Modélisation de réseaux IoT hétérogènes à des fins d'évaluation de sécurité

Jonathan Tournier

Numéro d'ordre : 2021LYSEI018

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON
Opérée au sein de :
(INSA de Lyon, CITI lab)

Ecole Doctorale InfoMaths EDA 512
Informatique Mathématique

Spécialité de doctorat : Informatique

Soutenue publiquement le 10/03/2021, par :

Jonathan Tournier

Modélisation de réseaux IoT hétérogènes à des fins d'évaluation de sécurité

Devant le jury composé de :

Maryline Laurent		**Rapporteure**
Professeur des Universités,	Telecom SudParis	
Marine Minier		**Rapporteure**
Professeur des Universités,	Université de Lorraine	
Sara Bouchenak		**Examinatrice**
Professeur des Universités,	LIRIS, INSA de Lyon	
Vincent Nicomette		**Examinateur**
Professeur des Universités,	INSA de Toulouse	
Eric Totel		**Examinateur**
Professeur des Universités,	IMT Atlantique	
Frédéric Le Mouël		**Directeur de thèse**
Professeur des Universités,	INSA de Lyon	
François Lesueur		**Co-Encadrant de thèse**
Maître de conférences,	INSA de Lyon	
Hicham Ben Hassine		**Invité**
Ingénieur,	AlgoSecure	

Département FEDORA – INSA Lyon - Ecoles Doctorales – Quinquennal 2016-2020

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** http://www.edchimie-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr INSA : R. GOURDON	**M. Stéphane DANIELE** Institut de recherches sur la catalyse et l'environnement de Lyon IRCELYON-UMR 5256 Équipe CDFA 2 Avenue Albert EINSTEIN 69 626 Villeurbanne CEDEX directeur@edchimie-lyon.fr
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** http://edeea.ec-lyon.fr Sec. : M.C. HAVGOUDOUKIAN ecole-doctorale.eea@ec-lyon.fr	**M. Gérard SCORLETTI** École Centrale de Lyon 36 Avenue Guy DE COLLONGUE 69 134 Écully Tél : 04.72.18.60.97 Fax 04.78.43.37.17 gerard.scorletti@ec-lyon.fr
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** http://e2m2.universite-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : H. CHARLES secretariat.e2m2@univ-lyon1.fr	**M. Philippe NORMAND** UMR 5557 Lab. d'Ecologie Microbienne Université Claude Bernard Lyon 1 Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69 622 Villeurbanne CEDEX philippe.normand@univ-lyon1.fr
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** http://www.ediss-lyon.fr Sec. : Sylvie ROBERJOT Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 INSA : M. LAGARDE secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Curien - 3ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tel : +33(0)4 72 44 82 32 sylvie.ricard-blum@univ-lyon1.fr
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** http://edinfomaths.universite-lyon.fr Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tel : 04.72.44.83.69 hamamache.kheddouci@univ-lyon1.fr
Matériaux	**MATÉRIAUX DE LYON** http://ed34.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction ed.materiaux@insa-lyon.fr	**M. Jean-Yves BUFFIÈRE** INSA de Lyon MATEIS - Bât. Saint-Exupéry 7 Avenue Jean CAPELLE 69 621 Villeurbanne CEDEX Tél : 04.72.43.71.70 Fax : 04.72.43.85.28 jean-yves.buffiere@insa-lyon.fr
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bât. Direction mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA de Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69 621 Villeurbanne CEDEX jocelyn.bonjour@insa-lyon.fr
ScSo	**ScSo*** http://ed483.univ-lyon2.fr Sec. : Véronique GUICHARD INSA : J.Y. TOUSSAINT Tél : 04.78.69.72.76 veronique.cervantes@univ-lyon2.fr	**M. Christian MONTES** Université Lyon 2 86 Rue Pasteur 69 365 Lyon CEDEX 07 christian.montes@univ-lyon2.fr

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

« *L'apprentissage est la seule chose que l'esprit n'épuise jamais, ne craint jamais et ne regrette jamais.* »

Léonard de Vinci

« *Il n'y a pas de question idiote, seulement une réponse idiote.* »

Albert Einstein

Résumé

L'Internet des objets évolue rapidement, avec toujours plus de protocoles et d'objets déployés, à grande échelle ou dans des environnements cloisonnés. Cependant, avec cette émergence de protocoles, viennent de nouvelles problématiques de sécurité. En effet, les protocoles IoT sont hétérogènes, spécifiques à des besoins particuliers et utilisés dans des domaines d'applications divers, les rendant complexes à sécuriser. Plusieurs solutions existent pour évaluer et améliorer la sécurité des systèmes complexes, notamment le test d'intrusion.

Dans cette thèse, nous décrivons une méthodologie de modélisation de réseaux IoT hétérogènes, utilisée comme support d'analyse dans la réalisation de tests d'intrusion. Nous centrons cette méthodologie sur des protocoles IoT à courte portée, tels que Zigbee, BLE et OS4I. Cependant, son approche modulaire lui permet d'être fonctionnelle pour le plus grand nombre de protocoles IoT, sans devoir être modifiée pour l'ajout ou l'évolution d'un protocole.

Pour cela, nous présentons d'abord une approche générique, reposant sur quatre critères, qui permet de décrire et comparer, selon un modèle homogène, plusieurs protocoles IoT. Nous exposons, ensuite, une classification des attaques concernant les protocoles IoT, en trois parties, afin de comprendre, préciser la cible de l'attaque et son impact.

Ces modèles abstraits et génériques nous permettent ainsi l'élaboration d'une structure générique, que nous appelons paquet générique. C'est sur ce dernier que repose notre processus de modélisation, composé de quatre graphes. Nous proposons une construction itérative de ces graphes, allant du graphe représentant les communications point à point du réseau, jusqu'à celui mettant en évidence les applications détectées dans le réseau. La génération de chaque graphe nécessite l'utilisation de fonctions, prenant en entrée plusieurs patterns et le graphe précédent.

Nous proposons enfin une plateforme d'expérimentations, composée de plusieurs objets, mélangeant des dispositifs propriétaires et d'autres configurables. Elle permet notamment l'évaluation de notre méthodologie de modélisation dans des conditions expérimentales différentes. Dans le cas idéal, nous constatons que la modélisation détecte toutes les applications déployées dans le réseau. Le résultat est également pertinent lorsque nous dégradons notre observation par l'utilisation du chiffrement dans les communications. En effet, nous constatons alors que toutes les applications sont détectées, malgré un nombre plus important de faux positifs.

L'ensemble des fonctions et patterns définis dans notre méthodologie de modélisation est implémenté dans IoTMap, un *framework*, que nous avons rendu open source.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

L'Internet des objets (IoT) [1, 24, 35, 95] fait référence à un réseau d'objets physiques (ou *things*) capables de communiquer entre eux et avec des entités externes, ainsi que d'interagir avec le monde réel. Chaque objet possède ses propres capacités de calcul et sensorielles entraînant des interactions complexes avec son environnement ou les utilisateurs. L'IoT évolue rapidement et le nombre d'objets connectés augmente fortement pour atteindre plusieurs milliards d'objets d'ici à 2020 [32].

De nombreuses applications ont vu le jour dans différents domaines tels que l'énergie, la santé, la gestion du trafic, etc. Ces applications ont des besoins et des contraintes différentes (échelle, couverture ou besoin énergétique) [51] ce qui accentue l'hétérogénéité dans l'IoT. Plusieurs protocoles ont été développés pour répondre à des besoins applicatifs ou métiers spécifiques et la forte concurrence du domaine implique un développement rapide et une mise sur le marché parfois précoce, reléguant souvent la sécurité au second plan.

Cependant, ces objets sont déployés dans le monde réel, parfois dans des infrastructures critiques (santé ou système industriel), ou encore en tant que domotique. La menace est d'autant plus grande que ces objets ne sont pas isolés mais plutôt connectés soit au réseau local soit directement à Internet. Sans cloisonnement efficace, la compromission d'un objet peut entraîner la compromission du réseau entier.

1.1 Motivations

L'écosystème IoT est hétérogène et composé de plusieurs sous-systèmes, tous interconnectés grâce à Internet. La figure 1.1 représente notre vision de cet écosystème IoT, avec des bâtiments intelligents, une usine connectée et plusieurs objets isolés (considérés ici comme une ville connectée).

Chaque acteur de cet écosystème peut utiliser les services accessibles depuis le Cloud et est éventuellement connecté aux autres acteurs par Internet. De plus, pour chaque acteur, plusieurs protocoles IoT et applications existent avec des exigences spécifiques. Par exemple, l'usine connectée et le bâtiment intelligent utilisent des protocoles IoT à courte portée, tels que Bluetooth low energy (BLE), ZigBee ou encore WirelessHart. Ils peuvent également être intégrés au réseau privé existant. Ces sous-systèmes intègrent des applications de surveillance (*monitoring*) et des applications capteur-actionneur (*sensor-actuator*). Les villes connectées utilisent des protocoles à longue portée, tels que LoRaWAN, SigFox ou NB-IoT. L'accès aux services hébergés dans le Cloud est possible par l'utilisation de passerelles connectées à Internet. Contrairement aux bâtiments intelligents et aux usines connectées, les dispositifs qui composent la ville connectée communiquent uniquement avec la passerelle et ne peuvent donc pas utiliser les communications pair à pair avec les autres objets. Un grand nombre d'applications existent [45], telles que la vidéosurveillance, la gestion des parkings et du trafic et toutes les applications liées aux compteurs connectés

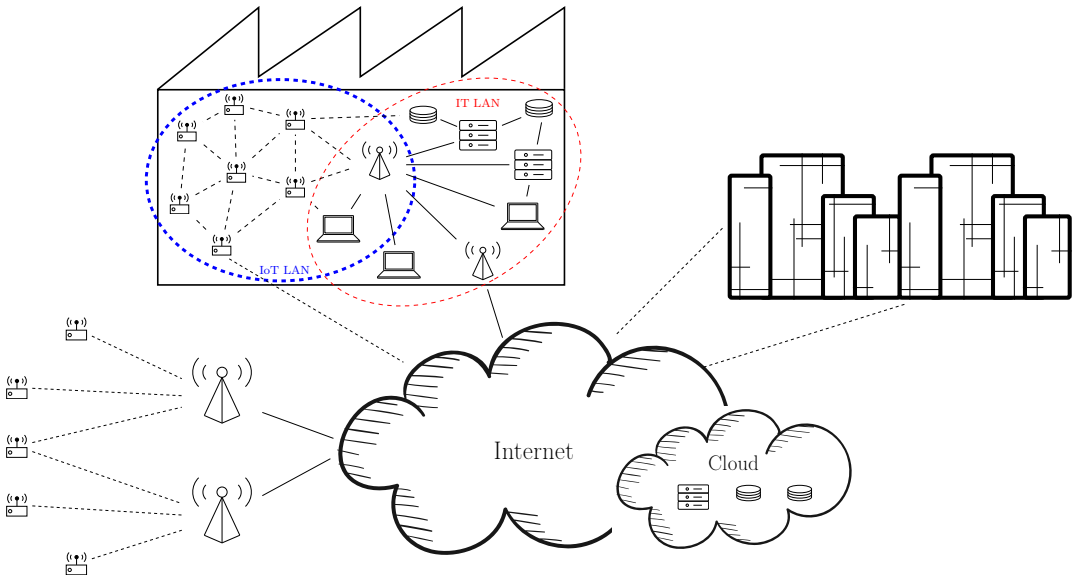

FIGURE 1.1 – Répresentation de l'écosystème IoT.

(eau, électricité, météo, etc.). Les sous-systèmes, représentés par le bâtiment intelligent et l'usine connectée, peuvent également faire partie des composants d'une ville connectée.

Avec autant d'applications disponibles apparaissent de nombreuses menaces. De nombreux travaux montrent qu'il est facile de prendre le contrôle d'objets et de réseaux IoT. Par exemple, plusieurs attaques [65, 67, 61, 64] permettent à un attaquant de prendre le contrôle total de tous les objets d'un réseau. Dans [65, 67], il est montré qu'un attaquant peut prendre le contrôle d'une serrure connectée ainsi que de l'alarme associée à la maison connectée. Dans [61], l'attaquant peut prendre le contrôle de n'importe quel objet utilisant le Bluetooth. Dès qu'un attaquant a le contrôle d'un objet, il peut alors accéder aux données sensibles stockées sur le dispositif, ou utiliser l'objet pour espionner l'utilisateur, ou encore l'utiliser en tant que relais afin de diffuser son attaque sur une plus grande surface (le malware Mirai [38] ou ses variantes). Un attaquant peut injecter une dose mortelle d'insuline depuis une pompe connectée [77], en utilisant des vulnérabilités classiques [63], telles que les attaques MITM ou *buffer overflows*. De plus, de nombreux articles détaillent comment des attaquants exploitent les objets IoT pour accéder aux informations personnelles de leurs utilisateurs [62, 66].

L'évolution des moyens de communication et des applications, dans le secteur des objets connectés, a conduit au développement de multiples protocoles. Ces derniers essayent soit de répondre aux exigences de plusieurs clients, soit de cibler un domaine spécifique. Cet excès de protocoles entraîne la définition de plusieurs spécifications, parmi lesquelles certaines peuvent ne pas être accessibles publiquement, si le protocole est propriétaire. Une majorité des travaux existants sur la sécurité dans l'IoT traite de protocoles spécifiques ou bien ciblés. Cependant, même si les protocoles ne sont pas compatibles et diffèrent dans leurs caractéristiques et fonctionnalités, ils héritent d'une architecture similaire définissant les systèmes IoT. Ces protocoles partagent donc des principes génériques et des vulnérabilités liées aux mêmes types d'attaques.

De plus, comme le secteur de l'IoT est relativement jeune, les principaux protocoles sont en constante évolution (nouveaux protocoles ou nouvelles versions). Ceci conduit à des déploiements hétérogènes que ce soit aujourd'hui avec différents

protocoles pour chaque sous-système de l'IoT, ou demain avec la présence de protocoles, dits hérités ou *legacy*, combinés à des protocoles plus récents. Certains protocoles sont conçus pour être sécurisés, répondant à tous les standards de sécurité. Cependant, l'environnement dans lequel ils sont utilisés, le nombre et le type d'objets déployés, ainsi que les besoins d'interopérabilités avec d'autres protocoles (récents ou anciens) entraînent de nouveaux risques et vulnérabilités de sécurité.

1.2 Sécurité dans l'Internet des objets

Les objets, qui composent l'IoT, sont des dispositifs électroniques incluant, pour la plupart, des systèmes d'exploitation légers et minimalistes, et autres solutions logicielles, développées et distribuées par les multiples fabricants. De plus, ces appareils sont interconnectés et utilisent les communications sans fils pour échanger des messages. Toutes ces caractéristiques impliquent la présence probable de menaces de sécurité dans l'IoT, pouvant être évaluées et classées en utilisant le principe de confidentialité, intégrité et disponibilité [90] :

— *Confidentialité* : Il s'agit de la capacité à cacher ou dissimuler de l'information à tous ceux qui ne sont pas autorisés à la consulter

— *Intégrité* : Il s'agit de la capacité à assurer que la donnée transmise n'a pas été modifiée ni altérée durant l'échange

— *Disponibilité* : Il s'agit de la capacité à rendre l'information toujours disponible, en lecture, pour toute personne autorisée.

Dans ce manuscrit, nous considérons les systèmes IoT comme étant des réseaux d'objets, avec une architecture ou une topologie spécifique. Chaque appareil peut échanger des données avec n'importe quel autre objet, à travers des communications utilisant un ou plusieurs protocoles. De plus, même s'il existe plusieurs problèmes liés aux logiciels embarqués dans les objets IoT, nous nous concentrons uniquement sur les vulnérabilités connues dans les communications et les protocoles. En effet, les difficultés logicielles ne sont pas spécifiques à l'IoT mais sont plutôt des problèmes d'IT classique.

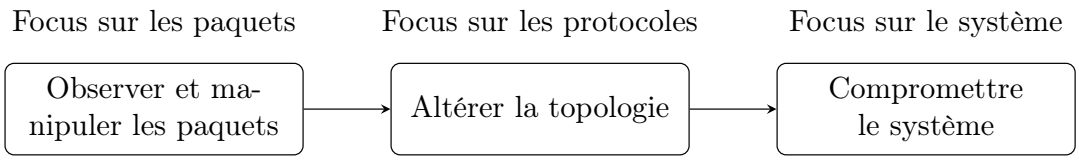

FIGURE 1.2 – La *killchain* de l'IoT.

La figure 1.2 expose un schéma d'attaque ou *killchain*, qu'un attaquant pourrait suivre pour gagner le contrôle total d'un système IoT. Cette *killchain* est divisée en 3 parties : les paquets, les protocoles et finalement le système. La première étape se concentre sur les objets et plus précisément sur les messages envoyés et reçus. La seconde étape de cette *killchain* concerne le protocole et les potentielles façons d'altérer la topologie afin de contrôler une partie ou la totalité d'un réseau. La dernière étape décrit les attaques contre le système dans sa globalité après que ce dernier soit compromis.

La sécurité des paquets est le premier élément dans la *killchain*, comme le montre la figure 1.2. Cette étape consiste à extraire des informations ou gagner potentiellement le contrôle d'un objet. Comme décrit plus tôt, seules les vulnérabilités contre

les communications et les protocoles sont considérées, excluant ainsi toutes les exploitations logicielles. Cette étape décrit donc les attaques contre les communications d'un objet vers un autre – en d'autres mots, les attaques dites cryptographiques. Nous distinguons les attaques cryptographiques dites actives et celles considérées comme passives.

Après la première étape de la *killchain*, montrée sur la figure 2.6, un attaquant est alors en mesure d'observer et de manipuler les communications dans le réseau. L'étape suivante de la *killchain* consiste à propager son contrôle ou simplement augmenter ses privilèges. Dans un système IoT, ceci consiste à altérer la topologie afin de contrôler une ou plusieurs sous-parties du réseau, en utilisant les faiblesses du protocole.

À partir de de la seconde étape, un attaquant est en mesure de modifier une partie ou l'ensemble de la topologie d'un réseau. Il peut alors essayer de modifier ou d'altérer le bon fonctionnement du système dans son ensemble.

1.3 Le test d'intrusion

Le test d'intrusion ou *penetration testing (pentest)* [14] est une solution d'évaluation de sécurité de système, dont l'approche repose sur la simulation autorisée d'une attaque. Il existe deux types de tests d'intrusion qui se différencient par le niveau d'informations connu avant le test : les pentests dits en contexte de "boîte noire", pour lesquels aucune information n'est connue ; et les pentests dits en contexte de "boîte blanche" durant lesquels l'équipe d'audit dispose de toutes les ressources nécessaires telles que des comptes utilisateurs ou administrateurs, des documents d'architecture, de réseaux, etc. Un test d'intrusion conventionnel suit une méthodologie définie en plusieurs étapes [1], illustrée sur la figure 1.3 : collecte d'informations, modèle de menaces, analyse des vulnérabilités, exploitation, post-exploitation et enfin restitution des résultats.

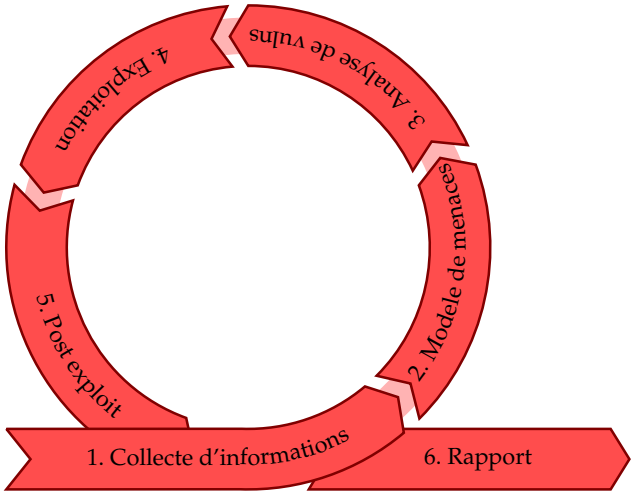

FIGURE 1.3 – *Killchain* d'un test d'intrusion.

1. http://www.pentest-standard.org/index.php/Main_Page

1.3.1 Collecte d'informations

Il s'agit de la première étape selon la méthodologie des tests d'intrusion. Elle consiste à identifier et comprendre l'environnement dans lequel l'auditeur ou pentesteur évolue. Cela commence par la découverte de tous les éléments publics, ou directement accessibles, par de simples recherches. Il faut réussir à cartographier le système cible avec le plus de détails possible. En fonction des systèmes audités, la cartographie peut se manifester de différentes façons :

— une liste des différents serveurs ou adresses IP accessibles, avec une énumération des différents services reconnus ;

— une représentation graphique, mettant en relation les différents éléments du système, interconnectés par différents canaux (WiFI, Ethernet, etc.) ;

— un diagramme UML, simplifié ou non, d'une application mettant en avant les différents services et ou fonctions utilisés.

1.3.2 Modèle de menaces

Après avoir récolté suffisamment d'informations sur la cible du test d'intrusion, la seconde étape consiste à définir les différentes menaces susceptibles d'affecter la cible. Le modèle constitue la feuille de route de l'auditeur. Il met en évidence les faiblesses du système et définit le plan d'attaque. Plusieurs scénarios sont alors envisagés pour attaquer la cible. Il convient d'être le plus exhaustif possible et d'identifier le scénario d'attaque le plus efficace contre le système cible. Pour chaque menace identifiée, il faut alors identifier si cette dernière est réalisable, en précisant les différents moyens utilisés pour y parvenir.

1.3.3 Analyse des vulnérabilités

Après l'identification des composants du système, ainsi que la définition du modèle de menaces, la troisième étape consiste à identifier et analyser les vulnérabilités potentielles du système. C'est dans cette étape, que le pentesteur statue sur la faisabilité des différents scénarios élaborés. A l'aide des informations obtenues et des vulnérabilités identifiées, l'auditeur considère si l'attaque est réalisable. Ainsi, à la fin de cette analyse, il ne reste que des scénarios d'attaques dont le potentiel de réussite est élevé.

1.3.4 Exploitation

Dans cette étape, le pentesteur va exploiter les différentes vulnérabilités afin de vérifier si ces dernières sont réellement présentes et qu'il ne s'agit pas de faux positifs. Cette étape est essentielle, mais également très sensible et nécessite des connaissances et une parfaite maîtrise des actions effectuées, pour atténuer les possibles effets de bords. En effet, l'exploitation d'une vulnérabilité peut entraîner le dysfonctionnement d'une partie ou de l'ensemble du système, si elle n'est pas maîtrisée. Lorsque l'exploitation d'une vulnérabilité est un succès, le système est alors compromis. L'attaquant a désormais accès au système, ou peut interagir avec ce dernier.

1.3.5 Post-exploitation

Cette étape n'est réalisable que si le système ciblé est compromis, autrement dit, lorsqu'une vulnérabilité est exploitée donnant un accès à l'auditeur. Depuis cet accès, plusieurs options sont alors possibles :

— L'accès obtenu dispose de droits à fort privilèges sur le système (administrateur du domaine, administrateur local, administrateur applicatif, etc.) ;

— L'accès obtenu dispose de faibles privilèges sur le système (droit utilisateur, compte de services restreint, etc.).

En fonction des privilèges obtenus, les objectifs suivants seront différents, mais la procédure pour y parvenir reste similaire. En effet, que ce soit pour élever ses privilèges ou pour obtenir des informations sensibles, l'auditeur va devoir, de nouveau, collecter des informations, en prenant en compte les nouvelles données obtenues. C'est le début d'un nouveau cycle, comme présenté dans la figure 1.3. L'auditeur effectue autant de cycles que nécessaire jusqu'à ce que l'objectif soit atteint.

1.3.6 Restitution des résultats

Dernière étape de la méthodologie d'un test d'intrusion, elle consiste à rédiger, dans un rapport, de manière précise et détaillée, les différents éléments trouvés lors du test. Ce document décrit à la fois l'historique de nos actions, le cheminement de nos tests et les résultats obtenus. Il doit présenter toute la démarche réalisée afin de donner un état des lieux, très concret, du niveau de sécurité du système audité.

Cette méthodologie est ensuite adaptée en fonction des différents types de pentests rencontrés. En effet, les étapes, bien que similaires, ne ciblent pas les mêmes informations. Par exemple, lors du test d'une application Android, les informations à récolter sont différentes que celles pour un pentest interne (ou LAN). Cependant, à notre connaissance, il n'existe aucune mise en œuvre spécifique de cette méthodologie pour la réalisation de tests d'intrusion sur des systèmes d'objets connectés.

1.4 Contributions

Dans cette thèse, nous souhaitons proposer une adaptation de la méthodologie standard d'un pentest, afin de répondre aux spécificités des environnements IoT. Nous traitons l'étape de collecte d'informations à travers trois contributions principales.

1.4.1 Approche générique des piles de protocoles IoT

Tout d'abord, nous observons que le nombre de protocoles croît avec l'émergence de l'IoT. Ils répondent à des besoins spécifiques, pour des environnements particuliers, rendant alors ces protocoles très différents les uns des autres. Cette diversité entraîne également une augmentation de la surface d'attaques et des risques de sécurité potentiels.

Nous proposons, alors, une approche générique pour décrire et comparer, selon un modèle homogène, plusieurs protocoles IoT. Elle repose sur l'utilisation de quatre critères, communs à tous les protocoles IoT : l'ouverture, l'interopérabilité, la topologie et les pratiques de sécurité. Notre approche s'appuie également sur une analyse des vulnérabilités des protocoles IoT, avec une vision globale. Ainsi, plutôt que de nous concentrer sur les vulnérabilités propres à un protocole, nous proposons un modèle qui regroupe et classifie les vulnérabilités en trois catégories : les paquets, les protocoles et le système.

Cette approche est utilisée comme base dans la définition et l'implémentation de l'étape de collecte d'informations. Nous la présentons dans le chapitre 2.

1.4.2 Modélisation de réseaux IoT hétérogènes

La collecte d'informations permet de cartographier l'ensemble du système à auditer. Pour cela, une phase d'observation est nécessaire, afin de découvrir les services et les composants déployés dans l'environnement observé. De plus, les systèmes évoluent et se complexifient, intégrant plusieurs protocoles IoT au sein d'un même réseau, permettant à des objets dédiés à un protocole d'interagir avec d'autres objets, qui utilisent des protocoles différents.

Ainsi, nous proposons une méthodologie de modélisation par des graphes, qui repose sur l'utilisation d'un format générique. Cette approche générique permet la modélisation des réseaux hétérogènes, dans lesquels plusieurs protocoles sont déployés et utilisés simultanément. Le processus de modélisation se décompose en quatre graphes, construits de manière itérative, par une détection de patterns. L'approche et la méthodologie de modélisation sont présentées dans le chapitre 3. Les fonctions utilisées pour générer les graphes, ainsi que les patterns utilisés lorsque l'observation de l'environnement est idéal, sont présentés dans le chapitre 4. Enfin, dans le chapitre 5, nous présentons plusieurs patterns permettant d'effectuer la modélisation du réseau lorsque l'observation de ce dernier est dégradée.

1.4.3 Implémentation et évaluation

Nous proposons une implémentation de notre méthodologie de modélisation, dans un *framework*, que nous appelons IoTMap. Nous avons conçu cette plateforme modulaire, afin de pouvoir s'adapter facilement à la modification ou à l'ajout de nouveaux protocoles.

Afin de déterminer l'efficacité de notre méthodologie à modéliser des réseaux IoT hétérogènes, nous développons une plateforme d'expérimentations, composée d'objets aux matériels et logiciels différents. Nous évaluons ainsi IoTMap grâce à cette plateforme, dans laquelle nous définissons plusieurs conditions expérimentales. Dans un premier temps, nous évaluons IoTMap lorsque les conditions sont idéales, avec un accès total aux communications. Puis, dans un second temps, nous dégradons notre observation par deux approches. La première consiste à réduire le niveau d'information obtenu lors de l'observation, avec l'utilisation du chiffrement par exemple. La seconde consiste à rendre l'observation partielle, soit par une perte de paquets, soit par des objets hors de portée de l'observation.

Nous présentons IoTMap dans le chapitre 6. La plateforme d'expérimentations et les résultats obtenus sont développés dans le chapitre 7.

L'approche générique des protocoles et des vulnérabilités a été publiée dans le journal *Internet of Things (IoT)* [104], de l'éditeur *Elsevier*. La méthodologie de modélisation, ainsi que les patterns utilisés dans le cas idéal, ont été publiés dans la conférence internationale *the Internet of Things* [105]. Par ailleurs, l'implémentation de cette modélisation a donné lieu à une présentation à la conférence *Pass The Salt* [103].

1.5 Plan du manuscrit

Dans le chapitre 2, nous introduisons l'état de l'art des protocoles IoT et de leur sécurité, ainsi que les méthodologies de modélisation existantes. Nous présentons également notre approche générique afin de décrire, selon un modèle homogène, les piles de protocoles observées. Puis, nous introduisons une *killchain* dans le but de présenter et catégoriser les différentes attaques contre les protocoles IoT.

Dans le chapitre 3, nous présentons notre méthodologie de modélisation de réseaux IoT hétérogènes. Nous décrivons un scénario en tant qu'exemple filant pour illustrer les divers concepts de la modélisation. Nous définissons un paquet générique sur lequel repose le processus de modélisation. Enfin, nous exposons les différents graphes qui composent cette nouvelle méthodologie.

Dans le chapitre 4, nous introduisons les différentes fonctions et patterns utilisés pour réaliser la modélisation du réseau, par un graphe de chaque couche, lorsque l'observation effectuée répond au cas idéal.

Dans le chapitre 5, nous décrivons et présentons différentes dégradations qu'une observation peut subir ainsi que les modifications apportées aux patterns pour y répondre.

Le chapitre 6 introduit IoTMap, notre *framework*, qui implémente la méthodologie de modélisation présentée dans cette thèse.

Le chapitre 7 introduit l'évaluation d'IoTMap. Nous décrivons notre plateforme d'expérimentations, constituée d'objets physiques, dans laquelle nous déployons trois protocoles et plusieurs applications. Nous présentons nos conditions expérimentales et les résultats de l'évaluation d'IoTMap, lorsque l'observation est idéale ou dégradée par une perte de paquets ou bien d'informations.

Enfin, le dernier chapitre conclut cette thèse et résume les contributions apportées. Enfin, dans une démarche constante d'amélioration de la sécurité des réseaux IoT, nous méditons sur les perspectives de recherches et les futurs travaux envisagés.

Chapitre 2

État de l'art

Dans ce chapitre, nous présentons le contexte scientifique de cette thèse et l'état de l'art des protocoles déployés et utilisés dans l'internet des objets ainsi que leur sécurité. Nous détaillons plusieurs protocoles IoT, dont la portée reste locale (LAN), avec une vision basée sur une pile générique, utilisée notamment pour les comparer selon un modèle homogène, qu'importe le protocole. Nous présentons également les architectures permettant de décrire l'écosystème IoT, ainsi que les différentes attaques pouvant être menées contre les protocoles IoT. Enfin, nous montrons que les multiples travaux et méthodologies de modélisation de réseaux IoT existants doivent évoluer pour convenir à un écosystème IoT, devenant de plus en plus hétérogène au fur et à mesure que cette technologie évolue.

2.1 Protocoles IoT

Dans cette section, nous introduisons nos critères de comparaison des piles de protocoles IoT ainsi que notre pile IoT générique. Nous utilisons ensuite ce modèle générique pour décrire plusieurs piles de protocoles IoT. Nous avons choisi de nous concentrer sur les protocoles, open stack for IoT (OS4I), BLE et Zigbee, car ils sont déployés à grande échelle. Ils ont été analysés par des chercheurs ou hackeurs et proposent des fonctions de sécurité. Nous proposons une analyse d'un plus grand nombre de protocoles dans le journal [104]. Pour chaque pile protocolaire présentée, nous analysons en premier sa structure, puis nous présentons les mécanismes de routage associés et nous terminons par une analyse du modèle de sécurité.

2.1.1 Critères de comparaison

Nous décrivons quatre critères : l'ouverture, l'interopérabilité, la topologie (architecture de réseau) et les pratiques de sécurité. Les trois premiers sont des critères techniques, et le dernier est un jugement qualitatif.

2.1.1.1 Ouverture

Le premier critère de classification est le niveau de transparence du protocole. Il correspond à la disponibilité et à l'accessibilité des informations et des ressources d'un protocole, telles que les spécifications techniques, les implémentations et le code source. À partir de ces informations, nous divisons le degré de transparence en trois niveaux : ouvert (tout est public), semi-ouvert (toutes les ressources ne sont pas disponibles) et fermé (peu d'informations sont disponibles).

Pile de protocoles ouverts Ce niveau offre un accès à chaque couche de la pile de protocoles. Tous les protocoles utilisés dans la pile sont publics et considérés comme

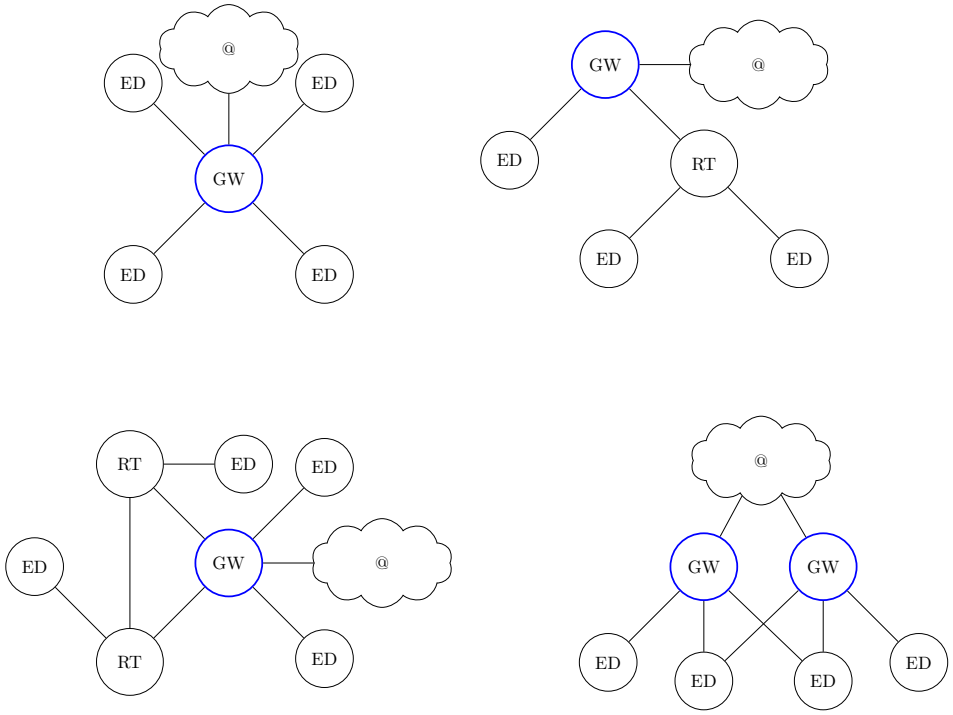

FIGURE 2.1 – Types de topologie : passerelle (GW), routeur (RT) et dispositif final (FD)

ouvert. La définition de la pile est également détaillée et accessible publiquement et les implémentations de cette dernière sont également disponibles. Chacun peut donc accéder, utiliser et modifier les implémentations, ou en proposer une à partir des informations techniques publiées.

Pile de protocoles semi-ouverts Ce niveau se situe entre un protocole ouvert et un protocole propriétaire. Il n'offre qu'un accès partiel aux informations du protocole. Cela signifie qu'il est possible de déterminer les protocoles utilisés dans la pile mais qu'un ou plusieurs d'entre eux sont propriétaires. Une autre possibilité est qu'une entreprise décide d'utiliser un protocole ouvert, mais le modifie pour mieux répondre à ses besoins et exigences, sans rendre publique les modifications.

Pile de protocoles propriétaires Ce degré de transparence n'offre que les informations que l'entreprise souhaite révéler. Aucune spécification n'est accessible publiquement. En outre, toute tentative de rétro-ingénierie du protocole sans autorisation peut entraîner des poursuites judiciaires.

2.1.1.2 Interopérabilité

Le second critère aborde l'interopérabilité de la pile de protocoles avec le monde existant. Le protocole Internet (IP) étant l'interconnexion standard, nous déterminons s'il est possible d'atteindre directement un objet, sans proxy, à partir d'Internet ou d'un autre réseau IoT.

2.1.1.3 Architecture réseau

L'architecture réseau, ou topologie, représente la façon dont les appareils sont organisés dans le réseau pour communiquer. Les piles IoT utilisent un ou plusieurs types d'architectures, avec quatre topologies principales. Cependant, selon la pile de protocoles, ces topologies peuvent être légèrement modifiées pour donner différentes possibilités. La figure 2.1 représente les quatre catégories de topologies que nous décrivons : étoile, arbre, pair-à-pair (P2P) et cellulaire.

Afin de fournir une meilleure vision générique de la topologie disponible, nous définissons trois types de nœuds, présents dans tous les protocoles avec des noms différents ou des caractéristiques spécifiques : passerelle (GW), routeur (RT) et dispositif final (ED). La passerelle est un nœud qui contrôle l'ensemble du réseau et permet la communication entre le réseau et Internet. Le routeur transmet les messages à l'intérieur du réseau entre les appareils. Enfin, le dispositif final est un nœud qui ne peut qu'envoyer ou recevoir des messages.

La topologie en étoile est basée sur un modèle *primaire-secondaire* [1], avec une passerelle (*primaire*) agissant comme un hub central unique avec lequel chaque dispositif final (*secondaire*) peut communiquer.

La topologie en arbre est basée sur les relations parents-enfants, dans lesquelles un routeur peut être un parent ou un enfant. Le dispositif final est exclusivement une feuille du réseau. Enfin, la passerelle qui contrôle l'ensemble du réseau est appelée le nœud *root*.

Dans la topologie *P2P*, il n'y a pas de nœud central par lequel chaque communication doit passer pour atteindre le nœud de destination. Tous les routeurs peuvent communiquer avec tous les autres routeurs à portée. Si chaque routeur est capable de communiquer avec tous les autres routeurs, nous parlons de topologie maillée ou *mesh*.

Enfin, la topologie cellulaire, également appelée topologie en étoile d'étoiles, est assez similaire à la topologie en étoile, avec de multiples passerelles directement liées à Internet. L'avantage de cette topologie est la possibilité pour un appareil d'être connecté à plusieurs passerelles. Ainsi, tous les messages sont envoyés plusieurs fois, et la redondance est traitée dans la partie Internet de la topologie.

2.1.1.4 Pratiques de sécurité

Le critère pratique de sécurité est un jugement personnel, idéalement basé sur des observations antérieures, concernant l'attitude des organisations, maintenant un protocole, vis à vis des problèmes de sécurité. Nous nous concentrons principalement sur les réponses aux vulnérabilités de sécurité signalées. En effet, certains protocoles sont ouverts, ont été étudiés et peuvent être améliorés par les chercheurs en sécurité. Pour d'autres, les organisations qui les soutiennent peuvent être réticentes à collaborer avec la communauté de sécurité.

Nous considérons que la communication, la transparence ou les *bug bounties* sont de bonnes pratiques pour répondre ou prévenir des problèmes de sécurité. En revanche, nous estimons que les organisations qui tentent de limiter l'analyse de la sécurité et la publication de leurs protocoles adoptent de mauvaises pratiques en matière de sécurité.

1. Dans ce manuscrit, nous utiliserons le terme primaire-secondaire plutôt que le terme maître-esclave

2.1.2 Pile IoT générique

Pour comparer les piles de protocoles IoT en détail, nous utilisons un modèle commun à plusieurs couches comme ceci est utilisé dans les réseaux traditionnels avec les modèle OSI [15] ou TCP/IP [30]. Notre modèle est largement inspiré par ces derniers, que nous avons adaptés afin de correspondre aux piles de protocoles IoT. Le modèle OSI (2), comme montré dans la figure 2.2, propose sept couches, allant de la couche physique jusqu'à la couche application. Bien que les couches présentation et session assurent des fonctionnalités adaptées pour les réseaux IT classiques, ces deux couches ne sont pas clairement définies et utilisées dans les systèmes IoT. Ainsi, nous décidons de supprimer ces couches de la pile IoT générique. Le modèle TCP/IP (3), représenté dans la figure 2.2, est décrit comme une pile à quatre couches, allant de la couche interface réseau jusqu'à la couche application. Cependant, ce modèle est utilisé pour déterminer comment un objet doit être connecté à Internet. De plus, la modification des protocoles, tels que TCP ou UDP, semble compliqué avec ce modèle.

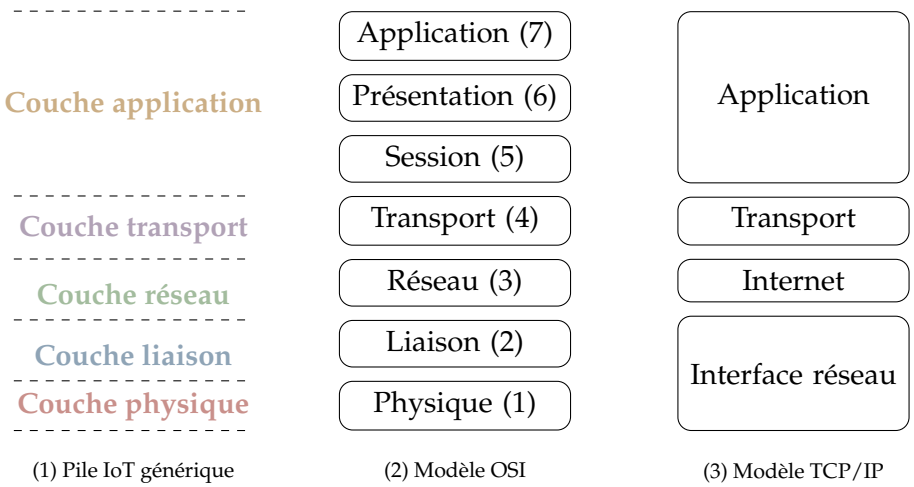

FIGURE 2.2 – (1) Pile de protocoles IoT générique comparé aux modèles (2) OSI et (3) TCP/IP.

La pile IoT générique (1) est représentée dans la figure 2.2. Nous utilisons ce modèle pour comparer toutes les piles de protocoles IoT présentées dans ce chapitre. Le modèle est composé de cinq couches, allant de la couche physique (PHY) jusqu'à la couche application (APP). Chaque couche apporte une caractéristique spécifique : les couches physique et liaison spécifient les fonctionnalités radio, la couche réseau définit le routage et les propriétés de sécurité et les couches transport et application spécifient les commandes dans le protocole.

2.1.3 OS4I

Dans cette section, nous décrivons OS4I, une pile qui repose uniquement sur des technologies ouvertes pour chaque couche. Nous appelons technologie ouverte, toute technologie dont les ressources sont accessibles publiquement et librement, que ce soit la spécification, la documentation ou le code source.

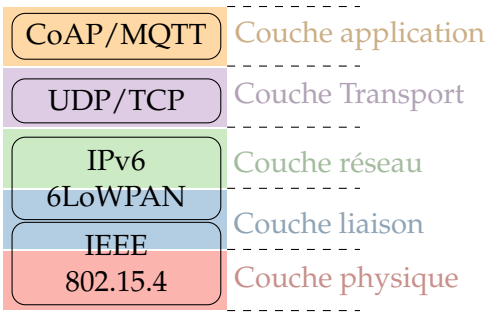

FIGURE 2.3 – Pile OS4I comparée à la pile IoT générique.

2.1.3.1 Description de la pile OS4I

La figure 2.3 représente la pile de protocoles IoT uniquement composée de technologies libres. La couche physique et la couche liaison sont définies par le standard IEEE (*Institute of Electrical and Electronics Engineers*) 802.15.4 [42]. Une couche d'adaptation, appelée 6LoWPAN, apparaît entre la couche liaison et la couche réseau. Cette dernière assure l'interopérabilité avec les réseaux non IoT qui utilisent IPv6. Plusieurs protocoles peuvent être utilisés pour la couche application tels que CoAP (*Constrained Application Protocol*) ou MQTT (*Message Queue Telemetry Transport*). En fonction du protocole IoT déployé, la couche transport est définie soit par UDP (si CoAP) soit TCP (si MQTT).

Couche IEEE 802.15.4 (PHY) La couche physique spécifie les caractéristiques radio du protocole. Cette couche assure que ce protocole opère sur les trois bandes de fréquences possibles : 2.4 GHz (avec 16 canaux), 915 MHz (avec 10 canaux) et 868 MHz (avec 1 canal). Le débit peut aller jusqu'à 250 Kbps avec une portée de communication de l'ordre de la dizaine de mètres. De plus, la couche physique offre une résistance aux collisions et d'autres fonctionnalités afin d'assurer une adaptation en temps réel.

Les bandes de fréquences disponibles peuvent être divisées en deux groupes : les bandes basses fréquences (868 ou 915 MHz) et les bandes hautes fréquences (2.4 GHz). Les deux groupes sont basés sur le DSSS (*Direct Sequence Spread Spectrum*), assurant une forte résistance contre les interférences. Cependant, en fonction de la bande de fréquences utilisée, les techniques de modulation diffèrent, impliquant des débits différents. La bande basse fréquence offre une débit de 40 Kbps et utilise la modulation BPSK (*Binary Phase Shift Keying*), alors que la bande haute fréquence offre un débit de 250 Kbps et utilise la modulation O-QPSK (*Offset Quadrature Phase Shift Keying*).

Couche IEEE 802.15.4 (MAC) La couche liaison assure des fonctionnalités telles que le transfère de données et le scanne de canaux. Cette couche définit également deux types de nœuds dans un réseau : RFD (*Reduced function Device*) et (*Full function Device*).

Un RFD agit uniquement comme un dispositif finale (ED) du réseau. Ces objets sont généralement équipés de capteurs ou d'actionneurs. Contrairement aux FFD, les RFD communiquent uniquement avec un unique FFD. D'après la figure 2.1, un RFD est un ED.

Un FFD est un objet polyvalent avec plus de puissance, qui agit soit comme un ED (comme pour un RFD), soit comme un routeur interne relayant les messages dans un réseau multi-saut (*router*) ou un coordinateur de réseau (GW). Ce dernier envoie des *beacons*, assurant les services de synchronisation, de communication et d'ajout de nouveaux objets dans un réseau. Un unique coordinateur par réseau contrôle le système et agit comme une passerelle entre les objets du réseau et Internet.

Le protocole 6LoWPAN Le protocole 6LoWPAN a été développé par l'IETF (*Internet Engineering Task Force*) [40, 71] en 2007. Sa principale fonctionnalité est d'utiliser le protocole IPv6 dans des réseaux IEEE 802.15.4. Ceci permet alors l'interconnexion des réseaux IPv6, de l'IT classique, aux réseaux 6LoWPAN.

L'utilisation d'IPv6 permet au protocole 6LoWPAN de tirer profit de ses avantages comme la détection des voisins, la résolution d'adresses par multidiffusion à portée locale, la détection d'adresses dupliquées et la découverte des routeurs du réseau. Toutefois, les paquets IPv6 sont plus volumineux que ceux de la norme IEEE 802.15.4. Par conséquent, le protocole offre divers mécanismes pour assurer une concordance entre les paquets IPv6 et 6LoWPAN, malgré d'éventuels problèmes :

— *Compression de l'en-tête* : La couche d'adaptation optimise l'espace disponible pour les données dans un paquet IPv6 en comprimant son en-tête. Cela signifie qu'il faut réduire la longueur de chaque en-tête utilisé dans le paquet, qu'il s'agisse d'un en-tête IPv6, UDP ou TCP.

