RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ MOULOUD MAMMERI DE TIZI-OUZOU

FACULTÉ DE GÉNIE ÉLECTRIQUE ET INFORMATIQUE
DÉPARTEMENT D'INFORMATIQUE

Mémoire de Fin d'Études de MASTER ACADÉMIQUE

Domaine : **Mathématiques et Informatique**

Filière : **Informatique**

Spécialité : **Réseaux, Mobilité et Systèmes Embarqués**

Présenté par :

Amar ABANE

Thème

Mise en œuvre des concepts NDN et IoT dans le domaine des Smart Cities : Exemple du parking intelligent

Mémoire soutenu publiquement le 21/06/2016 devant le jury composé de :

Président :	M. Mourad LAGHROUCHE - Université Mouloud MAMMERI de Tizi-ouzou
Encadreur :	M. Mehammed DAOUI - Université Mouloud MAMMERI de Tizi-ouzou
Co-encadreur :	Mme Samia BOUZEFRANE - Conservatoire National des Arts et Métiers, Paris
Examinatrice :	Mme Rachida AOUDJIT - Université Mouloud MAMMERI de Tizi-ouzou
Examinateur :	M. Mohamed RAMDANE - Université Mouloud MAMMERI de Tizi-ouzou

« Votre travail va remplir une grande partie de votre vie, et la seule manière d'être satisfait est de croire en ce que vous faites. Et la seule manière d'y arriver, c'est d'aimer ce que vous faites. Si vous n'avez pas encore trouvé, continuez à chercher. Ne vous reposez pas... »

- Steve Jobs, *Co-fondateur d'Apple*

Résumé

L'Internet of Things (IoT) est considéré comme la première véritable évolution de l'Internet. Mais c'est également le changement qui a le plus mis en évidence les limites de l'architecture IP sur laquelle repose Internet. Jusqu'ici, l'IP s'est greffé des protocoles et des couches logicielles pour répondre aux premières contraintes de l'IoT. Mais on commence à comprendre qu'une nouvelle architecture, conçue spécialement pour supporter toutes les nouvelles contraintes, est la solution ultime pour construire l'Internet du futur.

La NSF finance et dirige des projets concernant l'Internet du futur. L'un des projets les plus prometteur s'appelle NDN (Named-Data Networking). NDN est considéré comme une architecture ICN (Information-Centric Networking). L'approche ICN propose une architecture dans laquelle le point central est l'information (ou la donnée) non pas les adresses hôtes comme c'est le cas pour l'Internet actuel. NDN bénéficie d'un grand potentiel pour gérer les exigences de l'IoT, notamment par le fait qu'il soit conçu sur des principes complètement compatibles avec l'esprit et la nature des applications de l'IoT.

Les villes intelligentes sont un des aspects les plus complexes de l'Internet of Things, et l'un des services les plus importants pour les villes modernes (et celles du futur) est le stationnement intelligent.

NDN-SP est le système que nous proposons. C'est une plateforme IoT entièrement NDN. Il permet la recherche, la réservation et la gestion des places de stationnement dans une ville. Bien que notre solution s'inspire de certains travaux NDN sur l'IoT, elle est probablement la première à s'intéresser aux parkings intelligents avec NDN. Elle est également la première à utiliser la norme ZigBee comme moyen de communication sans fil (appelé *NDN-over-ZigBee*), élargissant ainsi les possibilités de NDN concernant les objets mobiles.

Abstract

The Interne of Things is certainly the first true evolution of Internet, and this innovation emphasizes the limitations of TCP/IP architecture which is the architecture of today's Internet. The first known constraints of IoT like scalability and mobility were addressed by adding some protocols and middlwares to TCP/IP architecture, but we start to understand that only a new architecture, designed to support IoT constraints and challenges, is the best solution to build Internet of the future.

The NSF founded and manages projects related to Internet of the future, in order to provide a network architecture that is better suited to today's use and future's use of Internet. NDN (Named-Data Networking) is one of these projects. It is based on the ICN (Information-Centric Networking) paradigm which focuses on contents to build network architectures rather than hosts addresses as in today's Internet architecture. NDN can offer some innovative solutions to address IoT issues because it's designed to accommodate emerging patterns of communication, and its principles are completely consistent with IoT's applications challenges.

Smart cities are one of the most interesting and intricate aspects of the IoT. One of the most important services provided by a smart city is smart parking.

