIDEO_CX25840           VIDEO_CX2341X
USB_VIDEO_CLASS         USB_GSPCA               USB_M5602
USB_STV06XX             USB_GL860               USB_GSPCA_BENQ
USB_GSPCA_CONEX         USB_GSPCA_CPIA1         USB_GSPCA_ETOMS
USB_GSPCA_FINEPIX       USB_GSPCA_JEILINJ       USB_GSPCA_KINECT
USB_GSPCA_KONICA        USB_GSPCA_MARS          USB_GSPCA_MR97310A
USB_GSPCA_NW80X         USB_GSPCA_OV519         USB_GSPCA_OV534
USB_GSPCA_OV534_9       USB_GSPCA_PAC207        USB_GSPCA_PAC7302
USB_GSPCA_PAC7311       USB_GSPCA_SE401         USB_GSPCA_SN9C2028
USB_GSPCA_SN9C20X       USB_GSPCA_SONIXB        USB_GSPCA_SONIXJ
USB_GSPCA_SPCA500       USB_GSPCA_SPCA501       USB_GSPCA_SPCA505
USB_GSPCA_SPCA506       USB_GSPCA_SPCA508       USB_GSPCA_SPCA561
USB_GSPCA_SPCA1528      USB_GSPCA_SQ905         USB_GSPCA_SQ905C
USB_GSPCA_SQ930X        USB_GSPCA_STK014        USB_GSPCA_STV0680
USB_GSPCA_SUNPLUS       USB_GSPCA_T613          USB_GSPCA_TV8532
USB_GSPCA_VC032X        USB_GSPCA_VICAM         USB_GSPCA_XIRLINK_CIT
USB_GSPCA_ZC3XX         VIDEO_PVRUSB2           VIDEO_HDPVR
VIDEO_EM28XX            VIDEO_EM28XX_ALSA       VIDEO_EM28XX_DVB
VIDEO_TLG2300           VIDEO_CX231XX           VIDEO_CX231XX_ALSA
VIDEO_CX231XX_DVB       VIDEO_USBVISION         USB_ET61X251
USB_SN9C102             USB_PWC                 USB_ZR364XX
USB_STKWEBCAM           USB_S2255               USB_DSBR
USB_SI470X              USB_MR800               DVB_USB
DVB_USB_A800            DVB_USB_DIBUSB_MB       DVB_USB_DIBUSB_MC
DVB_USB_DIB0700         DVB_USB_UMT_010         DVB_USB_CXUSB
DVB_USB_M920X           DVB_USB_GL861           DVB_USB_AU6610
DVB_USB_DIGITV          DVB_USB_VP7045          DVB_USB_VP702X
DVB_USB_GP8PSK          DVB_USB_NOVA_T_USB2     DVB_USB_TTUSB2
DVB_USB_DTT200U         DVB_USB_OPERA1          DVB_USB_AF9005
DVB_USB_AF9005_REMOTE   DVB_USB_DW2102          DVB_USB_CINERGY_T2
DVB_USB_ANYSEE          DVB_USB_DTV5100         DVB_USB_AF9015
DVB_USB_CE6230          DVB_USB_FRIIO           DVB_USB_EC168
DVB_USB_AZ6027          DVB_USB_LME2510         DVB_USB_TECHNISAT_USB2
SMS_SIANO_MDTV          SMS_USB_DRV             DVB_B2C2_FLEXCOP
DVB_B2C2_FLEXCOP_USB    DVB_STB0899             DVB_STB6100
DVB_STV090x             DVB_STV6110x            DVB_DRXK
DVB_TDA18271C2DD        DVB_CX24110             DVB_CX24123
DVB_MT312               DVB_ZL10036             DVB_ZL10039
DVB_S5H1420             DVB_STV0288             DVB_STB6000
DVB_STV0299             DVB_STV6110             DVB_STV0900
DVB_TDA8083             DVB_TDA10086            DVB_TDA8261
DVB_VES1X93             DVB_TUNER_ITD1000       DVB_TUNER_CX24113
DVB_TDA826X             DVB_TUA6100             DVB_CX24116
DVB_SI21XX              DVB_DS3000              DVB_MB86A16
DVB_TDA10071            DVB_SP8870              DVB_SP887X
DVB_CX22700             DVB_CX22702             DVB_S5H1432
DVB_DRXD                DVB_L64781              DVB_TDA1004X
DVB_NXT6000             DVB_MT352               DVB_ZL10353
DVB_DIB3000MB           DVB_DIB3000MC           DVB_DIB7000M
DVB_DIB7000P            DVB_DIB9000             DVB_TDA10048
DVB_AF9013              DVB_EC100               DVB_STV0367
DVB_CXD2820R            DVB_VES1820             DVB_TDA10021
DVB_TDA10023            DVB_STV0297             DVB_NXT200X
DVB_OR51211             DVB_OR51132             DVB_BCM3510
DVB_LGDT330X            DVB_LGDT3305            DVB_S5H1409
DVB_AU8522              DVB_S5H1411             DVB_S921
```