— *Fragmentation et rassemblement* : Le MTU (*Maximum Transmission Unit*) d'IPv6 ne correspond pas à la taille imposée par les paquets IEEE 802.15.4. Ainsi, tous les paquets IPv6 sont fragmentés en plusieurs segments et rassemblés au niveau de la destination.

— *Transmission couche 2* : La couche d'adaptation (6LoWPAN) permet à la couche liaison de transmettre les paquets IPv6.

La méthodologie de compression des en-têtes est définie dans la RFC 6282 (*Request for Comments*) [41], mettant à jour la RFC 4944 [60]. Elle améliore les méthodes utilisées pour compresser les en-têtes IPv6 et UDP. D'autres mécanismes existent pour effectuer la compression des en-têtes pour TCP [9], mais nous ne les avons pas abordés dans notre travail, en restant concentré sur UDP uniquement.

Le niveau de compression de l'en-tête peut varier selon le type de communication choisi parmi les trois options disponibles :

— *Communication monodiffusion locale* : Ce type de communication se produit entre deux appareils au sein d'un réseau 6LoWPAN. Les adresses des appareils sont générées à partir du préfixe local de liaison et des adresses IEEE 802.15.4. Dans ce contexte, les deux adresses peuvent être déterminées à partir de l'en-tête IPv6. Ainsi, l'en-tête IPv6 peut être comprimé à 2 octets.

— *Communication multidiffusion* : Ce type de communication concerne les dispositifs qui ont des interactions en dehors de la portée du lien local, mais toujours au sein du même réseau. Grâce à l'utilisation du contexte partagé, l'en-tête peut être réduit à 4 octets.

— *Communication globale* : La communication globale fait de multiples sauts IP jusqu'à atteindre une destination en dehors du réseau. Par conséquent, les adresses source et destination doivent être définies dans l'en-tête, en l'augmentant jusqu'à 7 octets.

Couche transport La couche transport repose sur UDP ou TCP. Le choix entre ces deux protocoles est lié au protocole utilisé sur la couche supérieure. Il y a des avantages et des inconvénients à l'utilisation de chacun. Par exemple, TCP permet une meilleure qualité de service (QoS) ainsi que la robustesse et la fiabilité dans l'envoie des messages. Les connexions TCP sont par définition toujours actives, limitant l'utilisation de ce protocole dans un environnement contraint. De même, UDP permet une transmission rapide et efficace des messages mais n'a pas la fiabilité et la garantie de service de TCP.

Couche application La couche application spécifie le protocole utilisé pour assurer la communication entre les hôtes. Cette communication peut se faire soit entre ED, soit entre le serveur et le dispositif final. Dans la pile OS4I, illustrée dans la figure 2.3, nous décrivons deux protocoles ouverts : CoAP basé sur UDP et MQTT basé sur TCP.

Le groupe CoRE (*Constrained Restful Environments*) a développé CoAP [92, 78], un protocole conçu pour les appareils aux ressources limitées (batterie, mémoire ou même puissance de calcul). Ce protocole est basé sur une architecture REST (*REpresentational State Transfer*), tout comme HTTP (*HyperText Transfer Protocol*), mais sa complexité est réduite par l'utilisation d'UDP plutôt que TCP. De plus, des méthodes telles que GET, POST, PUT et DELETE sont incluses dans le protocole, ce qui permet la compatibilité avec HTTP. Ainsi, CoAP étend l'unification entre l'IT classique et l'IoT, définie par 6LoWPAN, avec la possibilité pour des applications web, bien connues, d'interagir avec des applications IoT machine-à-machine (M2M).

MQTT est un protocole standard ouvert créé à l'origine par IBM. Ce protocole est basé sur un modèle *Publish-Subscribe*, conçu pour les communications classiques ainsi que pour les communications M2M légères [7]. Contrairement à CoAP, MQTT est construit sur TCP, de sorte que la sécurité du protocole repose sur les protocoles SSL (*traditional secure sockets layer*) et TLS (*transport layer security*).

2.1.3.2 Routage dans 6LoWPAN

Le routage multi-sauts permet la transmission de paquets entre des objets au delà de leur portée radio. En fonction de la couche par laquelle le processus de routage est effectué, nous pouvons diviser les protocoles de routage en deux catégories : *mesh-under* et *route-over* [53, 20]. Pour le schéma *mesh-under*, la décision de routage est prise sur la couche d'adaptation, elle-même basée sur la couche liaison ; tandis que la couche réseau contrôle la décision de routage pour un schéma *route-over*.

Routage de type *mesh-under* La couche d'adaptation 6LoWPAN effectue une fragmentation IPv6 pour transmettre les paquets IPv6 par le protocole IEEE 802.15.4. Cette méthode nécessite l'utilisation d'un en-tête spécifique, en plus de l'en-tête IEEE 802.15.4, afin de transmettre un paquet. Cet en-tête contient l'adresse du nœud courant ainsi que celui du prochain saut ; et l'en-tête *mesh addressing* contient les adresses source et destination du paquet. De cette façon, il est possible de savoir si le fragment, ou le paquet, a atteint sa destination ou s'il doit être transmis. Dans ce cas, le paquet n'est pas rassemblé le paquet, comme pour un routage de type *route-over*. Cependant, si un fragment vient à manquer, tous les fragments sont retransmis. D'après [20], il est possible de réduire le nombre de fragments à retransmettre en utilisant la méthode de retransmission sélective. L'IETF et le groupe de travail 6LoWPAN ont proposé plusieurs protocoles pour assurer une compatibilité avec les caractéristiques du protocole 6LoWPAN. Le premier est le protocole AODV (*Ad Hoc On Demand Distance Vector Routing Algorithm*) [75], sortie en 2003. Néanmoins, il n'est

pas adapté pour le schéma de routage *mesh-under* et les adresses MAC. Le second est LOAD [73], une version dérivée d'AODV, mais son développement a été suspendu. Finalement, LOADng [22] est celui qui propose les meilleurs résultats pour un routage de type *mesh-under*. Il est également dérivé du protocole de routage ad hoc on demand distance vector routing algorithm (AODV).

Routage de type *route-over* L'autre moyen d'assurer le routage dans un réseau 6LoWPAN est de transmettre les paquets au niveau de la couche réseau. Ce schéma de routage est basé sur le routage IP, dans lequel chaque saut de la couche liaison correspond à un saut IP. Deux options se distinguent dans ce type de routage. La première option consiste à router les paquets de saut en saut. Cette option nécessite d'ajouter des informations facultatives, devant être vérifiées par chaque saut sur la route. L'autre option consiste à utiliser une table de routage IP. Cette dernière fournit des informations permettant au saut actuel de savoir quel saut doit recevoir tous les fragments. Cependant, ces deux options impliquent que tous les fragments doivent être rassemblés à chaque saut pour récupérer les informations d'en-tête IPv6. Ainsi, lorsque la couche d'adaptation reçoit tous les fragments, elle crée un paquet IP et l'envoie à la couche réseau. Si la destination finale correspond au saut en cours, le paquet est alors transmis à la couche supérieure. En revanche, s'il ne correspond pas, le paquet IP repasse par la couche d'adaptation, qui le fragmente et le transmet au saut suivant en fonction des informations de la table de routage.

Comme dans le cas du routage de type *mesh-under*, si un fragment du schéma de routage manque lors du rassemblement, tous les fragments sont retransmis. La différence dans ce dernier schéma est que la retransmission ne se fait pas à partir de la source, mais à partir du dernier saut qui a transmis le paquet IP. D'après [20], la méthode de retransmission sélective, dans le cas d'un routage de type *route-over*, permet de retransmettre uniquement les fragments perdus. En 2008, le groupe de travail ROLL (*The Routing Over Low Power and Lossy*) formé par l'IETF a proposé un protocole de routage pour les réseaux à faible puissance et à faible perte, appelé RPL [43, 110, 107]. RPL est un protocole de routage IPv6 à vecteurs de distance. Il utilise un DODAG (*Destination Oriented Directed Acyclic Graph*) pour représenter le réseau. Il est donc plus facile de calculer des opérations telles que le meilleur chemin.

2.1.3.3 Sécurité dans la pile OS4I

OS4I est une pile basée sur plusieurs protocoles IoT, dont chacun définit une couche spécifique de la pile. Chaque couche est indépendante et fournit ses propres mécanismes de sécurité.

Sécurité dans les couches physique et liaison Nous considérons uniquement la sécurité fournie par la couche liaison IEEE 802.15.4. Elle offre un chiffrement des données en utilisant l'algorithme de chiffrement par blocs AES-CCM*. Ce protocole est une variante de l'algorithme AES-CCM; il offre les mêmes caractéristiques et ajoute la possibilité d'utiliser uniquement les capacités de chiffrement (pas de signature). La taille de la clé est fixée à 128 bits, tout comme la taille du bloc de texte en clair.

Sécurité dans IPv6 La couche réseau repose sur IPv6, c'est pourquoi nous ne considérons que la sécurité IP (IPSec) comme un mécanisme de sécurité pour assurer l'interopérabilité avec Internet. *Raza et al.* ont présenté une approche d'IPSec [80] et

une approche pour échanger les clés [83] afin de les adapter aux environnements contraints.

Sécurité dans la couche TCP/UDP La sécurité assurée par cette couche dépend du protocole utilisé au niveau de la couche application. Si CoAP est utilisé, alors le protocole de transport est UDP et le mécanisme de sécurité fourni est DTLS [59, 46, 82]. En revanche, si MQTT est déployé, alors TCP est utilisé comme protocole de transport et le mécanisme de sécurité correspondant est SSL/TLS.

2.1.4 BLE

BLE est un protocole développé par le SIG (*special interest group*) Bluetooth et présenté dans la version 4.0 de la spécification Bluetooth [96]. BLE est un protocole sans fil pour les communications à courte portée dont le champ d'application est similaire à celui du Bluetooth classique, tel que les applications de contrôle et de surveillance. De par sa conception, BLE est adapté aux appareils de faible puissance et répond aux exigences de l'IoT.

Il n'y a pas de compatibilité entre les appareils utilisant BLE et ceux qui utilisent le Bluetooth classique. Par conséquent, de nombreux appareils possèdent les deux protocoles et sont appelés appareils bi-modes.

2.1.4.1 Description de la pile BLE

La figure 2.4 représente la comparaison entre la pile BLE et notre pile IoT générique. La couche application est la seule qui n'est pas définie dans la spécification BLE. Pour les autres couches, nous pouvons les diviser en deux parties : l'hôte et le contrôleur.

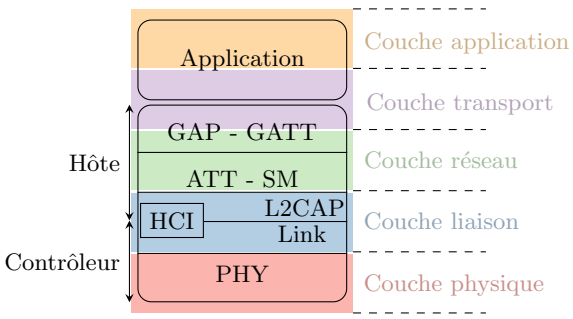

FIGURE 2.4 – Pile BLE comparée à la pile générique. Interface Hôte Contrôleur (HCI).

L'hôte gère les couches supérieures de la pile qui sont le L2CAP (*logical link control and adaptation protocol*), le protocole ATT (*Attribute*), le gestionnaire de sécurité (SM pour *Security Manager*), le GAP (*generic access profile*) et GATT (*generic attribute profile*).

Le contrôleur gère les couches les plus basses du protocole, qui sont les couches liaison et physique. Les deux parties peuvent communiquer l'une avec l'autre en utilisant l'interface Hôte-Contrôleur (HCI). Les sections suivantes présentent une vue d'ensemble de chaque couche de la pile BLE [33].

Couche physique. Comme la plupart des protocoles LAN sans fil présentés précédemment, BLE fonctionne sur la bande de fréquences ISM (*Industrial, Scientific and*

Medical) à 2,4 GHz avec 40 canaux et un saut de 2 MHz. BLE distingue deux types de canaux : les canaux d'annonces (*advertisement*) et les canaux de données. Trois canaux d'annonces sont utilisés pour la recherche d'appareils et 37 canaux de données pour les communications bidirectionnelles entre les appareils. Un mécanisme de saut de fréquences est utilisé sur les canaux de données afin de réduire les possibles interférences, pertes, trajets multiples, etc. Ce mécanisme choisit, au hasard, un canal parmi les 37 disponibles afin de communiquer à un moment donné.

La couche physique utilise la modulation GFSK (*Gaussian Frequency-Shift Keying*) pour tous les canaux et fournit un débit de données de 1 Mbps.

Couche liaison. Elle est responsable de la création et du maintien des communications, de l'annonce et du scan. Ces fonctionnalités sont assurées par trois couples de rôles définis :

— *Advertiser-scanner* : Un objet de type *advertiser* utilise les canaux d'annonces afin qu'il puisse être découvert. Un objet de type *scanner* écoute sur ces canaux en attendant de découvrir un appareil.

— *Primaire-secondaire* : Un appareil, qui possède le rôle *secondaire*, est un *advertiser* qui accepte une connexion. Un dispositif *Primaire* est un *scanner* qui a initié une connexion qui est acceptée.

— *Broadcaster-observer* : Un objet, avec le rôle *broadcaster*, envoie des annonces et refuse toute connexion. Un objet avec le rôle *observer* ne détecte que les objets qui s'annoncent sans créer de connexion.

Ces rôles sont définis en fonction du type de communication utilisé, qu'il s'agisse d'une communication monodiffusion ou *broadcast*. Dans une connexion monodiffusion, deux paires de rôles sont impliquées : les rôles *advertiser-scanner* et *primaire-secondaire*. La paire de rôle *broadcaster-observer* n'est utilisée que dans une communication *broadcast*.

Couche L2CAP. Cette couche est une version simplifiée du L2CAP du protocole Bluetooth classique. L'objectif principal de ce protocole est de multiplexer les données des couches ATT et SM dans la couche de liaison. Contrairement au L2CAP du protocole Bluetooth classique, la fonctionnalité de segmentation et de rassemblement n'est pas nécessaire dans BLE, car les couches supérieures peuvent s'occuper de la taille de leur charge utile.

Couches ATT et SM. Le protocole ATT est défini au dessus du protocole L2CAP afin de gérer la communication de type client-serveur entre des dispositifs. Le serveur stocke un ensemble de structures de données, appelées attributs, contenant les informations gérées par le GATT. Un client peut, quant à lui, accéder aux attributs en faisant une demande au serveur. De nombreux types d'opérations sont autorisés, comme la lecture et l'écriture de valeurs d'attributs. Cependant, toutes les transactions suivent un schéma strict d'arrêt et d'attente. Cela signifie qu'aucune autre demande ne peut être envoyée tant que la réponse à la demande précédente n'a pas été reçue ni traitée.

Le protocole SM vient en complément des différentes fonctionnalités de sécurité définies dans BLE. Le protocole SM est utilisé pour prendre en charge les trois phases traitées lors de la connexion des appareils ; en d'autres termes, la session de couplage. En effet, une session de couplage est déterminée par trois phases : (1) l'annonce des capacités de chaque dispositif, (2) la génération de la STK (*Short-Term Key*)

et (3) la distribution de la LTK (*Long-Term Key*), de la CSRK (*Connection Signature-Resolving Key)*) et de la clé de résolution d'identité.

Couches GAP et GATT. Le protocole GATT est construit au dessus du protocole ATT. Il détermine les rôles du client et du serveur, indépendamment des rôles *primaire* et *secondaire*. Il structure également les attributs sous forme de services et de caractéristiques, où un service est un ensemble de caractéristiques et une caractéristique un ensemble d'attributs. Ainsi, le GATT agit comme un *framework* qui utilise la brique ATT. Les principaux rôles du GATT sont la découverte de services et l'échange de caractéristiques entre les appareils.

Le GAP spécifie les rôles, les modèles et les procédures des objets pour permettre la découverte entre les nœuds, la diffusion des données et l'établissement de connexions sécurisées. Quatre rôles peuvent être attribués à un dispositif pour rejoindre un réseau : *broadcaster*, *observer*, *central* ou *peripheral*. Les rôles de *broadcaster* et d'*observer* sont les mêmes que ceux définis au niveau de la couche liaison. Ces rôles sont utilisés pour créer des communications unidirectionnelles et sans connexion. Les dispositifs avec les rôles *peripheral* et *central* sont impliqués dans des communications bidirectionnelles, dont la connexion est orientée. Le dispositif avec le rôle *peripheral* correspond à l'objet *secondaire*, et le dispositif avec le rôle *central* correspond au *primaire*. Une procédure est une séquence d'actions qu'un dispositif doit effectuer pour accomplir ses tâches. Un modèle est l'état dans lequel un appareil doit exécuter une procédure. Ainsi, un modèle dépend du rôle de l'appareil, et une procédure dépend du modèle et du rôle.

2.1.4.2 Routage dans BLE

À l'origine, le protocole BLE offrait uniquement une topologie en étoile, dans laquelle le dispositif jouant le rôle *primaire* est le nœud central et les objets, jouant le rôle *secondaire*, sont les points terminaux. Ainsi, toutes les communications passaient par le nœud central. La topologie BLE peut également être appelée topologie *primaire-secondaire* en raison du rôle joué par les dispositifs. Piconet est un autre terme utilisé dans les réseaux utilisant le protocole BLE, mais ce dernier n'est pas décrit dans ce document.

Avec la spécification de Bluetooth 5.0 [97], la topologie de type *mesh* est possible avec l'utilisation du protocole BLE [34]. De ce point de vue, de nombreux mécanismes de réseaux *mesh* ont émergé ; [25] fournit une taxonomie des différents réseaux *mesh* utilisant le protocole BLE.

Selon [34], le réseau *mesh* fournit par le protocole BLE suit un type de communications, dont les messages sont orientés, utilisant un système de messageries de type *publish-subscribe*. Cependant, il est adapté dans le protocole BLE à l'aide de relais afin de favoriser les réseaux couvrant de grands espaces. Ainsi, un message peut être retransmis par de nombreux dispositifs agissant comme relais, limité à 127 sauts. Enfin, le protocole BLE utilise une approche par *flooding* pour publier et relayer les messages. Ainsi, tous les dispositifs à portée de l'objet émetteur reçoivent le message, puis tous ceux agissant comme relais le retransmettent à tous les autres objets disponibles à portée de radio et ainsi de suite.

2.1.4.3 Sécurité dans BLE

La sécurité dans BLE comprend plusieurs modes de sécurité avec plusieurs niveaux de sécurité. Le mode et le niveau de sécurité sont basés sur la méthode d'appairage utilisée entre les objets.

Appairage. L'appairage, ou couplage, implique l'authentification de deux appareils à l'aide d'un secret partagé. Cette authentification permet le chiffrement de la connexion avec la STK et la distribution de la LTK dans le but d'assurer une communication chiffrée. Quatre procédures génèrent une STK lors d'un appairage :

— *Just works* : Utilise un code pin par défaut (000000).

— *Passkey* : Affiche un code pin sur un appareil qui doit être utilisé par l'autre.

— *OOB* : Utilise un canal auxiliaire pour transmettre les données (NFC, QRcode, etc.).

— *Numeric comparison* : Affiche un code pin identique sur les deux objets, dont la vérification doit être effectuée par l'utilisateur.

La sécurité dans BLE est divisée en deux modes, avec plusieurs niveaux de sécurité pour chaque mode :

Mode de sécurité 1. Ce mode offre une sécurité basée sur le chiffrement avec quatre niveaux de sécurité :

— Niveau 1 : Pas de sécurité (ni authentification ni chiffrement)

— Niveau 2 : Appairage sans authentification mais avec du chiffrement

— Niveau 3 : Appairage avec authentification et avec chiffrement

— Niveau 4 : Appairage authentifié avec chiffrement avant que la connexion soit établie

Mode de sécurité 2. Ce mode renforce la sécurité en se basant sur la signature des données et offre deux niveaux de sécurité :

— Niveau 1 : Appairage non authentifié avec signature des données

— Niveau 2 : Appairage authentifié avec signature des données

Le chiffrement est basé sur le protocole AES-CCM 128 bits et la clé est générée à partir de la LTK. L'authentification est assurée par l'utilisation de la CSRK.

2.1.5 Zigbee

Zigbee est un protocole basé sur la norme IEEE 802.15.4 et spécifié par l'Alliance Zigbee.

2.1.5.1 Description de la pile Zigbee

Le protocole Zigbee est basé sur la norme IEEE 802.15.4 pour les couches physique et liaison, présentées dans la section 2.1.3.1. Puis, la norme définit les couches supérieures, comme le montre la figure 2.5.

Selon [11], les couches supérieures sont composées de la couche réseau et d'une couche application, la couche application étant constituée d'un APF (*Application Framework*), d'un ZDO (*Zigbee Device Object*) et d'une APS (*Application Sub Layer*). Les sections suivantes détaillent chaque partie de ces couches.

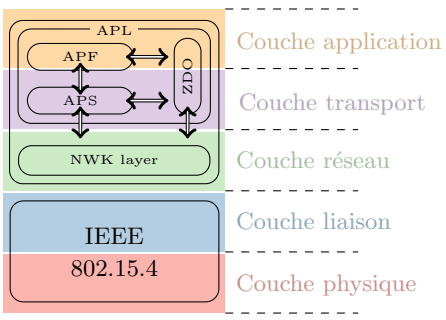

FIGURE 2.5 – Pile Zigbee comparée à la pile générique.

Couche réseau. Elle offre de nombreuses fonctionnalités, parmi lesquelles le routage multipoints, la découverte et la maintenance des routes, la sécurité et la possibilité de rejoindre ou de quitter le réseau.

APF. L'APF est composée de 254 APO (*Application Object*), dans lequel chaque APO est un logiciel qui contrôle une unité matérielle spécifique (interrupteur, lampe, etc.) disponible sur l'appareil. Tous les APO peuvent communiquer et interagir entre eux, un numéro unique étant attribué localement à chaque APO.

ZDO. Ce dernier offre des services aux APO et leur permet de découvrir les appareils et les services disponibles dans le réseau. Il fournit également des services de communications, de gestion de réseau et de sécurité.

APS Elle relie l'APO et le ZDO pour gérer les services de transfert de données entre eux. Pour créer une application, l'APS doit se conformer à un profil existant fourni par l'Alliance Zigbee. Le format du message et le protocole pour les interactions entre les APO sont définis par le profil de l'application. Le principal avantage du profil d'application est l'interopérabilité entre les applications développées par différents fabricants.

2.1.5.2 Routage dans Zigbee

Zigbee est basé sur la norme IEEE 802.15.4 pour spécifier les couches physique et liaison. Ainsi, Zigbee fournit des topologies en étoile, en arbre et *mesh*. De plus, chaque appareil qui compose un réseau Zigbee est soit un FFD soit un RFD. Cependant, leurs noms changent pour devenir Zigbee ED, Zigbee routeur, accompagné d'un coordinateur Zigbee (hub). Un Zigbee ED est un RFD ou un FFD agissant comme un simple appareil. Un routeur Zigbee est un FFD doté de capacités de routage. Un coordinateur Zigbee est propre au réseau Zigbee et correspond à un FFD contrôlant l'ensemble du réseau. L'algorithme de routage utilisé dans Zigbee dépend de la topologie du réseau déployée.

Topologie en arbre. Le seul algorithme possible dans une topologie en arbre se caractérise par des liens parents-enfants établis à la suite d'opérations de jointure, appelées *tree-based routing*. Les routeurs tiennent à jour une liste contenant l'adresse du routeur ainsi que les informations associées à leurs enfants et à leurs parents. Les communications fonctionnent avec un système de *beacon* ; le parent et l'enfant voulant communiquer doivent se synchroniser avant le début de la communication.

Topologie *mesh*. Bien que plus complexe à manipuler et avec une méthode de *beaconing* interdite, un réseau de type *mesh* est plus robuste et résilient aux défaillances qu'une topologie en arbre. Les routeurs maintiennent une table de routage et utilisent un algorithme de découverte de routes pour construire ou mettre à jour ces structures de données sur les nœuds du chemin. La table de routage n'est consultée que si le routage pour le prochain saut vers la destination n'est pas possible. Si la destination n'est pas trouvée, même dans la table de routage, alors l'algorithme de découverte de routes est lancé. Dans le pire des cas, si les ressources nécessaires à la découverte ne sont pas disponibles, le routage revient au mode de routage utilisé dans une topologie en arbre.

Algorithme de découverte de routes La découverte de routes est un processus nécessaire pour créer la table de routage en y entrant tous les nœuds disponibles et le chemin permettant de communiquer avec ces derniers. Cette table de routes est maintenue par les routeurs et le coordinateur. Le protocole Zigbee exploite le protocole AODV, en plus d'un mécanisme de *broadcast*, pour atteindre la destination.

2.1.5.3 Sécurité dans Zigbee

Le protocole Zigbee fournit des services, au niveau des couches réseau et APS, pour assurer les caractéristiques de sécurité suivantes : établissement des clés, sécurité du réseau, transport de la clé et sécurité du paquet. La pile Zigbee repose sur un modèle de confiance ouvert. Ainsi, chaque couche fait confiance aux autres.

La sécurité du réseau Zigbee est assurée par deux clés de chiffrement : la clé du réseau et la clé de liaison. La clé de réseau est utilisée pour sécuriser les communications de type *broadcast*. Cette clé, de 128 bits, est partagée entre tous les appareils du réseau. Il y a deux façons pour un appareil d'acquérir cette clé, soit par pré-installation, soit par transport de clé. La clé de liaison est utilisée pour sécuriser les communications monodiffusions au niveau la couche APL. Par conséquent, seuls deux appareils partagent la même clé de 128 bits. Une clé de liaison peut être acquise par pré-installation, par transport ou établissement de clé.

Comme Zigbee est basé sur un modèle de confiance ouvert, chaque couche fournit sa propre sécurité pour les paquets. La couche réseau fournit une clé de 128 bits avec l'algorithme AES-CCM pour assurer le chiffrement des communications et utilise l'algorithme CBC-MAC (*cipher block chaining message authentication code*) afin de garantir l'intégrité des données. L'APS assure la sécurité de sa trame en utilisant soit la clé du réseau, soit la clé de liaison. Si la première doit être utilisée, l'APS vérifie si la couche réseau assure déjà la sécurité ; si c'est le cas, l'APS n'ajoute pas de sécurité sur le paquet.

Toutes les piles protocolaires présentées dans cette section sont résumées dans le tableau 2.1, avec les critères utilisés pour les comparer.

2.2 Architecture et sécurité de l'IoT

Dans cette section, nous présentons l'état de l'art des différentes architectures proposées pour représenter l'écosystème IoT avec une vision orientée sécurité. Puis, dans un second temps, nous présentons l'état de l'art de la sécurité de l'écosystème IoT, en prenant pour modèle notre proposition de *killchain*, illustrée sur la figure 2.6.

		Piles Protocolaires		
		OS4I	BLE	ZigBee
Propriétés	Portée	LAN <100m	LAN <100m	LAN <100m
	Ouverture	Ouvert	Semi-ouvert	Semi-ouvert
	Interopérabilité	Oui	Non	Non
	Topologies	Étoile, arbre, mesh	Étoile, mesh	Étoile, arbre, mesh
	Pratiques de sécurité	Oui	Oui	Oui
	Débit	250 Kbps	1 Mbps	250 Kbps
	Fréquence	2.4 GHz	2.4 GHz	2.4 GHz
	Nœuds max	Milliers	32000	64000
	Multi-saut	Oui	Non/Oui	Oui
	Authentification	Oui	Oui/Non	Oui
	Chiffrement	AES-CCM	AES-CCM	AES-CCM

TABLEAU 2.1 – Résumé comparatif des piles protocolaires IoT.

2.2.1 Architecture de l'écosystème IoT

Nous comparons plusieurs études relatives aux protocoles IoT et leur sécurité. Ces différents travaux peuvent être divisés en trois groupes distincts :

— les travaux qui traitent de la sécurité IoT avec comme point de vue un unique protocole ;

— les travaux qui se concentrent sur la sécurité d'une seule couche de la pile IoT ;

— les travaux qui concernent la sécurité d'un système IoT dans son intégralité.

Le premier groupe d'études [31, 88, 81, 11, 89] traite de la sécurité des protocoles IoT, en se basant sur l'analyse d'un seul protocole. Par conséquent, les auteurs ne permettent pas aux lecteurs de comparer les protocoles ou de définir une vue globale des caractéristiques fondamentales des protocoles IoT. Dans ce manuscrit, nous proposons une vision abstraite, que nous estimons être un prérequis pour l'analyse et la compréhension des protocoles IoT, afin de développer de nouveaux protocoles et remédier aux vulnérabilités de sécurité fondamentale, particulièrement dans des systèmes hétérogènes.

Le second groupe d'études [68, 84, 16, 44, 52] aborde la comparaison des protocoles, mais chaque travail se concentre sur une ou deux couches du modèle OSI, comme pas exemple la couche liaison (MAC), la couche réseau, transport ou application. Par conséquent, ils n'étudient pas ou n'apportent aucune compréhension des interactions (de sécurité) entre les différentes couches. Nous sommes convaincus que les systèmes IoT sont très complexes. Par exemple, des clés réseaux peuvent être provisionnées par la couche applicative. Notre objectif est donc de fournir une compréhension globale de la pile des protocoles IoT, en décrivant et analysant toutes les couches.

Finalement, le troisième groupe d'études [57, 72, 2, 4] fournit une vue globale de la sécurité de l'IoT et des challenges associés. Cependant, ces travaux reposent sur une vision de l'architecture de l'IoT, présenté par Zhao et Ge [115]. Cette vision est basée sur une représentation générique des systèmes IoT en trois couches : la

couche perception, la couche réseau et la couche applicative. La couche perception rassemble toutes les données extraites depuis l'environnement dans lequel se trouve l'objet. La couche réseau définit les mécanismes de transmission et de traitements des données. La couche applicative fait le lien entre les données traitées et les utilisateurs finaux qui souhaitent les consulter. Cependant, selon nous, ce modèle IoT est trop abstrait qu'il en devient difficile de faire correspondre les problèmes de sécurité des protocoles IoT dans ce modèle. De plus, dans la couche perception, plusieurs confusions peuvent être faites entre les technologies et les protocoles, par exemple, RFID, WSN, ZigBee et *blockchain* se retrouvent dans le même groupe [57]. Dans ce manuscrit, plutôt qu'avoir une vision abstraite à l'échelle d'un système, nous prônons une abstraction intermédiaire à l'échelle des protocoles, qui nous permet de décrire et comparer précisément les protocoles IoT, ainsi que leurs problèmes de sécurité.

		[31]	[88]	[81]	[11]	[89]	[68]	[84]	[16]	[44]	[52]	[57]	[72]	[2]	[4]	Nos travaux
Pile (1)	OS4I					✓	(✓)	(✓)	(✓)	(✓)		(✓)			(✓)	✓
	BLE		✓								(✓)	(✓)		(✓)	(✓)	✓
	ZigBee				✓			(✓)			(✓)	(✓)		(✓)	(✓)	✓
Couche (2)	APP	(✓)	(✓)	(✓)	(✓)	(✓)			✓	✓						✓
	TRANS	(✓)	(✓)	(✓)	(✓)	(✓)	✓		✓	✓						✓
	NWK	(✓)	(✓)	(✓)	(✓)	(✓)	✓				✓					✓
	DL	(✓)	(✓)	(✓)	(✓)	(✓)		✓								✓
	PHY	(✓)	(✓)	(✓)	(✓)	(✓)		✓								✓
Système (3)	3 couches											✓	✓	✓	✓	
	Protocole	(✓)	(✓)	(✓)	(✓)	(✓)	(✓)									✓

TABLEAU 2.2 – Résumé des études en fonction des trois groupes : (1) travaux centrés sur une pile protocolaire, (2) travaux centrés sur une comparaison de protocoles en fonction d'une couche et (3) travaux centrés sur la sécurité des protocoles IoT avec comme point de vue le système entier.

L'ensemble des travaux est résumé dans le tableau 2.2. Les trois groupes abordés y sont représentés. Le symbole ✓ est utilisé lorsqu'une référence aborde le point spécifique et le symbole (✓) si le point est abordé partiellement. La cellule est laissée vide si la référence n'est pas concernée par le point. Par exemple, nous plaçons un symbole ✓ sur la ligne BLE (Pile protocolaire (1)) pour la référence [88], car cette étude analyse la pile protocole BLE dans sa globalité, allant de la couche physique à la couche applicative. Cependant, cette étude est centrée uniquement sur la pile protocolaire BLE et ne fait aucune comparaison ou d'analyse sur les autres protocoles IoT. Par conséquent, le second point, qui consiste à comparer plusieurs protocoles en fonction d'une couche spécifique (identifiée comme Couche (2) dans le tableau 2.2), est partiellement abordé, chaque cellule est donc remplie par le symbole (✓). Le dernier critère (Système (3)) se centre sur les travaux axés sur la sécurité des protocoles IoT en prenant en compte le système complet, qu'il soit représenté par le modèle en trois couches de Zhao et Ge [115], ou par une représentation intermédiaire abstraite au niveau protocolaire. Comme l'étude proposée dans [88] adresse la sécurité avec un point de vue par couche et non avec une vision globale du protocole BLE, il ne

répond que partiellement à ce dernier critère. Ainsi, la cellule est marquée par le symbole (✓).

2.2.2 Sécurité dans l'internet des objets

La figure 2.6 expose un schéma d'attaque ou *killchain* qu'un attaquant pourrait suivre pour gagner le contrôle total d'un système IoT. Cette *killchain* est divisée en 3 parties : les paquets, les protocoles et finalement le système. La première étape se concentre sur les objets et plus précisément sur les messages envoyés et reçus. La seconde étape de cette *killchain* concerne le protocole et les potentielles façons d'altérer la topologie pour contrôler une partie ou la totalité d'un réseau. La dernière étape décrit les attaques contre le système après que ce dernier soit compromis.

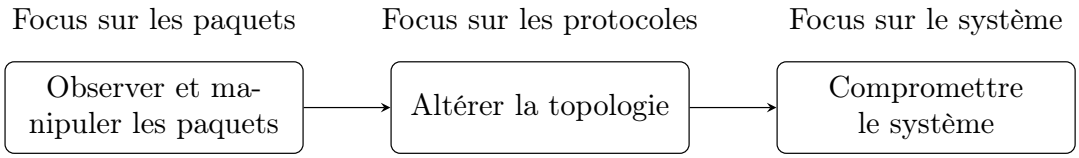

FIGURE 2.6 – La *killchain* de l'IoT.

2.2.2.1 Focus sur la sécurité des paquets

La sécurité des paquets est le premier élément dans la *killchain* comme il est montré sur la figure 2.6. Cette étape consiste à extraire des informations ou gagner potentiellement le contrôle d'un objet. Comme décrit plutôt, seules les vulnérabilités contre les communications et protocoles sont considérés, excluant ainsi toutes les exploitation logicielles. Donc, cette étape décrit les attaques contre les communications d'un objet vers un autre – en d'autres mots, les attaques dites cryptographiques. De plus, nous distinguons les attaques cryptographiques dites actives et celles considérées comme passives.

Les attaques cryptographiques passives consistent à extraire des secrets ou des données confidentielles, de messages interceptés, sans en émettre. Parce qu'aucune action n'est menée contre le réseau, le risque d'altérer l'intégrité ou la disponibilité de ce dernier est nul. Elles visent seulement à compromettre la confidentialité des données en accédant aux informations par l'écoute passive et furtive, autrement appelée *eavesdropping*.

Avec les attaques cryptographiques actives, un attaquant cherche à modifier le message transmis sur le réseau. Il peut alors injecter son propre trafic en forgeant ou en rejouant des paquets interceptés afin de perturber le réseau. Ainsi, une attaque cryptographique active vise à compromettre les trois principes de sécurité : la confidentialité, l'intégrité et la disponibilité.

Les attaques cryptographiques passives Les communications sans fils sont par nature faillibles à l'interception de trafic. Le chiffrement est une solution pour assurer que les données transmises soient protégées contre les accès non autorisés. Nous distinguons quatre contextes d'attaques cryptographiques : l'absence de chiffrement, le chiffrement étrange (*ugly*), le mauvais chiffrement (*bad*) et le bon chiffrement (*good*).

Absence de chiffrement. De nombreux systèmes IoT communiquent sans chiffrement lors de leurs échanges. Ceci rend l'obtention d'informations facile lorsque les paquets sont capturés. Dans ce cas, aucune action spécifique est requise de la part

de l'attaquant pour accéder à l'information contenue dans le message. Il examine simplement la communication sans fil et extrait les données.

Chiffrement étrange (*ugly*). Nous définissons un chiffrement étrange quand les communications sont chiffrées à l'aide d'algorithmes considérés comme faibles, tels que d'anciens algorithmes ou des algorithmes utilisant de petites clés. Dans les deux cas, ces algorithmes sont facilement cassables. [2] Dans ce cas, un attaquant peut obtenir la version en clair des communications depuis des messages chiffrés en utilisant une puissance de calcul limitée ou faible. Les communications utilisant du chiffrement étrange devraient être considérées comme équivalentes aux communications sans chiffrement.

Mauvais chiffrement (*bad*). Avec les chiffrements mauvais, l'algorithme utilisé pour chiffrer les messages est un standard à l'état de l'art. Cependant les développeurs s'appuient sur une implémentation ou un déploiement ad hoc de ces algorithmes. Souvent, ces implémentations ne sont pas conformes à tous les concepts nécessaires pour fournir un niveau de sécurité satisfaisant. Les faiblesses peuvent venir de la génération de la clé, du processus d'échange des clés ou du processus de sécurisation de la clé (chiffrement), etc.

Dans [31], les auteurs, après l'analyse des phases de chiffrement et de l'algorithme d'authentification dans un réseau Z-Wave, ont été en mesure de trouver une vulnérabilité. Cette dernière leur a permis de prendre le contrôle d'une porte connectée. Il apparaît que la clé de chiffrement n'est pas transmise en clair. Cependant, les auteurs affirment que la clé est générée en utilisant un générateur pseudo-aléatoire de nombres fournit par le chip Z-Wave, puis elle est chiffrée avec une clé temporaire inscrite dans le logiciel. Les auteurs ont trouvé que cette clé a pour valeur 16 octets de 0 et sont parvenus à la déchiffrer.

Bon chiffrement (*good*) Avec les chiffrements bons, l'algorithme utilisé pour chiffrer les communications est suffisamment robuste pour être difficile voire impossible à casser dans un temps fini. Ce chiffrement repose sur des algorithmes standards, tels qu'AES, avec les bons paramètres, une taille de clé suffisante, une implémentation valide et un déploiement éprouvé. Ici, l'attaquant peut ne pas être capable de déchiffrer le message, cependant il est toujours possible d'identifier certains motifs qui permettent à un attaquant d'obtenir de l'information.

Les attaques cryptographiques actives Une bonne implémentation d'un standard cryptographique peut parfois n'être pas suffisant pour assurer une parfaite sécurité des données. D'autres mécanismes sont également importants pour sécuriser les communications dans un réseau. Cependant, pour tester la présence de ces mécanismes, un attaquant doit injecter du trafic dans le réseau ou altérer les messages qu'il intercepte.

Les attaques *Same-nonce*. En cryptographie, un *nonce* est un nombre aléatoire qui doit être utilisé une seule fois. Ce nombre peut être généré par un générateur aléatoire ou pseudo-aléatoire. Le principal objectif de ce *nonce* est d'éviter la réutilisation d'un message lors d'une attaque rejeu. Une attaque *same-nonce* permet à un attaquant de récupérer partiellement de l'information depuis des messages chiffrés.

2. https://www.wired.com/story/hackers-steal-tesla-model-s-seconds-key-fob/

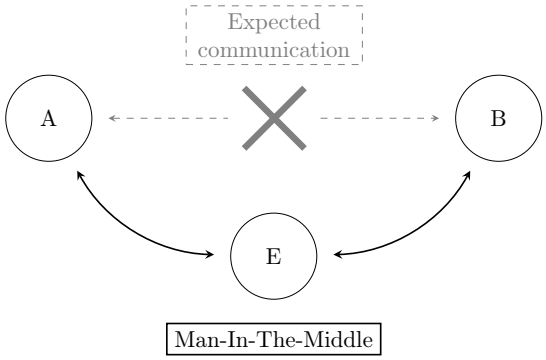

FIGURE 2.7 – Schéma classique d'une attaque MITM.

Les attaques rejeu Une attaque rejeu consiste à renvoyer le paquet intercepté au même objet, un instant plus tard. Les conséquences d'une attaque rejeu sont diverses, telles que le rejeu de commandes et les retransmissions de données. Par conséquent, une attaque rejeu ne permet pas à un attaquant d'obtenir de nouvelles informations depuis le paquet intercepté ; cette attaque représente plutôt un problème d'intégrité et non de confidentialité.

Les attaques de malléabilité Une attaque de malléabilité est une combinaison des deux attaques précédentes : le rejeu et le *same-nonce*. En d'autres mots, dans une attaque de malléabilité, un attaquant crée un nouveau paquet, le chiffre et l'envoie dans le réseau. Dans une attaque *same-nonce*, l'attaquant récupère le message en clair, puis à l'aide du même chiffré (celui correspondant au message en clair), l'attaquant peut retrouver la clé ou la suite de clé en utilisant l'opération XOR : $C = P \oplus K \rightarrow K = C \oplus P$, où C est le message chiffré, P correspond au message en clair et K la clé. Avec la clé, il est possible de créer de nouveaux messages chiffrés. Plutôt que d'utiliser une attaque rejeu, l'attaquant peut alors envoyer des paquets nouvellement créés contenant le même compteur.

2.2.2.2 Focus sur la sécurité des protocoles

Après la première étape de la *killchain* montré en figure 2.6, un attaquant est alors en mesure de récupérer certaines informations ou potentiellement prendre le contrôle d'un objet. L'étape suivante est de propager son contrôle ou simplement d'augmenter ses privilèges. Dans un système IoT, ceci consiste à altérer la topologie pour contrôler une ou plusieurs sous parties du réseaux en utilisant les faiblesses du protocole.

L'attaque MiTM L'attaque MiTM (*Man in The Middle*) est une attaque très connue dans les réseaux et protocoles de communications. Elle consiste à secrètement intercepter, modifier et relayer les informations entre deux entités (objets ou personnes) qui croient qu'elles communiquent ensemble. La figure 2.7 est une représentation du schéma classique d'une attaque MiTM, dans laquelle les entités A et B sont légitimes et l'entité E est un attaquant essayant d'intercepter le trafic.

L'attaque de *flooding* Une attaque de *flooding* consiste à envoyer une succession de requêtes à un objet en particulier, afin que ce dernier épuise la plupart de ses ressources dans le traitement de ces requêtes. Dans un contexte comme l'IoT, où les objets disposent de peu de ressources (batterie, puissance de calcul, etc), cette action

peut entraîner une réduction ou un arrêt d'activité de l'objet ciblé. De plus, cette attaque peut être divisée en plusieurs attaques avec des conséquences différentes, telles que l'attaque *Hello flood* [108]. Généralement, une attaque de *flooding* est considérée comme la plus simple attaque pouvant entraîner un déni de service (DoS).

L'attaque de *Spoofing* Une attaque de *spoofing* est une attaque bien connue en sécurité des réseaux. Elle décrit un attaquant ayant réussit à se faire passer pour un autre objet du réseau en altérant ou modifiant les données de la cible. Une attaque de *spoofing* consiste à utiliser un objet que l'attaquant contrôle afin d'usurper ou imiter un objet légitime dans le réseau pour effectuer différentes actions, comme par exemple une déconnexion ou l'envoie de fausses informations.

L'attaque *sybil* L'attaque *sybil* vise à créer et utiliser un grand nombre d'identités depuis un unique nœud du réseau, considéré comme malveillant. Cette attaque est principalement utilisée pour cibler la réputation des systèmes distribués, mais elle est également une menace pour les protocoles de routage distribués. Avec un grand nombre d'identités créées, un attaquant possède une très grande influence dans le réseau, ce qui peut affecter son efficacité, avec par exemple la création d'un *single point of failure* au sein du réseau.

Cette attaque est réalisable uniquement si le réseau ne possède pas d'entité agissant comme autorité centrale de confiance, qui vérifie l'identité de chaque entité avant son inclusion dans le réseau. Ce manque de contrôle est une faiblesse courante des réseaux sans fils ad hoc de capteurs.

L'attaque *wormhole* Dans les attaques *wormholes* [39], l'attaquant utilise un tunnel entre deux nœuds malveillants pour relayer plus rapidement des messages d'une zone A vers une zone B ou en faisant moins de sauts qu'il ne faudrait avec l'utilisation de l'algorithme de routage classique. L'attaquant peut utiliser un lien filaire, ou une technologie sans fil plus rapide, entre les deux points relais. Cette manipulation permet aux nœuds contrôlés par un attaquant d'être considérés comme le meilleur chemin entre les deux zones. Ainsi, la plupart des paquets entre ces deux zones sont transmis en suivant cette route. L'attaquant peut ensuite gagner des privilèges en écoutant ou en manipulant les paquets routés. Cette attaque est très dangereuse, car elle n'exige pas de connaître les clés cryptographiques du réseau. Donc, cette attaque est réalisable même si le réseau assure la confidentialité et l'intégrité des communications.

2.2.2.3 Focus sur la sécurité du système

Après la seconde étape de la *killchain*, présentée dans la figure 2.6, un attaquant est en mesure de modifier une partie ou l'ensemble de la topologie d'un réseau. L'attaquant peut alors essayer de modifier ou d'altérer le bon fonctionnement du système dans son ensemble.

L'attaque *sinkhole* Une attaque *sinkhole* se décrit par la création d'un point centralisé depuis un nœud contrôlé par l'attaquant pour attirer tout le trafic en provenance d'une zone spécifique. De plus, tous les objets de cette zone doivent communiquer en passant par cet objet malveillant. Cette attaque est également une condition d'exécution de nombreuses autres attaques, comme par exemple l'attaque *selective forwarding*.

Afin que le nœud contrôlé par l'attaquant soit considéré comme le point central par tous les autres nœuds, il doit être défini comme un bon choix par l'algorithme de routage. En fonction de l'algorithme utilisé, les critères pour devenir un nœud attractif diffèrent. Cependant, une route de grande qualité et des transmissions à faible latence sont deux points qui permettent de devenir un nœud central dans toutes les communications. L'usurpation ou le rejeu d'un paquet de contrôle (*advertisement*) du réseau avec comme contenu les spécifications d'un chemin de qualité sont de parfaits exemples pour réaliser cette attaque.

L'attaque *selective-forwarding* Une attaque *selective-forwarding* ne peut être réalisée dans un réseau que si les nœuds actifs du réseau ne modifient pas leur configuration de routage lorsqu'ils transfèrent des paquets. Cette attaque consiste à laisser passer uniquement le trafic désiré par les nœuds malveillants. Les paquets restants ne répondant pas aux critères de l'attaquant sont supprimés pour s'assurer qu'ils ne sont pas propagés.