NDN-SP is the system we propose. It is an IoT platform completely based on NDN. It allows handling, searching and making reservation of parking spaces within a smart city. Some NDN projects concerning IoT inspired us to design our proposed solution, but it's probably the first one that deals with smart parking using NDN. It's also the first solution that uses ZigBee specification as a communication support (called *NDN-over-ZigBee*) expanding NDN's possibilities toward mobile objects.

Keywords : Named-data, NDN, ICN, CCN, Internet of Things, smart cities, smart parking, future internet, mobility, networking architectures, TCP/IP, ZigBee, XBee, Raspberry Pi, ultrasonic sensors, embedded systems.

Remerciements

Ce travail, aussi modeste soit-il, n'aurait jamais pu être achevé sans l'aide de certaines personnes. Citer chacune d'elles m'est impossible. Je tiens néanmoins à remercier tous ceux qui ont contribué de près ou de loin à l'aboutissement de ce travail, en particulier :

- M. DAOUI et Mme BOUZEFRANE pour m'avoir fait confiance en me proposant ce sujet, et pour la qualité et l'ingéniosité de leur encadrement tout au long de ce travail. Je les remercie également d'avoir trouvé le temps nécessaire pour me conseiller et m'aider dans toutes mes initiatives malgré leur programme chargé.
- Les enseignants du Master « Réseaux, Mobilité et Systèmes Embarqués », pour leur soutien et leurs encouragements, et avec lesquels j'ai beaucoup appris.
- M. LAGHROUCHE pour son soutien, ses idées et le matériel qu'il a mis à ma disposition pour réaliser ce travail.
- L'administration et le personnel du département d'Informatique pour leur gentillesse et leur serviabilité.
- Mon père et ma mère pour leurs encouragements, leur soutien sans faille et leur compréhension malgré mon stress et ma mauvaise humeur. Je remercie également mes sœurs pour leur présence et leur aide incessante, en particulier *Lamia*.
- Mes camarades du département d'Informatique et mes amis : *Amir*, *Dahbia*, *Sarah* et les autres. Je remercie en particulier *Kenza* qui est toujours à l'écoute et qui m'encourage.

Sommaire

Introduction générale

En cinquante ans d'existence, l'Internet a connu bien des changements ; de nouveaux protocoles, équipements, algorithmes et autres adaptations issus de la recherche sont tour à tour venus renforcer l'architecture fragile et vulnérable qu'était l'Internet des années 60-70, pour donner ce grand réseau planétaire indispensable, qui connecte aujourd'hui des millions d'utilisateurs partageant des milliards de ressources. Mais ce grand réseau est une fois de plus mis à l'épreuve. En effet, l'Internet of Things (IoT) se démocratisant et devenant très accessible, impose de nouvelles contraintes concernant la connectivité, la mobilité, la mise à l'échelle et la sécurité, pour un nombre gigantesque d'objets présent désormais sur le réseau (50 à 100 milliards en 2020 [1]).

L'Internet actuel a été conçu comme un réseau de communication : les seules entités pouvant être identifiées par les paquets IP sont les adresses sources et destinations (endpoints). Mais les applications récentes (e-commerce, réseaux sociaux, domotique, IoT, etc.) en font une utilisation basée sur le contenu ; ce sont des requêtes pour les noms des données qui sont manipulées, ce qui rend l'Internet actuel inadapté. Les réseaux orientés information (Information-Centric Networking) dont NDN (Named-Data Networking) fait partie, représentent un concept plus général que les réseaux de communication. Par conséquent, un paquet NDN peut identifier autre chose qu'un hôte ; il peut identifier une vidéo, un fichier pdf, une commande de contrôle d'un dispositif, etc. NDN aspire à devenir l'architecture sur laquelle fonctionnera l'Internet du futur avec tous ses nouveaux aspects. Son avantage est qu'il a été conçu en combinant les forces de l'architecture TCP/IP actuelle en tenant compte des faiblesses et limites constatées avec l'évolution d'Internet et les exigences des applications émergentes.

Les villes intelligentes sont un des aspects les plus riches de l'Internet of Things. Plusieurs systèmes ont été imaginés et proposés pour offrir des services aux habitants et des prototypes sont testés dans certaines villes [2].

Face aux difficultés rencontrées avec le TCP/IP pour mettre en place des applications de l'IoT en général, nous nous sommes intéressés à l'architecture NDN afin de réaliser un système de parking intelligent. Ce système permet à un serveur de connaître l'état des places de stationnement, et permet aux conducteurs de réserver une place de stationnement depuis leurs véhicules. Ceci est possible grâce à des bornes de parking qui informent le serveur sur l'état des places, tout en servant de passerelle aux véhicules voulant communiquer avec le serveur.