```
DVB_DIB8000             DVB_MB86A20S            DVB_PLL
DVB_TUNER_DIB0070       DVB_TUNER_DIB0090       DVB_LNBP21
DVB_LNBP22              DVB_ISL6405             DVB_ISL6421
DVB_ISL6423             DVB_A8293               DVB_LGS8GL5
DVB_LGS8GXX             DVB_ATBM8830            DVB_TDA665x
DVB_IX2505V             DVB_IT913X_FE           LCD_CLASS_DEVICE
BACKLIGHT_CLASS_DEVICE  BACKLIGHT_GENERIC       SND
SND_TIMER               SND_PCM                 SND_HWDEP
SND_RAWMIDI             SND_SEQUENCER           SND_SEQ_DUMMY
SND_MIXER_OSS           SND_PCM_OSS             SND_HRTIMER
SND_RAWMIDI_SEQ         SND_MPU401_UART         SND_DUMMY
SND_ALOOP               SND_VIRMIDI             SND_MTPAV
SND_SERIAL_U16550       SND_MPU401              SND_BCM2835
SND_USB_AUDIO           SND_USB_UA101           SND_USB_CAIAQ
SND_USB_6FIRE           SOUND_PRIME             HID_A4TECH
HID_ACRUX               HID_APPLE               HID_BELKIN
HID_CHERRY              HID_CHICONY             HID_CYPRESS
HID_DRAGONRISE          HID_EMS_FF              HID_ELECOM
HID_EZKEY               HID_HOLTEK              HID_KEYTOUCH
HID_KYE                 HID_UCLOGIC             HID_WALTOP
HID_GYRATION            HID_TWINHAN             HID_KENSINGTON
HID_LCPOWER             HID_LOGITECH            HID_LOGITECH_DJ
HID_MAGICMOUSE          HID_MICROSOFT           HID_MONTEREY
HID_MULTITOUCH          HID_NTRIG               HID_ORTEK
HID_PANTHERLORD         HID_PETALYNX            HID_PICOLCD
HID_QUANTA              HID_ROCCAT              HID_ROCCAT_COMMON
HID_SAMSUNG             HID_SONY                HID_SPEEDLINK
HID_SUNPLUS             HID_GREENASIA           HID_SMARTJOYPLUS
HID_TOPSEED             HID_THRUSTMASTER        HID_WACOM
HID_WIIMOTE             HID_ZEROPLUS            HID_ZYDACRON
USB_LCD                 USB_LED                 USB_APPLEDISPLAY
W35UND                  PRISM2_USB              R8712U
ZRAM                    REISERFS_FS             JFS_FS
XFS_FS                  GFS2_FS                 OCFS2_FS
OCFS2_FS_O2CB           BTRFS_FS                NILFS2_FS
QUOTA_TREE              ECRYPT_FS               HFS_FS
HFSPLUS_FS              SQUASHFS                NFSD
9P_FS                   XOR_BLOCKS              ASYNC_CORE
ASYNC_MEMCPY            ASYNC_XOR               ASYNC_PQ
ASYNC_RAID6_RECOV       CRYPTO_AES              RAID6_PQ
LZO_COMPRESS            LZO_DECOMPRESS
```

B.2 Options built-in supprimées