Il y a deux méthodes pour implémenter cette attaque. la première est d'utiliser des nœuds malveillants, les faire devenir des trous noirs et les forcer à refuser de transférer tous les paquets. Cependant, ce comportement peut mener les nœuds légitimes à chercher une autre route pour envoyer leurs paquets. Par conséquent, la seconde méthode, plus efficace, consiste à supprimer uniquement les paquets considérés comme non essentiels pour le bon fonctionnement du réseau.

	Attaques \ Protocoles	OS4I	BLE	ZigBee
Paquets	Cryptographie passive	[8]	[88, 99, 114]	[29]
	Cryptographie active	[85]	[56]	[29, 70, 28]
Protocoles	MITM		[56]	[11]
	Flooding	[85, 76, 108]		[29]
	Sybil	[108]	✗	[50]
	Spoofing	[55]		[29]
	Wormhole	[74, 108]	✗	[11]
	denial-of-service (DoS)	[69]		[102, 28]
Systèmes	Sinkhole	[108]	✗	[48, 23]
	Selective forwarding	[108]	✗	[11]

TABLEAU 2.3 – Résumé des attaques selon la *killchain*.

Le tableau 2.3 résume toutes les attaques présentées et détaillées dans cette section pour chaque protocole étudié. Quand une attaque est réalisable en théorie ou en pratique pour chaque protocole, la cellule est remplie par une ou plusieurs références. Les cellules vides mettent en évidence les attaques pour lesquelles la documentation n'est pas suffisante, que ce soit sur le protocole ou sur l'attaque. Finalement,

lorsque nous estimons qu'une attaque ne peut pas être réalisée, la cellule est remplie par un ✗. L'hypothèse sur la faisabilité d'une attaque sans référence pour la justifier dépend de notre analyse des protocoles ainsi que de notre expérience.

2.3 Modélisation des réseaux IoT

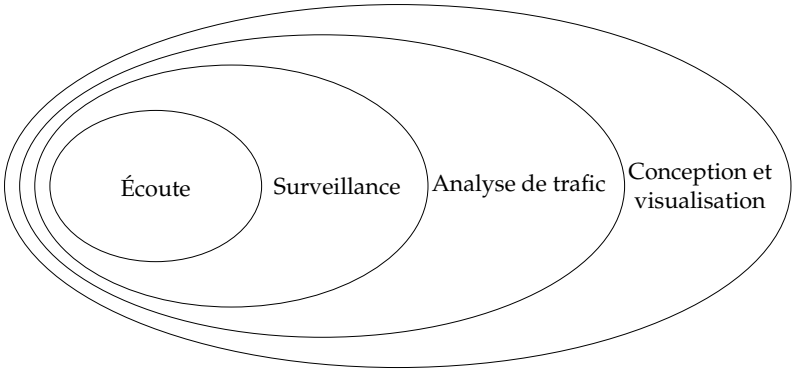

FIGURE 2.8 – Représentation des travaux en quatre groupes.

Il existe de nombreux travaux sur la capture et l'analyse des réseaux IoT. Ces études peuvent être catégorisées en quatre groupes différents, comme l'illustre la figure 2.8 :

— **Écoute** : les travaux sont centrés sur la capture et la manipulation de paquets ;

— **Surveillance** : les travaux sont centrés sur la surveillance des communications ;

— **Analyse de trafic** : les travaux sont centrés sur l'analyse de trafic ;

— **Conception et visualisation** : les travaux sont centrés sur la surveillance et l'analyse de trafic simultanément.

Dans la figure 2.8, nous représentons les différents groupes de travaux de façon encapsulée. Nous estimons que la réalisation des travaux nécessitent la connaissance et les méthodes développées et proposées dans les travaux présentés dans les groupes encapsulés. Ainsi, nous définissons l'ordre suivant : les travaux de d'écoute sont encapsulés par les travaux de surveillance, eux même encapsulés dans les travaux d'analyse de trafic, également encapsulés par les travaux de conception et de visualisation. Par exemple, dans la figure 2.8, la bulle couvrant les travaux d'analyse de trafic encapsule les travaux de surveillance ainsi que ceux d'écoute. Ceci implique que pour réaliser l'analyse de trafic, les méthodes et outils présentés dans les travaux de surveillance et ceux d'écoute sont indispensables.

2.3.1 Travaux sur l'écoute

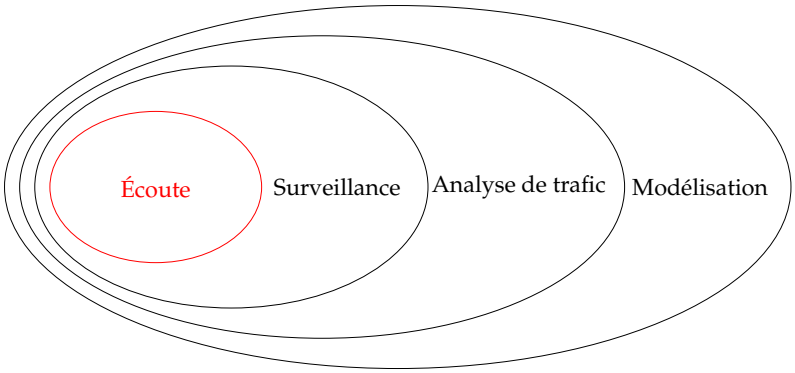

FIGURE 2.9 – Représentation des travaux en quatre groupes. Focus sur le groupe Écoute.

Ces premiers travaux présentent des *frameworks* qui fournissent une interface entre les signaux radios et l'information numérique. Ces *frameworks* sont des ensembles d'outils permettant aux utilisateurs d'interagir avec le protocole pour faciliter sa compréhension. Ils mettent à disposition plusieurs fonctionnalités complexes telles que l'écoute passive, la reconnaissance de réseaux, la découverte d'objets et des fonctions afin d'émettre et recevoir des messages par radio. Ils sont, par nature, spécifiques à un protocole en particulier. En effet, le *framework* repose généralement sur du matériel (*hardware*) particulier, correspondant parfaitement aux spécifications du protocoles.

Dans [111], Joshua Wright présente son *framework* d'attaque pour le protocole ZigBee, appelée KillerBee. Cet ensemble d'outils peut intercepter, injecter et manipuler du trafic 802.15.4 et ZigBee.

Hall Joseph *et al.* introduisent leur *framework* d'attaque du protocole Z-Wave dans l'étude [37]. Comme KillerBee, EZ-Wave est en mesure d'intercepter et manipuler le trafic de réseaux Z-Wave.

Damien Cauquil introduit Btlejack [17], un *framework* d'attaque spécifique au protocole BLE. Comme les deux *frameworks* précédents, BtleJack possède diverses fonctionnalités, dont des capacités d'interception de nouvelles connexions ou de connexions existantes. En effet, ce *framework* agit différemment de Killerbee, il ne peut pas intercepter toutes les communications BLE à proximité. La spécificité du protocole BLE ne permet pas à une seule instance de BtleJack d'intercepter les communications de plusieurs objets différents.

De notre point de vue, ces travaux sont relatifs aux couches physique et liaison de la pile IoT. Ainsi, ces travaux ne sont pas suffisant pour produire une modélisation complète des communications observées. Cependant, ils sont la base de tous les travaux suivants.

2.3.2 Travaux sur la surveillance

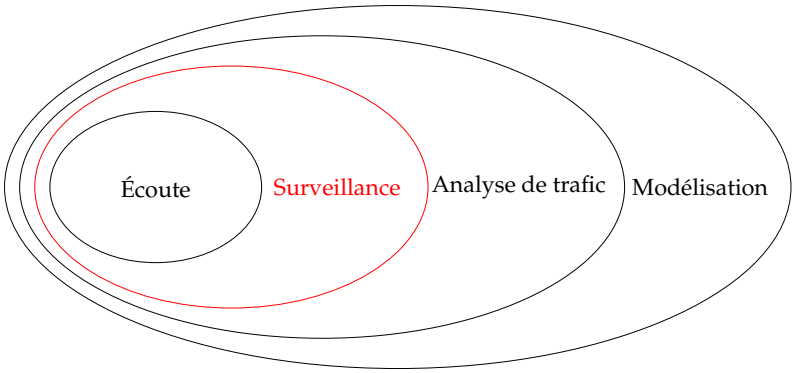

FIGURE 2.10 – Représentation des travaux en quatre groupes. Focus sur le groupe Surveillance.

Le second groupe d'études présente des travaux qui se concentrent sur la surveillance de trafic. Les travaux présentés dans cette section sont fortement liés à ceux présentés dans le groupe d'écoute. En effet, si les travaux d'écoute offrent une interface entre le physique et le numérique, au moyen de périphériques capables d'intercepter les signaux radios pour les transformer en informations numériques, les travaux de surveillance utilisent ces informations pour comprendre et analyser le réseau observé. Il existe deux approches différentes pour surveiller des réseaux : une approche passive et une active.

L'approche active repose sur l'installation de modules directement sur les nœuds du réseau afin de capturer les messages envoyés et reçus. Une méthode active offre de meilleures capacités d'interception, mais requiert une partie significative des ressources de l'objet. Plusieurs travaux [79, 106, 87] fournissent des solutions pour surveiller le trafic des réseaux. Ils peuvent détecter les défaillances des objets, lister des voisins, identifier la topologie et même visualiser l'état du réseau depuis une interface. Cependant, ces outils impliquent de modifier le comportement des objets, en ajoutant du code applicatif. Cette exigence n'est pas toujours réalisable : dans le cas où l'objet est déjà déployé ou qu'il n'est pas reprogrammable.

Une approche passive consiste à utiliser des périphériques pour intercepter les signaux radios et convertir ces derniers en informations numériques. Une méthode passive n'impacte pas les ressources de l'objet, mais elle est plus sujette aux interférences et à la perte de paquets. Dans [19, 112, 12, 47], de nombreux problèmes sont traités pour améliorer la surveillance de trafic en utilisant des approches passives. Ces travaux donnent des solutions pour procéder à la fusion de captures provenant de plusieurs antennes, comme par exemple la normalisation du temps et des paquets dupliqués. Ces travaux décrivent également des algorithmes permettant d'analyser le trafic, détecter les interactions entre les objets et inférer les chemins de routage. Finalement, ils fournissent tous une interface pour visualiser le réseau tel qu'il intercepté. Dans notre approche, nous considérons uniquement l'approche passive en tant que méthode de surveillance. De plus, comparé à notre approche, ces travaux ne font références qu'aux couches liaison et réseau de la pile IoT.

2.3.3 Travaux sur l'analyse de trafic

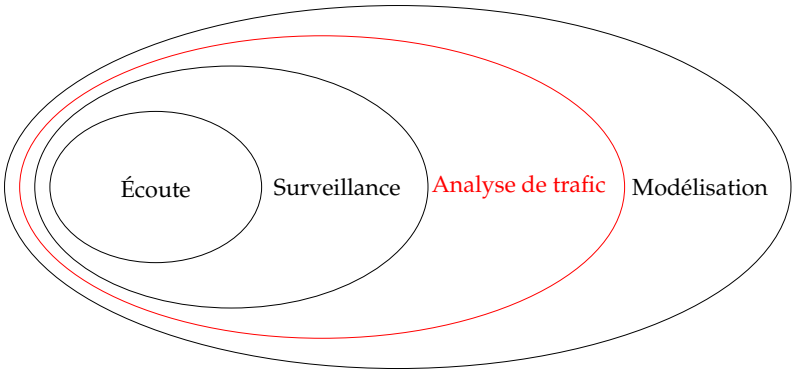

FIGURE 2.11 – Représentation des travaux en quatre groupes. Focus sur le groupe analyse de trafic.

Le troisième groupe d'études se concentre sur l'analyse de trafic. Certains travaux sont spécialisés dans l'identification ou la reconnaissance de dispositifs afin d'identifier les menaces d'un réseau ou des comportements anormaux dans le trafic. Les auteurs se basent sur les techniques d'empreintes numériques pour réaliser l'identification des dispositifs afin de distinguer les objets de confiances et ceux qui sont malveillants.

Miettinen *et al.* dans [58] décrivent une classification par type d'objets reposant sur une analyse de la séquence d'ajout d'un nouvel objet dans les réseaux. Dans un premier temps, les objets connectés sont identifiés par leur adresse MAC. Puis, les auteurs analysent les flux de données émis par un objet pour identifier le type de l'objet ainsi que ses potentielles vulnérabilités. Si un objet est détecté comme vulnérable, il est isolé des autres à l'aide de règles de filtrage dans le but d'éviter la potentielle contamination d'autres objets.

Cependant, cette approche soulève plusieurs problèmes si tous les objets ont déjà été appairés ou s'ils changent de comportement après leur ajout dans le réseau. De plus, l'isolement d'un objet ne pouvant pas être mis à jour le rend totalement inutilisable dans le réseau.

Dans [13, 91], les auteurs proposent de classifier les dispositifs d'après leur comportement suite à une observation des communications. Ils analysent le trafic des objets, collecté pendant une période de temps finie et observent les informations spécifiques pour déduire des comportement différents. Les auteurs de [91] s'intéressent uniquement aux flux bidirectionnels entre deux objets. Puis, ils se concentrent sur le nombre et la taille des paquets échangés (émis et reçus) pour chaque communication. En fonction du nombre de paquets échangés sur une période de temps, définie par une session TCP jusqu'à expiration, les auteurs sont alors en mesure de distinguer les objets grâce à l'algorithme t-SNE. Dans [13], les auteurs utilisent différents paramètres pour caractériser le comportement associé à un flux de trafic. Dans un premier temps, ils déterminent un modèle de comportement statique, qui s'appuie sur l'utilisation des protocoles et un modèle de comportement dynamique, qui repose sur des schémas de communications avec des commandes et des requêtes.

Ces approches reposent sur la connaissance des objets observés, ainsi que sur la définition et l'exemple d'un trafic régulier, afin de détecter si le trafic observé est légitime ou non.

D'autres études se focalisent sur la classification de trafic pour identifier des schémas d'applications au sein du réseau. Sivanathan *et al.*, dans [98] proposent une méthodologie et des solutions pour instrumentaliser et classifier le trafic généré par des dispositifs IoT déployés dans un campus universitaire. Leur analyse est multiple et se centre sur cinq critères.

Les trois premiers critères étudiés sont l'activité générée par les objets, les protocoles utilisés et l'identification de caractéristiques propres aux objets IoT. Ils sont définis pour distinguer les objets IoT des objets non-IoT dans le réseau. Ils permettent également de catégoriser les objets IoT selon des critères communs qui seront utilisés pour établir une classification.

Le quatrième critère analysé est le groupement par attributs. Leur approche est de créer un groupe d'attributs correspondant à un trafic en particulier et non par rapport aux fonctionnalités d'un objet. Ainsi, pour un dispositif en fonction du trafic émis, il est possible qu'il fasse parti de plusieurs groupes différents.

Le dernier critère étudié, la classification des objets IoT, propose une classification des objets en fonction du groupe d'attributs qu'ils possèdent. Les auteurs s'appuient sur des algorithmes de *machine learning* pour produire cette classification.

Cependant, ces travaux reposent sur l'installation d'un proxy, en amont, dans le réseau pour intercepter le trafic entrant et sortant. Toutes les études présentées dans ce paragraphe se positionnent au niveau de la couche transport de notre approche. Cependant, peu de protocoles IoT sont supportés par ces outils. Les dispositifs utilisés dans les expérimentations utilisent principalement la technologie IP ou alors utilisent des protocoles non spécifiques au contexte IoT comme HTTP, DNS, etc.

Notre approche est agnostique par rapport aux protocoles et nous traitons toutes les couches de la pile IoT à l'exception de la couche physique.

2.3.4 Travaux sur la modélisation

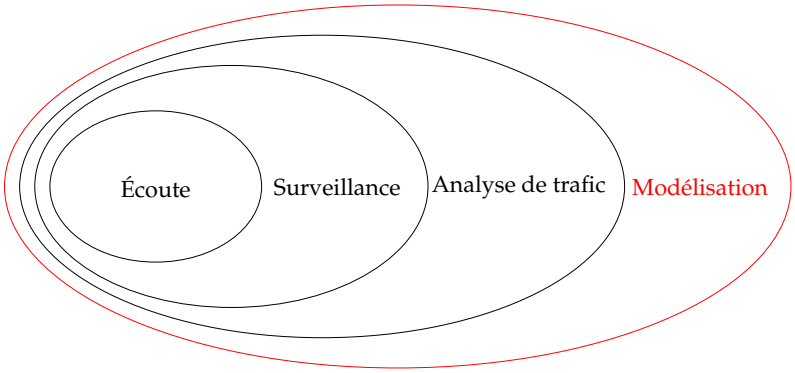

FIGURE 2.12 – Représentation des travaux en quatre groupes. Focus sur le groupe modélisation.

Le dernier groupe inclut les solutions tout-en-un, réunissant les méthodologies présentées dans les groupes précédents, comme l'illustre la figure 2.12. En effet, pour réaliser une modélisation de réseau, il faut dans un premier temps intercepter et capturer le trafic du réseau. Les traces obtenues passent ensuite par différents processus d'analyse afin d'extraire des informations qui seront utiles, voire nécessaires, pour réaliser la modélisation du réseau. La modélisation peut se manifester de différentes façon. Certains travaux proposent des modélisation mettant en avant les anomalies d'un réseau, d'autres modélisent le réseau en fonction des types d'objets, etc.

Siby *et al.* décrivent, dans [94], une approche qui couvre toutes les étapes allant de la capture de trafic jusqu'à son analyse. De plus, les objets déployés pour évaluer leurs travaux utilisent des protocoles spécifiques au contexte IoT comme BLE ou ZigBee.

Leur *framework* utilise une méthode de surveillance passive pour intercepter, capturer et visualiser le trafic. Il est également capable d'analyser les captures afin d'identifier et classifier les objets dans le but de déterminer de potentielles menaces au sein du réseau. Cette analyse repose sur une inspection des paquets et l'extraction de plusieurs informations, concentrées principalement dans l'en-tête et le pied du paquet. Cette phase est spécifique pour chaque protocole que le système est capable d'étudier, car la structure des paquets change en fonction du protocole analysé (entre BLE et Zigbee par exemple). Les données extraites sont ensuite stockées dans une base de données, utilisée notamment pour permettre la visualisation du réseau.

IoTScanner propose, en plus de sa visualisation du réseau, une classification des objets en fonction du trafic généré. Plusieurs méthodes d'analyse et de classification sont explorées dans ces travaux. Une méthode consiste à faire corréler le nombre d'octets émis et envoyés par objet, dans une fenêtre de temps spécifique, ainsi que le type de paquet (contrôle, management, données) afin de déduire le type et le rôle de l'objet. Une seconde analyse est centrée sur un type de paquets en particulier, la donnée. Pour cette analyse, les auteurs comparent le trafic émis avec celui reçu pour chaque objet. En fonction de la différence entre les deux valeurs, les auteurs en déduisent le type de l'objet.

Cependant, leur principale expérimentation implique des objets qui communiquent en utilisant la technologie WIFI. De plus, leur classification repose uniquement sur la détection du type de l'objet, avec comme prérequis, la définition au préalable de chaque type d'objets pouvant être détecté.

Nous utilisons, dans notre approche, une classification sur le rôle de l'objet dans le réseau. Nous décrivons, également, une analyse holistique des systèmes IoT, utilisant des protocoles IoT simultanément, avec des interactions multi-protocolaires.

IoTHound [5] est une autre solution tout-en-un, proposé par Anantharaman *et al.*, dont l'approche consiste à intercepter, capturer et analyser le trafic du réseau d'objets connectés afin d'identifier les différents types d'objets de ce réseau. Il repose également sur une approche de surveillance passive, qui s'appuie sur plusieurs dispositifs d'écoute pour gérer plusieurs protocoles.

Contrairement à IoTScanner, qui gère plusieurs protocoles non simultanément, le système IoTHound est capable d'intercepter les protocoles WiFI, ZigBee et BLE simultanément afin de les analyser par la suite. Pour chaque protocole, dès que le trafic est intercepté, le système IoTHound extrait, de chaque paquet, les informations nécessaires pour la suite de l'analyse. Le processus d'extraction ne s'effectue qu'après le groupement de paquets, par créneau de temps et par objet. Rassembler plusieurs paquets par créneau de temps permet de distinguer les paquets qui seraient moins intéressants pour l'analyse, comme les paquets d'acquittement (ACK) par exemple.

Après l'extraction des informations, le système IoTHound calcule un vecteur de caractéristiques pour chaque groupe (donc de chaque objet identifié par son adresse source), utilisé pour identifier le comportement ou pattern d'un objet en fonction du trafic qu'il a généré.

Les auteurs proposent également une solution pour différencier deux objets, qui auraient le même comportement, en utilisant une classification basée sur le RSSI.

Cette méthode permet donc de différencier plusieurs objets, au comportement identique, en les situant spatialement. La mesure du RSSI en provenance de chaque objet diffère en fonction de leur proximité et de leur position par rapport à l'antenne qui intercepte les ondes radios. Ainsi, avec cette méthode, les auteurs peuvent distinguer des objets, au pattern de trafic semblable, d'après leur position au sein de l'environnement.

Bien que ces travaux exploitent l'analyse de réseaux et se basent sur des réseaux multi-protocoles, ils restent centrés autour de l'objet. En effet, toute l'analyse du réseau est dédiée à la classification et à l'identification de l'objet dans le réseau. De plus, bien que plusieurs protocoles soient utilisés dans leur travaux, aucune interaction multi-protocoles n'est mise en avant. Dans cette thèse, nous nous sommes centrés sur la vision du réseau dans sa globalité plutôt que sur les objets qui le composent. De plus, notre vision, qui se retrouve dans notre expérimentation, est de modéliser des réseaux hétérogènes, dont certaines interactions utilisent plusieurs protocoles pour aboutir.

2.4 Résumé

Dans ce chapitre, nous avons présenté un état de l'art des protocoles IoT. Nous avons introduit une pile IoT générique composée de cinq couches et quatre critères de comparaison, nous permettant de décrire les protocoles IoT étudiés selon un même modèle.

Nous avons analysé les systèmes IoT en fonction de leur architecture ainsi que les problèmes de sécurité associés aux protocoles IoT. Nous avons dénoté trois différentes façons de concevoir l'écosystème IoT : vision centrée sur un protocole, vision centrée sur une couche de la pile protocolaire et vision centrée sur le système global. Nous avons étudié les protocoles IoT avec une vision à l'échelle du protocole sans se limiter à une couche particulière ou à une pile protocolaire.

Nous avons également présenté une *killchain* en trois parties pour décrire et catégoriser les attaques contre les protocoles IoT : attaques contre les paquets, attaques contre le protocole et attaques contre le système.

Cet état de l'art des protocoles et leur architecture, complété par deux autres protocoles IoT LAN (Z-wave [31], WirelessHart [81]) et deux protocoles IoT WAN (LoRaWAN [101] et SigFox [49]), ainsi qu'avec la catégorisation et la présentation des attaques selon une *killchain*, a été publié dans le journal [104].

Enfin, nous avons présenté un état de l'art des méthodes de modélisation des réseaux IoT existants. Nous avons classé l'ensemble des travaux étudiés en quatre catégories :

— les travaux centrés sur l'interception et la capture de trafic ;

— les travaux centrés sur la surveillance des communications ;

— les travaux centrés sur l'analyse de communications ;

— les travaux centrés sur la conception et la visualisation des réseaux

Chapitre 3

Modélisation : du paquet jusqu'au graphe applicatif

Dans ce chapitre, nous présentons la méthodologie de notre modélisation de réseau IoT. Plus précisément, nous nous intéressons aux différents modèles de graphes qui composent la modélisation finale du réseau. Dans un premier temps, pour faciliter et accompagner la compréhension du processus de modélisation et des couches qui le composent, nous définissons un exemple typique de réseau hétérogène IoT, dans lequel plusieurs applications sont déployées. Chaque couche présente dans le modèle est illustrée à l'aide d'un scénario expérimental. Dans un second temps, nous présentons le paquet générique, une représentation abstraite d'un paquet qui permet d'appliquer notre méthodologie de modélisation sur un format stable, sans devoir l'adapter spécifiquement à chaque protocole. Finalement, nous présentons les différents graphes qui composent le processus de modélisation de réseaux hétérogènes IoT.

3.1 Scénario expérimental

A travers cette section, nous présentons un exemple de scénario complexe et typique des configurations que nous attendons et voulons modéliser. Ce scénario est utilisé comme exemple filant tout au long de ce chapitre. Afin d'évaluer et jauger la généricité du modèle, il est composé de douze objets qui peuvent communiquer en utilisant trois protocoles IoT différents (certains objets agissent comme des passerelles entre les différents protocoles IoT). De plus, pour montrer les différentes interactions protocolaires, quatre applications de type actionneur-capteur et quatre applications de type agrégateur de données sont instanciées dans ce scénario, illustré en figure 3.1.

Les trois protocoles utilisés sont ZigBee, BLE et OS4I. ZigBee et BLE sont des protocoles IoT connus et déjà largement déployés. OS4I, pour "Open Stack For IoT", est le nom que nous avons choisi pour nommer la pile IoT open-source qui utilise les protocoles suivants : IEEE 802.15.4, 6LoWPAN, UDP/TCP et CoAP/MQTT.

Applications actionneur-capteurs. Quatre applications sont des applications de type actionneur-capteur, avec comme étiquette AC.

AC1 est une application qui utilise uniquement le protocole ZigBee et qui met en relation les objets étiquetés $d1$, $d3$ et $d4$. Le contrôleur $d4$ requête le capteur $d1$ toutes les 60 secondes pour obtenir les valeurs remontées par cet objet, puis met à jour l'actionneur $d3$ en fonction des données obtenues.

AC2 est une application dont les communications se font par le protocole OS4I et les objets impliqués sont les dispositifs étiquetés $d8$, $d10$ et $d12$. L'objet $d12$ agit en

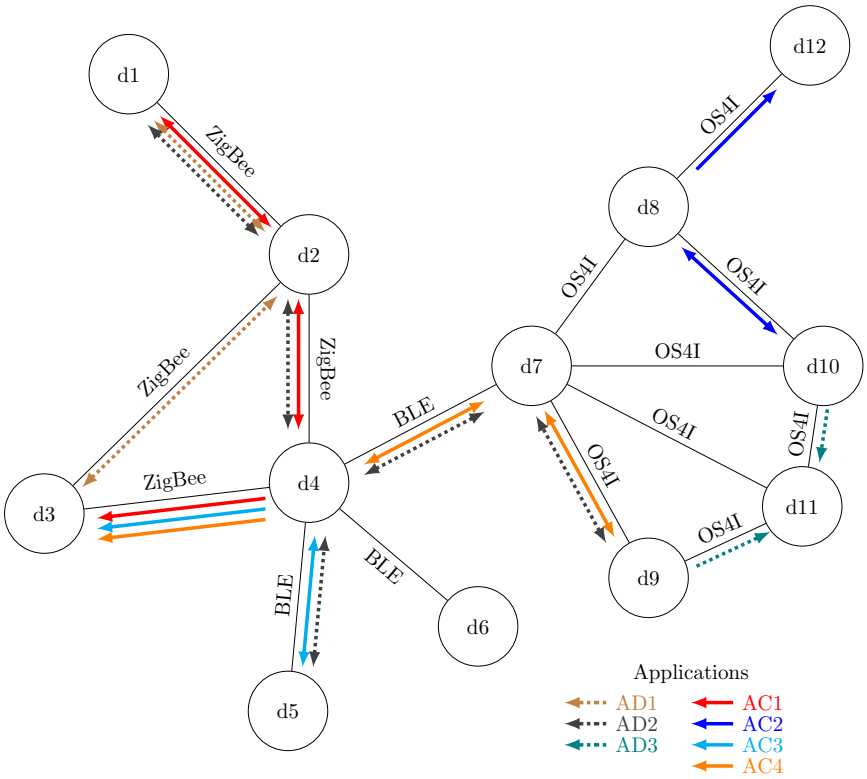

FIGURE 3.1 – Exemple de déploiement de réseau multi-protocolaires.

tant qu'actionneur et l'objet $d10$ en tant que capteur. Le contrôleur $d8$ requête donc le capteur toutes les 40 secondes et envoie une commande à l'actionneur ($d12$) en fonction de la valeur retournée par l'objet $d10$.

AC3 est une application, qui utilise les objets $d5$, $d3$ et $d4$, et dans laquelle les protocoles IoT, ZigBee et BLE, sont utilisés simultanément. Le contrôleur $d4$ requête le capteur $d5$ toutes les 60 secondes en utilisant le protocole IoT BLE et met à jour l'actionneur $d4$ en utilisant le protocole de communication ZigBee.

AC4 est une application qui mélange les trois protocoles de communications IoT et qui nécessite l'utilisation des objets $d3$, $d4$, $d7$ et $d9$. Le contrôleur $d4$ envoie une requête au contrôleur $d7$ afin de récupérer les données du capteur $d9$. Lors de la réception de la requête du contrôleur $d4$, le contrôleur $d7$ transmet la requête au capteur $d9$. Dès réception de la donnée, le contrôleur $d7$ renvoie directement cette dernière au contrôleur $d4$. Après analyse de la valeur obtenue par le contrôleur $d4$, ce dernier envoie, si nécessaire, une commande à l'actionneur $d3$. Les communications effectuées entre le contrôleur $d7$ et le capteur $d9$ sont portées par le protocole OS4I. Celles transmises entre les deux contrôleurs sont effectuées en utilisant le protocole BLE. Finalement, le contrôleur $d4$ communique avec l'actionneur $d3$ en utilisant le protocole ZigBee.

Applications agrégateur de données. Les trois autres applications sont des applications de type agrégateur de données (AD).

Dans AD1, $d2$ surveille les statuts du capteur $d1$, c'est à dire la température relevée par ce capture, et l'actionneur $d3$, c'est à dire l'état dans lequel se trouve l'actionneur. Les trois objets communiquent exclusivement via le protocole ZigBee.

AD2 est une application dans laquelle $d4$ agrège et stocke les données de température en provenance du capteur $d1$, par le biais de communications ZigBee, et du capteur $d5$, en utilisant, cette fois, des communications utilisant le protocole OS4I.

La dernière application, AD3, assure le monitorage des capteurs $d10$ et $d9$ par l'objet $d11$ en utilisant un seul protocole, OS4I.

D'après plusieurs scénarios présentés dans des travaux comme [94, 13], nous estimons que ce scénario est suffisamment pertinent pour évaluer l'efficacité de notre modélisation.

3.2 Paquet générique

Les protocoles IoT sont nombreux et très divers dans leur fonctionnement et leur représentation. Or, ils sont pourtant, de plus en plus, amenés à cohabiter au sein d'un même réseau. Cette hétérogénéité des protocoles IoT les rend donc difficiles à analyser conjointement.

Il est donc nécessaire de définir un modèle homogène sur lequel tous les protocoles peuvent s'adapter pour faciliter l'analyse de ces protocoles. Dans ce manuscrit, nous proposons un format unifié, appelé le paquet générique, qui permet de transposer le format des paquets spécifiques à chaque protocole vers un format unifié que nous avons défini. Il s'agit d'une interface qui permet de passer d'un environnement hétérogène avec plusieurs protocoles différents, avec leurs propres caractéristiques, à un format homogène sur lequel s'appuie la modélisation.

De cette façon, l'analyse et la méthodologie de modélisation ne sont pas spécifiques à un protocole, et l'ajout d'un nouveau protocole n'entraine aucun changement dans les algorithmes de modélisation. Ce modèle, abstrait et homogène, s'appuie sur la pile IoT générique, définie en section 2.1.2. Ainsi, le paquet générique est composé de cinq couches distinctes, qui peuvent être composées d'un ou plusieurs champs, comme illustré sur la figure 3.2.

Physique		Liaison		Réseau		Trans.	App.
Timestamp	Protocole	Adresse Source	Adresse Dest	Adresse Source	Adresse Dest	Type application	Donnée

FIGURE 3.2 – Structure du paquet générique.

Couche physique. La couche physique stocke les informations relatives au paquet, comprenant le *timestamp* et le protocole d'origine. Le *timestamp* permet d'identifier temporellement quand le paquet a été émis. Cette valeur est essentielle pour la suite du processus de modélisation. Le second champ de la couche physique renseigne le protocole d'origine du paquet. Cette information, bien qu'intéressante pour d'autres raisons, n'est pas utilisée par ce processus de modélisation.

Couche liaison. La couche liaison est composée de deux champs, l'adresse source et l'adresse destination. De même que la couche liaison, que de nombreux protocoles utilisent, celle-ci identifie les adresses physiques des objets (MAC). Cette adresse est spécifique et propre à chaque objet et est définie lors sa création. Ainsi, le champ adresse source stocke l'adresse MAC de l'objet qui envoie un message, alors que le champ adresse destination stocke l'adresse MAC de l'objet qui reçoit le message.

Couche réseau. Tout comme la couche liaison, la couche réseau se compose également des deux champs, adresse source et adresse destination. Elle reprend en grande partie les mêmes spécificités que celles définies dans des instances spécifiques de protocole. L'adresse réseau est une adresse définie par le routeur ou hub du réseau. Elle est propre au réseau et non à l'objet. Un même objet peut, par exemple, avoir deux adresses de réseau différentes, s'il est ajouté puis supprimé avant d'être ajouté de nouveau dans le réseau. L'adresse source, dans la couche réseau, fait référence à l'adresse réseau de l'objet ayant déclenchée la communication. L'adresse destination fait référence à l'adresse réseau de l'objet à qui le message est destiné.

Couche transport. La couche transport du paquet générique ne possède qu'un seul champ, le type d'application. Il s'agit d'une information extraite grâce à une analyse du paquet dans le format spécifique du protocole. Cette analyse permet de définir une base de conversion qui met en lien les éléments spécifiques du protocole avec le type d'application utilisé par le paquet générique. Actuellement, nous avons défini cinq types d'application :

— Contrôle : Il s'agit de paquets destinés au contrôle et au management du réseau ;

— ACK : Il s'agit de paquets d'acquittement (réseau ou MAC) pour vérifier que les paquets ont bien été émis ;

— Commande : Il s'agit de paquets dont le contenu fait référence à une action de type commande ;

— Donnée : Il s'agit de paquets qui transportent de la donnée utile, comme par exemple une valeur de température, de luminosité, taux de batterie, etc ;

— Inconnu : Il s'agit de paquets dont le contenu n'est pas accessible.

Couche application. La couche application est également constituée d'un unique champ, le champ donnée. Ce dernier stocke la valeur contenue dans le champ application du paquet au format spécifique. Cette valeur n'est remplie que lorsque le type d'application correspond au type "donnée".

3.3 Modèles de graphe

Cette section présente le modèle utilisé pour représenter les différentes couches d'un réseau IoT. Chaque couche correspond à un graphe spécifique et met en évidence un point précis dans la modélisation. Ainsi, pour tous les graphes, N est défini comme l'ensemble des nœuds du réseau et E représente l'ensemble des arcs correspondant au graphe. Le tout est défini comme suit :

— **Graphe liaison** $G_{DL}(N, E_{DL})$ représente les informations obtenues au niveau de la couche liaison et met en évidence les communications à un seul saut ;

— **Graphe réseau** $G_{NWK}(N, E_{NWK})$ représente les informations obtenues au niveau de la couche réseau et met en évidence les communications finales, appelées également bout à bout ;

— **Graphe transport** $G_{TRANS}(N, E_{TRANS})$ représente les informations obtenues au niveau de la couche transport et met en évidence le type d'application déployé dans le réseau ainsi que le rôle de chaque objet ;

— **Graphe application**$G_{APP}(N, E_{APP})$ représente les informations obtenues au niveau de la couche application et met en évidence les flux logiques entre les nœuds du graphe.

Certains nœuds du réseau, comme les routeurs par exemple, peuvent disposer d'une ou plusieurs interfaces réseau en fonction du nombre de protocoles de communication utilisés. Dans ce cas, ces objets possèdent plusieurs adresses MAC et réseaux, une pour chaque interface réseau. Un nœud est ainsi défini par les caractéristiques suivantes :

— un identifiant unique (ID) ;

— un ensemble d'adresses MAC (DL adresse) ;

— un ensemble d'adresses réseau (NWK adresse) ;

— un ensemble de rôles, vide initialement, puis complété au cours de l'analyse.

A travers cette section, chaque graphe est illustré par le scénario décrit dans en section 3.1. Les algorithmes permettant la création de ces graphes sont présentés dans le chapitre 4.

3.3.1 Graphe liaison

Ce graphe représente toutes les connexions entre les nœuds du réseau au niveau de la couche liaison. Il correspond également aux communications à saut unique. Une communication à saut unique est une communication directe ne passant par aucun autre intermédiaire (nœud routeur ou nœud classique). Il s'agit donc d'une communication avec un nœud voisin à portée de radio.

Chaque communication observée est représentée par un arc orienté entre deux nœuds : partant du nœud qui émet la communication jusqu'au nœud destinataire. Chaque arc est ensuite étiqueté avec le *timestamp* de la communication, les adresses MAC et le reste des informations utiles pour les couches supérieures.

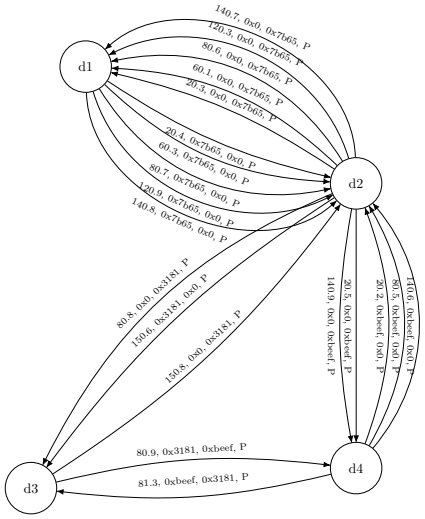

FIGURE 3.3 – Sous-partie du graphe liaison. Focus sur 4 nœuds (d1, d2, d3 et d4).

La figure 3.3 illustre une sous-partie du graphe liaison du réseau décrit dans le scénario expérimental. Chaque arc qui compose le graphe, correspond à une communication observée dans le réseau, et est étiqueté avec les informations nécessaires pour construire le graphe.

Par exemple, $d4$ initie trois communications à des *timestamps* différents (20.2, 80.5 et 140.6) à destinations du dispositif $d2$ et une communication à destination du dispositif $d3$ au *timestamp 81.3*. Les trois premières communications sont identifiées par trois arcs orientés de $d4$ vers $d2$. Chaque arc est étiqueté avec le timestamp associé, l'adresse MAC source de $d4$ (0xbeef), l'adresse MAC source de $d2$ (0x0) et les informations des couches supérieures (P). Cette action est itérée pour chaque nœud et communication observés.

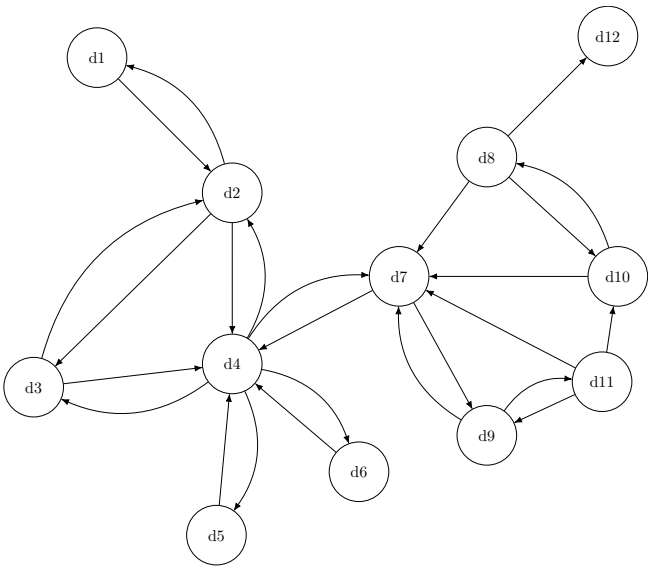

FIGURE 3.4 – Graphe liaison du scénario expérimental.

La figure 3.4 représente le graphe liaison du scénario décrit en section 3.1. Pour améliorer la lisibilité de la figure, le nombre d'arc entre deux nœuds est limité à un. En réalité, il y a autant d'arcs orientés que de communications observées.

3.3.2 Graphe réseau

Ce graphe représente toutes les communications entre les nœuds du point de vue de la couche réseau, aussi appelées communications finales (ou *end-to-end*). Par conséquent, ce graphe met en évidence les nœuds qui se comportent uniquement comme des routeurs, ou dispositifs ne servant qu'à retransmettre des messages pour une communication entre deux objets. Nous utilisons la même spécification que celle présentée dans la section 3.3.1 pour labelliser chaque arc du graphe réseau : le *timestamp*, adresse réseau source et adresse réseau destination.

La figure 3.5 illustre une sous-partie du graphe réseau du scénario expérimental. De nombreux arcs ont été ajoutés par rapport au graphe liaison tels que les arcs entre les nœuds $d1$ et $d4$. En comparant les deux graphes (liaison et réseau), nous pouvons en déduire, par exemple, que l'objet $d2$ agit comme un routeur dans le réseau. En effet, le graphe liaison nous donne l'information que $d1$ communique avec $d2$ et que $d2$ communique avec $d4$. Associé à la connaissance apportée par le graphe réseau, nous pouvons valider que $d2$ est un routeur qui retransmet des messages entre $d1$ et $d4$. Cependant, certains arcs peuvent également être identiques à ceux présents dans le graphe liaison. Par exemple, les arcs entre $d3$ et $d4$ sont identiques entre le graphe réseau et le graphe liaison. Ceci s'explique car les deux objets sont suffisamment proches physiquement pour ne pas nécessiter de saut intermédiaire pour communiquer ensemble.

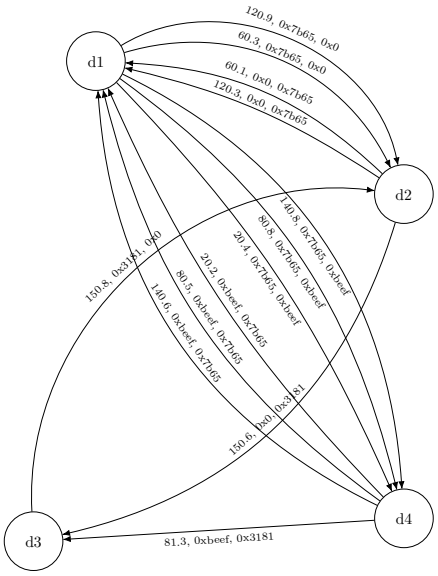

FIGURE 3.5 – Sous-partie du graphe réseau. Focus sur 4 nœuds (d1, d2, d3 et d4).

La figure 3.6 représente le graphe réseau du scénario présenté dans la section 3.1 avec tous les nœuds du réseau, également simplifié pour améliorer sa visibilité. Nous observons que des arcs sont apparus et d'autres ont été effacés par rapport au graphe liaison. Par exemple, l'arc qui reliait les objets *d*7 et *d*8 dans le graphe liaison, n'existe plus dans le graphe réseau. Au contraire, un nouvel arc est observé entre les objets *d*9 et *d*8. Nous pouvons donc en déduire que l'objet *d*7 sert de routeur entre les objets *d*9 et *d*8.

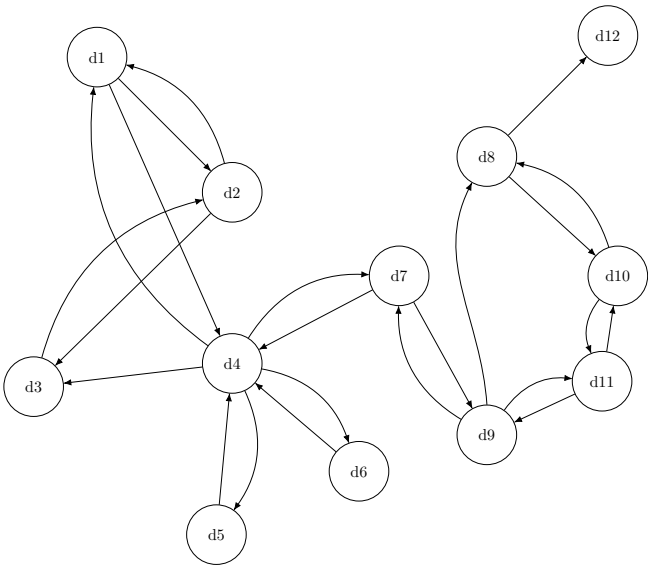

FIGURE 3.6 – Graphe réseau du scénario expérimental.

3.3.3 Graphe transport

Ce graphe reflète l'orientation du flux de données et appose un rôle pour chaque nœud du graphe. Ce graphe est construit en suivant deux phases :

1. chaque arc entre deux nœuds du graphe est remplacé par un unique arc orienté ;
2. chaque nœud du graphe se voit attribuer un rôle correspondant à son activité dans le réseau.

Les *timestamps*, de toutes les communications entre deux objets, sont regroupés dans un tableau de *timestamps* ; aux côtés des adresses réseaux source et destination. Pour chaque objet du réseau, nous définissons quatre rôles :

— **Source (Src)** : Ce rôle est attribué à tout objet qui **envoie de la donnée** ;

— **Sink (Sk)** : Ce rôle est attribué à tout objet qui **reçoit de la donnée** ;

— **Source-Sink (Src-Sk)** : Ce rôle est attribué à tout objet impliqué dans des communications où il **reçoit de la donnée** et d'autres où il **envoie de la donnée** ;

— **Controller (Ctrl)** : Ce rôle est attribué à tout objet qui agrège et analyse des données en provenance d'un ou plusieurs objets possédant le rôle *source* ; afin de contrôler ou envoyer des informations à un ou plusieurs objets possédant le rôle *sink*.

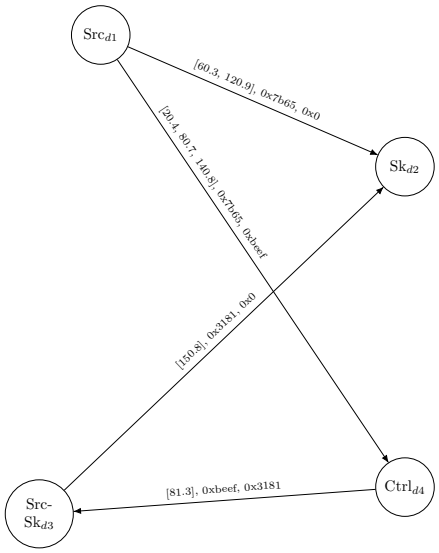

FIGURE 3.7 – Sous-partie du graphe transport avec les rôles et les arcs orientés. Focus sur 4 nœuds (d1, d2, d3 et d4).

La figure 3.7 illustre une sous-partie du graphe transport du réseau présenté dans le scénario expérimental. Tous les arcs bidirectionnels du graphe réseau ont été remplacés par un unique arc orienté entre chaque couple d'objets. Les rôles sont ensuite assignés à chaque objet en fonction de leur position dans l'orientation du flux de données. Par exemple, dans la figure 3.6, $d3$ et $d2$ utilisent un schéma de communication bidirectionnel. Cependant, après analyse, le flux de données entre ces deux dispositifs est orienté de $d3$ vers $d2$. Comme le flux de donnée part de $d3$ pour finir à $d2$, $d3$ est défini comme la *source* du flux de données et $d2$ comme le *sink*. La méthodologie reproduit cette action sur toutes les communications bidirectionnelles ou unilatérales observées dans le réseau.