Pour les communications mobiles du système (véhicules/passerelles), nous avons utilisé la norme ZigBee comme support de communication pour NDN. Cette technologie, bien qu'elle ne soit pas supportée dans l'implémentation actuelle de NDN, est très présente dans le domaine de l'IoT.

Ce mémoire est organisé comme suit : le Chapitre 1 présente l'architecture NDN et ses principes. Le Chapitre 2 traite de l'IoT, ses contraintes et ses caractéristiques. On y étudiera également la compatibilité de NDN avec l'IoT et on présentera certains travaux NDN sur l'IoT. Le Chapitre 3 décrit en détails l'architecture et la conception de notre parking intelligent. Le Chapitre 4 présente les détails de réalisation de

notre prototype, sa configuration et son démarrage, ainsi que les tests de ses fonctionnalités. Les équipements matériels utilisés dans notre prototype ainsi que leur configuration et les schémas de connexion sont décrits dans les Annexes A, B, C et D.

Chapitre 1

L'architecture NDN

1.1 Introduction

Bien que l'IPv6 ait été conçu pour faire face aux exigences de l'Internet du futur, les contraintes telles que la mobilité ou la sécurité ne pourront pas être satisfaites par de simples extensions de l'IP. Et donc, tout comme la téléphonie s'est révélée insuffisante pour transmettre la vidéo et la radio, l'architecture IP classique sera insuffisante pour assurer ses fonctions de base dans un avenir proche.

Ceci a poussé certains à repenser radicalement la conception d'Internet en proposant de nouvelles architectures. La National Science Foundation (NSF) finance et dirige des projets de ce type tels que Named-Data Networking (NDN). NDN suit le paradigme des Content-Centric Networking (réseaux orientés contenu) et suggère d'identifier la donnée plutôt que son détenteur, faisant ainsi de la donnée l'entité principale et de son nom son seule identifiant.

Ce chapitre traite des origines et des détails de ce concept et de l'architecture des réseaux NDN, mais aussi des fonctionnalités de base des réseaux (routage, sécurité, etc.) du point de vue de NDN. Nous présenterons également les outils utilisés pour développer et expérimenter cette architecture.

1.2 ICN et CCN : Les origines de NDN

NDN suit le paradigme des réseaux orientés contenu (appelés CCN) qui, eux-mêmes, font partie de la famille des réseaux orientés information (appelés ICN). Nous allons donc voir ces deux paradigmes et leurs principes, et nous allons situer NDN par rapport à tous les autres projets et architectures ICN.

1.2.1 ICN : Information-Centric Networking

Les principes d'ICN ou réseaux orientés information ont vu le jour en 1979 dans le projet Xanadu de Ted Nelson, sous forme d'un ensemble de règles [3]. En 2002, Brent Baccala publie un document décrivant les différences entre les réseaux orientés connexion (ou hôtes) et les réseaux orientés information et constate que dans l'Internet, le Web devenait de plus en plus une application orientée information, il dit : « *L'opération de base dans l'affichage d'une page web ne consiste plus en une connexion bout-en-bout mais c'est le fait de récupérer un bloc de données nommées.* »[4]. L'approche ICN [5] propose de remplacer le paradigme orienté hôtes de l'architecture actuelle d'Internet par une architecture dans laquelle le point central est l'information (ou la donnée). Ce principe nous permet d'outrepasser les restrictions imposées par les adresses IP, il dissocie l'information de son fournisseur et ouvre de nouveaux horizons pour les réseaux en supportant nativement la mobilité, le « storage in-network »(stockage dans le réseau), le multicast, le broadcast, etc.

Les bases d'ICN étant posées, des projets commencèrent à apparaître, dès 2006 avec le projet DONA (Data-Oriented Network Architecture) [6] de l'Université de Berkeley (Californie) qui est considéré comme la première architecture décrivant de manière claire une conception ICN. D'autres projets de même type sont apparus peu après : CCN (Content-Centric Networking) aux États-Unis, NetInf (Network of Information) et PERISP (Publish-Subscribe Internet Routing Paradigm) en Europe. La Figure 1.1 dresse un bref comparatif de ces architectures.