```
BSD_PROCESS_ACCT        BSD_PROCESS_ACCT_V3     FHANDLE
TASKSTATS               TASK_DELAY_ACCT         TASK_XACCT
TASK_IO_ACCOUNTING      AUDIT                   HAVE_SPARSE_IRQ
TINY_PREEMPT_RCU        PREEMPT_RCU             IKCONFIG
CGROUPS                 CGROUP_FREEZER          CGROUP_DEVICE
CGROUP_CPUACCT          RESOURCE_COUNTERS       CGROUP_SCHED
FAIR_GROUP_SCHED        BLK_CGROUP              NAMESPACES
UTS_NS                  IPC_NS                  USER_NS
PID_NS                  NET_NS                  SCHED_AUTOGROUP
PERF_EVENTS             VM_EVENT_COUNTERS       SLAB
```

```
PROFILING                       SLABINFO                        MODVERSIONS
MODULE_SRCVERSION_ALL           BLK_DEV_THROTTLING              IOSCHED_CFQ
CFQ_GROUP_IOSCHED               DEFAULT_CFQ                     FREEZER
CPU_HAS_PMU                     TICK_ONESHOT                    GENERIC_CLOCKEVENTS_BUILD
PREEMPT                         PREEMPT_COUNT                   HW_PERF_EVENTS
SECCOMP                         CPU_FREQ_TABLE                  CPU_FREQ_GOV_USERSPACE
CPU_FREQ_GOV_ONDEMAND           CPU_FREQ_GOV_CONSERVATIVE       SUSPEND
SUSPEND_FREEZER                 PM_SLEEP                        PM
PM_CLK                          ARM_CPU_SUSPEND                 XFRM_USER
IPV6_MULTIPLE_TABLES            NF_CONNTRACK_TIMESTAMP          NF_NAT_NEEDED
BRIDGE_IGMP_SNOOPING            VLAN_8021Q_GVRP                 NET_SCHED
NET_CLS                         CLS_U32_MARK                    NET_EMATCH
NET_CLS_ACT                     GACT_PROB                       NET_SCH_FIFO
DNS_RESOLVER                    HAMRADIO                        AX25_DAMA_SLAVE
IRDA_ULTRA                      IRDA_CACHE_LAST_LSAP            IRDA_FAST_RR
BT_L2CAP                        BT_SCO                          BT_RFCOMM_TTY
BT_BNEP_MC_FILTER               BT_BNEP_PROTO_FILTER            WIRELESS
WIRELESS_EXT                    WEXT_CORE                       WEXT_PROC
WEXT_SPY                        WEXT_PRIV                       CFG80211_DEFAULT_PS
CFG80211_WEXT                   WIRELESS_EXT_SYSFS              MAC80211_HAS_RC
MAC80211_RC_PID                 MAC80211_RC_MINSTREL            MAC80211_RC_MINSTREL_HT
MAC80211_RC_DEFAULT_MINSTREL    MAC80211_MESH                   MAC80211_LEDS
MISC_DEVICES                    BCM2708_VCHIQ                   MD
NET_VENDOR_BROADCOM             NET_VENDOR_CHELSIO              NET_VENDOR_FARADAY
NET_VENDOR_INTEL                NET_VENDOR_I825XX               NET_VENDOR_MARVELL
NET_VENDOR_MICREL               NET_VENDOR_MICROCHIP            NET_VENDOR_NATSEMI
NET_VENDOR_8390                 NET_VENDOR_SEEQ                 NET_VENDOR_STMICRO
PPP_FILTER                      PPP_MULTILINK                   SLIP_COMPRESSED
SLIP_SMART                      USB_NET_CDC_NCM                 USB_ALI_M5632
USB_AN2720                      USB_BELKIN                      USB_ARMLINUX
USB_KC2190                      WLAN                            RTL8187_LEDS
ATH9K_RATE_CONTROL              CARL9170_LEDS                   CARL9170_WPC
B43_SSB                         B43_PIO                         B43_PHY_N
B43_PHY_LP                      B43_LEDS                        B43LEGACY_LEDS
B43LEGACY_DEBUG                 B43LEGACY_DMA                   B43LEGACY_PIO
B43LEGACY_DMA_AND_PIO_MODE      P54_LEDS                        RT2800USB_RT33XX
RT2800USB_RT35XX                RT2800USB_RT53XX                RT2800USB_UNKNOWN
RT2X00_LIB_FIRMWARE             RT2X00_LIB_CRYPTO               RT2X00_LIB_LEDS
INPUT_MOUSEDEV                  INPUT_MISC                      VT_CONSOLE_SLEEP
CONSOLE_POLL                    I2C                             SSB_BLOCKIO
SSB_SDIOHOST_POSSIBLE           DVB_NET                         MEDIA_ATTACH
MEDIA_TUNER_CUSTOMISE           VIDEO_CAPTURE_DRIVERS           V4L_USB_DRIVERS
USB_VIDEO_CLASS_INPUT_EVDEV     VIDEO_PVRUSB2_SYSFS             VIDEO_PVRUSB2_DVB
VIDEO_EM28XX_RC                 VIDEO_CX231XX_RC                USB_PWC_INPUT_EVDEV
RADIO_ADAPTERS                  RADIO_SI470X                    DVB_CAPTURE_DRIVERS
DVB_FE_CUSTOMISE                FB                              FB_CFB_FILLRECT
FB_CFB_COPYAREA                 FB_CFB_IMAGEBLIT                FB_BCM2708
BACKLIGHT_LCD_SUPPORT           DUMMY_CONSOLE                   FRAMEBUFFER_CONSOLE
FONT_8x8                        FONT_8x16                       LOGO
LOGO_LINUX_CLUT224              SOUND                           SOUND_OSS_CORE
SOUND_OSS_CORE_PRECLAIM         SND_OSSEMUL                     SND_PCM_OSS_PLUGINS
SND_SEQUENCER_OSS               SND_SEQ_HRTIMER_DEFAULT         SND_SUPPORT_OLD_API
SND_VERBOSE_PROCFS              SND_DRIVERS                     SND_ARM
SND_SPI                         SND_USB                         HID_SUPPORT
HID                             USB_HID                         HID_PID
USB_HIDDEV                      USB_SUPPORT                     USB_COMMON
```