De plus, si un objet est impliqué dans plusieurs communications avec des rôles différents (*source* et *sink*), il sera alors étiqueté avec les deux rôles. Par exemple, dans le graphe réseau, illustré sur la figure 3.6, l'objet $d3$ est impliqué dans des communications avec l'objet $d2$ et l'objet $d4$. Il est considéré comme un objet ayant le rôle *source* dans la communication avec le dispositif $d2$ et le rôle *sink* lui est attribué, cette

fois, dans sa communication avec l'objet $d4$. De ce fait, le dispositif $d3$ est un objet jouant le rôle *source-sink*.

Un *controller*, tel que le dispositif $d4$, est un nœud du graphe qui est impliqué dans au moins deux communications, une où il agit en tant que *source* et l'autre en tant que *sink*. Cependant, contrairement à un objet avec le rôle *source-sink*, ses deux communications sont corrélées. En effet, l'objet $d4$ agrège et analyse les données en provenance de l'objet avec le rôle *source* et envoient les messages correspondants à l'objet identifié par le rôle *sink*. Dans la figure 3.8, $d4$ est identifié comme un *controller* où $d1$ est un objet identifié comme la *source* du flux de données et $d3$ est l'objet identifié comme le *sink* dans la communication.

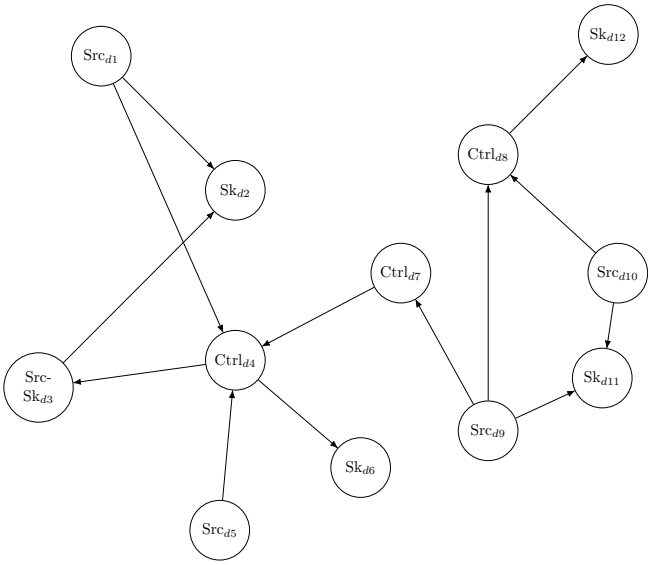

FIGURE 3.8 – Graphe transport avec les rôles et les arcs orientés.

La figure 3.8 présente le graphe transport du scénario présenté en section 3.1. Pour simplifier la lecture du graphe, nous avons enlevé les étiquettes de chaque arc, tout en laissant identifié les rôles de chaque objet. Dans ce scénario, trois objets sont considérés comme des *controllers* dans le réseau, les objets $d4$, $d7$ et $d8$. Ce graphe met ainsi en évidence l'importance de certains nœuds dans le fonctionnement applicatif du réseau.

3.3.4 Graphe application

Ce dernier graphe met en évidence les applications détectées dans le réseau. Les applications sont définies par des patterns décrits dans la section 4.4. Chaque pattern défini utilise une labellisation des arcs qui lui est propre. Ainsi, il est possible de séparer chaque application par utilisation d'un filtre, soit de laisser toutes les applications détectées dans le même graphe. La figure 3.9 illustre le graphe des applications déployées dans le scénario expérimental, présenté en section 3.1. Il représente les applications de type actionneur-capteur (AC) et agrégateur de données (AD) du réseau.

Applications AC. Tous les arcs des nœuds impliqués dans les applications AC du graphe transport ont été remplacés par de nouveaux arcs avec l'étiquette *INTERACT*. Comme attendu, ce graphe met en évidence les interactions transprotocolaires entre les objets du réseau. Par exemple, $d9$, qui est un objet qui communique en

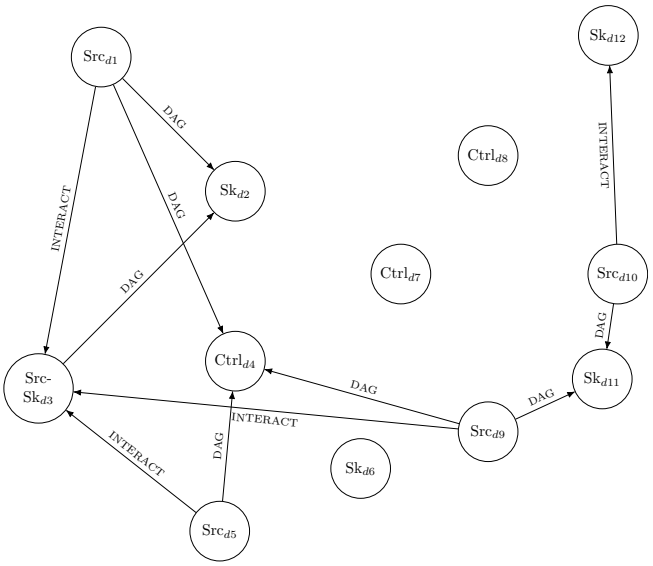

FIGURE 3.9 – Graphe application du scénario expérimental.

utilisant le protocole OS4I, interagit indirectement avec l'objet $d3$, qui, à son tour, communique uniquement via le protocole ZigBee.

Applications AD. De façon identique aux arcs liés aux applications AC, les arcs utilisés pour décrire les applications AD sont remplacés par de nouveaux arcs dont l'étiquette est *DAG*. De plus, nous constatons que cette application peut également utiliser plusieurs protocoles pour relier des objets entre eux. Par exemple, le dispositif $d9$, communiquant uniquement par le protocole OS4I, envoie ses données au dispositif $d4$, communicant soit en Zigbee soit en BLE, par l'intermédiaire du dispositif $d8$.

Par ailleurs, nous observons que divers dispositifs sont impliqués dans plusieurs applications. Il est alors possible de filtrer le label souhaité afin de visualiser plus précisément l'application à observer. Cela facilite l'analyse de l'application sans être pollué par les informations des autres applications. Cependant, ce filtre est valable uniquement pour un type d'application (*INTERACT* ou *DAG*).

3.4 Résumé

Ce chapitre présente notre méthodologie de modélisation de réseaux IoT hétérogènes. Cette modélisation se décline en quatre graphes distincts, chacun apportant son niveau d'informations, allant du graphe de liaison au graphe d'application.

Nous introduisons également le paquet générique, une interface unique sur laquelle repose la modélisation. Ce format générique s'appuie sur la définition d'une pile IoT générique, présentée en section 2.1.2, qui permet de convertir les paquets observés de protocoles hétérogènes en paquet au format homogène.

Pour comprendre le processus de modélisation, nous introduisons également un scénario expérimental servant d'exemple filant lors de la présentation des différents graphes de la modélisation.

Dans les chapitres suivants, nous présentons les algorithmes utilisés pour implémenter la modélisation. Nous proposons de diviser en deux cas la construction des graphes : le cas idéal et le cas dégradé.

Dans le cas idéal, nous faisons l'hypothèse que l'observation est parfaite et que la base de conversion, permettant de convertir les paquets au format spécifique à un protocole en paquets au format générique, est complète. Cette approche est présentée dans le chapitre 4.

Dans le cas dégradé, nous faisons plusieurs hypothèses décrivant la dégradation de l'observation effectuée, et proposons une modification des algorithmes afin de produire la modélisation. Notamment, nous présentons une dégradation de l'observation liée à une perte d'information ainsi qu'une dégradation liée à une observation partielle. Cette approche est décrite dans le chapitre 5.

Chapitre 4

Construction du modèle par détection de patterns : cas idéal

Dans ce chapitre, nous présentons le processus de modélisation reposant sur une approche itérative, basée sur des graphes. Cette approche est présentée dans la figure 4.1. Elle utilise quatre fonctions différentes, où chacune prend en paramètre la sortie de la fonction précédente et le pattern désiré.

La modélisation commence avec la fonction F_{DL} qui transforme les communications interceptées ($PCAPs$), en graphe liaison, grâce au pattern P_{DL}. La seconde fonction F_{NWK} prend en paramètre le graphe G_{DL} et le pattern P_{NWK} et produit le graphe G_{NWK}. Puis, le graphe G_{NWK} et le pattern P_{TRANS} alimentent la fonction F_{TRANS} pour modéliser le graphe G_{TRANS}. Finalement, la dernière fonction, F_{APP}, prend en paramètre le graphe G_{TRANS} et le pattern P_{APP} afin de générer le graphe G_{APP}.

Seule la fonction F_{DL} requiert l'intervention humaine pour faire correspondre les multiples interfaces des objets, qui utilisent plusieurs protocoles pour communiquer. La génération de graphe devient, par la suite, entièrement automatique. Tous les patterns et fonctions proposés sont présentés et détaillés dans les sections suivantes.

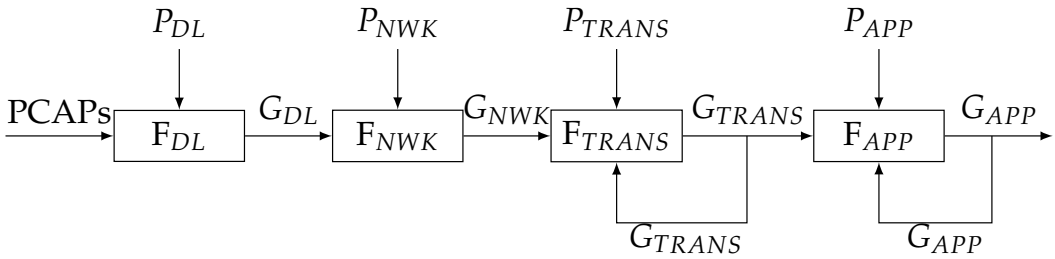

FIGURE 4.1 – Approche itérative basée sur les graphes.

4.1 Fonction de la couche liaison F_{DL}

La première étape, pour réaliser la modélisation du réseau, est de générer le premier graphe, G_{DL}, à partir des captures des communications interceptées dans le système IoT. La fonction F_{DL} prend en paramètre les captures spécifiques au protocole utilisé dans chaque réseau. Une étape intermédiaire est nécessaire pour unifier le format de chaque capture. Elle consiste à convertir chaque paquet d'une capture, spécifique à un protocole, vers notre format de paquet générique, présenté dans la section 3.2. Le passage par un format intermédiaire a pour objectif de faciliter l'ajout d'un nouveau protocole, sans devoir modifier le processus de modélisation.

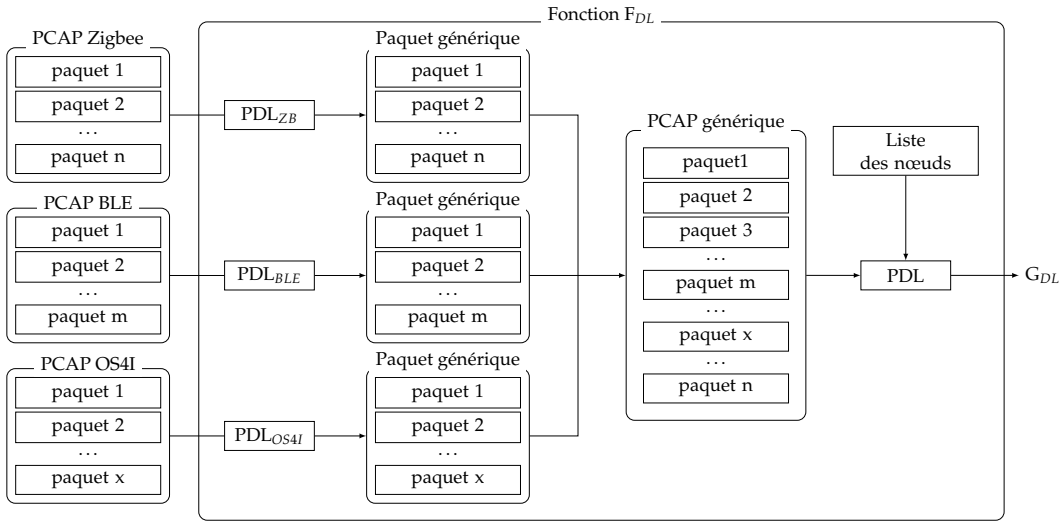

FIGURE 4.2 – Processus effectué par la fonction F_{DL}.

La figure 4.2 décrit le processus complet effectué par la fonction F_{DL}. Chaque fichier de captures (PCAP), spécifique à un protocole, nécessite l'utilisation d'un pattern pour chaque protocole étudié, tel que PDL_{OS4I} pour le protocole OS4I, PDL_{ZB} pour le protocole ZigBee et PDL_{BLE} pour le protocole BLE. Chaque pattern analyse et extrait les informations requises pour effectuer la modélisation. Ces patterns reposent également sur l'utilisation d'une table de conversion permettant de faire correspondre les informations spécifiques au protocole et celles demandées par le paquet générique. Ainsi, la définition d'un pattern, pour un nouveau protocole, exige une connaissance et une analyse avancée de ce protocole. Chaque pattern produit une liste de paquets au format identique. Toutes les listes générées sont ensuite fusionnées en un seul fichier de capture contenant tous les paquets au format générique. La fusion des paquets consiste à ordonner tous les paquets en fonction du *timestamp* de chaque paquet. Ce PCAP est alors utilisé par le pattern PDL pour produire le graphe G_{DL}.

4.1.1 Patterns de conversion $PDL_{protocole}$

Comme détaillé dans la figure 4.2, la fonction F_{DL} utilise un pattern spécifique pour chaque protocole étudié. Pour chaque itération de ce pattern (PDL_{OS4I}, PDL_{ZB}, PDL_{BLE}), une table de conversion, spécifique à chaque protocole, est nécessaire pour convertir les informations obtenues dans le paquet, au format du protocole, vers le format attendu dans le paquet générique. Cette table de conversion est utilisée pour associer chaque élément du paquet vers un élément précis dans le paquet générique. Le paquet générique est découpé en cinq couches, ainsi, chaque pattern effectue ce processus en cinq étapes, de manière itérative :

1. Traitement de la couche physique. Il s'agit de récupérer une seule information, le *timestamp* du paquet. Ensuite, le pattern ajoute le nom du protocole utilisé sur ce paquet. Une fois cette information obtenue, le pattern exécute la deuxième étape ;
2. Traitement de la couche liaison. Le pattern extrait les adresses source et destination dans le champ correspondant du paquet. Ces adresses correspondent aux adresses physiques (MAC) des objets, utilisées pour les communications point à point.

3. Traitement de la couche réseau. Le pattern extrait les adresses source et destination du paquet dans le champ correspondant. Ces adresses correspondent aux adresses réseau du protocole, utilisées pour les communications finales. Il s'agit, en fait, de la réelle destination du paquet.
4. Traitement de la couche transport. C'est ici que la table de conversion est nécessaire. Le pattern extrait les informations contenues dans les couches spécifiques du protocole, puis les associe à la table de conversion afin d'obtenir le type d'application du paquet.
5. Traitement de la couche application. Cette étape est fortement dépendante de l'étape précédente. En effet, il s'agit de récupérer la donnée contenue dans le paquet. Cependant, cette donnée est disponible uniquement si le type du paquet correspond à un paquet de données. S'il s'agit d'un paquet de données, alors le pattern extrait la valeur, sinon, une valeur par défaut est utilisée correspondant au type du paquet.

4.1.1.1 PDL_{ZB} : Pattern de conversion spécifique au protocole Zigbee

la figure 4.3 expose les informations que le pattern PDL_{ZB} cible lors du traitement d'un paquet Zigbee. L'extraction des adresses source et destination de la couche liaison s'effectue dans la couche 802.15.4, dans les champs du même nom. L'extraction des adresses source et destination de la couche réseau a lieu dans la couche *ZigBee Network Layer*, dans les champs du même nom.

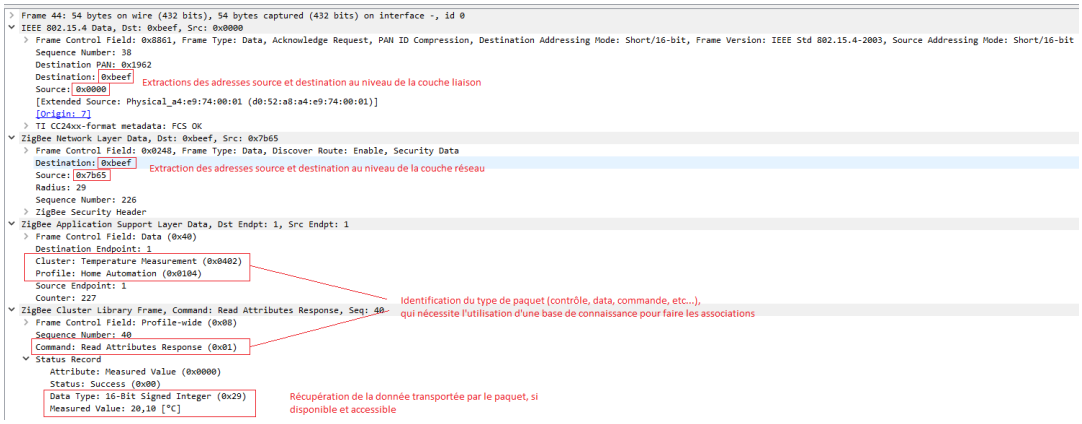

FIGURE 4.3 – Exemple d'un paquet ZigBee traité par le pattern PDL_{ZB}.

La génération du type d'application (*apptype*) s'appuie, dans un premier temps, sur les informations obtenues dans les couches *ZigBee Application Support* et *ZigBee Cluster library*. Puis une table de conversion est utilisée pour identifier le type d'application du paquet. Ce dernier est défini par trois valeurs principales : le *Profil*, le *Cluster* et le *frametype*. La table de conversion, illustrée dans le tableau 4.1, s'appuie sur ces trois valeurs pour identifier le type d'application correspondant pour le paquet générique.

Cette table de conversion n'est pas exhaustive et correspond aux besoins de la plateforme d'expérimentations. Notamment, nous nous basons uniquement sur le profil d'application Zigbee correspondant à *HA_Home_Automation*. De plus, notre plateforme d'expérimentations utilise des objets compatibles uniquement avec les deux clusters *Temperature_measurement* et *on_off*.

Type d'applications (Zigbee)			Type d'applications (Pile générique)
Profile	Cluster	Frametype	
HA_Home_Automation	Temperature_measurement	1	Commande
HA_Home_Automation	Temperature_measurement	0	Donnée
HA_Home_Automation	Temperature_measurement	2	ACK
HA_Home_Automation	on_off	1	Commande
HA_Home_Automation	on_off	0	Donnée
HA_Home_Automation	on_off	2	ACK
ZigBee Device Profile	*	*	Contrôle
Nul	Nul	Nul	Contrôle

TABLEAU 4.1 – Table de conversion du protocole Zigbee vers le format générique.

Lorsque le profil fait référence à *ZigBee Device Profile*, il n'y a pas de *cluster*, et tous les paquets sont destinés à contrôler et surveiller l'état du réseau et des objets. C'est pourquoi nous définissons tous les paquets de ce type comme des paquets avec le type d'application correspondant à contrôle.

La dernière ligne avec le *Profil*, le *Cluster* et le *frametype* à Nul fait référence à des paquets sans couches Zigbee, soit des paquets comprenant uniquement la couche liaison 802.15.4. Ces paquets font références à des paquets de contrôle, principalement, utilisés pour gérer l'état du réseau.

4.1.1.2 PDL_{OS4I} : Pattern de conversion spécifique au protocole OS4I

La figure 4.4 expose les informations que le pattern PDL_{OS4I} cible lors du traitement d'un paquet OS4I. La méthode d'extraction des informations, destinées à la couche liaison, est identique à celle opérée pour le protocole Zigbee. Les adresses source et destination de la couche réseau sont extraites depuis la couche *6LoWPAN*, dans les champs du même nom.

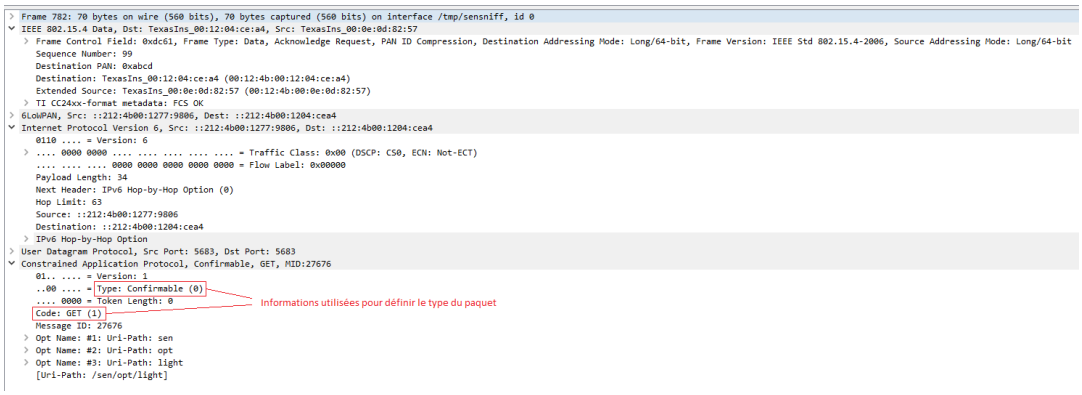

FIGURE 4.4 – Exemple d'un paquet OS4I traité par le pattern PDL_{OS4I}.

Le type d'application est défini en fonction des informations contenues dans la couche *CoAP* et à l'aide de la table de conversion. Le type d'application des paquets *CoAP* est défini en fonction du contenu des champs *type*, *payload* et *code* de la couche. La table de conversion, illustrée dans le tableau 4.2, s'appuie donc sur ces valeurs pour définir le type d'application correspondant pour le paquet générique.

Type d'applications (OS4I)			Type d'applications (Pile générique)
Type	Code	Payload	
CON	GET	*	Commande
CON	POST	*	Commande
ACK	2.05	Payload	Donnée
ACK	2.05	Nul	ACK
Nul	Nul	Nul	Contrôle

TABLEAU 4.2 – Table de conversion du protocole OS4I vers le format générique.

La table de conversion du protocole OS4I n'est pas exhaustive. Nous nous centrons sur certains *type* et *code*, définis dans le protocole CoAP, car ce sont ceux déployés et utilisés dans la plateforme d'expérimentations.

Le *type CON* fait principalement référence au type d'application *Commande* dans le paquet générique. En effet, le type CON est utilisé dès lors qu'un objet initie une connexion, que ce soit pour effectuer une requête *GET* ou une requête *POST*.

La dernière ligne du tableau 4.2 signifie que la couche CoAP n'est pas disponible dans le paquet. Par conséquent, il s'agit d'un paquet *UDP* ou *802.15.4*, tous deux utilisés pour contrôler le réseau et son état.

4.1.1.3 PDL_{BLE} : Pattern de conversion spécifique au protocole BLE

La figure 4.5 illustre les différentes couches composant un paquet BLE et les informations qui y sont stockées. Pour extraire les adresses source et destination spécifiques à la couche liaison, le pattern PDL_{BLE} inspecte la couche *Bluetooth Low Energy Link Layer* et les champs du même nom. C'est également dans cette couche et dans les mêmes champs que sont extraites les adresses source et destination spécifiques à la couche réseau. En effet, dans le protocole BLE, il n'y a pas de distinction entre les adresses au niveau des couches réseau et liaison. Elles sont donc identiques pour les deux couches.

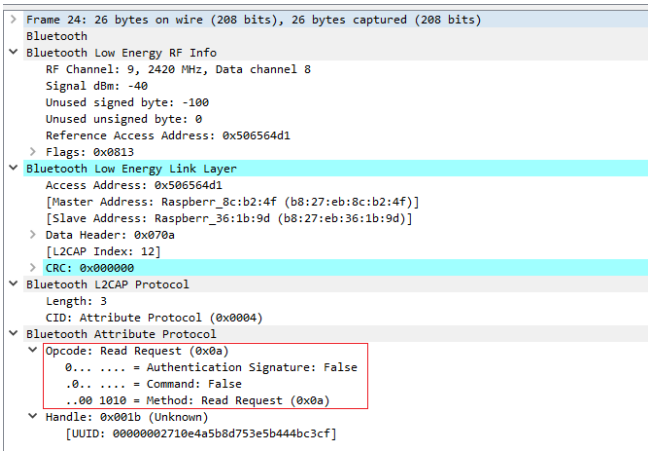

FIGURE 4.5 – Exemple d'un paquet BLE traité par le pattern PDL_{BLE}.

L'extraction du type d'application, par le pattern PDL_{BLE}, s'appuie sur les informations obtenues dans le champ *opcode* de la couche *Bluetooth Attribute Protocol* du protocole BLE, illustrée dans la figure 4.5. Le tableau 4.3 présente la table de

conversion des informations spécifiques au protocole BLE vers le type d'application attendu dans le paquet générique.

Type d'applications (BLE) Opcode	Type d'applications (Pile générique)
Read Request	Commande
Write Command	Commande
Read Response	Donnée
Handle Value Indication	Donnée
Exchange MTU Request	Contrôle
Find By Type Value Request	Contrôle
Find By Type Value Response	Contrôle
Nul	Contrôle

TABLEAU 4.3 – Table de conversion du protocole BLE vers le format générique.

Dans notre plateforme d'expérimentations, les applications utilisent sept principaux *opcode*, présentés dans le tableau 4.3. Tout comme les deux autres protocoles, cette table de conversion n'est pas exhaustive et représente uniquement les spécificités rencontrées dans les scénarios déployés. La dernière ligne du champ *opcode*, Nul, fait référence aux paquets BLE qui ne disposent pas de la couche *Bluetooth Attribute Protocol*. Ces paquets servent principalement à synchroniser les deux objets pour assurer le bon fonctionnement de la communication.

4.1.2 Pattern PDL : conversion du PCAP générique vers le graphe G_{DL}

Algorithme 1 Pattern *PDL* : Génération du graphe G_{DL}

Entrée: P le PCAP de paquets génériques
Sortie: G_{DL}

1: $N_{DL} \leftarrow \emptyset$
2: $E_{DL} \leftarrow \emptyset$
3: **for all** $p \in P$ **do**
4: **if** $p.apptype$!= ACK *and* $p.apptype$!= $Control$ **then**
5: $n_p \leftarrow (p.dlsrc, p.nwksrc)$
6: **if** $n_p \notin N_{DL}$ **then**
7: $n \leftarrow createNode(n_p, N_{DL})$
8: **else**
9: $n \leftarrow \{n \in N_{DL} | n_p.dlsrc \in n.dlsrc$ and $n_p.nwksrc \in n.nwksrc\}$
10: **end if**
11: $createEdge(n, p, E_{DL})$
12: **end if**
13: **end for**

```
Paquet générique : {
    timestamp;
    dlsrc;
    dldst;
    nwksrc;
    nwkdst;
    apptype;
    donnée;
}
```

```
Nœud : {
    ID;
    dlsrc;
    nwksrc;
    [roles];
}
```

Le pattern PDL, détaillé dans l'algorithme 1, extrait du PCAP générique tous les paquets utiles pour la modélisation. En effet, les paquets de type contrôle (ACK, routage, etc) ne sont pas considérés par notre modélisation. Tous les paquets restants sont ensuite transformés en arc de graphe, en utilisant le langage de graphe choisi, afin de générer G_{DL}.

C'est également pour ce pattern qu'une interaction humaine est nécessaire. En effet, lors de la création des arcs, si le nœud, correspondant au couple adresse de liaison et adresse réseau, n'existe pas, alors le pattern PDL le crée. Cependant, si ce nœud correspond à un objet physique possédant plusieurs interfaces, le pattern PDL n'est pas capable de lui associer toutes les interfaces. L'intervention de l'utilisation est donc nécessaire, afin de permettre au pattern PDL de produire un graphe conforme à ce qui est observé.

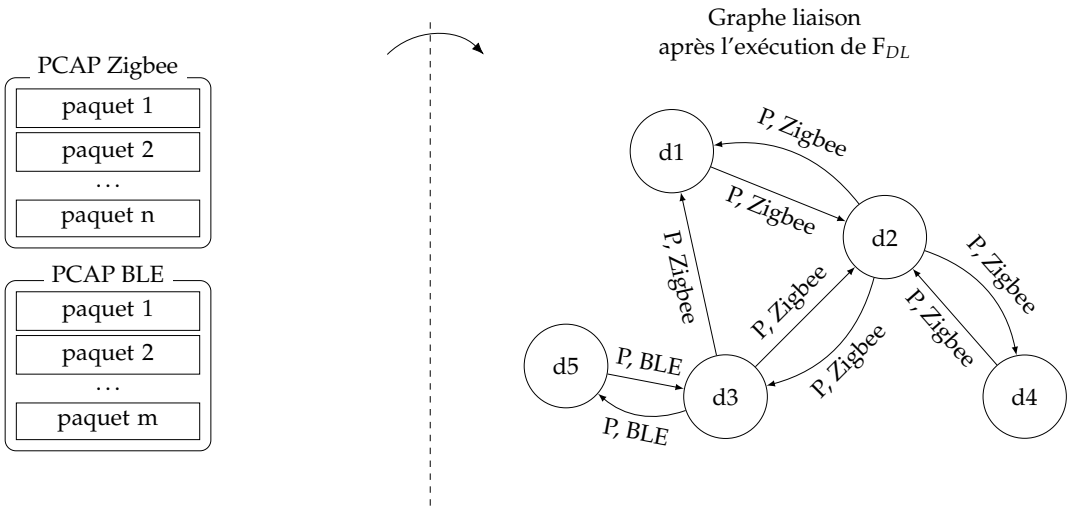

FIGURE 4.6 – Exemple de conversion de PCAPs vers le graphe liaison.

Un exemple d'exécution de la fonction F_{DL}, est illustré sur la figure 4.6. Dans cet exemple, deux fichiers de captures, un pour le protocole Zigbee et un autre pour le protocole BLE, sont utilisés comme paramètre d'entrée pour la fonction F_{DL}. Le résultat produit est illustré par le graphe dans la partie droite de la figure. Les patterns utilisés dans F_{DL} ont identifié cinq différents nœuds. De plus, un arc est créé entre deux objets, pour chaque paquet conforme aux attentes des patterns, avant d'être étiqueté. Pour simplifier la lecture de la figure, nous avons simplifié l'étiquette de l'arc, en regroupe dans la variable P les informations suivantes : *timestamp*, dlsrc, dldst, nwksrc, nwkdst, apptype, donnée.

4.2 Fonction de la couche réseau F_{NWK}

La fonction F_{NWK} utilise le pattern réseau P_{NWK} afin de construire le graphe G_{NWK} à partir de G_{DL}. P_{NWK} identifie les communications finales (*end-to-end*) entre les objets depuis le graphe liaison G_{DL}. Autrement dit, ce pattern détecte les communications entre deux nœuds, tout en excluant les nœuds intermédiaires, tels que les routeurs ou les passerelles. Nous utilisons les informations stockées dans chaque arc $e \in E_{DL}$ du graphe G_{DL} afin de créer les arcs du graphe G_{NWK}.

L'algorithme 2 représente le pattern P_{NWK}, utilisé pour modéliser les graphes de la couche réseau. Le pattern se base sur le graphe G_{DL} et utilise les informations stockées dans les arcs pour créer le graphe réseau. Il parcourt l'ensemble des arcs du graphe G_{DL}, noté E_{DL}. Si le couple d'adresses sources liaison et réseau, notées respectivement e_{dlsrc} et e_{nwksrc}, correspond à un nœud du graphe, noté n, alors un arc est tracé entre le nœud n et le nœud destination identifié par l'adresse réseau de destination stockée dans l'arc e, notée e_{nwkdst}.

Algorithme 2 Pattern P_{NWK} : Génération du graphe réseau

Entrée: G_{DL}
Sortie: G_{NWK}
1: N_{DL} l'ensemble des nœuds de G_{DL}
2: E_{DL} l'ensemble des arcs de G_{DL}
3: **for all** $(e \in E_{DL})$ **do**
4: $n \leftarrow \{n | n \in N_{DL}, e.nwksrc = n.nwksrc$ and $e.dlsrc = n.dlsrc\}$
5: $m \leftarrow \{m | m \in N_{DL}, e.nwkdst = m.nwksrc\}$
6: $createEdge(n, m, e)$
7: **end for**

Arc E_{DL} : {
timestamp;
dlsrc;
dldst;
nwksrc;
nwkdst;
payload;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

La figure 4.7 illustre le comportement de la fonction F_{NWK} et notamment le résultat produit par le pattern P_{NWK} à partir du graphe liaison. Le pattern parcourt le graphe de liaison, donné en entrée de la fonction F_{NWK}, puis extrait les informations spécifiques aux communications ayant lieu au niveau de la couche réseau. Par exemple, les objets $d3$ et $d4$, dans le graphe liaison, ne communiquent pas directement ensemble. En effet, $d3$ et $d4$ communiquent, entre autres, avec $d2$. Cependant, dans le graphe réseau, $d3$ et $d4$ apparaissent comme deux objets communicant ensemble. Ainsi, les communications échangées entre $d4$ et $d2$ contenaient des messages, non pas à destination de $d2$ mais pour $d3$ et réciproquement. La fonction F_{NWK} extrait cette information et utilise le pattern P_{NWK} pour créer un arc entre $d3$ et $d4$ afin de symboliser la communication entre les deux objets. Cette action est réitérée pour chaque arc et nœud du graphe liaison.

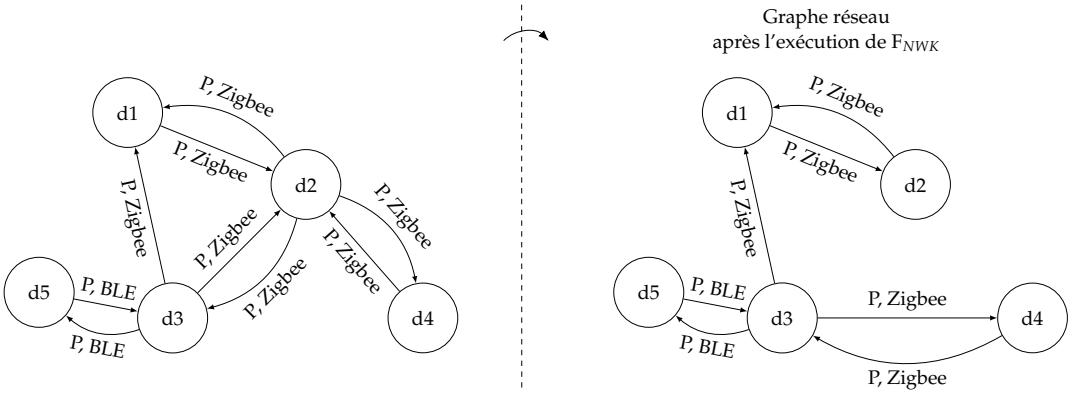

FIGURE 4.7 – Exemple de processus de création du graphe réseau à partir du graphe liaison.

4.3 Fonction de la couche transport F_{TRANS}

Dans cette section, nous présentons le processus de la fonction F_{TRANS}, ainsi que les trois patterns utilisés, pour produire le graphe G_{TRANS} à partir du graphe G_{NWK}, illustré sur la figure 4.8.

Les deux premiers patterns utilisés sont les algorithmes OWT (*One Way Transmission*) et PBC (*Periodic Bidirectionnal Communications*), illustrés respectivement par les algorithmes 3 et 4. Ils permettent d'identifier la direction du flux de données et le rôle à assigner pour chaque objet parmi : *source*, *sink*, *source-sink*, tous définis

en sections 3.3.3. Ces deux algorithmes utilisent le graphe G_{NWK} comme paramètre d'entrée et produisent chacun un graphe temporaire. Les deux graphes sont ensuite unifiés en un seul graphe, qui est utilisé par le dernier pattern, $CTRL$, illustré par l'algorithme 5, afin de générer le graphe G_{TRANS}.

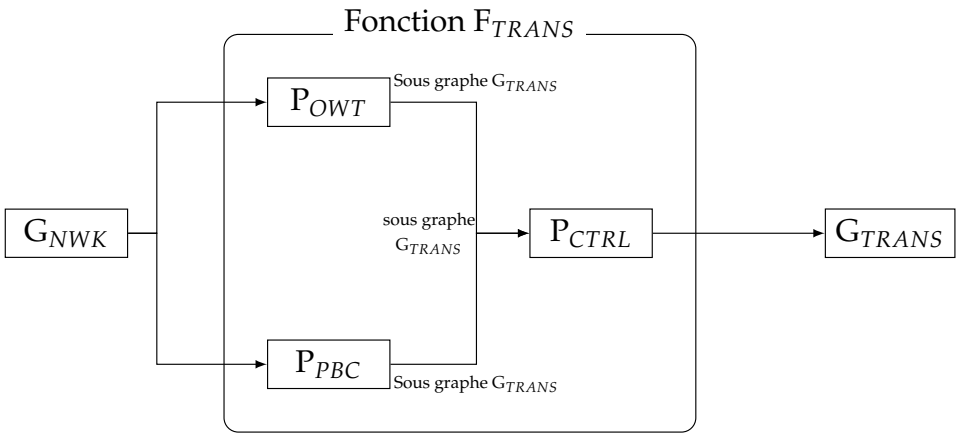

FIGURE 4.8 – Processus effectué par la fonction F_{TRANS}.

4.3.1 Algorithmes d'identification de *source* et de *sink* (OWT, PBC)

Deux algorithmes permettent d'identifier l'orientation du flux de données et de détecter le rôle d'un objet.

Algorithme 3 Pattern pour les communications unidirectionnelles (OWT)

Entrée: G_{NWK}
Sortie: Sous graphe G_{TRANS}

1: N_{NWK} l'ensemble des nœuds de G_{NWK}
2: E_{NWK} l'ensemble des arcs de G_{NWK}
3: $E_{TRANS} \leftarrow \emptyset$
4: **for all** $(n, m | n \in N_{NWK}, m \in N_{NWK}$ and $n \neq m)$ **do**
5: $\quad e_{(n,m)} \leftarrow \{e \in E_{NWK} | e.src = n$ and $e.dst = m\}$
6: $\quad e_{(m,n)} \leftarrow \{e \in E_{NWK} | e.src = m$ and $e.dst = n\}$
7: $\quad$ **if** $e(n,m) \neq \emptyset$ and $e(m,n) = \emptyset$ **then**
8: $\quad\quad addEdge(e_{n,m}, E_{TRANS})$
9: $\quad\quad addRole(n, source)$
10: $\quad\quad addRole(m, sink)$
11: $\quad$ **else if** $e(n,m) = \emptyset$ and $e(m,n) \neq \emptyset$ **then**
12: $\quad\quad addEdge(e_{m,n}, E_{TRANS})$
13: $\quad\quad addRole(m, source)$
14: $\quad\quad addRole(n, sink)$
15: $\quad$ **end if**
16: **end for**

Arc E_{NWK} : {
timestamp;
nwksrc;
nwkdst;
apptype;
payload;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

Le premier algorithme, OWT 3 (*One Way Transmission*), considère uniquement les communications unidirectionnelles entre deux objets. Puis, il détecte l'émetteur de la communication comme *source* et le destinataire comme *sink*. Ce pattern identifie les couples de nœuds, dont un seul des deux envoie des messages (lignes 7 et 11). Les objets qui échangent des messages ne sont pas considérés par ce pattern. Lorsque

qu'un couple, où un seul des objets communique, est identifié, ce pattern crée un arc orienté entre le nœud source et le nœud destination puis assigne le rôle *source* au nœud qui émet les messages et le rôle *sink* à celui qui les reçoit.

Le deuxième algorithme, PBC 4 (*Periodic Bidirectionnal Communications*), s'applique uniquement aux communications périodiques et bidirectionnelles entre deux nœuds, où l'un envoie une requête et l'autre répond à cette dernière. L'objet qui répond à une requête est identifié comme étant la *source* du flux de données et celui qui initie la communication, par l'envoie d'une requête, est identifié comme étant le *sink*. Nous choisissons de définir l'objet qui répond à la requête comme la source du flux de données, car il s'agit de l'objet qui délivre l'information. C'est à partir de cet objet que la donnée est générée puis envoyée sur le réseau. De même, l'objet qui initie la requête, est dans l'attente d'une information, d'où son rôle de *sink*, dans ce schéma de communication.

Algorithme 4 Pattern pour les communications périodiques et bidirectionnelles (PBC)

Entrée: G_{NWK}
Sortie: sous graphe G_{TRANS}

1: N_{NWK} l'ensemble des nœuds de G_{NWK}
2: E_{NWK} l'ensemble des arcs de G_{NWK}
3: $E_{TRANS} \leftarrow \emptyset$
4: **for all** $(n, m | n \in E_{NWK}, m \in E_{NWK}$ and $n \neq m)$ **do**
5: $\quad e_{(n,m)} \leftarrow \{e \in E_{NWK} | e.nwksrc = n.nwksrc$ and $e.nwkdst = m.nwksrc\}$
6: $\quad e_{(m,n)} \leftarrow \{e \in E_{NWK} | e.nwksrc = m.nwksrc$ and $e.nwkdst = n.nwksrc\}$
7: $\quad$ **for all** $e_1 \in e_{(n,m)}$ and $e_2 \in e_{(m,n)}$ **do**
8: $\quad\quad$ **if** $e_1.apptype = Commande$ and $e_2.apptype = Donnée$ **then**
9: $\quad\quad\quad addEdge(e_{m,n}, E_{TRANS})$
10: $\quad\quad\quad addRole(m, source)$
11: $\quad\quad\quad addRole(n, sink)$
12: $\quad\quad$ **else if** $e_2.apptype = Commande$ and $e_1.apptype = Donnée$ **then**
13: $\quad\quad\quad addEdge(e_{n,m}, E_{TRANS})$
14: $\quad\quad\quad addRole(n, source)$
15: $\quad\quad\quad addRole(m, sink)$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end for**

Arc E_{NWK} : {
timestamp;
nwksrc;
nwkdst;
apptype;
payload;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

Pour s'assurer que le bon rôle est assigné au bon objet, il faut déterminer l'orientation de l'échange, c'est à dire identifier l'objet qui initie la communication, et celui qui répond. C'est ici que nous utilisons le type d'application, défini dans le paquet générique. Nous distinguons deux différents types d'application dans ce pattern : le type d'application *Commande* et le type d'application *Donnée*. Le type d'application *Commande* est associé aux paquets considérés comme des requêtes d'informations. Le type d'application *Donnée*, quant à lui, est associé aux paquets contenant de la donnée. Ainsi, comme illustré par les lignes 8 et 12 de l'algorithme 4, l'objet qui émet un message de type *Commande* a pour rôle *sink* (lignes 11 et 15). Un objet qui répond

avec un message de type *Donnée* obtient le rôle *source* (lignes 10 et 14). De plus, le flux de donnée est orienté de l'objet ayant le rôle *source* à destination de l'objet ayant le rôle *sink*.

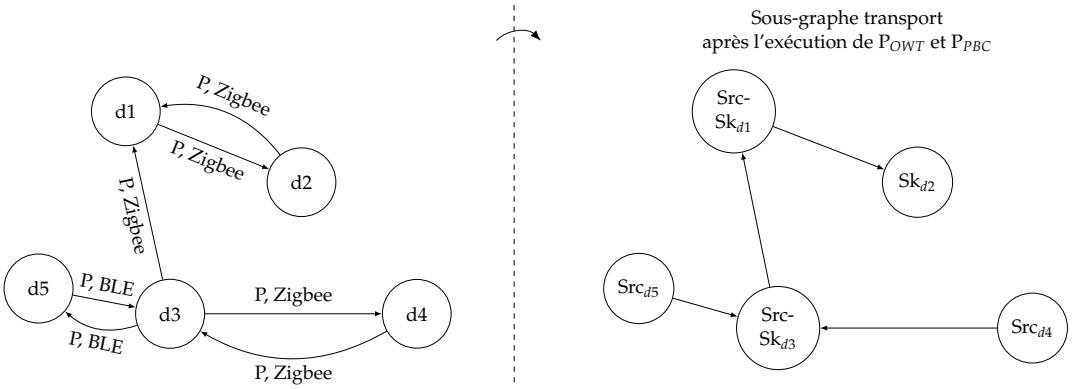

FIGURE 4.9 – Exemple de processus de création du sous-graphe transport à partir du graphe réseau.

La figure 4.9 illustre le comportement d'une partie de la fonction F_{TRANS} et notamment le résultat produit par les patterns P_{OWT} et P_{PBC}, à partir du graphe réseau. Ces deux patterns assignent un rôle à chaque objet du réseau et définissent l'orientation du flux de données à partir du graphe réseau. Le premier pattern, P_{OWT}, va appliquer cette logique aux communications unidirectionnelles (*one way*), c'est à dire, aux communications entre $d3$, jouant le rôle de *source* et $d1$ de *sink*, dans l'exemple illustré sur la figure 4.9. Le résultat est observable dans la partie droite de la figure, où les labels des objets $d3$ et $d1$ sont devenus, respectivement, Src$_{d3}$ et Sk$_{d1}$. Seulement, comme $d1$ et $d3$ sont également impliqués dans des communications où $d3$ a pour rôle *sink* (Sk$_{d3}$) et $d1$ le rôle *source* (Src$_{d1}$), leurs rôles finaux sont Src-Sk$_{d1}$ et Src-Sk$_{d3}$.

4.3.2 Algorithme d'identification de *controller* (CTRL)

Ce pattern, illustré par l'algorithme 5, ne s'applique que lorsque des communications impliquent des nœuds qui possèdent les deux rôles, *source* et *sink*. L'analyse commence par le nœud étiqueté *source*, puis suit le flux de données jusqu'à arriver aux nœuds identifiés comme *sink*. Nous définissons une variable δ, représentant le temps nécessaire pour qu'un nœud, identifié comme *source-sink*, obtienne l'information depuis une *source*, l'analyse et envoie une commande au nœud identifié comme *sink*.