Bien que ces approches diffèrent dans les détails de conception, elles partagent néanmoins le même objectif, celui de développer une architecture plus adaptée aux applications actuelles d'Internet ainsi que quelques principes et propriétés [7] :

Le contenu (ou l'information ou la donnée) est modélisé par un NDO (Named Data Object), celui-ci peut être une page web, une vidéo, une photo, un document, etc. Un NDO est indépendant de la localisation (i.e. il garde toujours le même nom), de la méthode de stockage et de la méthode de transport. Par conséquent, toutes les copies d'un même NDO se valent ; un nœud peut réutiliser une de ces copies pour répondre à une requête. L'identification des NDO doit assurer une certaine unicité car c'est grâce aux identifiants que se font les requêtes et la récupération du contenu. Toutefois, un NDO est considéré comme un paquet dans certaines architectures et comme un objet dans d'autres.

Dans toutes ces architectures, la communication est initiée par le consommateur qui envoie une requête pour un certain NDO. Il est indispensable pour toutes ces architectures de vérifier l'intégrité d'une donnée, c'est-à-dire la correspondance entre le nom d'un NDO et son contenu. Des informations sur l'origine (le fournisseur ou l'auteur) peuvent également être ajoutées à un objet.

Pour gérer les noms et les correspondances nom/contenu, deux schémas de nommage sont utilisés : espace de noms hiérarchique et espace de noms plat. Le schéma hiérarchique ressemble à celui des URI, il permet l'agrégation des noms et facilite la mise à l'échelle du routage. Parfois, il peut être dit « *human-readable* » ; c'est-à-dire que l'utilisateur peut le comprendre et le manipuler directement selon ses besoins. L'espace de nom plat possède une structure sous la forme P :L où P (principal) est un champ global unique et identifie le fournisseur, et L (label) est une étiquette unique pour l'objet.

Les architectures ICN ont également en commun le caching au sein du réseau ; tous les nœuds ont potentiellement une copie des objets transitant afin de satisfaire d'autres requêtes pour ce même objet sans remonter à chaque fois jusqu'au fournisseur.

Pour récupérer une information (NDO), on distingue généralement deux phases : routage des requêtes pour les NDO vers le(s) fournisseur(s) et acheminement des NDO vers leur(s) consommateur(s).

	DONA	CCN	PSIRP	NetInf
Namespace	Flat with structure	Hierarchical	Flat with structure	Flat with structure
Name-data integrity	Signature, PKI independent	Signature, external trust source	Signature, PKI independent	Signature or content hash, PKI indep.
Human-readable names	No	Possible	No	No
Information abstraction model	No	No	No	Yes
NDO granularity	Objects	Packets	Objects	Objects
Routing aggregation	Publisher/explicit	Publisher	Scope / explicit	Publisher
Routing of NDO request	Name-based (via RHs)	Name-based	NRS (rendezvous)	Hybrid NRS and name-based
Routing of NDO	Reverse request path or direct IP connection	Reverse request path using router state	Source routing using Bloom filter	Reverse request path or direct IP connection
API	Synchronous get	Synchronous get	Publish/subscribe	Synchronous get
Transport	IP	Many including IP	IP/PSIRP	Many including IP

FIGURE 1.1 – Comparatif de quelques architectures ICN [7]

1.2.2 CCN : Content-Centric Networking

Le projet CCN [8] [9] a démarré en 2007 à PARC (Palo Alto Research Center) dans le but de développer une architecture pouvant supporter l'agrégation des réseaux, la mobilité, la mise à l'échelle et la sécurisation des échanges axée sur les données. L'architecture CCN (Content-Centric Networking) se base sur les principes d'ICN. Elle met en avant le contenu en le rendant directement identifiable de manière unique grâce à un schéma de nommage hiérarchique.

Les contenus et leurs noms deviennent le cœur de la pile des protocoles réseaux, à la place des adresses IP dans l'Internet actuel (voir Figure 1.2). En plus, l'architecture CCN introduit une couche « *strategy* » permettant l'envoie et la réception des paquets aussi bien sur un protocole de couche liaison (Ethernet, etc.) que sur le protocole IP (couche réseau) ou encore sur des protocoles de couche transport (TCP, UDP).

CCN possède également une couche « *security* » assurant l'authenticité et l'intégrité des données indépendamment de leurs localisations ou de leurs fournisseurs.