```
USB_ARCH_HAS_HCD
USB_LIBUSUAL
LEDS_GPIO
EXT4_USE_FOR_EXT23
REISERFS_FS_XATTR
JFS_POSIX_ACL
XFS_QUOTA
OCFS2_FS_STATS
FS_POSIX_ACL
PRINT_QUOTA_WARNING
FSCACHE
CACHEFILES
CONFIGFS_FS
SQUASHFS_LZO
NFS_V3
ROOT_NFS
NFSD_V2_ACL
NFSD_V4
NFS_ACL_SUPPORT
SUNRPC_GSS
CIFS_POSIX
MAC_PARTITION
MAGIC_SYSRQ
SCHED_DEBUG
STACKTRACE
BOOT_PRINTK_DELAY
RING_BUFFER
BRANCH_PROFILE_NONE
KGDB_KDB
ARM_UNWIND
CRYPTO_MANAGER
CRYPTO_MD5
CRC_ITU_T
USB_DEVICE_CLASS
USB_EZUSB
RTC_HCTOSYS
EXT4_FS_XATTR
REISERFS_FS_POSIX_ACL
JFS_SECURITY
XFS_POSIX_ACL
OCFS2_DEBUG_MASKLOG
EXPORTFS
QUOTACTL
FSCACHE_STATS
TMPFS_POSIX_ACL
SQUASHFS_XATTR
SQUASHFS_XZ
NFS_V3_ACL
NFS_FSCACHE
NFSD_V3
LOCKD
NFS_COMMON
CIFS_WEAK_PW_HASH
9P_FS_POSIX_ACL
MSDOS_PARTITION
DEBUG_FS
SCHEDSTATS
DEBUG_BUGVERBOSE
LATENCYTOP
RING_BUFFER_ALLOW_SWAP
KGDB
KDB_KEYBOARD
KEYS
CRYPTO_CBC
CRYPTO_DES
LIBCRC32C
USB_UAS
LEDS_CLASS
XVMALLOC
EXT4_FS_POSIX_ACL
REISERFS_FS_SECURITY
JFS_STATISTICS
XFS_RT
BTRFS_FS_POSIX_ACL
QUOTA
GENERIC_ACL
FSCACHE_HISTOGRAM
TMPFS_XATTR
SQUASHFS_ZLIB
NFS_FS
NFS_V4
NFS_USE_KERNEL_DNS
NFSD_V3_ACL
LOCKD_V4
SUNRPC
CIFS_XATTR
PARTITION_ADVANCED
EFI_PARTITION
DETECT_HUNG_TASK
TIMER_STATS
DEBUG_MEMORY_INIT
SYSCTL_SYSCALL_CHECK
FTRACE
KGDB_SERIAL_CONSOLE
STRICT_DEVMEM
CRYPTO_BLKCIPHER
CRYPTO_HMAC
CRC_CCITT
AUDIT_GENERIC
```

Annexe C

Scripts de backup

C.1 Sauvegarde