Pour faciliter la compréhension de l'algorithme, définissons deux paquets :

— p_{ssk-sk} : le paquet échangé entre le nœud possédant le rôle *source-sink* et le nœud avec rôle *sink*

— $p_{src-ssk}$, le paquet échangé entre le nœud possédant le rôle *source-sink* et le nœud le rôle *source*

Si la différence entre les *timestamps* du paquet p_{ssk-sk} et du paquet $p_{src-ssk}$ est inférieure à la valeur de δ, alors nous supposons que cet objet agit comme un *controller*. Nous pouvons résumer l'explication comme suit :

$$p_{ssk-sk}.timestamp - p_{src-ssk}.timestamp < \delta \rightarrow n_{ssk}.roles = controller$$

Algorithme 5 Pattern d'identification de *controller* (CTRL)

Entrée: Sous graphe G_{TRANS}, δ # Seuil
Sortie: G_{TRANS}

1: N_{TRANS} l'ensemble des nœuds de G_{TRANS}
2: E_{TRANS} l'ensemble des arcs de G_{TRANS}
3: **for all** ($n \in N_{TRANS}|$ source-sink $\in n.roles$) **do**
4: $\quad e_{sk} \leftarrow \{e|e \in E_{TRANS}, e.nwksrc = n.nwksrc\}$
5: $\quad e_{src} \leftarrow \{e|e \in E_{TRANS}, e.nwkdst = n.nwksrc\}$
6: $\quad$ **for all** $e_1 \in e_{sk}$ **do**
7: $\quad\quad ts_1 \leftarrow e_1.timestamp$
8: $\quad\quad$ **for all** $e_2 \in e_{src}$ **do**
9: $\quad\quad\quad ts_2 \leftarrow e_2.timestamp$
10: $\quad\quad\quad$ **if** $ts_2 > ts_1$ and $ts_2 - ts_1 < \delta$ **then**
11: $\quad\quad\quad\quad setRole(n, "controller")$
12: $\quad\quad\quad$ **end if**
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: **end for**

Arc E_{TRANS} : {
timestamp;
nwksrc;
nwkdst;
apptype;
donnée;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

Cet algorithme est utilisé après que les rôles de *source* et *sink* aient été attribués à chaque dispositif du réseau. Finalement, le graphe G_{TRANS} est généré en fonction des différents rôles attribués à chaque nœud du graphe. En fonction des communications observées dans le réseau, un arc orienté est dessiné depuis un nœud dont le rôle est soit *source* soit *controller* à un nœud dont le rôle est soit *sink* soit *controller*.

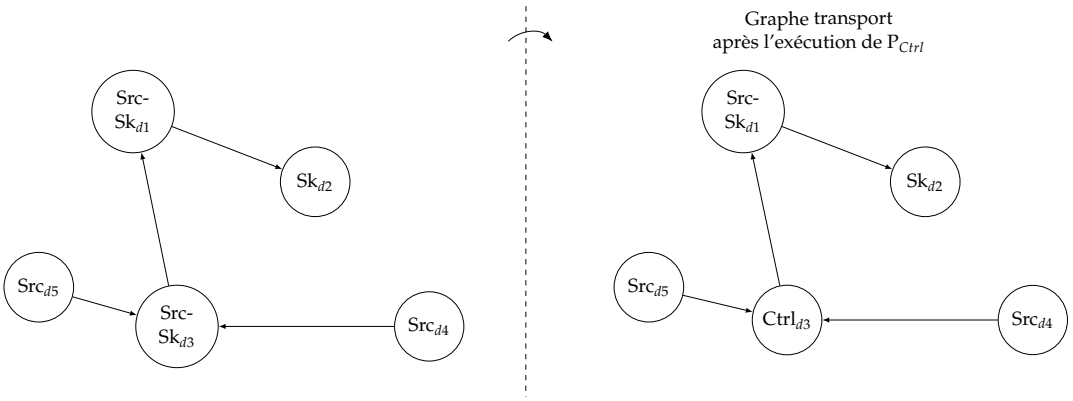

FIGURE 4.10 – Exemple de processus de création du graphe transport à partir du sous-graphe transport.

La figure 4.10 illustre le comportement de la seconde partie de la fonction F_{TRANS} ainsi que le résultat produit par le pattern P_{CTRL}, à partir du sous-graphe transport, produit par les deux autres patterns. Le pattern P_{CTRL} intervient lorsque tous les nœuds du graphe ont un rôle, afin d'identifier les nœuds qui possèdent des capacités d'analyse et de contrôle dans le réseau. Le pattern se base uniquement sur les objets qui possèdent le rôle *source-sink*.

Dans l'exemple illustré sur la figure 4.10, deux objets sont concernés, le nœud $d3$ et le nœud $d1$. Cependant uniquement l'objet $d3$ est considéré comme un *controller*, dont le rôle est modifié par $Ctrl_{d3}$. En effet, en plus d'observer le rôle de l'objet, le pattern P_{CTRL} va déterminer si ce dernier analyse un message reçu afin d'envoyer une commande par la suite. Ce comportement est analysé à l'aide d'une variable

δ, définie comme le temps nécessaire à l'objet pour analyser la donnée reçue avant d'envoyer un message. Dans l'exemple illustré sur la figure 4.10, l'objet $d4$ possède le rôle *source* et envoie des messages à l'objet $d3$. Dès la réception du message en provenance de $d4$, le nœud $d3$ analyse la donnée, et dans l'intervalle de temps défini par δ, envoie un message à $d1$. Ainsi, le pattern identifie l'objet $d3$ comme étant un *controller* dans ce réseau.

Le même traitement est appliqué à l'objet $d1$. Cependant, aucune corrélation n'est observée entre les messages envoyés depuis l'objet $d3$ et ceux émis à l'objet $d2$.

4.4 Fonction de la couche application F_{APP}

Les patterns d'applications sont utilisés et définis à des fins d'analyse des applications présentes dans un réseau IoT. Ils permettent la détection d'applications spécifiques qui pourraient être déployées, intentionnellement ou non, dans un réseau, à partir du graphe transport G_{TRANS}. Le processus de la fonction F_{APP} est illustré sur la figure 4.11.

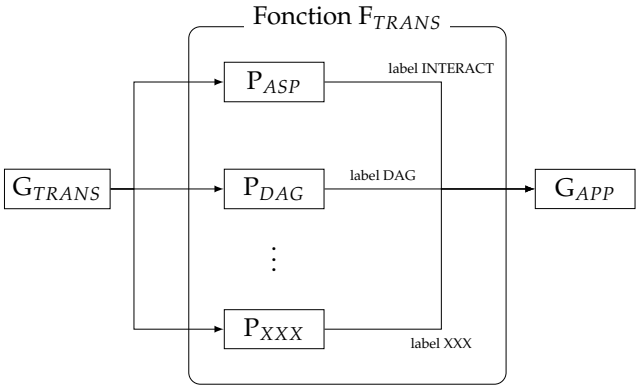

FIGURE 4.11 – Processus effectué par la fonction F_{APP}.

La fonction utilise les patterns de manière itérative en prenant le graphe G_{TRANS} à chaque fois comme entrée. Elle récupère, ensuite, tous les graphes produits par chaque pattern, puis les fusionne afin de générer le graphe G_{APP}. Chaque graphe produit par les patterns utilise un label d'arc spécifique, en fonction de l'application détectée. Ce label permet, par la suite, d'effectuer un filtre sur le graphe G_{APP}, pour effectuer une analyse d'une application spécifique.

Dans ce manuscrit, nous proposons deux patterns d'application. Le premier décrit un schéma applicatif représentant une application de type *capteur-actionneur*. Ce dernier met en évidence les interactions entre deux objets où l'un, le capteur, produit et envoie des données, et l'autre, l'actionneur, réagit en fonction de la donnée reçue. Le deuxième pattern détecte les schémas applicatifs *d'agrégateur de données*. Il identifie les dispositifs qui collectent et rassemblent les données dans le réseau afin de les stocker, de les analyser, etc.

4.4.1 Pattern capteur-actionneur (ASP)

Ce schéma d'application, représenté par l'algorithme 6, n'est utilisé que dans des schémas applicatifs *capteur-actionneur* multi-sauts. En effet, dans le cas d'une transmission directe entre le capteur et l'actionneur, nous pouvons identifier l'interaction dans le graphe transport, où le capteur est l'objet *source* et l'actionneur, l'objet *sink*.

Néanmoins, nous considérons les flux de données comprenant au moins un objet considéré comme *controller* entre un objet *source* et un objet *sink*. Puis, afin de garantir qu'une interaction existe entre la *source* et le *sink*, nous analysons le *timestamp* des paquets envoyés depuis la *source* et ceux envoyés par le *controller* à destination du *sink*. Si la différence entre eux est inférieure au seuil défini par la variable δ, alors nous supposons que le *controller* envoie des données ou des commandes au *sink*, en fonction des informations reçues depuis le dispositif étiqueté comme *source*.

Si cette condition est valide, alors il y a une interaction entre l'objet *source* et l'objet *sink* par l'intermédiaire du *controller*. Si plusieurs dispositifs *controller* existent sur le chemin, ou sur le flux de données, ce pattern est appliqué de façon récursive.

Algorithme 6 Pattern capteur-actionneur (ASP)

Entrée: G_{TRANS}, δ # Seuil
Sortie: G_{APP}

1: N_{TRANS} l'ensemble des nœuds de G_{TRANS}
2: E_{TRANS} l'ensemble des arcs de G_{TRANS}
3: $E_{APP} \leftarrow \emptyset$
4: **for all** $(n \in N_{TRANS} | n.role$ is $controller)$ **do**
5: $\quad e_{sk} \leftarrow \{e | e \in E_{TRANS}, e.nwksrc = n.nwksrc\}$
6: $\quad e_{src} \leftarrow \{e | e \in E_{TRANS}, e.nwkdst = n.nwksrc\}$
7: $\quad$ **for all** $(e_1 \in e_{sk}, (e_2 \in e_{src})$ **do**
8: $\quad\quad ts_1 \leftarrow e_1.timestamp$
9: $\quad\quad ts_2 \leftarrow e_2.timestamp$
10: $\quad\quad$ **if** $ts_2 > ts_1$ and $ts_2 - ts_1 < \delta$ **then**
11: $\quad\quad\quad s \leftarrow \{s | s \in N_{TRANS}, s.nwksrc = e_1.nwksrc\}$
12: $\quad\quad\quad d \leftarrow \{d | d \in N_{TRANS}, d.nwksrc = e_2.nwkdst\}$
13: $\quad\quad\quad createEdge(s, d, E_{APP})$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: **end for**

Arc E_{TRANS} : {
timestamp;
nwksrc;
nwkdst;
apptype;
donnée;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

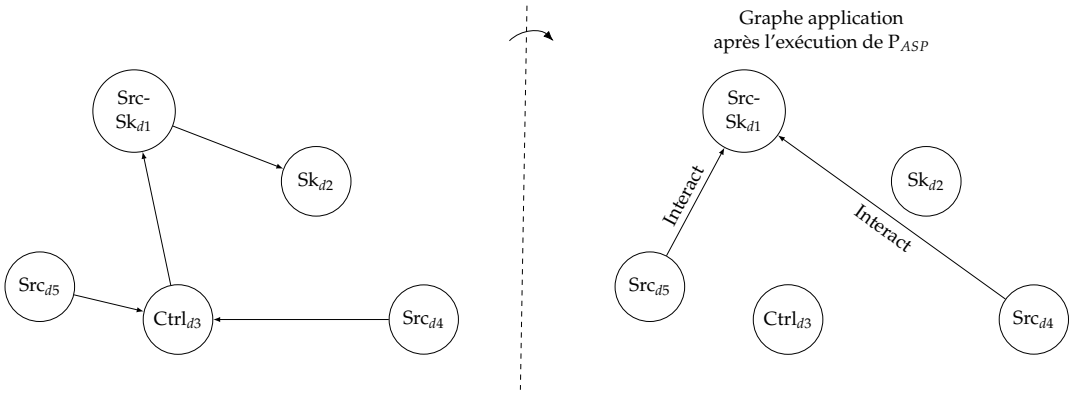

FIGURE 4.12 – Exemple de processus de création du graphe application ASP à partir du graphe transport.

La figure 4.12 illustre le comportement de la fonction F_{APP} et notamment le résultat produit par le pattern P_{ASP}, à partir du graphe transport. Le comportement de ce pattern est assez similaire à celui du pattern P_{CTRL}, utilisé par la fonction F_{TRANS}. Il faut nécessairement trois d'objets pour effectuer ce pattern : un objet possédant le

rôle *source*, qui sera l'élément déclencheur ; un ou plusieurs *controller* au milieu, pour propager le message ; et un objet avec le rôle *sink*, en fin de communication, élément qui effectuera la commande.

Dans l'exemple illustré sur la figure 4.12, les deux objets $d4$ et $d5$ envoient des messages à $d3$. Ce dernier, en tant que *controller*, analyse les messages et envoie les commandes correspondantes au nœud $d1$. Ainsi, les deux objets, $d4$ et $d5$, interagissent avec l'objet $d1$. Autrement dit, les données envoyées par ces deux objets ont un impact sur le comportement de l'objet $d1$.

4.4.2 Pattern d'agrégateur de données (DAG)

Ce pattern analyse les nœuds qui reçoivent des paquets en provenance d'au moins t objets identifiés avec le rôle *source*. Nous utilisons une variable t, pour définir le nombre d'objets requis pour que le nœud soit considéré comme un agrégateur de données. Nous supposons qu'un nœud qui rassemble les données depuis t *sources*, les analyse ou les stocke dans le but de les utiliser ultérieurement. Ce pattern est représenté par l'algorithme 7.

Algorithme 7 Pattern d'agrégateur de données (DAG)

Entrée: G_{TRANS}, t # seuil
Sortie: G_{APP}

1: N_{TRANS} l'ensemble des nœuds de G_{TRANS}
2: E_{TRANS} l'ensemble des arcs de G_{TRANS}
3: $N_{DAG} \leftarrow \{n | n \in N_{TRANS}, source - sink \in n.roles$ or $sink \in n.roles$ or $controller \in n.roles\}$
4: $E_{APP} \leftarrow \emptyset$
5: **for all** $(n \in N_{DAG})$ **do**
6: $\quad E_{subAPP} \leftarrow \emptyset$
7: $\quad e_{sk} \leftarrow \{e | e \in E_{TRANS}, e.nwkdst = n.nwksrc\}$
8: $\quad counter \leftarrow 0$
9: $\quad$ **for all** $e \in e_{sk}$ **do**
10: $\quad\quad s \leftarrow \{s | s \in N_{TRANS}, s.nwksrc = e.nwksrc\}$
11: $\quad\quad$ **if** *source* $\in s.roles$ or *controller* $\in s.roles$ **then**
12: $\quad\quad\quad counter \leftarrow counter + 1$
13: $\quad\quad\quad addEdge(e, E_{subAP})$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad$ **if** $counter < t$ **then**
17: $\quad\quad E_{subAPP} \leftarrow \emptyset$
18: $\quad$ **end if**
19: $\quad$ merge(E_{subAPP}, E_{APP})
20: **end for**

Arc E_{TRANS} : {
timestamp ;
nwksrc ;
nwkdst ;
apptype ;
donnée ;
}

Nœud : {
ID ;
dlsrc ;
nwksrc ;
[roles] ;
}

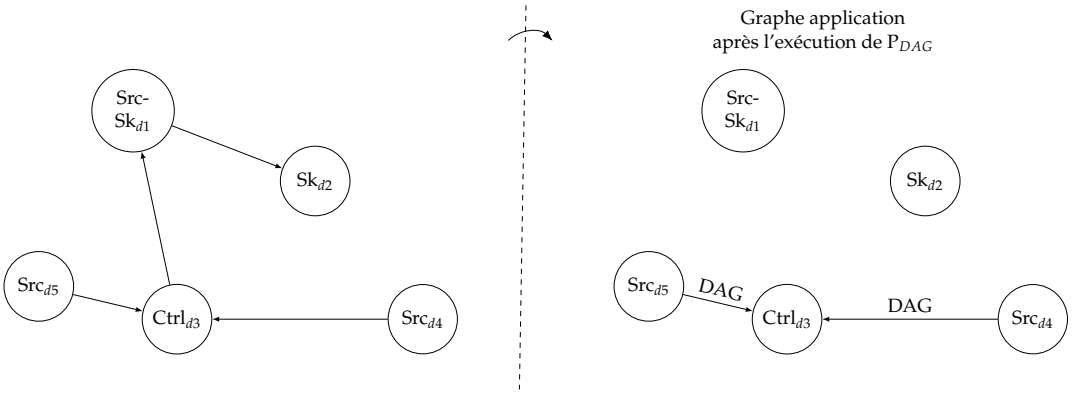

FIGURE 4.13 – Exemple de processus de création du graphe application DAG à partir du graphe transport.

La figure 4.13 illustre le comportement de la fonction F_{APP} et notamment le résultat produit par le pattern P_{DAG}, à partir du graphe transport. Ce pattern parcourt le graphe transport et identifie les objets considérés comme des agrégateurs de données. Pour ce faire, il cherche uniquement les objets qui possèdent les rôles *controller* ou *sink*. Puis, en fonction du nombre de communications entrantes, il les identifie comme étant des agrégateurs de données.

Dans l'exemple illustré sur la figure 4.13, nous définissons t, le nombre limite de communications, à 2. Ainsi, même si les objets $d1$ et $d2$ possèdent le rôle *sink*, ils ne remplissent pas la condition d'avoir à minima deux communications entrantes. Le seul objet qui remplit ces conditions est le *controller* $d3$. Ainsi, le pattern ne laisse paraître que les arcs à destination de $d3$, étiquetés DAG.

4.5 Résumé

Dans ce chapitre, nous avons présenté le processus complet de modélisation, en décrivant les fonctions et patterns utilisés pour générer chaque graphe.

Tout commence par la fonction F_{DL} qui construit le graphe G_{DL}, à partir des PCAPs obtenus. Nous avons introduit les patterns PDL qui permettent de convertir les paquets contenus dans les PCAPs au format spécifique, vers des paquets au format générique. Puis le pattern P_{DL} extrait les informations des paquets génériques pour créer le graphe G_{DL}.

La deuxième fonction F_{NWK} utilise le pattern P_{NWK} pour construire le graphe G_{NWK}, en s'appuyant sur les informations obtenues et contenues dans le graphe G_{DL}.

La fonction F_{TRANS} utilise trois patterns différents P_{OWT}, P_{PBC} et P_{CTRL} pour construire le graphe G_{TRANS}.

La dernière fonction F_{APP} repose sur l'utilisation de plusieurs patterns pour créer le graphe G_{APP}, contenant plusieurs applications, différenciées par un label associé à chaque arc du graphe.

Tous les patterns, présentés dans ce chapitre, sont élaborés avec comme condition initiale que l'observation de l'environnement est idéale et n'a subit aucune dégradation. Dans le cas contraire, les patterns actuels ne sont plus adaptés pour analyser les informations obtenues et pour procéder à la génération des graphes.

Dans le chapitre suivant, nous proposons des versions modifiées de certain pattern, afin de compléter le processus de modélisation, même quand l'observation de l'environnement est dégradée.

Chapitre 5

Construction du modèle par détection de patterns : cas dégradé

Dans ce chapitre, nous présentons les patterns utilisés pour générer la modélisation d'un réseau hétérogène d'objets connectés, lorsque l'observation de l'environnement est dégradé. Nous considérons qu'il y a une dégradation, lorsque l'information est manquante après l'analyse des communications entre les objets, ou que l'observation est partielle. Dans une première section, nous décrivons un environnement dont l'observation est dégradée par une absence d'informations. Une seconde section décrit le processus de modélisation lorsque l'observation de l'environnement est partielle. Dans chaque section, après avoir décrit notre vision de l'environnement, nous présentons les modifications apportées aux différents patterns, afin de réaliser le processus modélisation.

5.1 Observation de l'environnement dégradée par une absence d'informations

5.1.1 Description de l'environnement

Cette section présente l'environnement dans lequel nous considérons que l'observation est dégradée par une absence d'informations. Il s'agit d'une vision externe au réseau, tel que pourrait avoir un attaquant ou un auditeur. De leur point de vue, l'information n'est pas accessible directement, ou bien est incomplète. Nous considérons, dans ce chapitre, que le manque d'informations peut intervenir suite à deux possibilités : une base de conversion incomplète ou encore le chiffrement du paquet.

5.1.1.1 Base de conversion incomplète

Tout d'abord, nous nous intéressons à un environnement où l'observation est dégradée par l'utilisation d'une base de conversion incomplète. La base de conversion fait le lien entre les informations spécifiques à un protocole et celles attendues pour le format générique. Il est important de noter que cette base de conversion s'appuie sur une analyse et une étude des protocoles IoT, de leurs fonctionnalités et de leurs applications. Ainsi, lorsqu'une application n'a pas été étudiée dans un protocole connu, ou lorsqu'un nouveau protocole est utilisé dans le réseau, la base de conversion n'est pas en mesure d'extraire certaines informations (ex : le type d'application). Cette dernière est alors incomplète et les patterns, qui l'utilisent, doivent être modifiés.

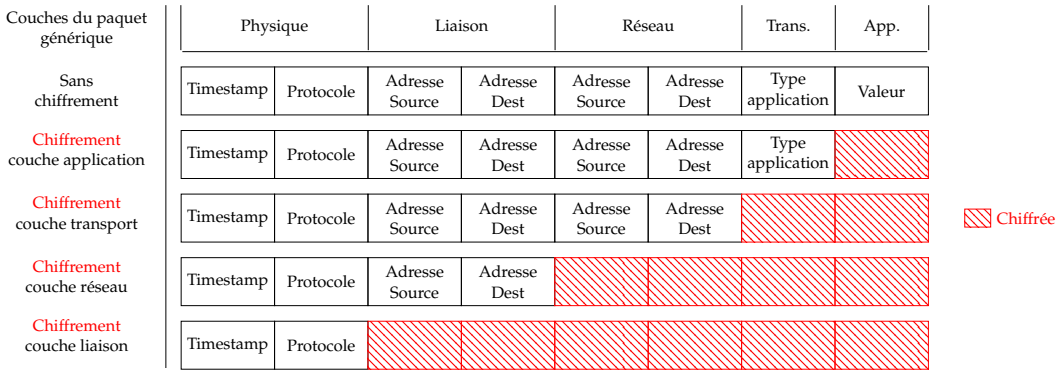

FIGURE 5.1 – Représentation du paquet générique en fonction du chiffrement par couche.

5.1.1.2 Utilisation du chiffrement

Nous présentons, dans cette section, un environnement dont l'observation est dégradée par l'utilisation du chiffrement dans les communications. En s'appuyant sur le découpage de la pile IoT générique, présentée en section 2.1.2, le message peut être chiffré de manière incrémentale, en commençant par la couche la plus haute, la couche applicative, jusqu'à la couche liaison. Dans ce manuscrit, nous ne considérons pas le chiffrement de la couche physique.

Le chiffrement d'une couche inférieure encapsule le chiffrement des couches qui lui sont supérieures. Par exemple, la figure 5.1 présente le fonctionnement du chiffrement par couche : lorsque la couche réseau est chiffrée, cela implique de fait, le chiffrement des couches supérieures, soit les couches transport et application. Dans la figure, la couche chiffrée est représentée par un bloc rouge hachuré exprimant alors que l'information est inaccessible.

5.1.2 Modification des patterns

Le processus de modélisation repose sur une construction itérative de graphes, en commençant par le graphe de liaison jusqu'au graphe application, tel que présenté dans le chapitre 4 et la figure 4.1.

Le processus global de modélisation n'est pas impacté par la dégradation de l'observation de l'environnement, laissant alors l'ordre d'exécution des fonctions tel quel. Cependant, si les fonctions ne sont pas modifiées par la dégradation, un ou plusieurs patterns doivent être adaptés afin de réaliser la modélisation.

La figure 5.2 illustre le processus d'exécution des fonctions pour modéliser un environnement dont l'observation est dégradée par une absence d'informations. Les patterns qui subissent une modification, ainsi que les fonctions qui les utilisent, sont représentés en rouge.

Deux patterns sont modifiés, les pattern P_{DL} et P_{TRANS}, renommés respectivement P'_{DL} et P'_{TRANS}.

La modification du pattern P'_{DL} impacte les patterns de conversion des protocoles spécifiques, vers le format générique : PDL_{BLE}, PDL_{ZB} et PDL_{OS4I}, renommés de fait PDL'_{BLE}, PDL'_{ZB} et PDL'_{OS4I}. Ces patterns reposent sur une base de conversion, qui lorsque l'observation est dégradée par une absence d'informations, devient soit incomplète, soit inadaptée aux paquets observés. Ainsi, une modification de ces bases de conversion et des patterns associés est nécessaire. La modification de la base

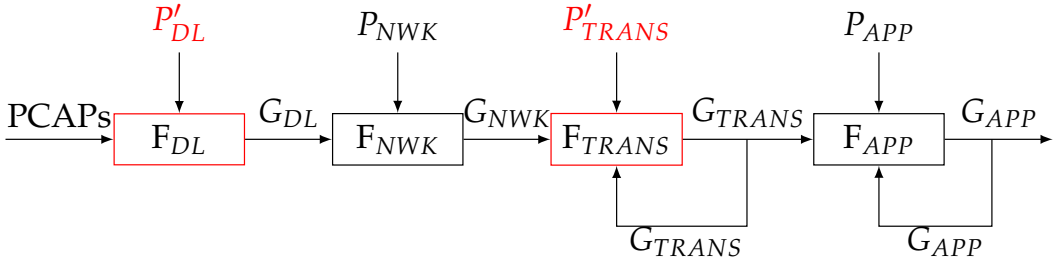

FIGURE 5.2 – Approche itérative basée sur les graphes, adaptée lorsque l'observation de l'environnement est dégradée par une absence d'informations.

de conversions et des patterns se fait par une analyse des paquets, au format spécifique d'un protocole, afin d'inférer des informations supplémentaires. De même, le pattern P_{TRANS} requiert l'utilisation du type d'application, présent dans le paquet générique. Cependant, cette information n'est plus accessible, il faut alors procéder d'une autre façon pour construire le graphe transport G_{TRANS}. Cette modification du pattern est effectuée dans le pattern P'_{TRANS}.

5.1.2.1 PDL'_{ZB} : modification du pattern PDL_{ZB}

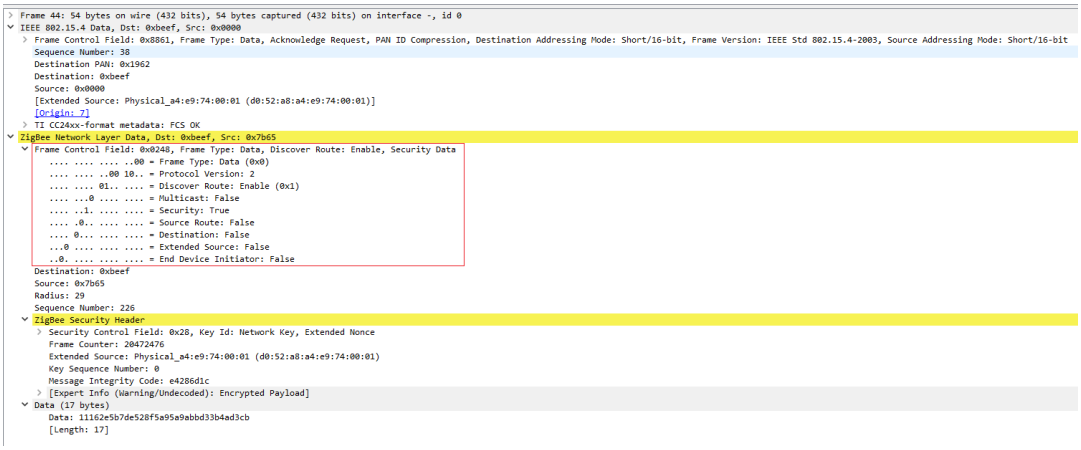

FIGURE 5.3 – Exemple d'un paquet ZigBee chiffré, traité par le pattern PDL_{ZB}.

La figure 5.3 montre les informations accessibles lorsque le paquet est chiffré, par rapport au même paquet non chiffré (figure 4.3). Les champs *ZigBee Application Support* et *ZigBee Cluster Library*, dans la version en clair, ont été remplacés par un unique champ, *ZigBee Network Layer*. Ainsi, la base de conversion doit s'appuyer uniquement sur ces informations.

Dans le cas idéal, il était possible d'identifier si un paquet était une commande ou une réponse contenant de la donnée. Dorénavant, nous identifions uniquement que le paquet est de type donnée. Cette perte d'informations impacte le processus effectué par le pattern P_{TRANS}, car il n'est plus aussi intuitif de déterminer la direction du flux de données.

Le tableau 5.1 présente la base de conversion des paquets spécifiques au protocole Zigbee, vers des paquets, au format défini par notre pile IoT générique, présentée en section 3.2. Nous divisons en deux étapes l'identification et l'extraction d'informations.

La première étape, illustrée par la partie haute du tableau, consiste à utiliser uniquement les informations fournies par le paquet, sans procéder à une inspection spécifique du paquet. Les quatre premières lignes du tableau illustrent les informations obtenues dans le paquet au format Zigbee (Type d'application Zigbee) et leurs correspondances dans le format générique (Type d'application Pile générique). En revanche, lorsque le paquet est de type *Data* dans le format Zigbee, ou *Inconnu* dans le format générique, il faut, dans une seconde étape, analyser plus précisément la taille du paquet.

Type d'applications (Zigbee)	Type d'applications (Pile générique)
Beacon	Contrôle
Command	Contrôle
Rejoin request	Contrôle
Rejoin response	Contrôle
Data	Inconnu

Inspection du paquet

Taille du paquet	Type d'applications (ZigBee)	Type d'applications (Pile générique)
$\leq$ 48 octets	ACK	Contrôle
52 octets	ZCl (Default ou Report attributes)	Contrôle
53 octets	Link status	Contrôle
$\geq$ 70 octets	Link Quality Response	Contrôle
Autres	ZCL (Read attributes, ...)	Inconnu

TABLEAU 5.1 – Base de conversion du protocole Zigbee vers le paquet générique, lorsque l'observation est dégradée par une absence d'informations.

Cette analyse est illustrée par la partie basse du tableau. L'inspection approfondie de la taille du paquet de type *Data*, nous permet d'extraire le type d'applications Zigbee, correspondant si les communications n'étaient pas chiffrées. Dans notre analyse, nous parvenons à différencier cinq tailles de paquets. Pour chaque taille présentée dans le tableau, nous identifions le type d'application Zigbee correspondant, ainsi que leur valeur dans le format générique.

La dernière ligne du tableau fait référence à des paquets que nous considérons comme étant de l'information utile, à conserver pour effectuer la modélisation. Cependant, nous ne sommes pas en mesure d'identifier si le paquet correspond à un paquet de type commande ou de type donnée, dans notre format générique, d'où le type *Inconnu*.

5.1.2.2 PDL'_{OS4I} : modification du pattern PDL_{OS4I}

La figure 5.4 illustre un paquet OS4I chiffré. Le champ CoAP, qui permettait de déterminer si le paquet était une commande ou une réponse contenant de la donnée, est remplacé par un champ DTLS (*Datagram Transport Layer Security*). Ce champ ne différencie plus les types d'applications OS4I (ACK, CON, GET ou POST). Il précise uniquement que le paquet est un paquet de données. Ainsi, comme pour le cas Zigbee, la modification de cette information impacte principalement le pattern P_{TRANS}.

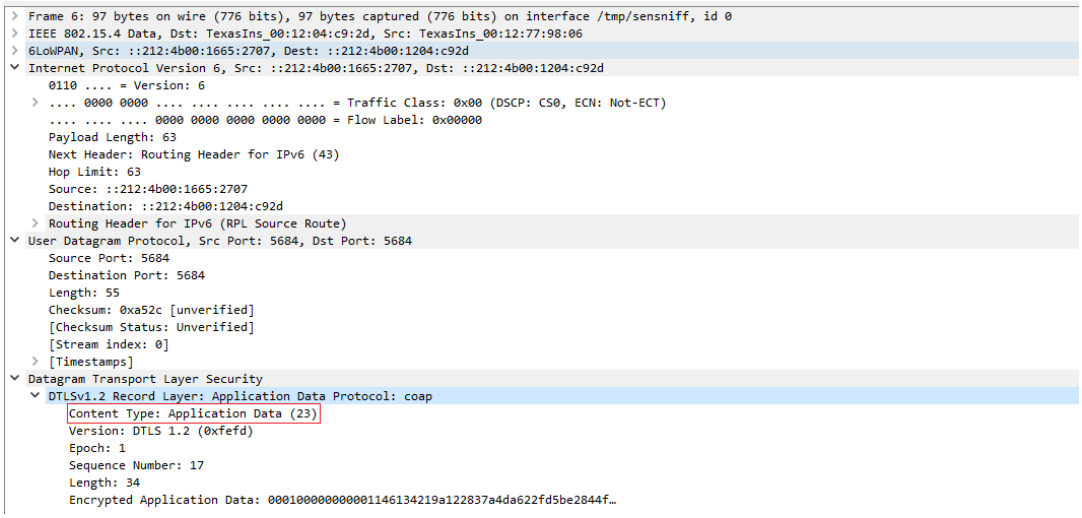

FIGURE 5.4 – Exemple d'un paquet OS4I chiffré, traité par le pattern PDL_{OS4I}.

Que ce soit pour Zigbee ou OS4I, les paquets de contrôle, gérés par les couches liaison ou réseau, restent inchangés, car le chiffrement n'est actif qu'à la couche application. Ainsi, la base de conversion est toujours efficace pour les différencier des paquets de données.

Le tableau 5.2 présente l'approche pour convertir les paquets chiffrés au format spécifique de la pile de protocoles OS4I, vers le format générique. Tous les paquets de données sont encapsulés dans des paquets au format DTLS. Cette approche s'effectue également en deux étapes. La première étape, présentée dans la partie haute du tableau, extrait les informations disponibles uniquement en affichant le paquet. Cela permet de distinguer le processus d'établissement d'une liaison (*handshake*), composé de plusieurs étapes, des paquets de données.

DTLS content-type (OS4I)	Type d'applications (Pile générique)
Client Hello	Contrôle
Server Hello	Contrôle
Hello Verify Request	Contrôle
Server Hello Done	Contrôle
Client Key Exchange	Contrôle
Handshake	Contrôle
Change Cipher Spec	Contrôle
Encrypted Alert	Contrôle
Application Data	Inconnu

Inspection du paquet

Taille du paquet	Type d'applications (OS4I)	Type d'applications (Pile générique)
82 octets	ACK	Contrôle
83 octets	ACK	Contrôle
Autres	GET ou POST ou ACK de données	Inconnu

TABLEAU 5.2 – Base de conversion du protocole OS4I, vers le paquet générique, lorsque l'observation est dégradée par une absence d'informations.

La seconde étape consiste à analyser la taille des paquets lorsque le paquet est

de type *Application Data* dans le format OS4I, afin d'obtenir des informations supplémentaires, dans le but de filtrer plus de paquets ne contenant pas de données. En CoAP, chaque objet recevant une requête, répond à l'aide d'un paquet d'acquittement (ACK). Cependant, en fonction de la requête d'origine, le paquet d'acquittement dispose d'un contenu (*ACK de données*) différent. Avec une analyse du paquet en fonction de sa taille, nous sommes en mesure de déterminer les paquets d'acquittement ne contenant aucune information (ACK). Nous pouvons, ainsi, les filtrer, afin qu'ils ne soient pas considérés lors de la modélisation. La partie basse du tableau 5.2 présente les caractéristiques de cette seconde étape.

5.1.2.3 PDL'_{BLE} : modification du pattern PDL_{BLE}

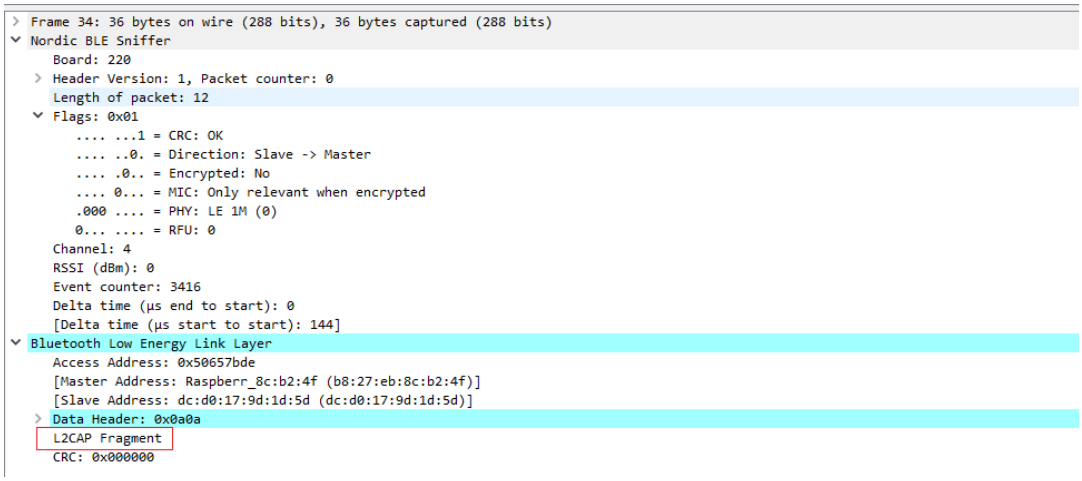

FIGURE 5.5 – Exemple d'un paquet BLE chiffré, traité par le pattern PDL_{BLE}.

La base de conversion du pattern PDL_{BLE} devient très restreinte lorsque le chiffrement est actif sur le protocole BLE. En effet, comme il est montré par la figure 5.5, la seule information restante dans le paquet, au niveau de la couche application, est *L2CAP fragment*. Par conséquent, dès que de la donnée ou des commandes sont envoyées entre deux objets qui utilisent le chiffrement, les paquets auront la même information : *L2CAP fragment*. Autrement dit, tous les autres paquets sont ignorés par le pattern PDL_{BLE}. Le pattern PDL'_{BLE} considère alors uniquement les paquets dont le type d'application est *L2CAP fragment*.

Contrairement aux autres protocoles étudiés, l'analyse de la taille des paquets n'est pas aussi précise et pertinente, dans le cas du protocole BLE. En effet, une des raisons à cela est que le protocole BLE ne dispose pas de paquet de contrôle en raison de sa topologie en *maître-esclave*. De plus, les paquets pouvant être considérés comme du contrôle par notre méthodologie ne sont pas différenciable par leur taille, dans le cas où les communications sont chiffrées.

5.1.2.4 P'$_{TRANS}$: modification du pattern P$_{TRANS}$

L'observation dégradée de l'environnement impacte les patterns d'extraction et de conversion vers le format générique et entraîne une modification du pattern P$_{PBC}$, utilisé par la fonction F$_{TRANS}$. Ce dernier, présenté et détaillé en section 4.3.1, utilise le champ *apptype* du paquet générique pour définir l'orientation du flux de données ainsi que pour assigner le rôle aux objets du réseau.

Cependant, dans le cas où les communications sont chiffrées, les patterns d'extraction d'informations, ne peuvent plus déterminer avec précision le type d'application utilisé, ce qui rend ce pattern moins efficace. Il faut donc opérer certaines modifications dans ce pattern, afin qu'il ne s'appuie plus uniquement sur le champ *apptype* pour définir les rôles et l'orientation du flux de données.

Algorithme 8 Pattern pour les communications périodiques et bidirectionnelles (PBC')

Entrée: G_{NWK}, θ # Seuil
Sortie: Part of G_{TRANS}

1: N_{NWK} l'ensemble des nœuds de G_{NWK}
2: E_{NWK} l'ensemble des arcs de G_{NWK}
3: $E_{TRANS} \leftarrow \emptyset$
4: **for all** $(n, m | n \in E_{NWK}, m \in E_{NWK}$ and $n \neq m)$ **do**
5: $\quad e_{(n,m)} \leftarrow \{e \in E_{NWK} | e.nwksrc = n.nwksrc$ and $e.nwkdst = m.nwksrc\}$
6: $\quad e_{(m,n)} \leftarrow \{e \in E_{NWK} | e.nwksrc = m.nwksrc$ and $e.nwkdst = n.nwksrc\}$
7: $\quad$ **for all** $e_1 \in e_{(n,m)}$ and $e_2 \in e_{(m,n)}$ **do**
8: $\quad\quad ts_1 \leftarrow e_1.timestamp$
9: $\quad\quad ts_2 \leftarrow e_2.timestamp$
10: $\quad\quad$ **if** $ts_1 > ts_2$ and $ts_1 - ts_2 < \theta$ **then**
11: $\quad\quad\quad addEdge(e_{n,m}, E_{TRANS})$
12: $\quad\quad\quad addRole(n, source)$
13: $\quad\quad\quad addRole(m, sink)$
14: $\quad\quad$ **end if**
15: $\quad\quad$ **if** $ts_2 > ts_1$ and $ts_2 - ts_1 < \theta$ **then**
16: $\quad\quad\quad addEdge(e_{m,n}, E_{TRANS})$
17: $\quad\quad\quad addRole(m, source)$
18: $\quad\quad\quad addRole(n, sink)$
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**

Arc E_{NWK} : {
timestamp;
nwksrc;
nwkdst;
apptype;
payload;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

L'algorithme 8 illustre le pattern PBC', c'est à dire la modification du pattern PBC, lorsque les communications sont chiffrées. Plutôt que d'utiliser le champ *apptype*, défini dans le paquet générique, qui dans ce cas a pour valeur *inconnu*, ce pattern s'appuie sur une analyse temporelle pour d'identifier la *source* et le *sink* de la communication ainsi que l'orientation du flux de données. Afin d'effectuer cette analyse temporelle, nous introduisons une variable θ, définie comme le temps maximal pour un objet de répondre à une requête reçue. De cette façon, cet algorithme est alors capable d'identifier la *source* et le *sink* lorsque deux objets communiquent ensemble. Cet algorithme sélectionne toutes les communications qui impliquent deux objets, n et m, illustré en ligne 5 de l'algorithme 8. Puis, il regroupe, dans deux variables distinctes, les communications émises par chacun des objets : e_1 pour les communications émises par l'objet n (ligne 6) et e_2 pour les communications émises par l'objet m (lignes 7). Une comparaison est ensuite effectuée pour déterminer si les deux communications sont corrélées (lignes 11 et 16). Elle s'appuie sur les *timestamps* de chaque communication avec les prérequis suivants :

— $ts_1 > ts_2$ and $ts_1 - ts_2 < \theta$: L'objet m envoie une requête à l'objet n et ce dernier répond à cette requête. Le pattern définit donc l'orientation du flux de données comme allant de l'objet n vers l'objet m et attribue le rôle de *source* à n et le rôle *sink* à m.

— $ts_2 > ts_1$ and $ts_2 - ts_1 < \theta$: L'objet n envoie une requête à l'objet m et ce dernier répond à cette requête. Le pattern définit donc l'orientation du flux de données comme allant de l'objet m vers l'objet n et attribue le rôle de *source* à m et le rôle *sink* à n.

5.2 Observation partielle de l'environnement

5.2.1 Description de l'environnement

Cette section définit une autre dégradation possible de l'environnement, celle où l'observation est partielle. Différents éléments peuvent induire que l'observation réalisée n'est pas complète. Le premier élément, concerne la limitation de la porté d'écoute du matériel d'interception. En effet, des objets déployés ou encore des communications effectuées au delà de la portée du dispositif d'interception, ne sont plus interceptés et détectés par cette dernière. Ceci rend partielle l'observation du système. Les autres éléments pouvant intervenir et ayant pour conséquence une observation incomplète, sont les problèmes liés à l'interception elle-même.

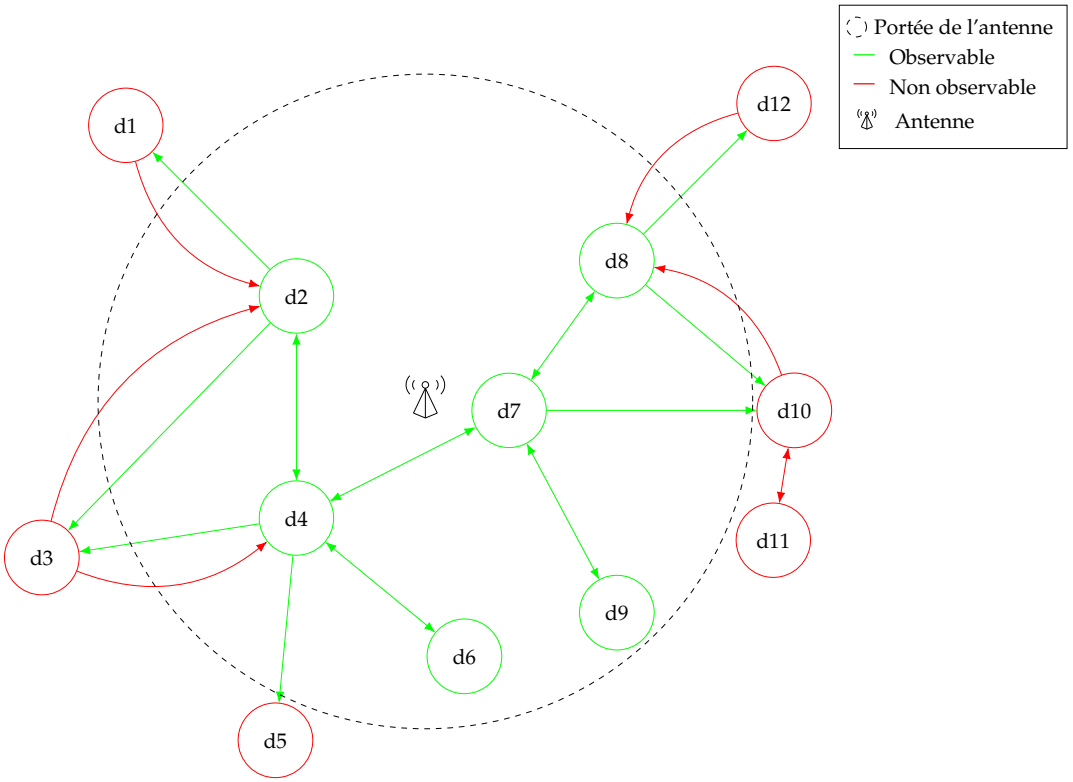

FIGURE 5.6 – Exemple de graphe de liaison d'un environnement dégradé due à une observation partielle.

5.2.1.1 Objets ou communications hors de portée du matériel d'écoute

La figure 5.6 illustre, par un graphe de liaison, un schéma dans lequel l'observation ne comprend pas tous les objets et communications du réseau. Le cercle en

pointillé noir représente la portée du dispositif d'interception. Les nœuds affichés en vert sont considérés comme étant observables et ceux affichés en rouge comme hors de portée de l'observation.

Une communication peut être interceptée si l'objet qui émet la communication est à portée du matériel d'interception et donc visible. Toutes les communications émises par un objet hors de portée de l'observation, ne peuvent être capturés, car elles ne sont pas visibles pour le matériel d'interception.

Par exemple, dans la figure 5.6, les communications entre les objets $d2$ et $d4$ sont toutes interceptées et capturées par le dispositif d'écoute, représenté par une antenne. En revanche, entre les objets $d1$ et $d2$, seules les communications émises par $d2$ sont interceptées par le matériel d'écoute. En effet, $d2$ est visible et donc tout le trafic en provenance de $d2$ peut être capturé. Cependant, les messages envoyés depuis $d1$ ne peuvent être interceptés par le matériel d'écoute, car $d1$ est hors de portée de l'observation et donc non visible.

De plus, lorsque deux objets hors de portée du matériel d'interception communiquent ensemble, $d10$ et $d11$ dans la figure 5.6, aucun message ne peut être capturé.

5.2.1.2 Problèmes d'interception

Dans un environnement où des objets et des communications sont observables, même de façon incomplète, d'autres événements peuvent dégrader le niveau d'observation obtenu. En effet, plusieurs problèmes peuvent intervenir au niveau du dispositif d'écoute, que ce soit au niveau du matériel ou du logiciel, empêchant l'interception et la détection des communications.