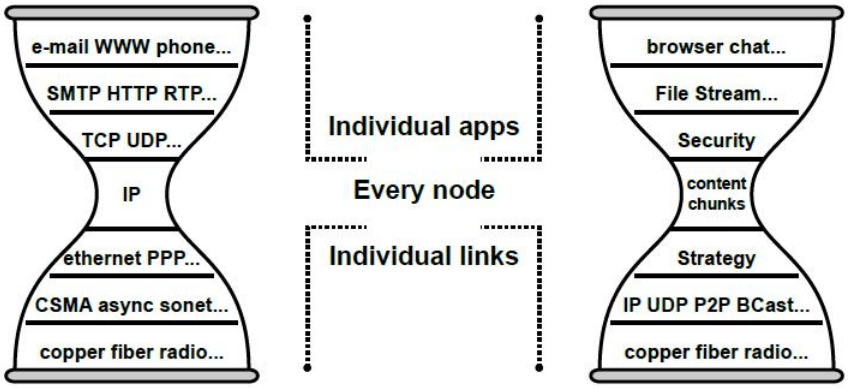

FIGURE 1.2 – Comparaison architectures CCN et IP [9]

L'implémentation actuelle de l'architecture CCN est appelée CCNx 1.0. Elle consiste en un ensemble de

protocoles assurant toutes les opérations fournies par CCN. Un réseau CCNx [10] est constitué des entités suivantes : le fournisseur de contenu, l'éditeur de contenu et le consommateur. Le fournisseur est celui qui génère le contenu en créant les données ; il peut être un individu qui prend une photo, un capteur qui collecte une donnée, etc. L'éditeur convertit ces données brutes en objets pouvant être transportés par le réseau et les sécurise grâce à des mécanismes cryptographiques, puis il les publie (à l'aide de protocoles CCNx) pour qu'ils soient connus des autres. Le consommateur utilise les protocoles CCNx pour récupérer ces objets (données) authentifiés publiés par l'éditeur.

Le protocole CCNx utilise principalement deux types de messages : Interest et Content Object. L'Interest est utilisé par le consommateur pour demander une donnée. Cette donnée est fournie sous forme d'un Content Object. Lorsque l'Interest est routé à travers le réseau, il laisse une trace dans tous les nœuds qu'il traverse ; c'est cette trace que suivra le Content Object pour atteindre le consommateur.

CCNx 1.0 utilise une représentation des messages en format TLV (Type-Length-Value). Il existe deux principes de base dans l'implémentation CCNx 1.0 : le routage et le caching dans le réseau. Un nœud effectue le routage en utilisant (a) sa FIB (Forwarding Interest Base) pour router les Interest, (b) sa PIT (Pending Interest Table) pour garder les Interests en attente jusqu'à avoir le contenu correspondant et (c) un cache optionnel (Content Store) pour sauvegarder temporairement les Content Object qu'il transmet. Le CS permet à un nœud de réutiliser un objet pour satisfaire un autre Interest sans le router plus loin, et d'assurer la retransmission d'un Content Object en cas de perte.

Un Interest comporte un nom désignant le contenu demandé, un nom d'éditeur optionnel (*KeyIdRestriction*) permettant de choisir un éditeur spécifique, et un nom de contenu optionnel (*ContentObjectHashRestriction*) ; c'est un hash permettant de cibler un contenu précis. Un champ *HopLimit* (sur 1 octet) est utilisé pour fixer le nombre maximal de sauts que l'Interest peut effectuer afin d'éviter le bouclage des Interest.

Un Content Object comporte un nom et un contenu (*Payload*). Un champ *Validator* peut également y être inclus pour fournir des informations sur la sécurité, il décrit l'algorithme de vérification utilisé et la valeur calculée (MAC, signature, etc.), l'algorithme de validation indique également la clé (*KeyId*) utilisée qui identifie l'éditeur du Content Object. Le Content Object comporte également un *ContentObjectHash* calculé à partir de tout le contenu de l'objet.

Un Interest reçoit au plus un Content Object (0 si erreur). Pour décider si un Content Object correspond à un Interest et le renvoyer comme réponse, un nœud aura à effectuer deux types d'opérations :

- L'égalité entre le nom de l'Interest et celui du Content Object. Si l'Interest indique un *KeyIdRestriction* spécifique alors il doit également correspondre au *KeyId* contenu dans le Content Object.
- Si l'Interest indique un *ContentObjectHashRestriction*, alors le nœud devra recalculer le hash de l'objet pour vérifier qu'il correspond bien à l'objet souhaité.

Le routage et l'acheminement se font à l'aide de trois tables : la FIB (Forwarding Interest Table), la PIT (Pending Interest Table) et le CS (Content Store). La FIB est peuplée manuellement avec des routes statiques ou avec des routes obtenues par des algorithmes de routage. Elle contient des préfixes associés aux interfaces correspondantes avec un coût. Les Interest routés en attente de réponse sont gardés dans la PIT de manière à ce que le Content Object retourné puisse être acheminé au bon consommateur en suivant le chemin inverse de l'Interest. Le CS sert à garder momentanément les Content Object ayant servi récemment pour satisfaire d'éventuels futurs Interest.