```
#!/bin/sh

if [ $USER != root ]
then
  echo "This script should be run as root."
  exit 1
fi

if [ $# != 1 ]
then
  echo "usage: $0 <device>"
  exit 1
fi

device=$1
if [ ! \( -b $device -a -r $device \) ]
then
  echo "The device $device is not valid."
  exit 1
fi

set -e

echo -n "Create working environment... "
archive_name="$(pwd)/rpi-backup-$(date '+%s').tar"
tmpdir=/tmp/$(mktemp -u tmpXXXXXX)
work=$tmpdir/archive
mkdir -m 700 $tmpdir
mkdir -m 770 $work
echo "done"

echo -n "Backup MBR... "
dd if=$device of=$work/mbr.bin bs=512 count=1 >/dev/null 2>&1
sfdisk -d $device > $work/mbr.sfdisk
echo "done"

echo -n "Mount partitions... "
```

```
for part in $device*
do
  partnum=$(basename $part| sed 's/[a-z]//g')

  if [ -z "$partnum" ]
  then
    continue
  fi

  partid=$(cat $work/mbr.sfdisk | grep $part | cut -d',' -f 3 | cut -
     d'=' -f 2)
  if [ "$partid" = 82 ]
  then
      continue
  fi

  mount_point=$work/$partnum
  mkdir $mount_point
  mount $part $mount_point
done
echo "done"

echo -n "Create archive... "
tar --preserve-permissions -cf "$archive_name" --exclude=/lost+found
   -C $tmpdir archive
echo "done"

echo -n "Compress archive... "
bzip2 $archive_name
echo "done"

echo -n "Checksum... "
shasum "$archive_name.bz2" > "$archive_name.bz2.sha"
echo "done"

echo -n "Umount partitions... "
for part in $device*
do
  partnum=$(basename $part| sed 's/[a-z]//g')

  if [ -z "$partnum" ]
  then
    continue
  fi

  partid=$(cat $work/mbr.sfdisk | grep $part | cut -d',' -f 3 | cut -
     d'=' -f 2)
  if [ "$partid" = 82 ]
  then
      continue
  fi

  mount_point=$work/$partnum
  umount $mount_point
  rmdir $mount_point
done
```

```
echo "done"

echo -n "Remove working environment... "
rm $work/mbr.bin
rm $work/mbr.sfdisk
rmdir $work
rmdir $tmpdir
echo "done"
```

C.2 Restauration