Le premier élément à considérer est la qualité du matériel d'interception. Si ce dernier est de mauvaise qualité, plusieurs paquets peuvent être omis lors de l'interception, rendant l'observation de moins bonne qualité. En effet, si l'antenne est située dans une zone à fort taux de communications, avec beaucoup de trafic, en provenance de nombreux objets, ou dans une zone avec de nombreuses interférences, certains messages ou paquets peuvent être manqués lors de l'interception.

De même, un dispositif d'interception qui utilise un logiciel défectueux, ou mal optimisé, ne peut pas traiter une trop grande quantité de paquets simultanément, et une surcharge peut ainsi entraîner une perte de paquets, à cause d'un temps de traitement trop long.

5.2.2 P'$_{NWK}$: modification du patterns P$_{NWK}$

La dégradation par l'observation partielle, tout comme pour l'absence d'information, n'impacte pas le processus global de modélisation. Ainsi, l'ordre d'exécution et la méthodologie des fonctions ne sont pas perturbés. Cependant, nous modifions un pattern dans le processus pour parvenir à modéliser le réseau, même si plusieurs éléments ne sont pas compris dans l'observation. La figure 5.7 illustre le processus de modélisation, en précisant le pattern modifié afin de répondre à la dégradation étudiée.

La dégradation par une observation partielle a des conséquences à tous les niveaux de la modélisation. Si les objets ne sont pas visibles, il ne peuvent être détectés par la fonction F$_{DL}$. De même, la fonction F$_{NWK}$ ne peut pas, avec ses patterns actuels, relier des objets non visibles entre eux. Les fonctions F$_{TRANS}$ et F$_{APP}$ renvoient de nombreuses erreurs en raison d'un nombre de communications insuffisant ou nul.

Dans le cas où les conséquences sont liées à une absence où à un nombre insuffisant de communications, nous ne pouvons pas adapter les patterns pour combler

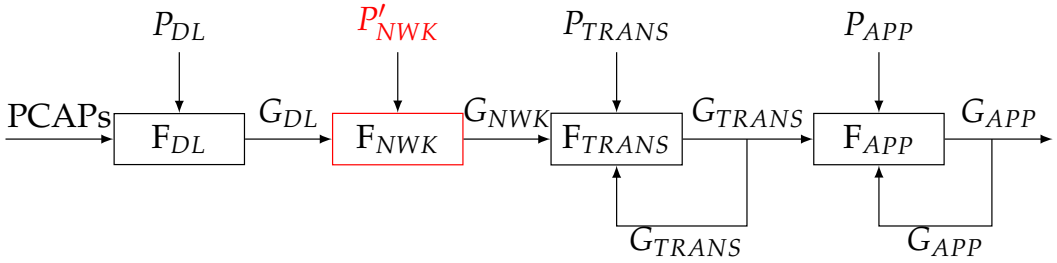

FIGURE 5.7 – Approche itérative basée sur les graphes, adaptée lorsque l'observation de l'environnement est partielle.

le déficit. Cependant, nous proposons de modifier le pattern P_{NWK}, renommé en P'_{NWK}, afin que ce dernier soit capable de détecter et de créer des nœuds, initialement non visible, par inspection des paquets et des informations obtenues grâce au graphe liaison.

Par exemple, dans la figure 5.6, le schéma de communications impliquant les objets $d12$, $d8$ et $d10$ montre que $d12$ et $d8$, ainsi que $d8$ et $d10$, communiquent de façon bidirectionnelle. Cependant, les objets $d10$ et $d12$ et les communications émises par ces dernier, ne sont pas identifiés et interceptés par l'antenne. La figure 5.8 illustre le résultat obtenu après l'exécution de la fonction F_{NWK} lorsque cette dernière utilise le pattern P_{NWK} et celui obtenu avec l'utilisation du pattern P'_{NWK}.

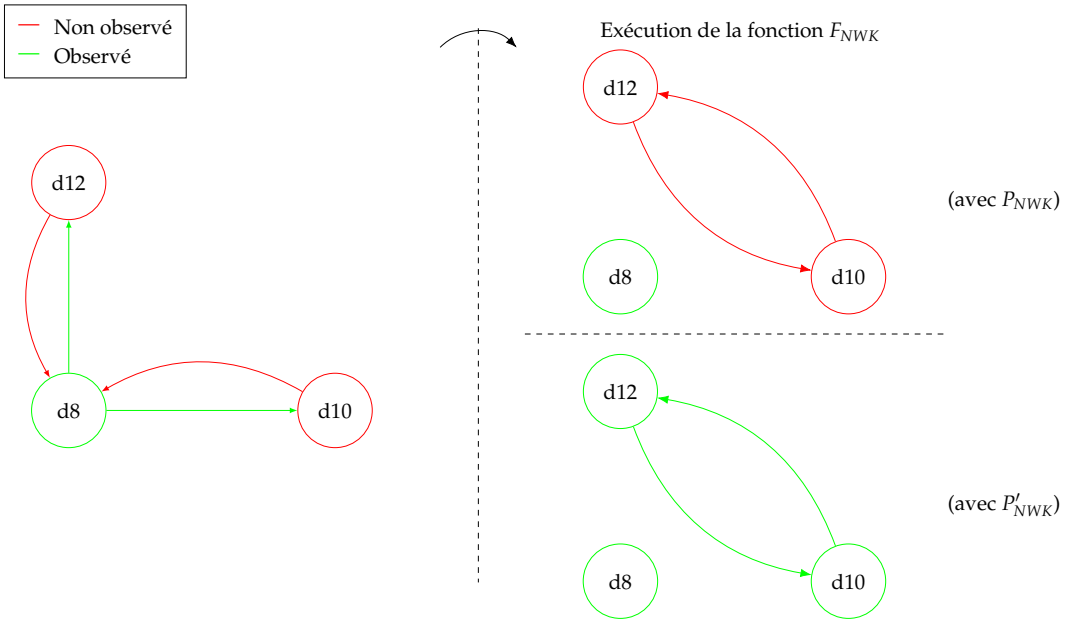

FIGURE 5.8 – Graphes G_{NWK} générés après l'exécution de la fonction F_{NWK} avec les deux patterns P_{NWK} et P'_{NWK} lorsque l'environnement est dégradé par une observation partielle.

L'utilisation du pattern P_{NWK} ne permet pas de créer d'arc entre les deux objets $d12$ et $d10$, si ces derniers ne sont pas compris dans l'observation. Le pattern P'_{NWK}, représenté par l'algorithme 9, détecte que des objets non identifiés sont utilisés dans un schéma de communication. Dès lors, P'_{NWK} est capable de créer le nœud afin qu'il puisse être utilisé dans la modélisation.

Dans le pattern P'_{NWK}, nous ajoutons une vérification sur l'existence d'un nœud en fonction des informations obtenues sur les arcs du graphe G_{DL}. Aux lignes 3 et 4

Algorithme 9 Pattern P'_{NWK}

Entrée: G_{DL}
Sortie: G_{NWK}

1: N_{DL} l'ensemble des nœuds de G_{DL}
2: E_{DL} l'ensemble des arcs de G_{DL}
3: $mac \leftarrow \{n.dlsrc, \forall n \in N_{DL}\}$
4: $nwk \leftarrow \{n.nwksrc, \forall n \in N_{DL}\}$
5: **for all** $(e \in E_{DL})$ **do**
6: $n \leftarrow \{n | n \in N_{DL}, e.nwksrc = n.nwksrc$ and $e.dlsrc = n.dlsrc\}$
7: $m \leftarrow \{m | m \in N_{DL}, e.nwkdst = m.nwksrc\}$
8: **if** $e.dlsrc \notin mac$ **then**
9: $createNode(n)$
10: $mac.append(e.dlsrc)$
11: **end if**
12: **if** $e.nwkdst \notin nwk$ **then**
13: $createNode(m)$
14: $nwk.append(e.nwkdst)$
15: **end if**
16: $createEdge(n, m, e)$
17: **end for**

Arc E_{NWK} : {
timestamp;
dlsrc;
dldst;
nwksrc;
nwkdst;
payload;
}

Nœud : {
ID;
dlsrc;
nwksrc;
[roles];
}

de l'algorithme, nous constituons deux ensembles : *mac* contenant toutes les adresses liaison de chaque nœud de N_{DL} (ligne 3) et *nwk* contenant toutes les adresses réseau de chaque nœud de N_{DL} (ligne 4). Lors du parcours des arcs du graphe G_{DL}, si le nœud source identifié par n est un nœud qui n'existe pas dans *mac*, le pattern crée ce nœud avec la fonction *createNode*, puis l'ajoute à l'ensemble *mac* pour éviter de dupliquer le nœud (lignes 8 à 10). De façon identique, la pattern ajoute le nœud identifié par m, si ce dernier n'est pas dans l'ensemble *nwk* (lignes 12 à 14). Lorsque les deux nœuds impliqués dans la communication sont identifiés, le pattern peut créer l'arc entre les deux nœuds.

5.3 Résumé

Dans ce chapitre nous avons présenté deux dégradations qui peuvent venir perturber l'observation d'un environnement : la perte d'informations et l'absence de communications ou de nœuds.

Lorsque la dégradation est une perte d'informations dans les paquets observés, nous avons proposé des modifications de plusieurs patterns pour que la méthodologie de modélisation puisse fonctionner. Dans un premier temps, nous avons présenté la modification apportée aux bases de conversion des protocoles Zigbee, OS4I et BLE afin de convertir les paquets du format spécifique d'un protocole, vers le format générique.

Dans un second temps, nous avons présenté le pattern P'_{TRANS}, qui est une modification du pattern P_{TRANS}, pour que ce dernier n'utilise plus le type d'application comme seul moyen d'inférer le rôle de chaque objet ainsi que l'orientation du flux de données. Nous avons proposé une approche temporelle qui permet d'identifier quel dispositif initie une communication et celui qui répond, lorsque la communication est bidirectionnelle.

Nous avons également proposé une modification du pattern P_{NWK}, que nous avons appelé P'_{NWK}, afin de proposer une solution lorsque l'observation de l'environnement est partielle, avec des communications et des objets hors de portée du matériel d'interception.

Chapitre 6

IoTMap

Dans ce chapitre, nous introduisons le *framework* IoTMap, une plateforme de modélisation basée sur de la détection d'applications déployées dans un réseau. IoTMap est une plateforme facilitant la compréhension et l'analyse de réseaux IoT hétérogènes avec une vision sécurité. Il peut être utilisé, aussi bien dans une évaluation de sécurité du réseau avec un point de vue offensif (test d'intrusion par exemple), que défensif (détection ou prévention d'attaques). Nous avons conçu ce *framework* de façon modulaire afin de pouvoir répondre aux différents critères des visions offensives et défensives, permettant l'évaluation de sécurité. Ce projet est également libre et open source, disponible sur le répertoire Github IoTMap [1].

Tout d'abord, nous décrivons une présentation générale du *framework*. Dans une seconde section, nous présentons l'architecture logicielle IoTMap, en exposant ses classes et leurs relations. Puis, nous décrivons plus précisément les modules importants d'IoTMap, en commençant par le paquet générique. Nous présentons ensuite le module permettant l'interception des communications, suivi du module responsable de la modélisation des graphes. Enfin, nous décrivons le module responsable de la base de données.

6.1 Présentation générale d'IoTMap

IoTMap est une plateforme qui couvre toutes les couches des travaux présentés sur la figure 2.8, en section 2.3. Ce *framework* est capable d'intercepter le trafic du réseau observé, de le capturer et de le sauvegarder dans un format de type PCAP. À partir des captures enregistrées, IoTMap analyse le trafic afin de détecter certains comportements définis par des patterns, pouvant représenter l'orientation du flux de données ainsi que les rôles des objets du réseau. Enfin, IoTMap propose une visualisation du réseau observé par des graphes.

La figure 6.1 illustre l'architecture générale d'IoTMap, décomposée en trois parties principales : l'écoute, l'analyse et la visualisation. La partie écoute, représentée à gauche sur la figure 6.1, s'occupe de l'interception des communications, la capture du trafic et la conversion des paquets, spécifiques au format du protocole observé, dans le format du paquet générique, présenté en section 3.2.

La partie analyse prend en charge la modélisation des différents graphes. Le processus de modélisation est itératif, où la création d'un graphe nécessite les informations obtenues et générées par le graphe précédent. IoTMap utilise le graphe courant ainsi que les patterns associés afin de construire le graphe suivant. Chaque graphe généré est stocké en base de données, en suivant le formalisme proposé par le langage Neo4J [2].

1. https://github.com/AlgoSecure/iotmap
2. https://neo4j.com/

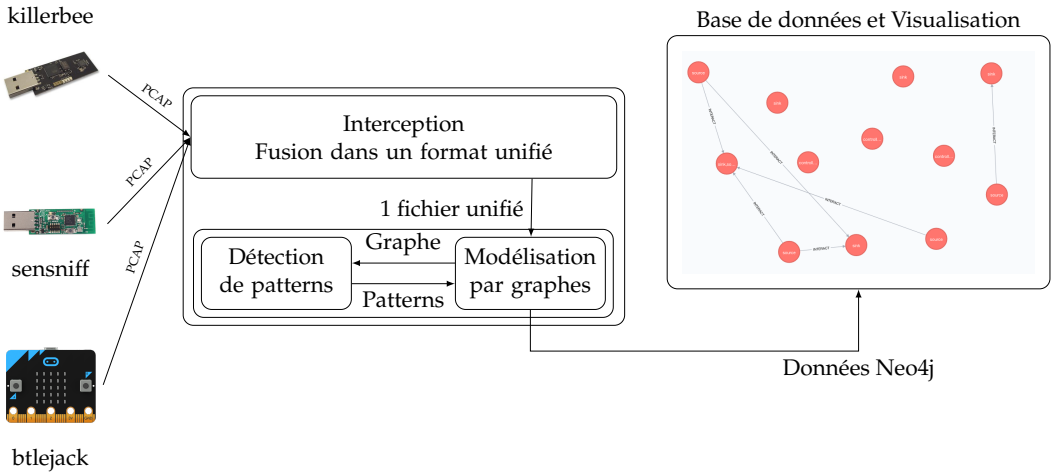

FIGURE 6.1 – Architecture du *framework* IoTMap.

La dernière partie de l'architecture d'IoTMap, la visualisation, permet à l'utilisateur d'avoir une représentation graphique du réseau observé. L'interface est accessible depuis un navigateur Web, en accédant directement au service exposé par Neo4J. La partie visualisation offre plusieurs possibilités. Une liste de requêtes préconfigurées permet l'affichage de tous les graphes générés lors de la modélisation ou l'accès à différents éléments détectés par les patterns. Il est également possible d'effectuer des requêtes au format *Cypher*, pour interroger la base de données, afin d'obtenir des graphes ou toutes autres données plus spécifiques dans le but d'effectuer une analyse plus approfondie.

IoTMap s'appuie sur divers projets existants, notamment pour la partie écoute. En effet, la philosophie allant avec IoTMap, n'est pas de proposer un outil supplémentaire, aux côtés de tous les autres projets, mais plus de proposer une approche qui unifie et combine l'analyse de différents protocoles dans un *framework* unique, plus facile à déployer et à utiliser. Avec cette approche, nous souhaitons populariser la sécurité des réseaux IoT en facilitant son évaluation et son amélioration. C'est pourquoi, nous avons pensé IoTMap de façon modulaire, afin qu'il puisse être maintenu et amélioré plus facilement, sans devoir modifier toute la logique du programme. Notamment, pour ajouter des patterns et des nouveaux protocoles.

L'ajout de nouveaux protocoles ne demande pas la modification de la logique de modélisation. Cependant, cela demande une certaine compréhension du protocole, afin de produire une base de conversion des paquets au format spécifique du protocole, vers celui du paquet générique. De même, l'ajout d'un nouveau pattern demande que les formalismes imposés par IoTMap et Neo4J soient respectés pour que ce pattern puisse être intégré dans le *framework*.

6.2 Architecture détaillée d'IoTMap

Dans cette section, nous présentons l'architecture détaillée d'IoTMap. Nous exposons les différentes classes qui composent le *framework*, ainsi que leurs relations. L'architecture est illustrée par la figure 6.2.

Les fonctions proposées par IoTMap sont regroupées en quatre modules spécifiques : *Paquet générique*, *Interception*, *Modélisation* et *Base de données*.

Même si les modules sont indépendants, certains composants doivent interagir ensemble pour assurer le bon fonctionnement d'IoTMap. Par exemple, le module

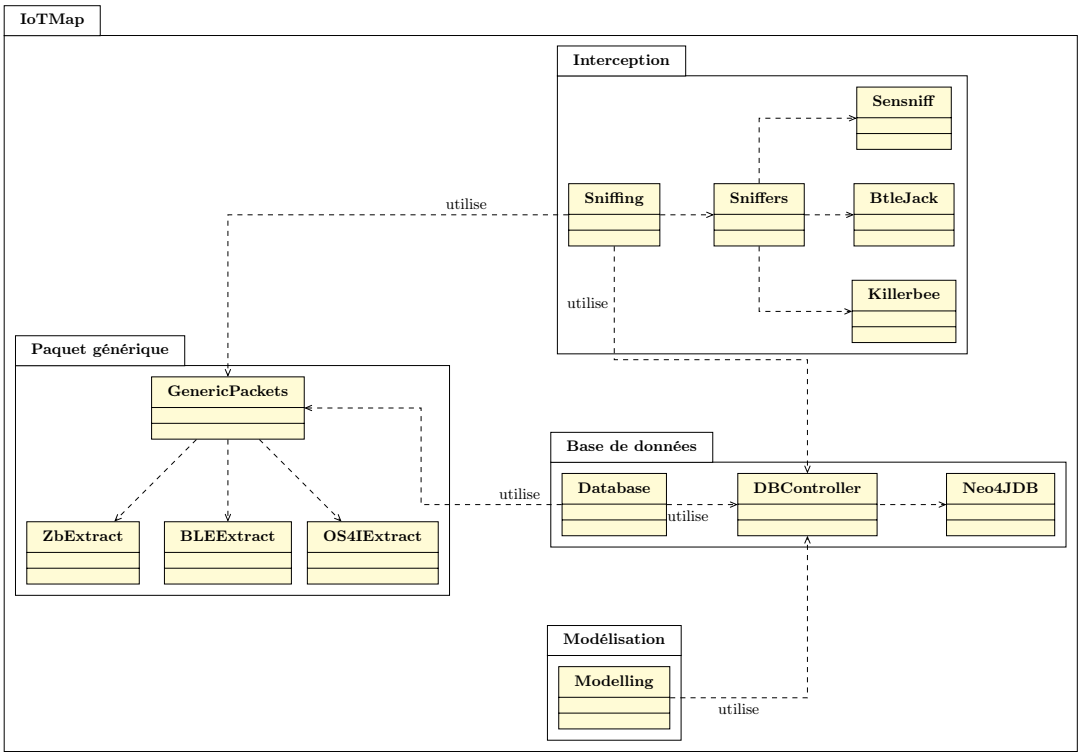

FIGURE 6.2 – Architecture d'IoTMap. Représentation simplifiée des composants d'IoTMap

Interception qui gère les fonctionnalités d'écoute doit pouvoir utiliser la classe *GenericPackets*, du module *Paquet générique*, pour convertir les paquets spécifiques d'un protocole vers le format générique. De plus, le module *Interception* nécessite également l'accès aux fonctions de la classe *DBController*, du module *Base de données*, pour stocker les paquets capturés dans la base de données Neo4J.

6.3 Le paquet générique

Dans cette section, nous présentons notre implémentation du paquet générique dans le *framework* IoTMap. La figure 6.2 illustre la relation entre les classes, qui définissent les extracteurs spécifiques à chaque protocole (ZbExtract, OS4IExtract et BLEExtract), et l'interface *GenericPackets* qui convertit en une liste de paquets génériques, les paquets obtenus par les extracteurs et les bases de conversions associées.

6.3.1 Les extracteurs de paquets

Chaque protocole étudié nécessite le développement d'un extracteur. Ce dernier permet de lire, d'analyser et enfin d'extraire les informations contenues dans un paquet. La figure 6.3 illustre le squelette de code à compléter pour chaque extracteur.

Chaque extracteur dispose de cinq fonctions. La fonction principale, représentée en ligne 2, *extract_pkt_layers()*, est utilisée pour chaque paquet contenu dans un PCAP. C'est depuis cette fonction que sont appelées les fonctions restantes. Ces dernières correspondent à des fonctions d'extraction, pour une couche spécifique dans le paquet, en commençant par la couche physique, jusqu'à la couche application. Nous divisons l'extraction du paquet selon le même découpage que le paquet générique.

```python
class Extract():
  def extract_pkt_layers(self, pkt):
    """
    Fonction  princiaple, qui retourne l'extraction des informations du paquet pour toutes les couches
    """

  def extract_pkt_rflayer(self, pkt):
    """
    Fonction qui extrait les informations spécifiques à la couche physique
    """

  def extract_pkt_layer2(self, pkt):
    """
    Fonction qui extrait les informations spécifiques à la couche liaison
    """
  def extract_pkt_layer3(self, pkt):
    """
    Fonction qui extrait les informations spécifiques à la couche réseau
    """
  def extract_pkt_layer4(self, pkt):
    """
    Fonction qui extrait les informations spécifiques aux couches transport et application
    """
```

FIGURE 6.3 – Squelette de code pour définir un extracteur de protocole.

L'inspection de paquets au format spécifique d'un protocole repose sur l'utilisation de la bibliothèque Scapy [3]. Elle permet de disséquer le paquet par couche, contenant tous les champs spécifiques au protocole. Nous procédons ensuite à l'analyse du paquet disséqué, champ par champ, afin d'extraire les informations qui nous semblent utiles. La figure 6.4 illustre un exemple de code utilisé dans la fonction *extract_pkt_layer3* de l'extracteur spécifique au protocole Zigbee. En fonction des possibilités liées au protocole, l'extracteur compte un nombre plus ou moins important de lignes de code. Par exemple l'extracteur dédié au protocole Zigbee compte plus de 500 lignes de codes, là où celui pour le protocole BLE tient dans 160 lignes, sans compter les dictionnaires de constantes.

Actuellement, IoTMap compte trois extracteurs pour les trois protocoles supportés. Cependant, cette approche est possible uniquement si le protocole observé est supporté par Scapy. Dans le cas contraire, il faudrait procéder par nous même à la dissection du paquet bit par bit.

6.3.2 Les bases de conversion

Après l'exécution des extracteurs, IoTMap dispose d'une liste de paquets, découpés en cinq couches. Cette structure est ensuite utilisée par la fonction contenant la base de conversion, pour convertir le paquet au format générique.

La figure 6.5 illustre un exemple d'implémentation de la base de conversion du protocole BLE, présentée en section 4.1.1.3. Dans cet exemple, nous identifions la variable *e* comme la structure d'un paquet, après l'exécution de l'extracteur. La variable *row* correspond au paquet converti dans le format générique. Les blocs de code conditionnels représentent l'implémentation de la table de conversion. Dans cet exemple, nous vérifions si le paquet en question contient l'*opcode* correspondant à la valeur *Read Request* ou *Write Command*. Si l'une ou l'autre des conditions est

3. https://github.com/secdev/scapy

```
# code avant
layer = decrypted_pkt[ZigbeeAppDataPayload]
transmission4['src_endpoint']= layer.src_endpoint if hasattr(layer, 'src_endpoint') else -1
transmission4['dst_endpoint']= layer.dst_endpoint if hasattr(layer, 'dst_endpoint') else -1
transmission4['delivery_mode'] = layer.delivery_mode if hasattr(layer, 'delivery_mode') else -1
transmission4['counter'] = layer.counter if hasattr(layer, 'counter') else -1


transmission4['cluster'] = clusters[layer.cluster] if (hasattr(layer, 'cluster') and
↪  layer.cluster in clusters) else "cluster unknown"
transmission4['profile'] = profiles[layer.profile] if (hasattr(layer, 'profile') and
↪  layer.profile in profiles) else "profile unknown"
transmission4['aps_frametype'] = aps_frametypes[layer.aps_frametype] if (hasattr(layer,
↪  'aps_frametype') and layer.aps_frametype in aps_frametypes) else "aps_frametype unknown"
transmission4['type'] = [transmission4['cluster'], transmission4['profile']]
#code après
```

FIGURE 6.4 – Exemple de code extrait de la fonction *extract_pkt_layer3* de l'extracteur Zigbee.

```
if (e['layer4']['opcode'] == 'Read Request' or \
   e['layer4']['opcode'] == 0x0a or \
   e['layer4']['opcode'] == 'Write Command' or \
   e['layer4']['opcode'] == 0x52):
    row.append(apptype['command'])
    row.append('get_data')
elif (e['layer4']['opcode'] == 'Read Response' or \
      e['layer4']['opcode'] == 0x0b or \
      e['layer4']['opcode'] == 'Handle Value Indication' or \
      e['layer4']['opcode'] == 0x1d):
    row.append(apptype['data'])
    row.append('value')
```

FIGURE 6.5 – Extrait de l'implémentation de la base de conversion pour le protocole BLE.

vraie, alors d'après la base de conversion, ce paquet correspond au type d'application *Commande* du paquet générique. Nous effectuons ainsi cette opération pour toutes les conditions de la base de conversion, et ce, pour tous les protocoles supportés.

6.4 Interception

Le module *Interception* d'IoTMap est responsable de toutes les fonctionnalités liées aux capacités d'écoute. Il s'agit d'une interface, permettant à l'utilisateur de spécifier le matériel d'écoute à utiliser et le protocole à observer, sans devoir préciser le logiciel à utiliser. En effet, c'est dans ce module que nous centralisons les logiciels dédiés à l'interception des communications spécifiques à un protocole. Ainsi, en fonction du matériel disponible et du protocole à observer, ce module détermine quel logiciel doit être utilisé pour réaliser l'interception des communications.

Nous définissons une interface pour instancier chaque logiciel d'interception, spécifique à un protocole, selon un format homogène. Une implémentation de cette interface est ensuite proposée pour chaque logiciel d'interception. En effet, la problématique est que les solutions d'interception existantes sont des logiciels qui s'utilisent en tant qu'outil seul, et non comme une bibliothèque. Ainsi, il est compliqué d'intégrer la solution souhaitée directement dans IoTMap.

```
1  class sixlowpanSniffer(Sniffer):
2    def __init__(self, options):
3      name, self.device, self.nbpkts, self.channel = options
4      if len(self.device) != 1:
5        print(f'[i] This sniffer can handle only one device at the same time. This sniffer will
         ↪  use the device {self.device[0]}')
6      self.device = self.device[0]
7      if name is None:
8        name = '6lowpan'
9      Sniffer.__init__(self, name)
10     self.redirect = io.StringIO()
11     self.outputFile = './sensniff.pcap'
```

FIGURE 6.6 – Exemple d'implémentation du logiciel d'interception *Sensniff* pour le protocole OS4I.

Lorsque le code source de l'outil est disponible, nous pouvons alors comprendre et adapter une interface, qui nous permet d'utiliser les fonctions de l'outil. La figure 6.6 illustre l'implémentation du logiciel d'interception *Sensniff* pour écouter le trafic OS4I. IoTMap intègre également les logiciels BtleJack (pour BLE) et Killerbee (pour Zigbee).

6.5 Modélisation

Le module *Modélisation* d'IoTMap est responsable de la génération des différents graphes qui composent la modélisation. Il offre une interface utilisateur où sont définies toutes les fonctions de génération de graphes, permettant aux utilisateurs de définir plusieurs options en fonction du pattern. Chaque fonction produit un graphe qui est ensuite sauvegardé dans la base de données.

L'implémentation dans IoTMap nécessite l'utilisation de deux langages de programmation. Le premier est Python, il est utilisé pour définir l'interface permettant à l'utilisateur de choisir le graphe à générer et de spécifier les options du pattern, si nécessaire. Le second est Cypher, le langage spécifique de Neo4J, permettant d'effectuer des requêtes dans la base de données, représentée par des graphes. Lorsque le pattern est trop complexe, nous utilisons les deux langages de programmation simultanément : Cypher pour interagir avec la base de données, et Python pour effectuer des calculs plus complexes.

La figure 6.7 illustre l'implémentation du pattern CTRL, présenté en section 4.3.2. Ce pattern est trop complexe pour être implémenté uniquement en utilisant Cypher. C'est pourquoi, nous utilisons Cypher (lignes 5 à 8) pour extraire les informations dont nous avons besoin afin de réaliser le pattern. Puis nous utilisons le langage Python pour effectuer les étapes plus complexes, comme l'identification des objets avec le rôle *controller* (lignes 15 à 17).

Dès que l'utilisateur génère un graphe, il peut consulter le résultat obtenu depuis l'interface web, illustrée par la figure 6.8. Elle est accessible dès l'exécution d'IoTMap et est mise à jour automatiquement. Elle affiche plusieurs informations permettant de manipuler ou d'analyser le graphe obtenu. Cette interface n'est pas développée dans le cadre du projet IoTMap, il s'agit d'un service web mis à disposition par Neo4J dès que la base de données est active. Nous estimons que cette interface est suffisamment complète pour nos besoins actuels.

```python
@classmethod
def CTRL_pattern(cs, tx, delta):
  results = tx.run("""
  match (n: Node{label: 4})-[r1]-(m: Node{label: 4})-[r2]-(d: Node{label: 4})
  where ('source' in n.role or 'controller' in n.role) and ('source' in m.role and 'sink' in
↪  m.role) and ('sink' in d.role or 'controller' in d.role) and n <> d
  with r1.nwksrc as src, r1.nwkdst as ctrl, r2.nwkdst as sink, r1.timestamp as ts1,
↪  r2.timestamp as ts2
  return src, ctrl, sink, ts1, ts2
  """).values()
  delta = delta
  for line in results:
    source, controller, sink, ts1, ts2 = line
    if isinstance(controller, list):
      controller = controller[0]
    for t2 in ts2:
      for t1 in ts1:
        if t2 > t1 and t2 - t1 < delta:
          tx.run("""
          match (n: Node{label: 4})
          where $ctrl in n.nwksrc
          set n.role = ['controller']
          """, ctrl=controller)
  return toreturn
```

FIGURE 6.7 – Exemple d'implémentation du pattern CTRL, en utilisant Cypher et Python.

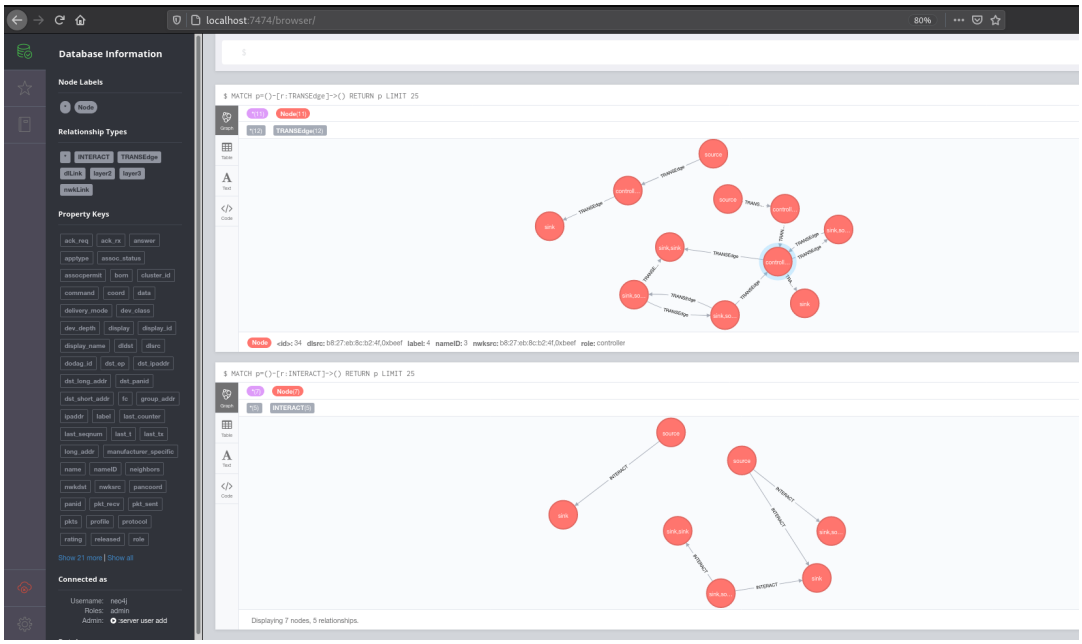

FIGURE 6.8 – Interface web d'IoTMap.

6.6 Base de données

Le module *Base de données* d'IoTMap propose une interface de gestion de la base de données. Il permet de consulter, modifier et remplir la base de données. C'est également par ce module que l'intégration des PCAPs s'effectue. Dès l'exécution de la fonction d'import, le module utilise la classe *GenericPackets* afin d'obtenir une liste de paquets au format générique, à partir des PCAPs passés en paramètre.

```
1  @command
2  def importPcaps(self, pcap: list, protocol: list, output: str, thread: int, nodesFile:
   ↪  str=None):
3
4    print(f"[i] Pcaps: {pcap}\nProtocols: {protocol}\nOutput: {output}\nThread: {thread}")
5    if check_protocol(protocol):
6      try:
7        pcaps_list = unify_pcaps(protocol, pcap)
8        print(f"[i] Pcaps_list: {pcaps_list}\nOutput: {output}\nThread: {thread}")
9        self.dbc.update(pcaps_list, output, thread, nodesFile)
10
11     except FileNotFoundError:
12       print("File not found")
13   else:
14     print(f"{protocol} is not a protocol available")
```

FIGURE 6.9 – Exemple de la fonction d'import de PCAPs pour peupler la base de données.

La figure 6.9 illustre la fonction d'import de PCAPs du module *Base de données*. La ligne 10 du programme fait référence au contrôleur de la base de données. Nous utilisons sa fonction *update* pour mettre à jour la base de données, avec le contenu des PCAPs passés en argument de la fonction, après différentes vérifications. La ligne 8 est un appel à la fonction de conversion des paquets vers le format générique.

6.7 Résumé

Dans ce chapitre nous avons introduit IoTMap, notre plateforme de modélisation par des graphes, basée sur la détection de patterns.

Nous avons présenté de manière générale le fonctionnement et l'utilité d'IoTMap. En effet, ce dernier intègre tous les éléments de notre méthodologie de modélisation. Ce *framework* s'appuie sur des projets existants, afin de capturer les communications. Puis, il utilise la méthodologie de modélisation pour générer les graphes, qui peuvent être visualisés depuis une interface web.

Nous avons également introduit l'architecture détaillée d'IoTMap, qui présente comment les composants du *framework* interagissent entre eux.

Par la suite, nous avons présenté les différents modules implémentés dans IoTMap, tels que le paquet générique, les mécanismes d'interception et de modélisation, ainsi que la base de données.

Une documentation plus détaillée sur le fonctionnement et l'utilisation d'IoTMap est disponible sur le répertoire Github du projet IoTMap [4].

4. https://github.com/AlgoSecure/iotmap/blob/master/doc/started.md

Chapitre 7

Évaluations

Dans ce chapitre, nous présentons notre approche d'évaluation de la modélisation de réseaux IoT hétérogènes. Une première section est dédiée à la présentation de notre environnement de tests. Elle présente une plateforme IoT utilisant trois protocoles IoT simultanément, ainsi que les conditions expérimentales définies pour les évaluations. Dans une seconde section, nous présentons les résultats obtenus par notre modélisation lorsque le cas étudié est idéal afin de valider différents paramètres utilisés dans les patterns. Finalement, une dernière section présente les résultats obtenus lorsque le cas étudié est dégradé selon trois éléments : une base de connaissance incomplète, l'utilisation de chiffrement et une perte de paquets.

7.1 Plateforme d'expérimentations

Dans cette section, nous présentons notre environnement de tests utilisés pour évaluer notre méthodologie de modélisation de réseaux IoT hétérogènes. Dans une première partie, nous présentons les composants matériels et logiciels déployés dans la plateforme d'expérimentations, permettant la construction d'un réseau de protocoles IoT hétérogènes. Ce dernier est composé de trois protocoles différents : Zigbee, BLE et OS4I. Dans une seconde partie, nous présentons l'infrastructure complète de la plateforme d'évaluation. Une troisième partie décrit les applications mono- et multi- protocolaires, déployées et utilisées dans la plateforme de tests. Finalement, une quatrième partie décrit la configuration expérimentale définie afin d'évaluer notre méthodologie de modélisation.

FIGURE 7.1 – Objets physiques utilisés dans la plateforme d'expérimentations.

7.1.1 Composants de la plateforme d'expérimentations

Dans cette section, nous présentons les éléments utilisés pour réaliser notre laboratoire de tests. Nous décrivons les composants matériels (objets et dongles) déployés pour chaque protocole, de même que les composants logiciels utilisés par les objets du réseau. Le tableau 7.1 résume tous les éléments utilisés dans notre plateforme d'expérimentations.

Dispositifs physiques du réseau déployé

ID	Adresse MAC	Adresse Réseau	Protocole	Hardware	Logiciel
d0	0x0	0x0	Zigbee	SmartThings Samsung hub	Propriétaire
d1	0x7b65	0x7B65	Zigbee	SmartThings Samsung motion sensor	Propriétaire
d2	0x3181	0x3181	Zigbee	SmartThings Samsung outlet	Propriétaire
d3	0xbeef b8 :27 :eb :8c :b2 :4f	0xbeef b8 :27 :eb :8c :b2 :4f	Zigbee BLE	Raspberry Pi 3	DeCONZ Open source
d4	00 :12 :4b :00 :12 :04 :cb :03	: :212 :4b00 :1204 :cb03	OS4I	CC2650	Contiki-ng
d5	dc :d0 :17 :9d :1d :5d	dc :d0 :17 :9d :1d :5d	BLE	MicroBit	Open source
d6	00 :12 :4b :00 :0e :0d :82 :57 b8 :27 :eb :36 :1b :9d	: :212 :4b00 :0e0d :8257 b8 :27 :eb :36 :1b :9d	OS4I BLE	Raspberry Pi 3 + CC2650	Contiki-ng Open source
d7	e0 :14 :9e :14 :11 :72	e0 :14 :9e :14 :11 :72	BLE	MicroBit	Open source
d8	00 :12 :4b :00 :16 :65 :27 :07	: :212 :4b00 :1665 :2707	OS4I	CC2650	Contiki-ng
d9	00 :12 :4b :00 :12 :04 :c9 :2d	: :212 :4b00 :1204 :c92d	OS4I	CC2650	Contiki-ng
d10	00 :12 :4b :00 :12 :04 :ce :a4	: :212 :4b00 :1204 :cea4	OS4I	CC2650	Contiki-ng

Dispositifs destinés à l'interception des communications

Protocole	Matériel	Logiciel
Zigbee	RSUSBStick	Killerbee
OS4I	CC2531	Sensniff
BLE	Microbit	BtleJack

TABLEAU 7.1 – Liste des composants (physiques et logiciels) du laboratoire de tests.

La plateforme d'expérimentations est inspirée du scénario expérimental présenté dans la section 3.1. Elle est composée de onze objets, illustrés sur la figure 7.1, et trois protocoles IoT sont utilisés pour permettre aux objets de communiquer entre eux. De plus, plusieurs périphériques sont utilisés pour maximiser l'hétérogénéité de la plateforme. Nous avons choisi d'utiliser, dans un même environnement, des objets commercialisés, dont le firmware est propriétaire et non modifiable, des objets programmables avec de faibles capacités de calculs et des objets avec des capacités de calculs plus importantes.

Réseau Zigbee. La création d'un réseau Zigbee peut s'effectuer par plusieurs approches. D'un côté, nous trouvons plusieurs solutions commerciales, qualifiées également de tout-en-un et *plug-and-play*. Ces derniers proposent des packs contenant le hub, plusieurs objets et les applications qui servent à les contrôler, et ne demandent aucune configuration pour fonctionner. Ces solutions propriétaires laissent peu de place à la modification et la configuration du protocole, de même que les paramètres de sécurité. Les applications dédiées servent uniquement d'interface pour utiliser les objets.

D'un autre côté, il existe des hubs indépendants et collaboratifs, dont le firmware est open source, assurant une certaine transparence dans les fonctionnalités

proposées par l'objet. Ils peuvent parfaitement s'adapter aux solutions commerciales, quand les objets à associer le permettent, ou d'autres objets indépendants. Dans notre plateforme d'évaluation, nous utilisons un hub SmartThings ainsi que plusieurs objets de la même gamme [1], tels que la prise connectée et le capteur de température. A cela s'ajoute un RaspberryPi 3 doté d'une passerelle USB ConBee et formaté avec la distribution RaspBee (une version modifiée de Raspbian compatible avec ConBee et DeCONZ). Nous avons également développé un plugin [2] pour le logiciel DeCONZ, dans le but de déployer notre propre application au sein du réseau Zigbee.

Réseau BLE. Le protocole BLE est largement déployé et de nombreux objets intègrent, par défaut, cette pile protocolaire. Néanmoins, il y a une distinction à faire entre les objets maîtres et les objets secondaires. Les objets secondaires possèdent une faible capacité de puissance et de batterie, et sont dédiés à effectuer une tâche unique.

À l'opposé, l'objet maître est multitâche et utilise plusieurs protocoles en plus de BLE. Tout comme Zigbee, il existe des solutions commerciales proposant des objets qui intègrent nativement le protocole BLE et dont les applications sont déjà installées. Encore une fois, ces solutions *plug-and-play* sont propriétaires et ne favorisent pas la modification de ces dites applications. Même si des packs existent, la force de ce protocole est de pouvoir utiliser les objets secondaires commerciaux avec une grande variété d'objets maîtres, du moment que ces derniers utilisent le protocole BLE.

En effet, l'objet maître peut être une solution commerciale proposée avec les objets secondaires. Cependant, plusieurs alternatives, commerciales ou libres, existent pour permettre à l'utilisateur de choisir ce qu'il souhaite. Il en va de même pour les objets secondaires. Cependant, tout comme pour la pile OS4I, les objets proposant des applications open source ne sont pas largement développés. Il faut, encore une fois, utiliser des périphériques et logiciels individuels pour obtenir la solution qui nous convient.

Dans la plateforme d'expérimentations, nous utilisons deux BBC MICRO :BIT, flashés avec différents programmes open-source et développés par nos soins ; ainsi que les deux RaspberryPi 3, déjà présentés, configurés pour utiliser le protocole BLE en plus du protocole déjà attribué par leur sous-réseau respectif.

Réseau OS4I. La pile de protocoles, que nous appelons OS4I, repose sur une encapsulation de protocoles open source. Peu d'objets sont développés et commercialisés avec OS4I disponible initialement. Cependant, il existe, un système d'exploitation (OS) simplifié open source, appelé Contiki-NG [3], qui permet d'utiliser cette pile protocolaire pour tout objet compatible et pouvant être flashé. Pour connaître les objets compatibles avec cet OS, une page est mise à disposition. Il est également possible de contribuer au projet en proposant des configurations afin d'augmenter le nombre de matériels compatibles.

Dans notre plateforme d'évaluation, nous utilisons les éléments suivants :

— Quatre SensorTag CC2650 de Texas Instrument flashés avec le programme Contiki-NG ;

1. https://www.samsung.com/us/smart-home/smartthings/kits/samsung-smartthings-home-monitoring-kit-f-mon-kit-1/
2. https://github.com/JnTournier/basic-aps-plugin
3. https://github.com/contiki-ng/contiki-ng

— Un RaspberryPi 3 configuré comme un 6LoWPAN Border Router (6lbr) avec un SensorTag CC2650 en tant que slip radio.

Nous avons également développé nos propres applications [4], pour les capteurs, ainsi que pour les serveurs.

Capacité d'interception À notre connaissance, il n'existe pas d'antenne capable d'intercepter des communications ZigBee, BLE et OS4I simultanément. C'est pourquoi, nous devons utiliser plusieurs logiciels et matériels pour intercepter les communications de chaque réseau présenté. Comme pour les objets, il existe plusieurs alternatives, matérielles et logicielles, pour intercepter les communications d'un protocole.

Pour les réseaux Zigbee, plusieurs approches conviennent pour intercepter les ondes radio. La première approche consiste à utiliser GNU radio et un émetteur-récepteur IEEE802.15.4 O-QPSK [5]. D'autres consistent à utiliser du matériel spécifique pouvant utiliser le protocole Zigbee. (une liste non exhaustive est disponible à cette adresse. [6]). Cependant, il convient de modifier le logiciel embarqué dans ces dispositifs de façon à ce qu'ils se comportent comme un périphérique d'interception, et non plus comme un objet du réseau.

Nous avons opté pour la seconde approche en utilisant le matériel Atmel RZ Raven USB stick, flashé avec le logiciel Killerbee. Il s'agit d'une configuration efficace et facile à déployer, ne nécessitant aucune modification. Cependant, le périphérique Atmel RZ Raven USB stick n'est actuellement plus commercialisé (en date de novembre 2020).

Pour les réseaux OS4I, nous avons trouvé peu de solutions logicielles fonctionnelles permettant la capture de paquets, si ce n'est le programme Sensniff [7]. Cette solution est compatible avec plusieurs matériels différents, notamment de la gamme CC25xx et CC26xx de Texas Instrument, facilitant ainsi son utilisation. Nous avons décidé d'utiliser l'émetteur-récepteur TI CC2530 pour intercepter et capturer les communications de réseau OS4I.

Pour les réseaux BLE, il existe pléthore de solutions pour intercepter les communications. Nous pouvons citer des applications mobiles compatibles avec de nombreux smartphones. Nous écartons cette approche de notre configuration, car l'utilisation d'un smartphone, au delà du prix plus conséquent qu'un matériel spécifique, requiert des interactions humaines pour être parfaitement opérationnelle.

Les autres approches utilisent des périphériques dédiés utilisant des logiciels adaptés. Nous distinguons des solutions propriétaires, comme celles proposées par Nordic Semiconducteur, dont l'utilisation n'est pas intuitive et ne correspond pas à nos besoins. Des alternatives existent et sont open source, fonctionnant sur plusieurs matériels compatibles, facilement intégrables dans d'autres projets.

Dans notre plateforme d'expérimentations, nous utilisons des cartes de programmation BBC MICRO :BIT flashées et contrôlées par le programme Btlejack pour intercepter et manipuler les communications BLE. Il y a, cependant, des particularités lors de l'interception des communications BLE. En effet, il faut au minimum un dispositif d'interception par communication observée. C'est une spécificité du protocole BLE ; chaque communication est identifiée par un *access address*, qu'il faut suivre pour intercepter et analyser les messages de cette communication.

4. https://github.com/dynamid/youpi-6lowpan
5. https://github.com/bastibl/gr-ieee802-15-4
6. https://github.com/riverloopsec/killerbee
7. https://github.com/g-oikonomou/sensniff

7.1.2 Infrastructure du laboratoire de tests

Cette section présente l'infrastructure de la plateforme d'expérimentations. Elle est illustrée sur la figure 7.2, où les trois réseaux Zigbee, 0S4I et BLE sont déployés. Ils sont identifiés avec un label au dessus de chaque arc, qui représente les communications, entre les objets du réseau. Pour le réseau Zigbee, les arcs sont étiqueté *ZigBee*, ceux étiquetés par *6lowpan* identifient le réseau 0S4I et enfin les communications ayant pour label *BLE* représentent le réseau BLE.