Lorsque plusieurs Interest pour un même Content Object arrivent à un nœud (i.e. même nom, même *KeyIdRestriction* si présent et même *ContentObjectHashRestriction* si présent), une agrégation est faite : seul le

premier est routé (i.e. enregistré dans la PIT) et les suivants sont supprimés mais l'interface par laquelle ils sont arrivés est ajoutée à la PIT.

Les signatures ne sont vérifiées que par les systèmes à l'extrémité de la communication (i.e. le consommateur et l'éditeur). Toutefois, chaque nœud doit recalculer le *ContentObjectHash* d'un Content Object si l'Interest qui l'a demandé spécifie un *ContentObjectHashRestriction*.

Un Content Object peut avoir une taille maximale de 64 KB, et la MTU dans les réseaux est de 1500 octets en général (pour Ethernet) ou 1280 octets (pour IPv6). Quand un Content Object est plus grand que la MTU, le système le divise automatiquement en plusieurs morceaux correspondant à la MTU grâce à la fragmentation. Toutefois, afin d'éviter la fragmentation, CCNx propose d'autres mécanismes permettant de gérer la subdivision d'un objet volumineux : Le « *Chunking* » utilise des étiquettes spécifiques pour indiquer au consommateur qu'un ensemble de données constitue le même objet et lui permettre de les remettre dans le bon ordre. Le « *Manifest* » est aussi utilisé pour décrire une agrégation d'objets constituant une même unité de donnée ainsi que des méta-informations sur ses objets.

La Figure 1.3 montre les deux grands paradigmes des réseaux informatiques et les principales architectures qui suivent ces paradigmes. Elle montre également la place de l'architecture NDN parmi ces architectures. NDN est dérivé de l'architecture CCN et repose en grande partie sur ses principes, mais il est enrichi d'autres protocoles (routages, etc.), de mécanismes (Modèles de sécurité, etc.), de services (NDNS, etc.) et d'applications. La section suivante présente en détails l'architecture NDN et ses principes.

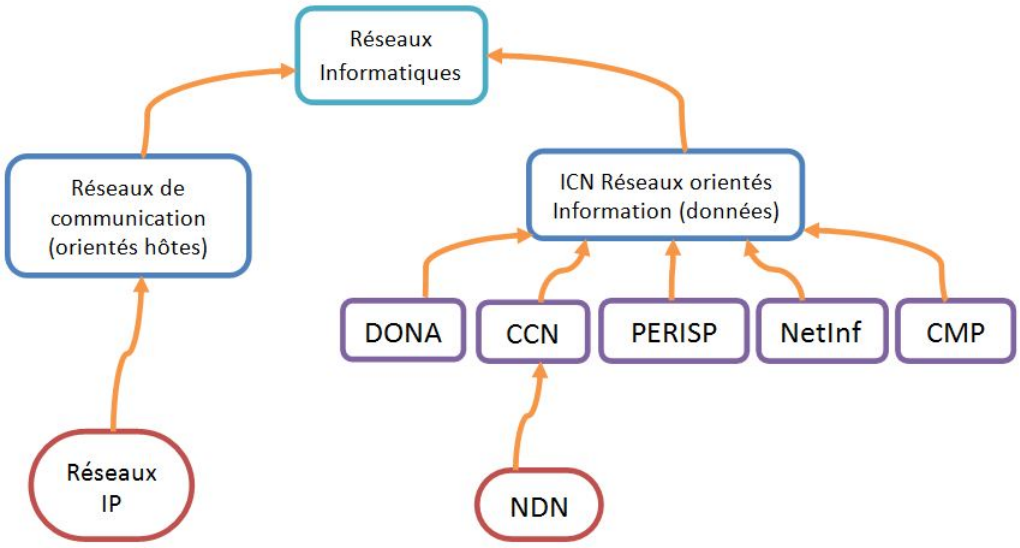

FIGURE 1.3 – NDN et les principales architectures réseau

1.3 NDN : Named-Data Networking

1.3.1 Le projet NDN

Le projet NDN [11] a été fondé par la NSF (National Science Foundation) en septembre 2010, dans le cadre du programme FIA (Future Internet Architecture) [12]. Le but de ce projet est de développer une nouvelle architecture d'Internet mieux adaptée aux modèles de communication émergents et aux nouvelles applications.