```
#!/bin/sh

if [ $USER != root ]
then
  echo "This script should be run as root."
  exit 1
fi

if [ $# != 2 ]
then
  echo "usage: $0 <archive> <device>"
  exit 1
fi

archive=$1
device=$2
swapsize=$3

if [ ! -r $archive ]
then
  echo "Cannot read the archive \"$archive\"."
  exit 1
fi

if [ ! \( -b $device -a -r $device \) ]
then
  echo "The device $device is not valid."
  exit 1
fi

set -e

echo -n "Create working environment... "
tmpdir=/tmp/$(mktemp -u tmpXXXXXX)
tmparch=$(mktemp -u tmpXXXXXX.tar)
work=$tmpdir/archive
mkdir -m 700 $tmpdir
mkdir -m 770 $work
echo "done"

echo -n "Decompress archive... "
bzip2 -d -k $archive -c > $tmparch
```

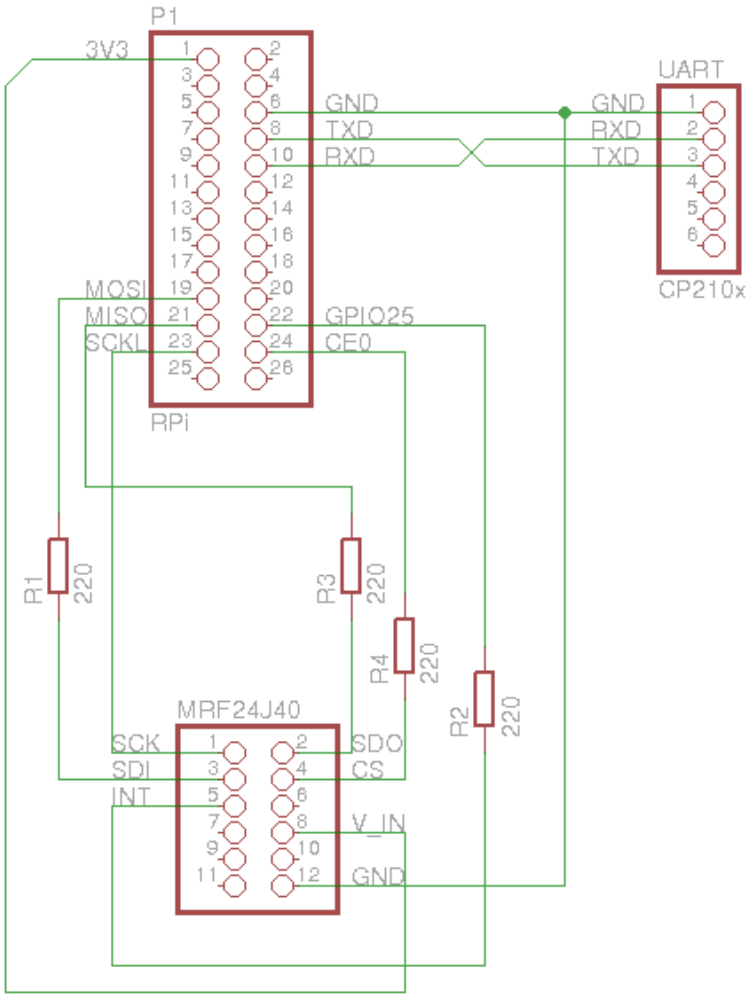
```
echo "done"

echo -n "Restore MBR... "
tar -xf $tmparch -C $tmpdir archive/mbr.bin archive/mbr.sfdisk
dd if=$work/mbr.bin of=$device bs=512 count=1 >/dev/null 2>&1
# sfdisk $device < $work/mbr.sfdisk > /dev/null
echo "done"

echo -n "Resize... "
partstart=$(fdisk -l $device | grep ${device}2 | awk '{ print $2 }')
fdisk $device <<EOF>/dev/null
d
2
n
p
2
$partstart

t
2
83
w
EOF
echo "done"

echo -n "Probing device... "
partprobe $device
echo "done"

echo -n "Format partitions... "
for part in $device*
do
  partnum=$(basename $part | sed 's/[a-z]//g')

  if [ -z "$partnum" ]
  then
    continue
  fi

  case "$partnum" in
    1) mkfs.vfat $part >/dev/null;;
    2) mkfs.ext4 $part -q -m 2 >/dev/null;;
    3) mkswap $part > /dev/null;;
    *) echo "Unknown partition $part left untouched";;
  esac
done
echo "done"

echo -n "Mount partitions... "
for part in $device*
do
  partnum=$(basename $part| sed 's/[a-z]//g')