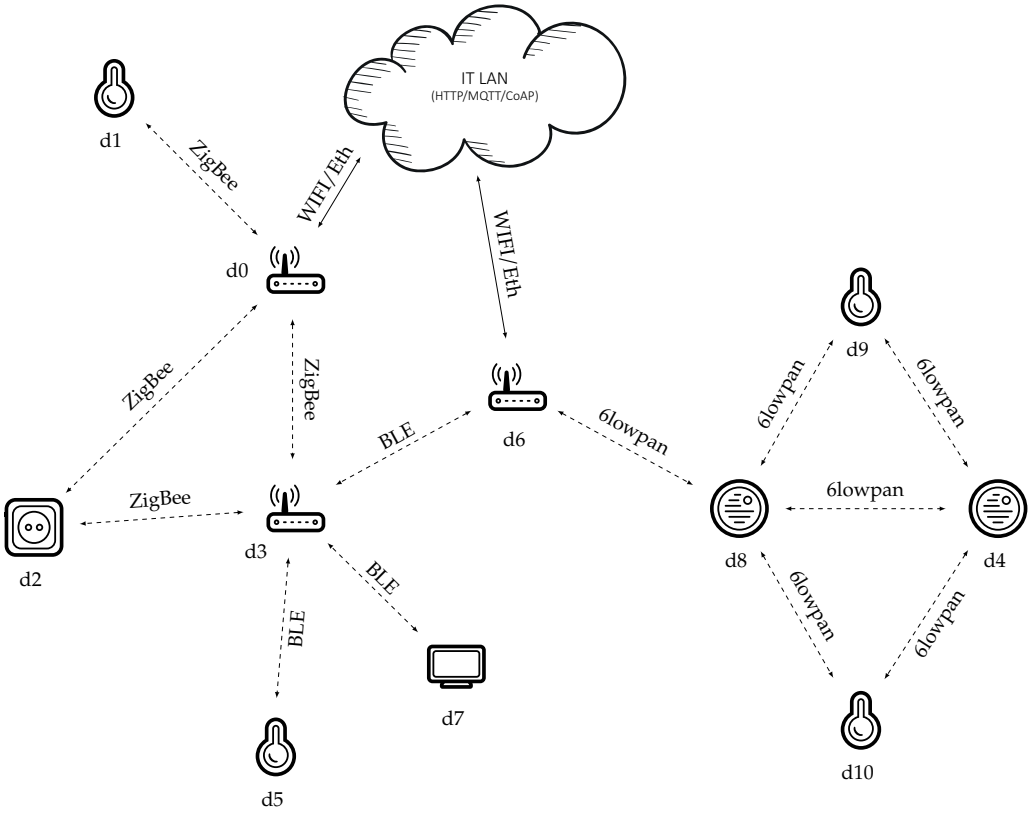

FIGURE 7.2 – Infrastructure de la plateforme d'évaluation.

Sur la partie gauche de la figure 7.2, nous représentons le réseau Zigbee, composé de quatre objets : un capteur de température, deux routeurs et une prise. Nous avons placé les objets de telle sorte que le capteur de température ne soit à portée que du routeur connecté à Internet (hub). Les deux routeurs et la prise peuvent communiquer ensemble de façon similaire aux objets d'un réseau maillé. Finalement, le hub est connecté par Ethernet à Internet, pour utiliser l'application mobile depuis l'extérieur du réseau afin d'interagir directement avec le capteur de température et la prise, par l'intermédiaire du hub.

Sur la partie droite de la figure 7.2, nous représentons le réseau OS4I, composé de cinq objets : un capteur de température, un capteur de luminosité, deux serveurs, dont un dispose d'une LED et un routeur. Dans un environnement restreint (environ 50m²), il est compliqué d'espacer suffisamment les objets pour obtenir la topologie souhaitée. Chaque objet est à portée radio des autres objets, pouvant ainsi communiquer sans saut. Cependant, pour réaliser les schémas de communications souhaitées, nous avons introduit une liste noire d'adresses dans les applications de chaque objet afin d'obtenir le comportement suivant :

— le capteur de luminosité et le capteur de température communiquent avec le serveur (sans LED) ;

— les deux serveurs communiquent ensemble ;

— le routeur communique uniquement avec le serveur (sans LED).

Le routeur (6LBR) est également connecté à Internet par une connexion WiFi.

Finalement, le réseau BLE n'est pas aussi regroupé que les deux autres réseaux. En effet, le fonctionnement de BLE ne favorise pas l'élaboration de réseaux *classiques* avec la présence d'un hub connecté à Internet, qui centralise les informations en provenance ou à destination d'Internet. Au contraire, dans les réseaux BLE, même si un nœud maître peut centraliser de nombreuses connexions avec des objets secondaires, les interactions entre les objets secondaires ne sont pas natives, et doivent passer par le développement d'applications spécifiques.

Dans notre plateforme, nous déployons un capteur de température connecté avec le routeur Zigbee, un afficheur de température, lui aussi, connecté avec le routeur ZigBee. Enfin nous connectons le routeur Zigbee et le routeur OS4I à travers une connexion BLE.

7.1.3 Applications du laboratoire de tests

À travers cette section, nous présentons les différentes applications que nous déployons dans notre plateforme d'expérimentations. Certaines applications sont mono-protocolaires, c'est à dire, qu'elles n'interviennent que dans le réseau concerné. À l'inverse, d'autres sont multi-protocolaires, entraînant ainsi l'utilisation de deux ou trois protocoles pour être exécutées.

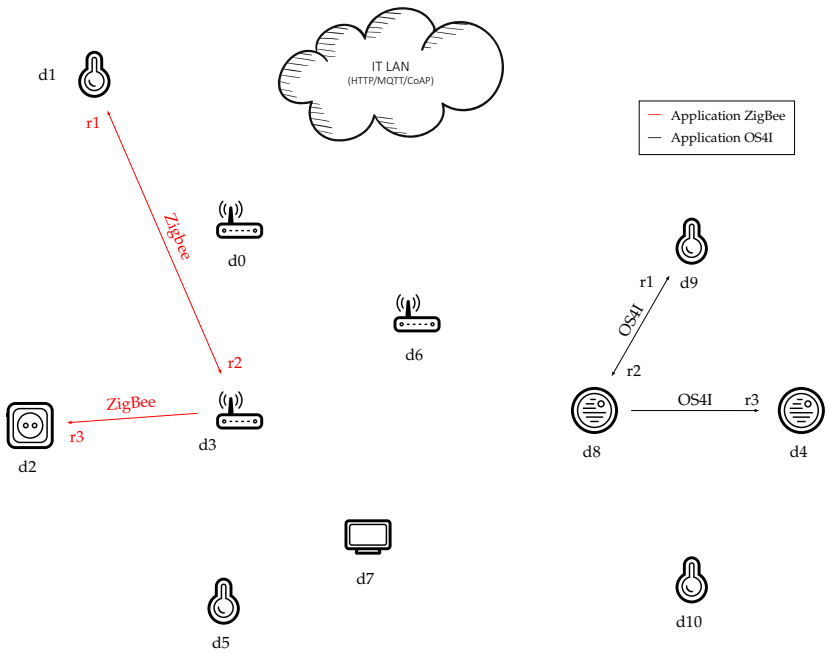

FIGURE 7.3 – Déploiement des applications mono-protocolaires dans la plateforme d'expérimentations.

7.1.3.1 Applications mono-protocolaires

La figure 7.3 illustre les deux applications mono-protocolaires déployées dans la plateforme d'expérimentations.

La première application, celle représentée par —, utilise uniquement le protocole Zigbee. L'application est initialisée par le routeur qui envoie une requête (r1) au capteur de température afin d'obtenir cette donnée. Le capteur de température répond au routeur par un message (r2) contenant la température observée. Dès réception de la donnée, le routeur l'analyse et la compare aux données précédentes, si cette donnée est inférieure au seuil de température fixé dans son logiciel et que l'actionneur (ici la prise) n'est pas activé, alors une commande d'activation (r3) est envoyée à la prise. Le tableau 7.2 résume les conditions requises pour envoyer une commande à la prise.

Température observée	État de la prise	Commande à envoyer
< au seuil	Éteinte	On
> au seuil	Éteinte	Aucune commande
< au seuil	Allumée	Aucune commande
> au seuil	Allumée	Off

TABLEAU 7.2 – Conditions d'activation ou d'extinction de la prise en fonction de la température observée.

La seconde application, celle représenté par —, utilise uniquement des objets communiquant avec le protocole OS4I. Le serveur initie l'application par l'envoie d'une requête CoAP GET (r1) afin d'obtenir la dernière valeur du capteur de luminosité. Le capteur répond au serveur par un paquet CoAP de type CON ACK (r2) contenant la valeur observée. De manière identique aux données présentées dans le tableau 7.2, le serveur analyse la donnée obtenue et envoie, ou non, une commande à l'autre serveur disposant d'une LED. Ainsi, cette LED s'allume lorsque la luminosité observée par le capteur est inférieure au seuil défini dans l'application, et s'éteint lorsqu'une valeur supérieure au seuil est observée.

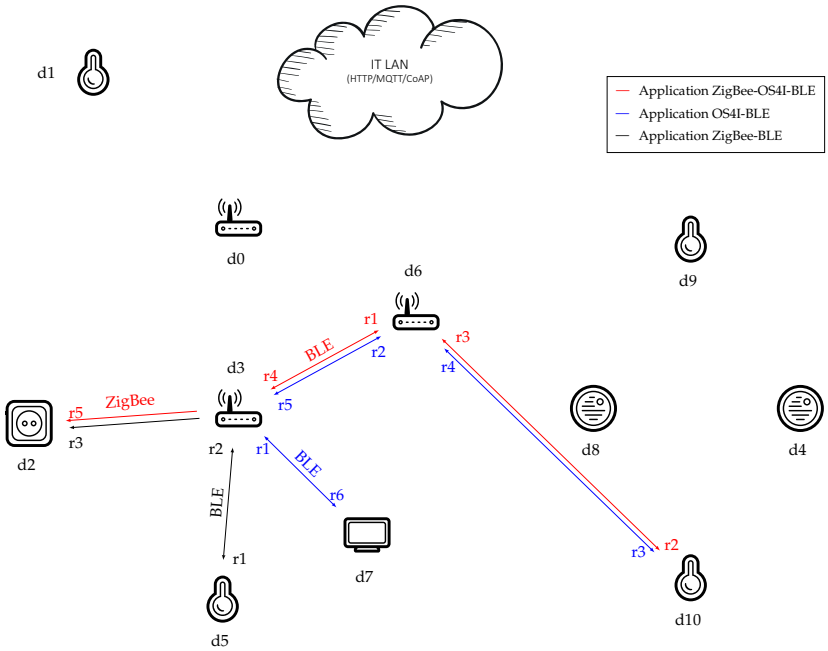

FIGURE 7.4 – Déploiement des applications multi-protocolaires dans la plateforme d'expérimentations.

7.1.3.2 Applications multi-protocolaires

Cette section présente les trois applications multi-protocolaires déployées dans la plateforme d'évaluation. Toutes les applications sont illustrées sur la figure 7.4 et utilisent leur propre code couleur pour les différencier.

La première application, celle représentée par —, entraîne l'utilisation des trois protocoles de communications, OS4I, Zigbee et BLE. Le routeur Zigbee initie l'application par l'envoie d'une requête (r1) au routeur OS4I via le protocole BLE. Dès la réception de la requête, le routeur OS4I adapte la requête provenant du réseau BLE pour l'envoyer (r2) au capteur de température du réseau OS4I. Puis, il répond (r3) au routeur OS4I avec la température observée, ce dernier convertit la réponse, provenant du réseau OS4I, pour envoyer la réponse (r4) au routeur Zigbee via BLE. Enfin, le routeur Zigbee analyse la donnée obtenue, puis, en utilisant la logique présentée dans le tableau 7.2, décide d'envoyer (r5) ou non, une commande afin d'agir sur la prise, par une requête au format Zigbee.

La seconde application, celle représentée par —, demande l'utilisation des deux protocoles BLE et OS4I pour être déroulée. L'application commence par l'envoie d'une requête (r1) de l'afficheur BLE au routeur Zigbee, via le protocole BLE. Dès réception de la requête, le routeur Zigbee envoie une requête (r2) au routeur OS4I à destination du capteur de température en utilisant le protocole BLE. Le routeur OS4I convertit la requête provenant du routeur Zigbee, via le protocole BLE, pour l'envoyer (r3) au capteur de température. Il modifie, ensuite, la réponse obtenue (r4) afin de la renvoyer (r5) au routeur Zigbee. Aucune analyse n'est faite lorsque le routeur Zigbee reçoit cette température : il renvoie (r6) simplement la valeur à l'afficheur qui l'affichera sur son écran.

La dernière application, celle représentée par —, nécessite, elle aussi, l'utilisation de deux protocoles, BLE et Zigbee. Le routeur Zigbee initie l'application en envoyant une requête (r1) au capteur de température via le protocole BLE. Ce dernier lui répond (r2) avec la valeur obtenue. Le routeur Zigbee analyse ensuite cette donnée et envoie (r3) ou non, une commande à la prise, en fonction des conditions détaillées dans le tableau 7.2, via le protocole Zigbee.

7.1.4 Configurations expérimentales

Nous présentons dans cette section le contexte dans lequel nous réalisons les expérimentations. La plateforme utilisée est celle présentée en section 7.1.3.

Paramètres des applications. Les applications déployées dans la plateforme d'évaluation sont présentées en section 7.1.3. Nous présentons dans cette section les paramètres configurés dans les applications.

Pour chaque application, mono- ou multi- protocolaire, nous définissons une période d'activation automatique toutes les 30 secondes, auxquelles s'ajoute ou se soustrait un certain nombre de secondes, généré aléatoirement et défini entre 1 et 10 secondes. Cette configuration permet, notamment, de réduire le chevauchement des communications provenant de différentes applications.

Captures des communications. Nos expérimentations détaillées dans les sections suivantes se basent sur les mêmes captures de trafic. Nous réalisons ces captures en deux temps et de façon distincte, que ce soit lorsque l'environnement utilise des communications en clair, que des communications chiffrées.

Chaque observation du réseau a une durée de 25 minutes. Ainsi, lorsque les captures de trafic sont réunies, le temps total de trafic observé et capturé est de 50 minutes. Cette méthodologie de captures permet également de faire un focus sur de plus petites traces de communications ou sur une capture de trafic plus importante.

Lorsque les communications sont en clair, les captures réalisées pour BLE font en moyenne 60 paquets, pour chaque communication observée. Les captures du protocoles OS4I contiennent environ 300 paquets. Enfin, plus de 5000 paquets sont contenus dans les captures Zigbee. Lorsque les communications sont chiffrées, nous observons une augmentation du nombre de paquets par capture. Nous comptons 100 paquets en moyenne dans les captures BLE. Les captures du protocole OS4I passent à 1500 paquets. Enfin, les captures du protocoles Zigbee disposent de 6500 paquets.

Nous faisons le choix de reproduire un environnement de tests réaliste, avec peu de trafic et peu d'objets, tel qu'il pourrait être déployé pour un usage personnel, ou professionnel, dans un but de surveillance domotique. Les principales conséquences de ces paramètres sont la faible surcharge du réseau et le faible taux de chevauchement de communications.

Dans ces évaluations, nous ne souhaitons pas mettre en évidence la capacité de notre méthodologie à identifier des applications lorsque le trafic observé est dense et dont de nombreuses communications se mélangent, rendant difficile leur détection.

Nos évaluations se concentrent sur l'importance d'effectuer l'analyse et l'observation des réseaux IoT hétérogènes, en ciblant les protocoles simultanément et non pas indépendamment.

7.2 Évaluation dans le cas idéal

Nous présentons dans cette section l'évaluation de notre méthodologie de modélisation lorsque le cas observé est idéal, présenté en section 4. Nous proposons dans un premier temps une validation du paramètre utilisé dans le pattern CTRL, présenté en section 4.3.2. Puis, dans un second temps, nous présentons les résultats obtenus par notre méthodologie de modélisation, en fonction du trafic observé dans notre plateforme d'expérimentations.

7.2.1 Calibration du pattern CTRL

Dans cette section, nous validons l'utilisation du paramètre δ, utilisé dans le pattern CTRL, comme valeur de temps de routage unique pour tous les protocoles, au lieu d'un paramètre spécifique pour chaque protocole observé. Pour cela, nous analysons le temps de routage appliqué pour chaque objet, qui dispose de capacité de routage. Nous évaluons ensuite l'efficacité du pattern CRTL à l'aide de deux mesures qui sont la précision et le rappel.

7.2.1.1 Évaluation du temps de routage applicatif

Nous considérons qu'un objet est un routeur applicatif, lorsque ce dernier est impliqué dans une schéma d'application et sert uniquement de routeur ou de passerelle dans l'application. Selon notre méthodologie, seuls les objets portant le rôle *controller* sont des routeurs applicatifs. Ils sont déterminés dans la génération du graphe transport par le pattern CTRL, présenté en section 4.3.2.

L'objectif de cette évaluation est de montrer la pertinence d'avoir une variable globale pour caractériser le temps nécessaire à un routeur applicatif de recevoir une

donnée, l'analyser et envoyer ou transférer un message ; par rapport à l'utilisation d'une valeur de temps de routage spécifique pour chaque protocole étudié.

Protocoles	Dispositif ciblé (ID)	Schéma de communication
ZigBee	d0	d1 → d0 → d3 d3 → d0 → d1
ZigBee	d3	d0 → d3 → d2 d2 → d3 → d0 d1 → d3 → d2
BLE	d3	d7 → d3 → d6 d6 → d3 → d7
OS4I	d6	d8 → d6 → d9 d9 → d6 → d8 d8 → d6 → d4
OS4I - BLE	d6	d3 → d6 → d10 d10 → d6 → d3
ZigBee - BLE	d3	d6 → d3 → d2 d5 → d3 → d2

TABLEAU 7.3 – Configuration de la plateforme d'expérimentations pour effectuer l'évaluation du temps de routage des dispositifs en fonction du protocole observé.

Nous observons trois principaux dispositifs, impliqués dans des schémas de communications soit mono-protocolaires, soit multi-protocolaires. L'objet d0 est utilisé pour router des communications Zigbee uniquement. Les deux objets restants, avec les identifiants d3 et d6, sont impliqués dans des schémas de communications, utilisant un ou deux protocoles. L'objet d3 est impliqué dans des communications qui utilisent soit uniquement le protocole Zigbee, soit exclusivement le protocole BLE, soit les deux. De même pour l'objet d6, ce dernier est impliqué dans des communications qui utilisent le protocole OS4I exclusivement, ou alors dans des communications qui mélangent les protocoles OS4I et BLE. Tous les schémas d'applications évalués sont également présentés dans le tableau 7.3.

Pour évaluer le temps de routage, nous comparons plusieurs valeurs entre les différents dispositifs et les protocoles observés. Notamment, nous allons relever le temps minimal (Min), le temps maximal (Max), le temps moyen (Moyenne) ainsi que l'écart type du temps de routage effectué. Notre évaluation se base sur un panel de 40 schémas de communications pour chaque objet observé.

Afin d'obtenir ces schémas de communications par dispositif et protocole observé, nous utilisons plusieurs sources. La première est la capture de trafic de la plateforme telle que présentée en section 7.1.3, avec les mêmes applications et objets. Cependant, pour certains schémas, le trafic n'est pas suffisant. C'est pourquoi nous complétons l'échantillon par des captures de trafic contenant uniquement le schéma de communications ciblé.

Les résultats du temps de routage sont présentés dans le tableau 7.4. Nous observons que le temps de routage minimal est similaire peu importe le protocole observé

Protocole	Dispositif ciblé (ID)	Min (s)	Max (s)	Moyenne (s)	Écart type (s)
Zigbee	d0	0.051	1.277	0.247	0.399
Zigbee	d3	0.046	1.059	0.950	0.300
BLE	d3	0.003	0.480	0.051	0.097
OS4I	d6	0.010	0.382	0.066	0.075
OS4I - BLE	d6	0.013	0.067	0.040	0.015
Zigbee - BLE	d3	0.029	1.980	0.961	0.627

TABLEAU 7.4 – Résultats du temps de routage de plusieurs routeurs en fonction du protocole observé dans un environnement non chiffré : (1) routage de communications spécifiques à un protocole et (2) routage de communications utilisant plusieurs protocoles.

ou le dispositif ciblé. Cependant, nous observons une nette différence des temps de routages maximaux entre les dispositifs utilisant le protocole ZigBee et les autres. En effet, les temps maximaux relevés sur les dispositifs aux ID d0 et d3 sont supérieurs à 1 seconde, soit deux fois plus que ceux observés lorsque le protocole Zigbee n'est pas utilisé.

Nous observons également que le temps moyen est supérieur pour tous les dispositifs utilisant le protocole Zigbee. Comme le temps de routage des autres dispositifs est négligeable par rapport à celui observé avec le protocole Zigbee, la valeur de la variable globale qui définit le temps de routage générique dans les réseaux hétérogènes se base sur les valeurs du temps de routage des dispositifs utilisant le protocole Zigbee.

De plus, l'écart type relevé sur ces temps d'observation permet de raffiner l'ensemble des valeurs possibles de notre variable globale. L'écart type des temps de routage des dispositifs qui utilisent le protocole Zigbee varie de 0.3s à 0.6s. Ainsi, pour définir un ensemble de valeurs acceptables, nous choisissons l'ensemble suivant :

$$\delta \in [\bar{t} - \sigma; \bar{t} + \sigma]$$

Où δ est la variable utilisée dans les patterns CTRL et ASP, $\bar{t}$ est la moyenne des temps de routage observés pour le dispositif d3 sur le protocole Zigbee uniquement, et σ son écart type.

D'après les observations du temps de routage, pour chaque dispositif et pour chaque protocole observé, nous concluons qu'il n'est pas nécessaire d'avoir une valeur dédiée pour chaque protocole, afin de déterminer le temps de routage nécessaire. Ainsi, notre proposition d'une variable globale utilisée, notamment dans les patterns CTRL et ASP, présentés respectivement en section 4.3.2 pour le pattern CTRL et en section 4.4.1 pour le pattern ASP, est tout à fait appropriée et acceptable.

Cependant, pour que cette variable donne des résultats acceptables, sa valeur doit se rapprocher au maximum des temps de routage observés lorsque le protocole Zigbee est utilisé. En effet, les temps de routage observés lorsque le protocole Zigbee est utilisé sont les plus élevés que les communications soient en clair ou chiffrées.

7.2.1.2 Évaluation du pattern CTRL

Dans cette section, nous évaluons la pertinence du modèle CTRL(5). Ce pattern détecte si un nœud, ayant le rôle de *source-sink*, est considéré comme un *controller*. Nous utilisons les méthodologies de précision et de rappel pour mesurer la pertinence de la détection du pattern. La précision permet d'évaluer le nombre d'éléments pertinents obtenus parmi l'ensemble proposé. Le rappel évalue le nombre d'éléments pertinents obtenus parmi l'ensemble des éléments pertinents. Nous définissons trois paramètres, tels que VP pour vrai positif, FP pour faux positif et FN pour faux négatif, définis comme suit :

VP : les nœuds détectés comme *controller* où les deux autres nœuds impliqués dans la communication sont corrélés.

FP : les nœuds détectés comme *controller* lorsque les deux autres nœuds impliqués dans la communication ne sont pas corrélés.

FN : les nœuds non détectés en tant que *controller* bien que les deux autres nœuds impliqués dans la communication soient corrélés.

Sur la base de ces paramètres, nous utilisons deux mesures, la précision(7.1) et le rappel(7.2), définies comme suit :

$$Précision = \frac{VP}{VP + FP} \qquad (7.1) \qquad\qquad Rappel = \frac{VP}{VP + FN} \qquad (7.2)$$

La valeur de vérité est représentée par un graphe défini en fonction de la connaissance des applications déployées dans la plateforme d'expérimentations. Ce graphe est illustré dans la figure 3.8. Chaque nœud et chaque arc de ce graphe correspondent à la valeur correcte et attendue.

La figure 7.5 démontre qu'IoTMap est très pertinent lorsque la valeur de δ est petite, voire très petit (< 0.25s), avec une précision de 100%. Nous observons une légère diminution de la précision dès que la valeur de δ dépasse 0.25s, jusqu'à atteindre un pallier à 90% avec une valeur de δ allant de 0.75s à 2s. Nous pouvons imaginer, d'après la formule de précision, que cette dernière continue de baisser au fur et à mesure que la valeur de δ augmente. En effet, plus la valeur de δ augmente, plus le nombre de faux positifs augmente.

Comme attendu, la courbe de rappel croît rapidement pour atteindre quasiment 100% avec une valeur de δ proche de 0.25s. La courbe de rappel atteint les 100% lorsque la valeur de δ est égale à 0.55s, puis la courbe se maintient à 100% avec l'augmentation de la valeur de δ. Ce résultat s'explique par le temps court où les faux négatifs sont trouvés. Cependant, avec une valeur de δ proche de 0.6s, tous les éléments faux négatifs sont remplacés par des éléments vrais positifs.

Nous observons que les courbes de précision et rappel ne sont pas lisses, sur la figure 7.5, ainsi que sur les autres. Nous constatons une évolution par marche. Ce comportement s'explique, car notre évaluation se base sur notre plateforme d'expérimentations, contenant un nombre limité d'objets. Aussi, le nombre de communications observées reste relativement peu élevé, bien que suffisant pour évaluer notre méthodologie. Ainsi, les valeurs de vrais et faux positifs sont petits et la détection d'un nouvel élément crée cette marche.

7.2.2 Éléments détectés en fonction du nombre de protocoles observés

Nous proposons une analyse sur les patterns détectés, en fonction du nombre de protocoles observés simultanément. Nous concentrons l'évaluation sur les patterns

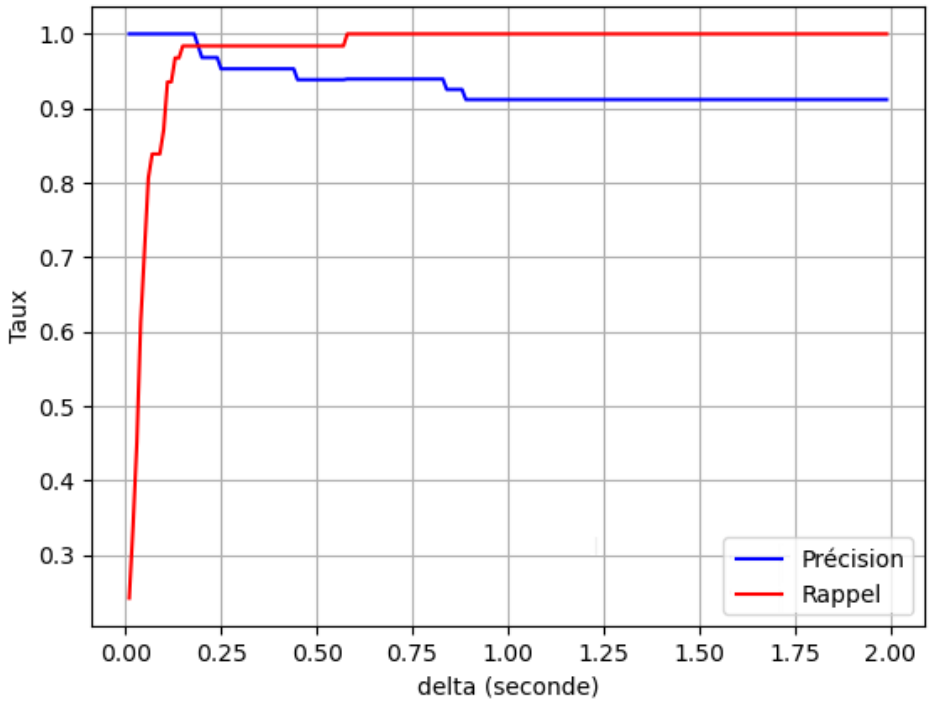

FIGURE 7.5 – Taux de nœuds détectés comme *controller* parmi ceux ayant le rôle *source-sink*.

utilisés pour construire le graphe de transport G_{TRANS} (OWT, PBC4 et CTRL5) et le graphe application G_{APP} (ASP6), présentés en section 4.

Pour les patterns OWT, PBC(4) et CTRL(5), nous comparons le nombre de nœuds correctement marqués (CT) et ceux qui sont incorrectement marqués (WT) parmi les quatre valeurs possibles : *source*, *sink*, *source-sink* et *controller*. Nous évaluons ensuite le modèle ASP 6 à l'aide de trois paramètres :

VP : Les vrais positifs sont des arcs créés entre deux nœuds, existant dans notre valeur de vérité

FP : Les faux positifs sont des arcs créés entre deux nœuds, n'existant pas dans notre valeur de vérité

FN : Les faux négatifs sont des arcs non créés entre deux nœuds, mais existant dans notre valeur de vérité

Trois protocoles sont déployés dans notre plateforme IoT, nous réalisons donc cinq expérimentations pour couvrir l'ensemble des combinaisons de protocoles possibles. Nous utilisons le séparateur '-' pour identifier les protocoles observés simultanément alors que le symbole '∪' est utilisé pour symboliser que les protocoles sont observés individuellement.

Nous réalisons cette évaluation sur deux captures de trafic de 25 minutes, que les communications soient chiffrées ou en clair. De plus, afin de générer les graphes, chaque expérimentation est basée sur les mêmes données de communications interceptées et nous utilisons la même valeur pour chaque paramètre utilisé. Nous définissons également un graphe en tant que valeur de vérité (ligne *Valeurs attendues*) correspondant aux applications déployées dans le réseau et illustré, à la fois par la

figure 3.8 pour le graphe G_{TRANS} et par la figure 3.9 pour le graphe G_{APP}. Nous comparons ensuite ce graphe à chacun de ceux produits durant l'expérimentation. Les résultats sont présentés dans le tableau 7.5.

Protocole(s) \ Patterns	PCAP 1					PCAP 2				
	OWT/PBC/CTRL		*ASP*			*OWT/PBC/CTRL*		*ASP*		
	CT	WT	VP	FP	FN	CT	WT	VP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	10	1	2	2	3	9	2	2	2	2
ZigBee-BTLE ∪ OS4I	10	1	2	2	3	9	2	2	2	2
ZigBee-OS4I ∪ BTLE	10	1	2	2	3	9	2	2	1	2
OS4I-BTLE ∪ ZigBee	11	0	2	0	3	10	0	3	1	1
ZigBee-OS4I-BTLE	11	0	4	0	1	10	0	3	2	1
Valeurs attendues	11	0	5	0	0	11	0	4	0	0

TABLEAU 7.5 – Détection de patterns en fonction du nombre de protocoles observés simultanément sur des communications en clair.

Nous observons qu'IoTMap détecte à tort le rôle d'un nœud dans presque toutes les expérimentations, où seulement un ou deux protocoles sont observés simultanément. Ce résultat montre que les nœuds impliqués dans plusieurs sous-réseaux, utilisant des protocoles différents, sont difficiles à classer lorsque des informations sont manquantes.

En revanche, comme attendu, lorsque tous les protocoles sont observés simultanément, IoTMap identifie correctement le rôle de chaque nœud, même ceux qui sont impliqués dans de nombreux sous-réseaux. Nous constatons le même résultat lorsque nous nous centrons sur la détection des patterns pour le graphe G_{APP}. Lorsque les protocoles sont observés indépendamment ou par paire, IoTMap n'identifie que les interactions spécifiques à chaque protocole.

Nous nous attendons, également, à des faux positifs, car IoTMap utilise trois paramètres pour générer la modélisation complète. En fonction de la valeur de chaque paramètre, les résultats donnés peuvent être plus ou moins pertinents, comprenant peu ou beaucoup de faux positifs. De fait, comme IoTMap s'appuie sur une analyse temporelle pour identifier les patterns CTRL et ASP, deux communications avec des timestamps très proches (par exemple avec une différence de 0,001s) peuvent correspondre à un pattern même si ce n'est pas prévu.

7.3 Évaluation dans le cas dégradé

Dans cette section, nous présentons l'évaluation de notre méthodologie de modélisation lorsque l'environnement observé a subi différentes dégradations. Nous proposons trois dégradations différentes, présentées en section 5. La première concerne l'utilisation d'une base de conversion incomplète. La seconde est l'utilisation du chiffrement dans les communications entre les objets. La dernière met en évidence une observation partielle de l'environnement, que ce soit lié à une perte de paquets ou à une visibilité incomplète du réseau.

7.3.1 Dégradation par l'utilisation d'une base incomplète

Nous proposons dans cette section d'évaluer la modélisation d'un environnement lorsque l'observateur ne dispose pas de la base de conversion complète pour chaque protocole. Nous entendons par base de conversion incomplète, l'incapacité d'identifier le type d'application associé à une communication observée. Le principal pattern impacté par cette dégradation est le pattern PBC. Ce dernier permet

d'attribuer un rôle et définit l'orientation du flux de données lorsque deux objets sont impliqués dans un schéma de communications bidirectionnelles.

Nous proposons ainsi d'évaluer la modification de ce pattern, présenté en section 5.1.2.4, qui repose sur l'utilisation d'un paramètre θ permettant d'identifier quel objet initie une communication et celui qui y répond.

Afin d'évaluer ce paramètre, nous utilisons les mesures de précision et de rappel, présentées respectivement par les équations 7.1 et 7.2. Pour utiliser ces mesures, nous définissons les paramètres suivants :

VP : Les nœuds identifiés comme *source* et *sink* sont corrélés dans la communication bidirectionnelle.

FP : Les nœuds identifiés comme *source* et *sink* ne sont pas corrélés dans la communication bidirectionnelle.

FN : Les nœuds ne sont pas identifiés comme *source* et *sink* alors qu'ils sont corrélés dans la communication bidirectionnelle.

Nous considérons que deux nœuds sont corrélés lorsque le premier paquet est une requête et que le message en retour est la réponse à la requête. Ainsi, nous estimons que les nœuds ne sont pas corrélés lorsque le premier paquet observé est une réponse, ou lorsque la réponse observée ne fait pas référence à la requête demandée.

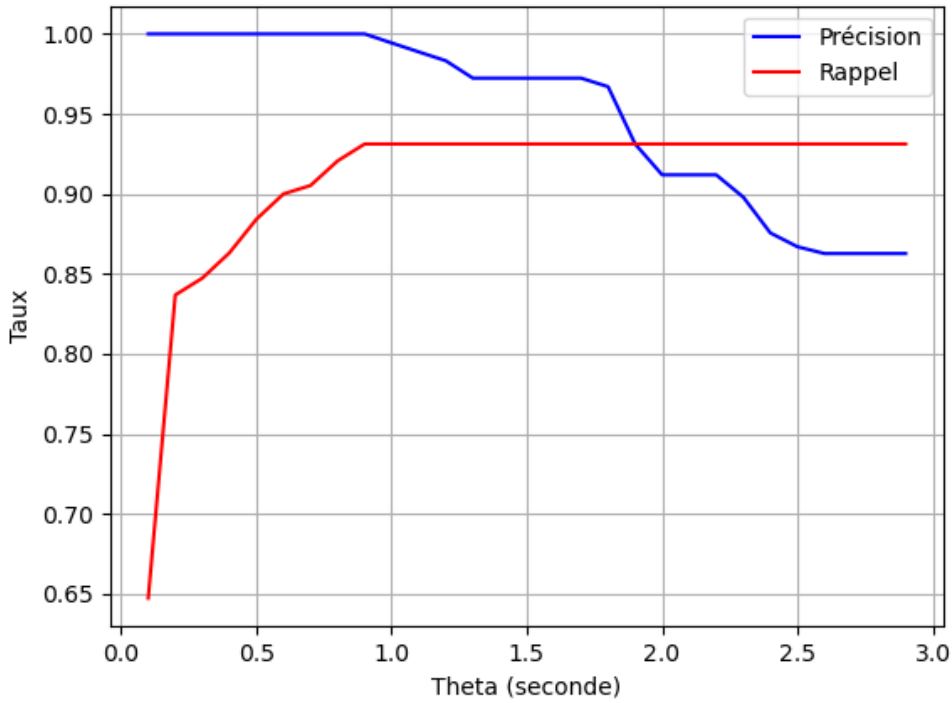

FIGURE 7.6 – Évaluation du pattern PBC'.

La figure 7.6 illustre l'efficacité du pattern PBC', présenté en section 5.1.2.4. Nous observons qu'IoTMap obtient des résultats pertinents, avec une précision parfaite (100%) et une courbe de rappel maximale sur l'ensemble des valeurs observées, lorsque la valeur de θ avoisine 0,9 seconde.

La précision est maximale lorsque la valeur de θ est inférieure à 0.9 seconde, puis diminue progressivement lorsque la valeur du paramètre θ augmente. Ainsi, nous

constatons que le nombre de faux positifs augmente fortement dès lors que la valeur de θ dépasse 1 seconde.

Parallèlement, la courbe de rappel croît rapidement jusqu'à atteindre son maximal sur l'ensemble des valeurs observées de θ (92%). Ainsi, nous constatons que le nombre de faux négatifs, soit le nombre d'éléments non détectés, diminue lorsque la valeur de θ augmente, puis se stabilise lorsque la valeur de θ atteint 0.9 seconde. La courbe de rappel n'atteint pas les 100% car le nombre de faux négatifs n'est pas nul, limité par l'ensemble des valeurs observées. Cependant, nous constatons que la précision diminue fortement au delà de 1 seconde. Ainsi, nous pouvons donc conclure que lorsque la valeur de θ est de 0.9 seconde, la pattern PBC' est pertinent.

Protocole(s) \ Patterns	PCAP 1					PCAP 2				
	OWT/PBC/CTRL		*ASP*			*OWT/PBC/CTRL*		*ASP*		
	CT	WT	VP	FP	FN	CT	WT	VP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	10	1	2	2	3	9	2	2	2	2
ZigBee-BTLE ∪ OS4I	10	1	2	2	3	9	2	2	2	2
ZigBee-OS4I ∪ BTLE	10	1	2	2	3	9	2	2	1	2
OS4I-BTLE ∪ ZigBee	11	0	2	0	3	10	0	3	1	1
ZigBee-OS4I-BTLE	11	0	4	0	1	10	0	3	2	1
Valeurs attendues	11	0	5	0	0	11	0	4	0	0

TABLEAU 7.6 – Détection de patterns en fonction du nombre de protocoles observés simultanément lorsque la dégradation est liée à l'utilisation d'une base de conversion incomplète.

Le tableau 7.6 présente les résultats de la modélisation des deux captures de trafic en clair, avec l'utilisation d'une base de conversion incomplète.

Nous observons que les résultats obtenus sont identiques à ceux du cas idéal, présentés dans le tableau 7.5.

Avec ces résultats, nous constatons que même si la base de conversion est incomplète, le pattern PBC' permet d'identifier avec précision le rôle et l'orientation du flux de données, lorsque les communications observées ne sont pas chiffrées.

7.3.2 Dégradation par l'utilisation du chiffrement

Dans cette évaluation, nous évaluons la modélisation lorsque l'observation de l'environnement est dégradée par l'utilisation du chiffrement. Cette évaluation se concentre sur les éléments détectés par notre méthodologie de modélisation lorsque les trois protocoles sont observés simultanément.

Protocole(s) \ Patterns	PCAP 1					PCAP 2				
	OWT/PBC/CTRL		*ASP*			*OWT/PBC/CTRL*		*ASP*		
	CT	WT	VP	FP	FN	CT	WT	VP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	7	4	2	2	3	3	8	2	2	3
ZigBee-BTLE ∪ OS4I	7	4	2	2	4	3	8	2	2	3
ZigBee-OS4I ∪ BTLE	7	4	2	2	2	3	8	2	7	3
OS4I-BTLE ∪ ZigBee	8	3	3	2	2	3	8	3	1	2
ZigBee-OS4I-BTLE	8	3	3	3	2	4	7	4	5	1
Valeurs attendues	11	0	5	0	0	11	0	5	0	0

TABLEAU 7.7 – Détection de patterns en fonction du nombre de protocoles observés simultanément sur des communications chiffrées.

Le tableau 7.7 présente les résultats obtenus lorsque les communications observées sont chiffrées.

Nous observons qu'IoTMap parvient à détecter la quasi totalités des applications déployées dans le réseau. Ainsi, même si le chiffrement réduit la facilité d'analyse des informations, et augmente le nombre d'objets incorrectement détectés, notre méthodologie identifie correctement trois applications sur les quatre déployées.

Nous observons également que le nombre de faux positifs renvoyés par IoTMap sur le pattern ASP, est plus élevé lorsque les communications sont chiffrées. Ce résultat s'explique principalement par le nombre d'objets dont le rôle n'est pas correctement identifié. Les objets avec le rôle *source* et *sink* sont ceux impactés par une mauvaise détection de rôle. En effet, plutôt que d'avoir un rôle bien distinct, ces objets se voient attribuer les deux rôles par le pattern PBC'. Avec un filtrage des paquets moins précis, il est alors possible d'avoir des paquets de contrôle, normalement éludés par les patterns PDL, faussant le résultat car non identifiés comme tels en raison du chiffrement.

Par ailleurs, nous observons que le nombre de faux positifs de l'application ASP retournés, est impacté par les rôles correctement ou incorrectement identifiés. En effet, dans le PCAP1, le nombre d'objets dont le rôle est correctement identifié est largement supérieur à celui du PCAP2, et le nombre de faux positifs de l'application ASP dans le PCAP1 est nettement inférieur à celui du PCAP2.

Nous observons également que les objets et les applications sont correctement identifiés pour les protocoles Zigbee et OS4I. En revanche, dès lors que le protocole BLE est impliqué dans le réseau, le nombre de faux positifs augmente fortement. Ainsi, nous observons que le chiffrement appliqué dans chaque protocole, a un impact différent sur l'analyse et les informations obtenues.

Différences de chiffrement en fonction des protocoles Nous observons que parmi les trois protocoles déployés dans notre plateforme d'expérimentations, aucun ne propose la même méthodologie de chiffrement. De plus, à l'exception du protocole BLE, chaque paquet chiffré en Zigbee ou OS4I laisse fuiter un minimum d'informations, permettant ainsi de déduire un type d'application spécifique.

En effet, malgré le chiffrement des protocoles Zigbee et OS4I, la taille des paquets et des en-têtes, non chiffrés, permettent de déduire trop d'informations sur le type de paquets échangés entre les objets. Même si certains paquets, non désirés, comme des ACK applicatifs ou autres paquets de contrôle, ajoutent du bruit et rendent la détection des rôles moins précise, les applications exclusives aux protocoles Zigbee et OS4I sont détectées sans erreur à chaque itération.

Nous observons que les faux positifs relevés dans les résultats du tableau 7.7 sont essentiellement dû à la présence du protocole. En effet, le chiffrement proposé par ce protocole laisse peu d'informations fuiter, réduisant ainsi le filtre des paquets non désirés par le processus de modélisation. De plus, le chiffrement que propose le protocole BLE perturbe également la détection de l'objet qui émet la donnée et celui qui la reçoit. Ce fonctionnement représente une limite importante à l'utilisation de la modélisation proposée par IoTMap.

Une fausse information sur l'émetteur et le destinataire d'un paquet implique une mauvaise détection du rôle des objets qui communiquent. Lorsque les rôles sont incorrectement ou partiellement détectés, les patterns qui s'exécutent par la suite, sont fortement impactés et produisent des résultats erronés, rendant alors la modélisation moins précise.

Il est donc important pour chaque protocole IoT de fournir une couche de chiffrement suffisante pour empêcher toute tentative de déduction d'informations par une analyse auxiliaire du paquet, comme la taille par exemple. Le chiffrement des couches inférieures à la couche application est une bonne solution pour réduire la

fuite d'informations des en-têtes non chiffrés. Une autre solution est de définir une taille fixe des paquets chiffrés pour éviter toute déduction d'informations via cette méthode.

7.3.3 Dégradation par une observation partielle

Pour cette évaluation, nous proposons deux scénarios. Le premier concerne une dégradation de l'observation due à une perte de paquets dans le réseau. Le second met en évidence une observation partielle dans laquelle un ou plusieurs objets du réseau ne sont pas observables. De plus, pour chaque scénario, nous considérons que les communications ne sont pas chiffrées et que les bases de conversion sont incomplètes.

7.3.3.1 Perte de paquets

Nous évaluons, dans cette section, la pertinence et la précision de notre méthodologie de modélisation face à une perte de paquets. Nous utilisons les deux mesures, précision et rappel, pour illustrer l'efficacité de notre modélisation. Elles reposent sur les paramètres suivants :

VP : les nœuds sont détectés comme *controller* et les deux autres nœuds impliqués dans la communication sont corrélés.

FP : les nœuds sont détectés comme *controller* mais les deux autres nœuds impliqués dans la communication ne sont pas corrélés.

FN : les nœuds ne sont pas détectés comme *controller* bien que les deux autres nœuds impliqués dans la communication sont corrélés.

Nous évaluons l'efficacité de notre modélisation sur la génération du graphe transport. En effet, ce graphe exploite plusieurs patterns et possède un impact sur la génération des graphes applications. Ceci justifie l'utilisation de ce graphe pour notre évaluation.

Afin de représenter la perte de paquets dans notre réseau, nous supprimons, de manière aléatoire, un pourcentage de paquets directement depuis la liste de paquets génériques. Chaque pourcentage de perte de paquets, allant de 5% à 80%, par pallier de 5%, est effectué cinq fois.

La figure 7.7 présente les résultats obtenus. Nous observons, comme attendu, que la précision diminue très fortement dès lors que le pourcentage de perte de paquets augmente. Avec un taux de perte de paquets à 5%, la précision est proche de 100%. Elle diminue jusqu'à 50% avec 10% de perte de paquets. Puis la courbe continue de décroître avec l'augmentation du pourcentage de perte de paquets jusqu'à atteindre moins de 10%. Ce résultat est attendu, une perte de paquets entraîne une diminution du nombre de vrais positifs, alors que le nombre de faux positifs, lui, n'augmente pas. Ainsi, la précision décroit en fonction du nombre de paquets perdus.

Comme attendu, le résultat est similaire pour la courbe de rappel. En effet, nous observons une décroissance de la courbe identique à celle de la mesure précision. Comme la courbe de rappel atteint une valeur proche de 0% très rapidement, nous constatons que le nombre de faux négatifs augmente alors que le nombre de vrais positifs diminue, lorsque le taux de perte de paquets augmente.