Étant inspiré de l'architecture CCN, NDN fait également de la donnée (et non des hôtes) l'entité principale du réseau ; ce qui facilite la sécurisation de la donnée, la gestion de son unicité et offre un espace de nom illimité. Pour illustrer ce principe, imaginons que nous voulons regarder une vidéo se trouvant chez *Youtube*, au lieu de se connecter à l'adresse du site pour avoir la vidéo, nous allons simplement et directement demander cette vidéo en envoyant une requête contenant son nom naturel (*e.g. youtube/videos/maVideo.mpg*).

Une telle vision d'Internet facilitera la tâche aux développeurs en leur permettant de créer des applications sans se soucier des routines de sécurité (car NDN s'en occupe déjà) et sans avoir recours aux middlwares ou autres couches intermédiaires, et aux utilisateurs en leur offrant un plus grand contrôle sur le contenu mais surtout une meilleure protection de leur vie privée. Mais cette approche engendre des défis techniques à relever pour faire de NDN un standard reconnu, ce sont ces défis qui font l'objet d'étude et de recherche du projet NDN [13] [14] [15] : le schéma de nommage, le routage, l' acheminement, les modèles de vérification des données et mécanismes de sécurité, la protection du contenu et de la vie privée, etc.

1.3.2 L'architecture

NDN conserve la même architecture Hourglass (en forme de sablier, Figure 1.4) que TCP/IP. Partant de cette architecture Hourglass, NDN regroupe en son centre (*thin waist*) tous les aspects importants du réseau jusque-là gérés par des couches supérieurs : la sécurité est implémentée au cœur de l'architecture ; ceci en signant et vérifiant systématiquement les données. De même pour le contrôle de flux qui est directement géré par la couche réseau. Dans l'architecture actuelle d'Internet, ces deux pratiques ne sont pas gérées par la couche réseau (IP) mais ont été pensées plus tard et implémentées dans certains protocoles de couches supérieurs (transport et application).

Un autre principe de base de NDN est le bout-en-bout [1], qui permet de réaliser des applications réseaux plus fiables et plus sécurisées mais également augmente la neutralité du réseau et minimise sa vulnérabilité.

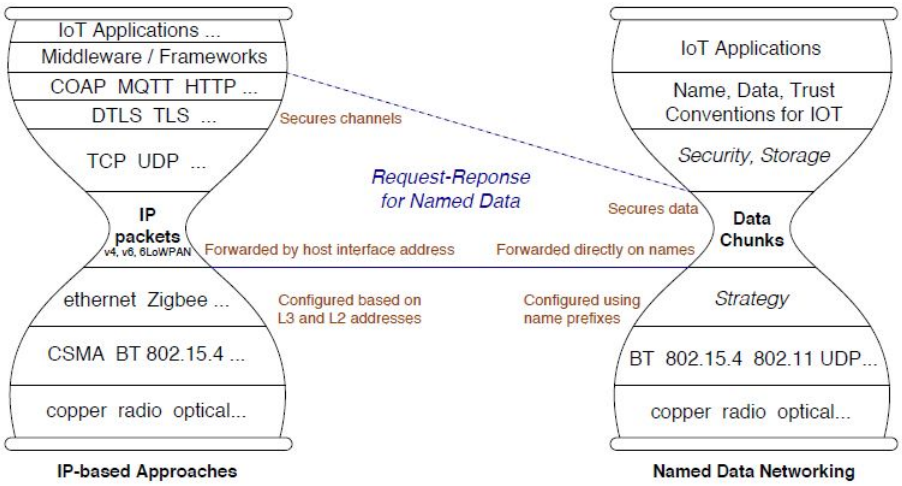

FIGURE 1.4 – Architecture Hourglass NDN et IP [16]

1. Il énonce que « plutôt que d'installer l'intelligence au cœur du réseau, il faut la situer aux extrémités : les ordinateurs au sein du réseau n'ont à exécuter que les fonctions très simples qui sont nécessaires pour les applications les plus diverses, alors que les fonctions qui sont requises par certaines applications spécifiques seulement doivent être exécutées en bordure de réseau. Ainsi, la complexité et l'intelligence du réseau sont repoussées vers ses lisières. Des réseaux simples pour des applications intelligentes. » (Wikipedia)

1.3.3 La communication

La communication dans NDN est initiée par le consommateur (i.e. celui qui demande la donnée), elle se déroule de la manière suivante (Figure 1.5) :

1. Le consommateur désirant une donnée, envoie un paquet Interest portant, entre autres, le nom de la donnée (*e.g. ummto/videos/v1.mpg*).