  if [ -z "$partnum" ]
  then
    continue
```

```
  fi

  partid=$(cat $work/mbr.sfdisk | grep $part | cut -d',' -f 3 | cut -
      d'=' -f 2)
  if [ "$partid" = 82 ]
  then
      continue
  fi

  mount_point=$work/$partnum
  mkdir $mount_point
  mount $part $mount_point
done
echo "done"

echo -n "Deflate archive... "
tar --preserve-permissions -xf "$tmparch" -C $tmpdir
echo "done"

echo -n "Sync... "
sync
echo "done"

echo -n "Umount partitions... "
for part in $device*
do
  partnum=$(basename $part| sed 's/[a-z]//g')

  if [ -z "$partnum" ]
  then
    continue
  fi

  partid=$(cat $work/mbr.sfdisk | grep $part | cut -d',' -f 3 | cut -
      d'=' -f 2)
  if [ "$partid" = 82 ]
  then
      continue
  fi

  mount_point=$work/$partnum
  umount $mount_point
  rmdir $mount_point
done
echo "done"

echo -n "Remove working environment... "
rm $tmparch
rm $work/mbr.bin
rm $work/mbr.sfdisk
rmdir $work
rmdir $tmpdir
echo "done"
```

Annexe D

Montage

Cette annexe reprend le schéma de montage du transceiver MRF24J40 et de l'UART sur le Raspberry Pi. Dans le cas du MRF24J40 le pin GPIO 25 du Raspberry Pi est utilisé pour l'interruption. Le montage est réalisé sur une bread board et ne pose pas de problème particulier pour les transferts SPI.



Annexe E

Scripts

Cette annexe reprend différents scripts utilisés pour configurer plus facilement les interfaces 6LoWPAN, mettre à jour les modules et redémarrer l'interface.

E.1 Démarrage de l'interface 6LoWPAN

start–lowpan.sh

```
#!/bin/sh

if [ $# != 4 ]
then
  echo "usage: $0 LONG_ADDR SHORT_ADDR PAN_ID CHANNEL"
  exit 1
fi

long_addr=$1
short_addr=$2
pan_id=$3
channel=$4

phy=$(iz listphy | head -n 1 | cut -d' ' -f 1)

iz add $phy; sleep 1
ip link set wpan0 address $long_addr; sleep 1
ifconfig wpan0 up; sleep 1

bin/iz set wpan0 $pan_id $short_addr $channel; sleep 1

ip link add link wpan0 name lowpan0 type lowpan; sleep 1
ip link set lowpan0 address $long_addr; sleep 1
ip link set lowpan0 up; sleep 1
echo "OK!"
```

E.2 Redémarrage de l'interface

reset–lowpan.sh

```
#!/bin/sh

rmmod 6lowpan
modprobe 6lowpan
rmmod mrf24j40
modprobe mrf24j40 $*
```

restart.sh

```
#!/bin/sh

# Mise a jour des pilotes
cd linux
echo "MRF24J40"
sh install-mrf24j40
echo "6LOWPAN"
sh install-6lowpan
cd ..

# Reset de l'interface
echo "RESET"
sh reset-lowpan.sh $*
echo "Ok!"
```

E.3 Mise à jour des modules

install–6lowpan.sh

```
#!/bin/sh
echo "Was $(sha1sum /lib/modules/$(uname -r)/kernel/net/ieee802154/6
    lowpan.ko | cut -d' ' -f1)"
cp net/ieee802154/6lowpan.ko /lib/modules/$(uname -r)/kernel/net/
    ieee802154
echo "Now $(sha1sum /lib/modules/$(uname -r)/kernel/net/ieee802154/6
    lowpan.ko | cut -d' ' -f1)"
```

install–mrf24j40.sh

```
#!/bin/sh
echo "Was $(sha1sum /lib/modules/$(uname -r)/kernel/drivers/net/
    ieee802154/mrf24j40.ko | cut -d' ' -f1)"
cp drivers/net/ieee802154/mrf24j40.ko /lib/modules/$(uname -r)/kernel
    /drivers/net/ieee802154
echo "Now $(sha1sum /lib/modules/$(uname -r)/kernel/drivers/net/
    ieee802154/mrf24j40.ko | cut -d' ' -f1)"
```