7.3.3.2 Objets non observables

Nous proposons une évaluation de notre méthodologie de modélisation dans le cas où certains objets du réseau ne peuvent être observés. Nous présentons les

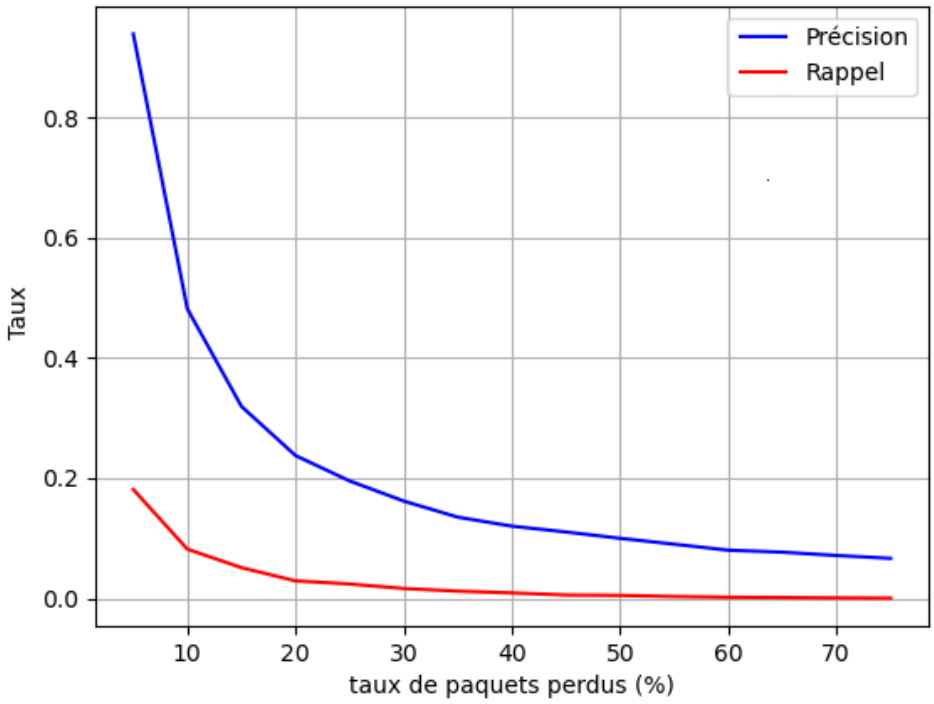

FIGURE 7.7 – Efficacité de la modélisation (graphe transport) en fonction du taux de paquets perdus.

résultats obtenus des différents patterns, en fonction des protocoles observés simultanément.

Dans un premier temps, nous proposons une observation où les objets aux IDs 2 et 5 sont exclus de l'observation, illustré par le tableau 7.8. L'objet 2 est un objet dont le rôle devrait être *sink*, et l'objet 5 devrait être une *source*.

ID : d2	0x3181	Zigbee
ID : d5	dc :d0 :17 :9d :1d :5d	BLE

TABLEAU 7.8 – Dispositifs exclus de l'observation car hors de portée de l'antenne d'interception.

Le tableau 7.9 présente les résultats obtenus de la modélisation du réseau lorsque les deux objets sont exclus de l'observation. Nous constatons des résultats similaires à ceux obtenus dans le cas idéal, présentés dans le tableau 7.5. Nous notons toutefois un objet incorrectement identifié dans cette observation. Il s'agit de l'objet 5, ce dernier est un dispositif qui devrait avoir le rôle *source*. Or dans ce scénario, cet objet n'est pas visible, ainsi toutes les communications émises par cet objet ne peuvent être interceptées. Uniquement les communications à destination de l'objet 5 sont observées. C'est pourquoi dans ce résultat il possède le rôle *sink*. En revanche, l'objet 2 est correctement identifié. Ainsi, toutes les interactions dans lesquelles cet objet est impliqué sont normalement détectées.

Dans un second temps, nous évaluons notre modélisation avec le scénario que nous considérons comme étant le pire scénario possible dans notre réseau. C'est à

Protocole(s) \ Patterns	*OWT/PBC/CTRL*		*ASP*		
	CT	WT	VP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	9	2	2	2	3
ZigBee-BTLE ∪ OS4I	9	2	2	2	3
ZigBee-OS4I ∪ BTLE	9	2	2	2	3
OS4I-BTLE ∪ ZigBee	10	1	2	0	3
ZigBee-OS4I-BTLE	10	1	4	0	1
Valeurs attendues	11	0	5	0	0

TABLEAU 7.9 – Détection de patterns en fonction du nombre de protocoles observés simultanément lorsque des objets ne sont pas observés.

ID : d10	: :212 :4b00 :1204 :cea4	OS4I
ID : d9	: :212 :4b00 :1204 :c92d	OS4I
ID : d1	0x7b65	Zigbee
ID : d5	dc :d0 :17 :9d :1d :5d	BLE

TABLEAU 7.10 – Dispositifs exclus de l'observation car hors de portée de l'antenne d'interception.

dire que tous les objets impliqués dans des communications bidirectionnelles, et qui devraient être détectés comme des objets avec le rôle *source*, sont hors de portée de notre observation. Les conséquences de cette dégradation sont la suppression de tous les paquets émis par ces objets, car non visibles. La liste des dispositifs sélectionnés est donnée dans le tableau 7.10.

Protocole(s) \ Patterns	*OWT/PBC/CTRL*		*ASP*		
	CT	WT	VP	FP	FN
ZigBee ∪ BTLE ∪ OS4I	4	7	0	4	5
ZigBee-BTLE ∪ OS4I	4	7	0	2	5
ZigBee-OS4I ∪ BTLE	4	7	0	2	5
OS4I-BTLE ∪ ZigBee	4	7	0	2	5
ZigBee-OS4I-BTLE	4	7	0	4	5
Valeurs attendues	11	0	5	0	0

TABLEAU 7.11 – Détection de patterns en fonction du nombre de protocoles observés simultanément lorsque des objets ne sont pas observés.

Le tableau 7.11 présente les résultats obtenus par notre méthodologie de modélisation en fonction du nombre de protocoles observés. Nous constatons que peu importe le nombre de protocoles observés, le nombre de dispositifs correctement identifiés est similaire. En effet, il s'agit d'objets ayant pour rôle *sink*, qui ne sont perturbés ni par le nombre de protocoles, car spécifique à un unique protocole, ni par la dégradation. Ainsi, même lorsque les protocoles sont observés indépendamment, le résultat est identique sur la détection des rôles.

Nous constatons également qu'aucune interaction existante n'est détectée par notre pattern ASP. En effet, la dégradation cible uniquement les objets ayant le rôle *source*, impliqués dans des communications bidirectionnelles. Ainsi, de nombreuses interactions provenant de ces objets ne sont pas détectées. De ce fait, tous les objets non visibles sont détectés à tort comme *sink* et ceux visibles sont détectés comme

source. Ces erreurs de détection provoquent alors une augmentation d'interactions et de rôles considérés comme faux positifs.

7.4 Résumé

Dans ce chapitre, nous avons évalué notre méthodologie de modélisation sur un environnement réaliste, et dans des conditions d'observation différentes.

Tout d'abord, nous avons présenté notre plateforme d'expérimentations, composée de onze objets IoT avec des matériels hétérogènes. Nous avons, également, décrit les différentes applications mono- et multi-protocolaires, déployées dans la plateforme. Enfin, nous avons exposé les conditions expérimentales utilisées pour l'évaluation.

Nous avons réalisé une première évaluation, lorsque les conditions d'observations sont idéales. Nous avons validé l'utilisation de la valeur δ dans le pattern CTRL, avant d'évaluer son efficacité. Les résultats obtenus ont montré que le pattern CTRL possède une précision et un rappel quasi optimal, lorsque la valeur de δ est proche de 0.25s. Nous avons également évalué le nombre d'éléments détectés par nos patterns en fonction du nombre de protocoles observés. Nous avons remarqué que nos patterns détectent correctement, le rôle et les applications, lorsque les protocoles sont observés simultanément.

Nous avons ensuite réalisé une seconde évaluation, lorsque les conditions d'observation sont dégradées, par une perte d'informations. Si la perte d'informations est liée à l'utilisation d'une base de conversion incomplète, les résultats obtenus montre que notre modélisation est aussi efficace que dans le cas idéal. Dans le cas, où la perte d'informations est liée à l'utilisation du chiffrement, nos évaluations ont montré que notre modélisation détecte toujours la majorité des applications déployées, malgré une hausse du nombre de faux positifs.

Enfin, nous avons évalué notre modélisation lorsque l'observation est partielle, soit par une perte de paquets, soit par des objets hors de portée. Dans le premier cas, nos résultats montre que plus le nombre de paquets perdus augmentent et moins notre modélisation est efficace. Dans le second cas, nous avons évalué deux scénarios. Le premier scénario identifie deux objets comme hors de portée de l'observation. Les résultats obtenus sont identiques à ceux observés dans le cas idéal, ce qui montre qu'IoTMap est capable de modéliser le réseau, même lorsque des objets sont hors de portée de l'observation. Le second scénario présente le pire scénario possible pour notre modélisation, et comme attendu, notre modélisation ne détecte aucune application correctement.

Chapitre 8

Conclusion

Pour conclure, nous proposons un bilan de cette thèse. Nous présentons les différents éléments abordés dans le manuscrit ainsi que nos contributions. Nous résumons notre approche pour décrire les protocoles IoT et leurs problèmes fondamentaux de sécurité. De plus, nous abordons l'intérêt d'une modélisation de réseaux IoT et la nécessité de l'utiliser, afin de comprendre et améliorer la sécurité de ces réseaux. Nous listons ensuite dans les travaux futurs, les points importants que nous souhaitons réaliser. Ces approches permettront d'enrichir, à court ou moyen terme, notre modélisation afin d'améliorer la sécurité des réseaux IoT. Enfin, nous concluons cette thèse par une discussion et des perspectives, que nous estimons réelles et importantes à présenter dans une démarche d'amélioration globale de la sécurité des réseaux IoT.

8.1 Bilan

Dans cette section, nous établissons un bilan des contributions présentées en section 1.4. Dans une premier temps, nous reprenons les différents éléments abordés pour chaque contribution. Puis, nous décrivons les difficultés associées à la contribution, avant de présenter notre approche.

8.1.1 Approche générique des piles de protocoles IoT

Dans cette thèse, nous avons proposé une approche générique permettant de décrire et de comparer les piles de protocoles IoT.

Nous avons proposé, dans un premier temps, d'analyser les protocoles IoT pour en extraire plusieurs critères communs. Cette approche nous a permis de souligner la difficulté d'identifier des critères communs et de définir une structure abstraite pour comparer efficacement différentes piles de protocoles IoT. Selon le domaine d'activité, les usages et les applications, les caractéristiques et les spécificités des piles de protocoles IoT diffèrent [27]. Notre approche a permis de définir une pile IoT générique décomposée en cinq couches et quatre critères de comparaison. Ces critères sont utilisés pour présenter et analyser différentes piles de protocoles IoT dans un formalisme similaire.

Afin d'améliorer notre analyse générique, nous avons proposé, dans un second temps, une catégorisation et une classification des vulnérabilités. Comme décrit dans ce document, les attaques publiées contre les protocoles IoT sont toujours présentées spécifiquement pour un protocole. Nous pouvons expliquer ce résultat de multiples façons. Tout d'abord, la diversité des protocoles rend l'analyse de chaque attaque difficile et fastidieuse. De plus, les conditions d'exploitation des attaques peuvent être difficiles, voire impossibles à reproduire sur des environnements réels. Enfin, la

disponibilité et l'accessibilité des informations sur les mécanismes de sécurité diffèrent selon les piles de protocoles IoT. Ceci rend l'analyse générique des attaques difficile pour toutes les piles de protocoles IoT. Nous avons proposé, pour répondre à cette difficulté, une *killchain* divisée en trois parties : les attaques axées sur les paquets, les attaques axées sur les protocoles et les attaques axées sur les systèmes.

Nous avons analysé chaque attaque par rapport au formalisme proposé pour la pile IoT générique. Ainsi, nous avons pu inventorier les attaques similaires sur différents protocoles, les relier et estimer quelles attaques pouvaient être exécutées de manière similaire contre un autre protocole et quels protocoles étaient intrinsèquement résistants à certaines attaques.

8.1.2 Modélisation de réseaux IoT hétérogènes

Les réseaux IoT deviennent de plus en plus complexes. Ils se composent d'objets aux exigences diverses et variées. De plus, ces réseaux utilisent plusieurs protocoles simultanément, dans un même environnement. Le contrôle, la supervision et la gestion de ces systèmes, très hétérogènes, deviennent plus difficiles. Plusieurs solutions existent pour aider les utilisateurs à comprendre leur système IoT. Cependant, elles se concentrent souvent sur l'analyse et la sécurité d'un unique protocole. Ainsi, elles ne peuvent être utilisées, ou partiellement, si le système utilise des protocoles différents.

Nous avons proposé dans ce manuscrit une méthodologie de modélisation de réseaux IoT hétérogènes, qui repose sur une description générique des piles de protocoles IoT. Elle utilise une approche en quatre graphes distincts, allant du graphe liaison jusqu'au graphe application, où chacun des graphes apporte un niveau d'information utile pour l'observateur.

Le processus de construction de la modélisation est itératif. Ainsi, la création de chaque graphe nécessite l'utilisation du graphe précédent. Nous avons proposé pour chaque fonction de construction de graphe, une liste de patterns, qui est utilisée pour détecter des éléments permettant la génération du graphe. Nous avons proposé des patterns permettant de modéliser des réseaux dont l'observation peut varier. En effet, nous avons proposé, dans un premier temps, une liste de patterns pour modéliser des réseaux dont l'observation est idéale, sans contraintes ni limites. Puis, nous avons proposé des modifications à certains patterns, afin de répondre à différentes dégradations que pourrait subir l'observation, telles que la perte d'informations ou l'invisibilité de certains objets.

8.1.3 Implémentation et évaluation

L'ensemble de notre méthodologie a été implémenté dans le *framework* IoTMap, afin de pouvoir l'utiliser dans un contexte réel. Cette implémentation est décomposée en plusieurs parties. La première est responsable de la partie interception et s'appuie sur des solutions tierces. La seconde partie intègre notre méthodologie de modélisation par des graphes. Enfin, une troisième partie permet la visualisation des résultats obtenus, afin d'analyser et de comprendre le réseau observé. IoTMap est également rendu open source pour faciliter son utilisation par la communauté.

Nous avons évalué IoTMap et notre méthode de modélisation à travers deux cas très distincts. Dans un premier temps, nous avons présenté les résultats obtenus lorsque IoTMap est utilisé dans un cas idéal, où toutes les communications et les objets sont connus et accessibles. Puis, dans un second temps, nous avons dégradé l'observation via deux facteurs, la perte d'informations et la perte de paquets.

Pour réaliser les tests et l'évaluation, nous avons déployé une plateforme d'expérimentations, composée d'objets physiques représentant un réseau domotique réel mélangeant plusieurs protocoles IoT. Ces objets utilisent des matériels et logiciels différents afin de favoriser l'hétérogénéité de la plateforme. De plus, nous avons développé et déployé plusieurs applications, mono- et multi-protocolaires.

Comme attendu, IoTMap est parvenu à modéliser et retrouver les interactions du réseau hétérogène dans le cas idéal. Dans les cas dégradés, nous avons observé qu'IoTMap est également parvenu à détecter la plupart des applications déployées. Cependant, nous avons constaté la présence de nombreux faux positifs. Lorsque l'information est chiffrée, l'identification des rôles n'est plus aussi précise.

En présence de perte de paquets, IoTMap a rapidement perdu en précision. Ce résultat est normal dans notre évaluation. En effet, sans information à analyser, IoTMap ne peut pas modéliser le réseau.

Lorsque l'observation partielle est liée à des objets non visibles, nous avons différencié deux cas particuliers. Le premier a concerné la détection d'objets non visibles, qui n'entraîne pas de modifications de leur rôle. Le second cas a présenté des scénarios où les objets non visibles sont détectés avec des mauvais rôles.

8.2 Travaux futurs

Notre approche de modélisation des réseaux IoT hétérogènes est plus que nécessaire pour comprendre et analyser les réseaux déployés, afin d'en améliorer la sécurité. IoTMap présente de bons résultats, notamment lorsque les protocoles observés sont analysés simultanément plutôt qu'indépendamment, que les conditions soient idéales ou dégradées. De plus, nous avons rendu ce *framework* open source et accessible à la communauté pour qu'ensemble, nous puissions améliorer la sécurité des réseaux IoT. Afin d'améliorer et de faire évoluer les travaux entrepris dans cette thèse, nous proposons les travaux futurs suivants :

— **Intégration du protocole industriel WirelessHart**. Nous souhaitons ajouter des protocoles LAN dont le domaine d'application est spécifique. Actuellement, les protocoles supportés dans notre méthodologie n'ont pas de secteur d'utilisation imposé. Ils peuvent être déployés dans tous les environnements et proposent des applications très générales. Nous souhaitons donc intégrer des protocoles spécialement conçus pour un domaine en particulier, comme WirelessHart [100] pour l'IoT industriel (IIoT).

  L'intégration de nouveaux protocoles spécifiques à un domaine d'application nécessite l'ajout de nouveaux patterns. Actuellement, nos patterns sont centrés sur les interactions et les comportements des objets sur le réseau. Avec des protocoles spécifiques à un domaine, nous devons penser des patterns qui détectent des comportements observés uniquement dans un environnement spécifique (IIoT, Santé, etc.).

— **Modélisation des protocoles WAN**. Nous souhaitons étendre les types de protocoles supportés dans notre modélisation aux protocoles WAN. Dans cette thèse, nous avons proposé une méthodologie qui supporte les protocoles LAN, tels que ZigBee, OS4I et BLE. Nous constatons que plusieurs autres protocoles sont déployés aujourd'hui, comme LoRaWAN, SigFox ou encore NB-IoT [104]. Ces derniers sont des protocoles à longue portée, déployés dans des environnements ouverts, à l'échelle d'une ville par exemple.

Ainsi, pour intégrer ces protocoles dans notre méthodologie, nous devons ajouter à notre approche générique la définition de nouveaux schémas de communications [6]. De plus, nous devons également proposer de nouveaux patterns, pour détecter ces schémas de communications, mais également pour détecter de nouvelles applications propres à ce type de protocoles.

— **Appliquer le chiffrement aux couches inférieures**. Dans cette thèse, nous avons défini plusieurs dégradations possibles qu'une observation peut subir. Dans le cas du chiffrement, nous faisons l'hypothèse qu'uniquement la couche applicative du paquet est chiffrée. Nous souhaitons amplifier la dégradation par le chiffrement en l'appliquant sur des couches inférieures, comme par exemple la couche transport, réseau, voire physique. En effet, en fonction de la couche où est appliqué le chiffrement, le comportement des objets dans le réseau change. Les paquets vont également être modifiés pour que les objets puissent continuer à communiquer.

  Avec un niveau d'informations encore plus réduit, nous devrons adapter nos bases de conversion pour essayer d'identifier le maximum d'informations dans des paquets chiffrés. De plus, nous devrons développer d'autres patterns pour détecter les nouveaux comportements des objets qui communiquent. En présence de moins d'informations, une solution par l'approche temporelle peut être efficace. D'autres solutions comme le suivi de flux de trafic [18] peuvent également être utilisées pour identifier les objets qui émettent des données et ceux qui la reçoivent. Dans le cas du chiffrement de la couche physique, il faudrait proposer des patterns permettant d'identifier des caractéristiques spécifiques d'un protocole [26].

— **Implémenter d'autres étapes du pentest**. Enfin, dans cette thèse, nous avons proposé une méthodologie de modélisation, considérée comme étant la première étape d'une méthode de pentest. Nous souhaitons explorer les étapes restantes de la méthodologie du pentest, en commençant par la définition d'un modèle de menaces et l'analyse des vulnérabilités. Plusieurs travaux proposent des méthodologies de pentests pour l'IoT [113, 36], mais ces derniers restent trop spécifiques à un ou deux protocoles.

  Notre approche consistera toujours à utiliser un formalisme générique et homogène pour décrire le modèle de menaces, afin qu'il puisse s'adapter au plus grand nombre de protocoles. Pour cela, nous souhaitons utiliser notre classification des vulnérabilités, présentée dans le chapitre 2. Ce modèle permet de définir les vulnérabilités par catégories. Nous proposerons ensuite des patterns qui auront pour objectifs de détecter la présence de ces menaces. Nous proposerons également d'autres patterns qui pourront détecter si une vulnérabilité est réalisable dans la configuration actuelle du réseau.

Ces travaux futurs sont une feuille de route sur du long terme. Cependant, nous estimons qu'ils permettront de contribuer au renforcement et à l'amélioration des problèmes de sécurité dans l'IoT.

8.3 Perspectives pour améliorer la sécurité des réseaux IoT

Nous pouvons mettre en évidence certaines causes profondes de ces problèmes de sécurité. Premièrement, bien que les piles de protocoles évoluent, les impératifs commerciaux impliquent qu'elles doivent conserver une certaine compatibilité avec les objets déjà déployés, ce qui entraîne des limites de sécurité héritées du passé.

Ce problème peut être atténué en isolant certaines parties du système ou en promouvant les logiciels open source, qui comportent généralement moins d'exigences héritées du passé, car les anciens objets peuvent être mis à jour.

Deuxièmement, les objets ne peuvent souvent pas être mis à jour, et ceux qui le peuvent exigent généralement des fabricants, des mises à jour inappropriées avec la politique de sécurité (rarement disponible, ou ne sont pas fournies pendant une durée suffisante, compte tenu de la durée prévue de ces objets). C'est pourquoi, nous sommes fermement convaincus que les fournisseurs d'objets IoT doivent favoriser la communication avec les chercheurs en sécurité (spécifications ouvertes et *bug bounty*), plutôt que d'essayer de les bloquer dans les travaux et leur publication.

Troisièmement, la pression commerciale fait passer la facilité d'utilisation avant la sécurité, et tout acteur qui voudrait changer cette vision verrait probablement ses ventes baisser par rapport à celles de ses concurrents. Une approche par les tests d'intrusion [14] peut donner des résultats intéressants en mettant l'accent, pour un déploiement donné, sur des mesures de sécurité qui ne doivent pas être ignorées. Enfin, bien qu'offrant des fonctionnalités similaires, il existe trop de protocoles hétérogènes, et les chercheurs et opérateurs en sécurité sont donc répartis entre les divers systèmes, au lieu d'unir leurs forces. Nous avons besoin d'un cadre commun pour mutualiser les efforts de sécurité et faciliter les échanges entre les communautés travaillant sur des protocoles différents.

À partir de ce travail, nous identifions qu'une nouvelle méthode pourrait être définie pour assurer les mêmes mécanismes de sécurité, quelle que soit la pile de protocole utilisée. La définition de cette nouvelle méthode soulève plusieurs nouveaux points de recherche intéressants :

— **Identifier des critères de sécurité communs**, en dressant une liste de caractéristiques de sécurité parmi tous les protocoles afin d'établir une base globale qui pourrait être appliquée à chaque pile de protocoles. Les travaux existants [3, 93, 109] fournissent des classifications et des taxonomies de caractéristiques de sécurité qui pourraient être appliquées de manière générique. Toutefois, ces travaux sont axés sur un aspect ou une application spécifique de l'IoT et non sur les piles de protocoles IoT. Ces travaux pourraient être utiles pour fournir une taxonomie des critères de sécurité, adaptée à chaque pile de protocoles IoT.

— **Harmoniser les mécanismes de sécurité** en trouvant une solution pour les appliquer, quelle que soit la pile de protocoles utilisée. Une solution peut consister à définir une certification que chaque protocole doit appliquer [54, 10]. S'appuyant sur des critères de sécurité communs, cette certification devrait être suffisamment générique pour s'adapter à un maximum de piles de protocoles IoT et couvrir les plus grandes demandes déposées. On pourrait imaginer une autre solution telle qu'un *framework* [86] qui identifie les mécanismes de sécurité à appliquer, en fonction du domaine d'application visé.

— **Garantir un niveau de sécurité constant**, même si les piles de protocoles évoluent. Cela signifie un suivi régulier de toutes les versions déployées pour un protocole afin d'assurer le même niveau de sécurité [21]. Cela devrait permettre d'éviter certains problèmes de compatibilité entre deux versions d'un même protocole.

L'IoT de demain sera multi-protocolaire, mélangeant des systèmes complémentaires déployés à différents moments, et devra faire face à la fois à l'héritage d'un protocole commun et à une série de protocoles différents. L'analyse à l'échelle d'un protocole unique ne sera plus pertinente.

Bibliographie

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things : A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4) :2347–2376, 2015.

[2] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi. Internet of things security : A survey. *J. Network and Computer Applications*, 88 :10–28, 2017.

[3] F. Alkhabbas, R. Spalazzese, and P. Davidsson. Characterizing internet of things systems through taxonomies : A systematic mapping study. *Internet of Things*, 7 :100084, 2019.

[4] M. Ammar, G. Russello, and B. Crispo. Internet of things : A survey on the security of iot frameworks. *J. Inf. Sec. Appl.*, 38 :8–27, 2018.

[5] P. Anantharaman, L. Song, I. Agadakos, G. Ciocarlie, B. Copos, U. Lindqvist, and M. E. Locasto. Iothound : environment-agnostic device identification and monitoring. In *Proceedings of the 10th International Conference on the Internet of Things*, pages 1–9, 2020.

[6] L. T. Ancian and M. Cunche. Re-identifying addresses in LoRaWAN networks. Research Report RR-9361, Inria Rhône-Alpes ; INSA de Lyon, 2020.

[7] Andrew Banks and Rahul Gupta. MQTT version 3.1.1. OASIS Standard, 29 October 2014.

[8] S. Andy, B. Rahardjo, and B. Hanindhito. Attack scenarios and security analysis of MQTT communication protocol in IoT system. In *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, pages 1–6. IEEE, 2017.

[9] A. Ayadi, P. Maillé, D. Ros, L. Toutain, and T. Zheng. Implementation and evaluation of a TCP header compression for 6lowpan. In *International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1359–1364, 2011.

[10] G. Baldini, A. Skarmeta, E. Fourneret, R. Neisse, B. Legeard, and F. Le Gall. Security certification and labelling in internet of things. In *3rd World Forum on Internet of Things (WF-IoT)*, pages 627–632, 2016.

[11] P. Baronti, P. Pillai, V. W. C. 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.

[12] B. Benmoshe, E. Berliner, A. Dvir, and A. Gorodischer. A joint framework of passive monitoring system for complex wireless networks. In *Consumer Communications and Networking Conference (CCNC)*, pages 55–59. IEEE, 2011.

[13] B. Bezawada, M. Bachani, J. Peterson, H. Shirazi, I. Ray, and I. Ray. Iotsense : Behavioral fingerprinting of iot devices. 2018.

[14] M. Bishop. About penetration testing. *IEEE Security & Privacy*, 5(6) :84–87, 2007.

[15] G. Bora, S. Bora, S. Singh, and S. M. Arsalan. Osi reference model : An overview. *International Journal of Computer Trends and Technology (IJCTT)*, 7(4) :214–218, 2014.

[16] M. Brachmann, O. Garcia-Morchon, and M. Kirsche. Security for practical CoAP applications : Issues and solution approaches. *10th GI/ITG KuVS Fachgespraech Sensornetze (FGSN11)*, pages 15–16, 2011.

[17] D. Cauquil. You'd better secure your ble devices or we'll kick your butts. *DEF CON*, 26, 2018.

[18] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac. Sensitive information tracking in commodity iot. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1687–1704, 2018.

[19] B. Chen, G. Peterson, G. Mainland, and M. Welsh. Livenet : Using passive monitoring to reconstruct sensor network dynamics. In *Distributed Computing in Sensor Systems, 4th IEEE International Conference (DCOSS)*, pages 79–98. Springer, 2008.

[20] A. H. Chowdhury, M. Ikram, H.-S. Cha, H. Redwan, S. Shams, K.-H. Kim, and S.-W. Yoo. Route-over vs mesh-under routing in 6lowpan. In *International conference on wireless communications and mobile computing : Connecting the world wirelessly (IWCMC)*, pages 1208–1212. ACM, 2009.

[21] R. N. Christina Skouloudi, Apostolos Malatras and G. Dede. Guidelines for Securing the Internet of Things. Research Report TP-02-20-946-EN-N, ENISA, 2020.

[22] T. H. Clausen, J. Yi, and A. C. de Verdiere. Loadng : Towards AODV version 2. In *Vehicular Technology Conference (VTC)*, pages 1–5. IEEE, 2012.

[23] L. Coppolino, V. DAlessandro, S. DAntonio, L. Levy, and L. Romano. My smart home is under attack. In *International Conference on Computational Science and Engineering (CSE)*, pages 145–151. IEEE, 2015.

[24] L. Da Xu, W. He, and S. Li. Internet of things in industries : A survey. *IEEE Transactions on industrial informatics*, 10(4) :2233–2243, 2014.

[25] S. M. Darroudi and C. Gomez. Bluetooth low energy mesh networks : A survey. *Sensors*, 17(7) :1467, 2017.

[26] J. de Jesus Rugeles, E. P. Guillen, and L. S. Cardoso. A technical review of wireless security for the internet of things : Software defined radio perspective, 2020.

[27] J. Ding, M. Nemati, C. Ranaweera, and J. Choi. Iot connectivity technologies and applications : A survey. *IEEE Access*, 8 :67646–67673, 2020.

[28] J. Durech and M. Franekova. Security attacks to ZigBee technology and their practical realization. In *International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 345–349. IEEE, 2014.

[29] X. Fan, F. Susan, W. Long, and S. Li. Security analysis of zigbee. Technical report, MIT, 2017.

[30] B. A. Forouzan. *TCP/IP protocol suite*. McGraw-Hill, Inc., 2002.

[31] B. Fouladi and S. Ghanoun. Security evaluation of the z-wave wireless protocol. *Black hat USA*, 24, 2013.

[32] Gartner. Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020. https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io, 2019. Accessed on 17 June 2020.

[33] C. Gomez, J. Oller, and J. Paradells. Overview and evaluation of bluetooth low energy : An emerging low-power wireless technology. *Sensors*, 12(9) :11734–11753, 2012.

[34] B. S. . M. W. Group. Bluetooth Mesh Networking specification. https://www.bluetooth.com/specifications/mesh-specifications. [Online ; accessed on 21 August 2019].

[35] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of things (iot) : A vision, architectural elements, and future directions. *Future Generation Comp. Syst.*, 29(7) :1645–1660, 2013.

[36] A. Gupta. Performing an iot pentest. In *The IoT Hacker's Handbook*, pages 17–37. Springer, 2019.

[37] J. Hall, B. Ramsey, M. Rice, and T. Lacey. Z-wave network reconnaissance and transceiver fingerprinting using software-defined radios. In *International Conference on Cyber Warfare and Security*, page 163. Academic Conferences International Limited, 2016.

[38] R. Hallman, J. Bryan, G. Palavicini, J. DiVita, and J. Romero-Mariona. IoDDoS - the internet of distributed denial of sevice attacks - A case study of the mirai malware and iot-based botnets. In *IoTBDS*, 2017.

[39] Y.-C. Hu, A. Perrig, and D. B. Johnson. Packet leashes : a defense against wormhole attacks in wireless networks. In *Annual Joint Conference of the IEEE Computer and Communications INFOCOM*, volume 3, pages 1976–1986. IEEE, 2003.

[40] J. Hui, D. Culler, and S. Chakrabarti. 6LoWPAN : Incorporating IEEE 802.15. 4 into the IP architecture. *Internet Protocol for Smart Objects Alliance (IPSO)*, 2009.

[41] J. Hui and P. Thubert. Compression format for ipv6 datagrams over ieee 802.15.4-based networks. RFC 6282, RFC Editor, September 2011.

[42] IEEE Standards Association. 802.15.4-2015 - IEEE Standard for Low-Rate Wireless Networks. Technical report.

[43] O. Iova, G. P. Picco, T. Istomin, and C. Király. RPL : the routing standard for the internet of things... or is it ? *IEEE Communications Magazine*, 54(12-Supp) :16–22, 2016.

[44] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1) :11–17, 2015.

[45] M. J. Kaur and P. Maheshwari. Building smart cities applications using iot and cloud-based architectures. In *International Conference on Industrial Informatics and Computer Systems (CIICS)*, pages 1–5. IEEE, 2016.

[46] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle. DTLS based security and two-way authentication for the internet of things. *Ad Hoc Networks*, 11(8) :2710–2723, 2013.

[47] G. Kunzel, J.-M. Winter, I. Muller, C.-E. Pereira, and J.-C. Netto. A passive monitoring tool for evaluation of routing in wirelesshart networks. In *4th International Embedded Systems Symposium (IESS)*, pages 159–170. Springer, 2013.

[48] M. Kurniawan and S. Yazid. Mitigation strategy of sinkhole attack in Wireless Sensor Network. In *International Workshop on Big Data and Information Security (IWBIS)*, pages 119–125. IEEE, 2017.

[49] A. Lavric, A. I. Petrariu, and V. Popa. Long range sigfox communication protocol scalability analysis under large-scale, high-density conditions. *IEEE Access*, 7 :35816–35825, 2019.

[50] G. Lee, J. Lim, D.-k. Kim, S. Yang, and M. Yoon. An approach to mitigating sybil attack in wireless networks using zigBee. In *International Conference on Advanced Communication Technology (ICACT)*, volume 2, pages 1005–1009. IEEE, 2008.

[51] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao. A survey on internet of things : Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5) :1125–1142, 2017.

[52] K. Lounis and M. Zulkernine. Attacks and defenses in short-range wireless technologies for iot. *IEEE Access*, 8 :88892–88932, 2020.

[53] A. Ludovici, A. Calveras, and J. Casademont. Forwarding techniques for IP fragmented packets in a real 6lowpan network. *Sensors*, 11(1) :992–1008, 2011.

[54] S. N. Matheu, J. L. Hernandez-Ramos, and A. F. Skarmeta. Toward a cybersecurity certification framework for the internet of things. *IEEE Security & Privacy*, 17(3) :66–76, 2019.

[55] M. Mavani and K. Asawa. Modeling and analyses of ip spoofing attack in 6lowpan network. *Computers & Security*, 70 :95–110, 2017.

[56] T. Melamed. An active man-in-the-middle attack on bluetooth smart devices. *International Journal of Safety and Security Engineering*, 8(2) :200–211, 2018.

[57] D. M. Mendez, I. Papapanagiotou, and B. Yang. Internet of things : Survey on security and privacy. *CoRR*, abs/1707.01879, 2017.

[58] M. Miettinen, S. Marchal, I. Hafeez, N. Asokan, A. Sadeghi, and S. Tarkoma. Iot SENTINEL : automated device-type identification for security enforcement in iot. In *37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE Computer Society, 2017.

[59] N. Modadugu and E. Rescorla. The design and implementation of datagram TLS. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

[60] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of ipv6 packets over ieee 802.15.4 networks. RFC 4944, RFC Editor, September 2007.

[61] T. H. news. Blueborne : Critical bluetooth attack puts billions of devices at risk of hacking. https://thehackernews.com/2017/09/blueborne-bluetooth-hacking.html, 2019. Accessed on 21 August 2019.

[62] T. H. news. Casino Gets Hacked Through Its Internet-Connected Fish Tank Thermometer. https://thehackernews.com/2018/04/iot-hacking-thermometer.html, 2019. [Online ; accessed on 21 August 2019].

[63] T. H. news. Hackers can remotely access syringe infusion pumps to deliver fatal overdoses. https://thehackernews.com/2017/09/hacking-infusion-pumps.html, 2019. Accessed on 21 August 2019.

[64] T. H. news. Hackers could turn LG smart appliances into remote-controlled spy robot. https://thehackernews.com/2017/10/smart-iot-device-hacking.html, 2019. Accessed on 21 August 2019.

[65] T. H. news. How drones can find and hack internet-of-things devices from the sky. https://thehackernews.com/2015/08/hacking-internet-of-things-drone.html, 2019. Accessed on 21 August 2019.

[66] T. H. news. How to Hack Smart Bluetooth Locks and IoT Devices — Check this Out. https://thehackernews.com/2016/09/hacking-bluetooth-locks.html, 2019. [Online ; accessed on 21 August 2019].

[67] T. H. news. Z-wave downgrade attack left over 100 million iot devices open to hackers. https://thehackernews.com/2018/05/z-wave-wireless-hacking.html, 2019. Accessed on 21 August 2019.

[68] K. T. Nguyen, M. Laurent, and N. Oualha. Survey on secure communication protocols for the internet of things. *Ad Hoc Networks*, 32 :17–31, 2015.

[69] C. P. O'Flynn. Message denial and alteration on IEEE 802.15. 4 low-power radio networks. In *International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5. IEEE, 2011.

[70] O. Olawumi, K. Haataja, M. Asikainen, N. Vidgren, and P. Toivanen. Three practical attacks against zigbee security : Attack scenario definitions, practical experiments, countermeasures, and lessons learned. In *International Conference on Hybrid Intelligent Systems (HIS)*, pages 199–206, 2014.

[71] J. Olsson. 6lowpan demystified. Technical report, Texas Instruments, 2014.

[72] A. Oracevic, S. Dilek, and S. Özdemir. Security in internet of things : A survey. In *2017 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6, 2017.

[73] S. D. Park. 6LoWPAN ad hoc on-demand distance vector routing (LOAD). Internet-Draft draft-daniel-6lowpan-load-adhoc-routing-03, IETF Secretariat, June 2007.

[74] P. Perazzo, C. Vallati, D. Varano, G. Anastasi, and G. Dini. Implementation of a wormhole attack against a RPL network : Challenges and effects. In *14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, pages 95–102. IEEE, 2018.

[75] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, RFC Editor, July 2003. http://www.rfc-editor.org/rfc/rfc3561.txt.

[76] P. Pongle and G. Chavan. A survey : Attacks on RPL and 6LoWPAN in IoT. In *International Conference on Pervasive Computing (ICPC)*, pages 1–6. IEEE, 2015.

[77] J. Radcliffe. Hacking medical devices for fun and insulin : Breaking the human SCADA system. In *Black Hat*, 2011.

[78] R. A. Rahman and B. Shah. Security analysis of IoT protocols : A focus in CoAP. In *International Conference on Big Data and Smart City (ICBDSC)*, pages 1–7. IEEE, 2016.

[79] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin. Sympathy for the sensor network debugger. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*, pages 255–267. ACM, 2005.

[80] S. Raza, S. Duquennoy, T. Voigt, and U. Roedig. Demo abstract : Securing communication in 6lowpan with compressed ipsec. In *International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–2. IEEE, 2011.

[81] S. Raza, A. Slabbert, T. Voigt, and K. Landernäs. Security considerations for the WirelessHART protocol. In *International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2009.

[82] S. Raza, D. Trabalza, and T. Voigt. 6lowpan compressed DTLS for CoAP. In *International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 287–289. IEEE, 2012.

[83] S. Raza, T. Voigt, and V. Jutvik. Lightweight ikev2 : a key management solution for both the compressed ipsec and the ieee 802.15. 4 security. In *Proceedings of the IETF workshop on smart object security*, volume 23. Citeseer, 2012.

[84] A. Reziouk, E. Laurent, and J.-C. Demay. Practical security overview of IEEE 802.15.4. In *International Conference on Engineering & MIS (ICEMIS)*, pages 1–9. IEEE, 2016.

[85] A. Reziouk, A. Lebrun, and J.-C. Demay. Auditing 6lowpan networks using standard penetration testing tools. *DEF CON 24*, 2016.

[86] D. A. Robles-Ramirez, P. J. Escamilla-Ambrosio, and T. Tryfonas. Iotsec : Uml extension for internet of things systems security modelling. In *International Conference on Mechatronics, Electronics and Automotive Engineering (ICMEAE)*, pages 151–156. IEEE, 2017.

[87] S. Rost and H. Balakrishnan. Memento : A health monitoring system for wireless sensor networks. In *3rd Annual Communications Society on Sensor and Ad Hoc Communications and Networks (SECON)*, volume 2, pages 575–584. IEEE, 2006.

[88] M. Ryan. Bluetooth : With low energy comes low security. In *7th USENIX Workshop on Offensive Technologies, WOOT*. USENIX Association, 2013.

[89] M. Sain, Y. J. Kang, and H. J. Lee. Survey on security in internet of things : state of the art and challenges. In *International Conference on Advanced Communication Technology (ICACT)*, pages 699–704. IEEE, 2017.

[90] S. SCSSI. Glossaire de la sécurité des systèmes d'information. Technical Report CNTI/CN 27, N23, SCSSI, octobre 1990.

[91] M. Shahid, G. Blanc, Z. Zhang, and H. Debar. Iot devices recognition through network traffic analysis. In *International Conference on Big Data (Big Data)*, pages 5187–5192. IEEE, 2018.

[92] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014.

[93] M. Shrestha, C. Johansen, J. Noll, and D. Roverso. A methodology for security classification applied to smart grid infrastructures. *International Journal of Critical Infrastructure Protection*, 28 :100342, 2020.

[94] S. Siby, R. R. Maiti, and N. O. Tippenhauer. Iotscanner : Detecting and classifying privacy threats in iot neighborhoods. *CoRR*, abs/1701.05007, 2017.

[95] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini. Security, privacy and trust in internet of things : The road ahead. *Computer networks*, 76 :146–164, 2015.

[96] B. SIG. Bluetooth specification version 4.0 vol 0 (6/2010). https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=229737. [Online ; accessed on 21 August 2019].

[97] B. SIG. Bluetooth specification version 5.0. https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=421043. [Online ; accessed on 21 August 2019].

[98] A. Sivanathan, D. Sherratt, H.-H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman. Characterizing and classifying iot traffic in smart cities and campuses. In *Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 559–564. IEEE, 2017.

[99] J. Slawomir. Gattacking bluetooth smart devices. *Black Hat USA*, 2016.

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

[101] N. Sornin, M. Luis, T. Eirich, T. Kramp, and O. Hersent. LoRaWAN Specification. Technical Report V1.0.1 Draf 3, LoRa Alliance, October 2015.

[102] B. Stelte and G. D. Rodosek. Thwarting attacks on ZigBee-Removal of the KillerBee stinger. In *International Conference on Network and Service Management (CNSM)*, pages 219–226. IEEE, 2013.

[103] J. Tournier, F. Lesueur, F. Le Mouël, L. Guyon, and H. Ben-Hassine. Iotmap, a modelling system for heterogeneous iot networks. In *Pass The Salt*, 2020.

[104] J. Tournier, F. Lesueur, F. Le Mouël, L. Guyon, and H. Ben-Hassine. A survey of iot protocols and their security issues through the lens of a generic iot stack. *Internet of Things*, page 100264, 2020.

[105] J. Tournier, F. Lesueur, F. L. Mouël, L. Guyon, and H. Ben-Hassine. Iotmap : A protocol-agnostic multi-layer system to detect application patterns in iot networks. In *Proceedings of the 10th International Conference on the Internet of Things*, pages 1–8, 2020.

[106] M. Turon. Mote-view : A sensor network monitoring and management tool. In *2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, pages 11–17. IEEE, 2005.

[107] J. Vasseur, N. Agarwal, J. Hui, Z. Shelby, P. Bertrand, and C. Chauvenet. RPL : The IP routing protocol designed for low power and lossy networks. *Internet Protocol for Smart Objects Alliance (IPSO)*, 2011.

[108] L. Wallgren, S. Raza, and T. Voigt. Routing attacks and countermeasures in the RPL-based internet of things. *International Journal of Distributed Sensor Networks (IJDSN)*, 9, 2013.

[109] P. Williams, P. Rojas, and M. Bayoumi. Security taxonomy in iot – a survey. In *62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 560–565, 2019.

[110] 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, RFC Editor, March 2012.

[111] J. Wright. Killerbee : practical zigbee exploitation framework. In *ToorCon*, 2009.

[112] X. Xu, J. Wan, W. Zhang, C. Tong, and C. Wu. PMSW : a passive monitoring system in wireless sensor networks. volume 21, pages 300–325, 2011.

[113] G. Yadav, A. Allakany, V. Kumar, K. Paul, and K. Okamura. Penetration testing framework for iot. In *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 477–482. IEEE, 2019.

[114] W. K. Zegeye. Exploiting bluetooth low energy pairing vulnerability in telemedicine. In *International Telemetering Conference Proceedings*. International Foundation for Telemetering, 2015.

[115] K. Zhao and L. Ge. A survey on the internet of things security. In *International Conference on Computational Intelligence and Security (CIS)*, pages 663–667. IEEE, 2013.

FOLIO ADMINISTRATIF

THESE DE L'UNIVERSITE DE LYON OPEREE AU SEIN DE L'INSA LYON

NOM : Tournier DATE de SOUTENANCE : 10/03/2021

Prénoms : Jonathan, Frédéric, Valérien

TITRE : Modélisation de réseaux IoT hétérogènes à des fins d'évaluation de sécurité

NATURE : Doctorat Numéro d'ordre : 2021LYSEI018

Ecole doctorale : InfoMaths

Spécialité : Informatique

RESUME : L'Internet des objets évolue rapidement, avec toujours plus de protocoles et d'objets déployés, à grande échelle ou dans des environnements cloisonnés. Cependant, avec cette émergence de protocoles, viennent de nouvelles problématiques de sécurité. En effet, les protocoles IoT sont hétérogènes, spécifiques à des besoins particuliers et utilisés dans des domaines d'applications divers, les rendant complexes à sécuriser. Plusieurs solutions existent pour évaluer et améliorer la sécurité des systèmes complexes, notamment le test d'intrusion.
Dans cette thèse, nous décrivons une méthodologie de modélisation de réseaux IoT hétérogènes, utilisée comme support d'analyse dans la réalisation de tests d'intrusion. Nous centrons cette méthodologie sur des protocoles IoT à courte portée, tels que Zigbee, BLE et OS4I. Cependant, son approche modulaire lui permet d'être fonctionnelle pour le plus grand nombre de protocoles IoT, sans devoir être modifiée pour l'ajout ou l'évolution d'un protocole.
Pour cela, nous présentons d'abord une approche générique, reposant sur quatre critères, qui permet de décrire et comparer, selon un modèle homogène, plusieurs protocoles IoT. Nous exposons, ensuite, une classification des attaques concernant les protocoles IoT, en trois parties, afin de comprendre, préciser la cible de l'attaque et son impact.
Ces modèles abstraits et génériques nous permettent ainsi l'élaboration d'une structure générique, que nous appelons paquet générique. C'est sur ce dernier que repose notre processus de modélisation, composé de quatre graphes. Nous proposons une construction itérative de ces graphes, allant du graphe représentant les communications point à point du réseau, jusqu'à celui mettant en évidence les applications détectées dans le réseau. La génération de chaque graphe nécessite l'utilisation de fonctions, prenant en entrée plusieurs patterns et le graphe précédent.
Nous proposons enfin une plateforme d'expérimentations, composée de plusieurs objets, mélangeant des dispositifs propriétaires et d'autres configurables. Elle permet notamment l'évaluation de notre méthodologie de modélisation dans des conditions expérimentales différentes. Dans le cas idéal, nous constatons que la modélisation détecte toutes les applications déployées dans le réseau. Le résultat est également pertinent lorsque nous dégradons notre observation par l'utilisation du chiffrement dans les communications. En effet, nous constatons alors que toutes les applications sont détectées, malgré un nombre plus important de faux positifs.
L'ensemble des fonctions et patterns définis dans notre méthodologie de modélisation est implémenté dans IoTMap, un framework, que nous avons rendu open source.
MOTS-CLÉS : IoT, réseaux IoT, modélisation IoT, réseaux hétérogènes, sécurité IoT, détection de patterns

Laboratoire (s) de recherche : CITILab

Directeur de thèse: Frédéric Le Mouël

Composition du jury :
Maryline Laurent, Professeur des Universités, Telecom SudParis, Evry
Marine Minier, Professeur des Universités, Université de Lorraine, Nancy
Sara Bouchenak, Professeur des Universités, LIRIS, INSA de Lyon, Lyon
Vincent Nicomette, Professeur des Universités, INSA de Toulouse, Toulouse
Eric Totel, Professeur des Universités, IMT Atlantique, Rennes
Frédéric Le Mouël, Professeur des Universités, INSA de Lyon, Lyon
François Lesueur, Maître de conférences, INSA de Lyon, Lyon