2. Le routeur qui reçoit cet Interest garde en mémoire l'interface ou l'application (appelées globalement *Face*) par laquelle il est arrivé et dirige le paquet vers le nœud suivant en se basant sur sa FIB (Forwarding Interest Table), elle-même remplie et maintenue à jour par un protocole de routage et/ou manuellement. Le routeur garde temporairement l'Interest et l'interface (Face) par laquelle il est arrivé dans sa PIT (Pending Interest Table).

3. Une fois l'Interest arrivé au nœud contenant la donnée, celui-ci renvoi un paquet Data contenant, entre autre, le nom de la donnée et son contenu. Ce paquet Data suit le chemin inverse emprunté par l'Interest (grâce à la PIT). Un Data est la réponse à un seul Interest.

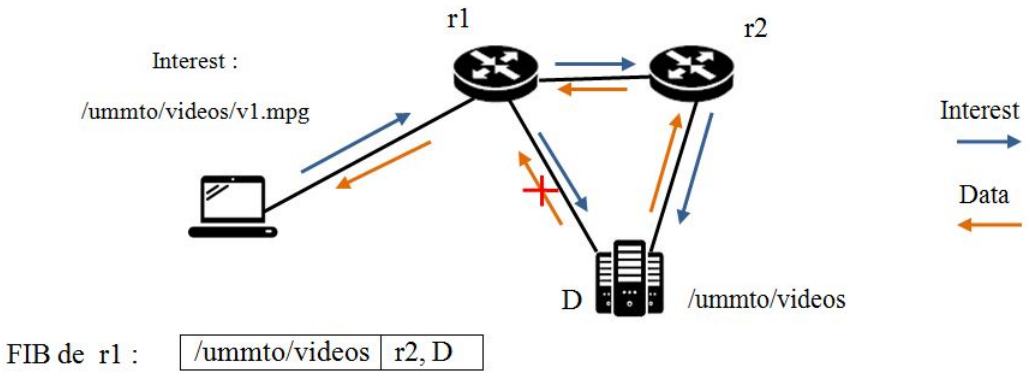

FIGURE 1.5 – Communication dans un réseau NDN

Aucun des deux paquets impliqués dans la communication ne porte d'adresse source ou destination. Le routage du paquet Interest se base sur son nom, et le paquet Data repasse par les mêmes nœuds (dans l'ordre inverse) en suivant les traces laissées par l'Interest. En effet, chaque routeur NDN sauvegarde temporairement les paquets Interest dans sa PIT dans laquelle chaque entrée contient le nom de l'Interest ainsi que l'interface (ou les interfaces) par laquelle il est arrivé. Quand un paquet Data arrive, la table PIT est consultée et les données sont acheminées par les interfaces correspondantes.

Après ça, le routeur supprime l'Interest de sa table PIT, et garde temporairement le paquet Data dans son CS (Content Store). Le Content Store est comparable au cache Web : il sauvegarde temporairement les paquets Data transitant par un nœud, suivant une politique de remplacement basée sur le temps, le nombre de paquets, etc. Avant d'acheminer un paquet Interest, le routeur consulte d'abord son CS pour vérifier s'il dispose de la donnée souhaitée. Si ce n'est pas le cas il le dirige vers un nœud voisin.

De ce fait, NDN fourni également le caching. Les paquets Data étant indépendant de toute adresse source ou destination, ils peuvent être réutilisés par les routeurs pour satisfaire d'autres Interests sans avoir à remonter jusqu'au fournisseur. Le caching fait que NDN peut intrinsèquement supporter des fonctionnalités telles que la retransmission des paquets perdus, la distribution de contenu, le multicast, la mobilité et la tolérance au pannes. Pour montrer la pertinence de ce principe, imaginons un utilisateur regardant une vidéo en streaming dans sa voiture : étant mobile, il peut demander une donnée puis changer de réseau. La donnée arrivera à son ancienne localisation et sera supprimée, mais elle aura été mise en cache dans tous les nœuds qu'elle a

traversé. Quand l'utilisateur renouvèlera son Interest, la donnée lui parviendra depuis un routeur très proche et donc avec un temps de coupure infime.

Cependant, le caching soulève des préoccupations concernant la vie privée. Les noms NDN sont explicites, ce qui rend les données plus facile à analyser. Mais NDN supprime toute donnée faisant référence à l'identité des utilisateurs (en appliquant des politiques de gestion du cache développées par les FAI par exemple). À moins d'être relié à l'utilisateur directement par une liaison point-à-point, le routeur saura uniquement qu'un utilisateur U a demandé une donnée D, sans connaître son identité ou sa localisation.
La Figure 1.6 montre une représentation des structures de données de base de NDN (FIB, PIT et CS).

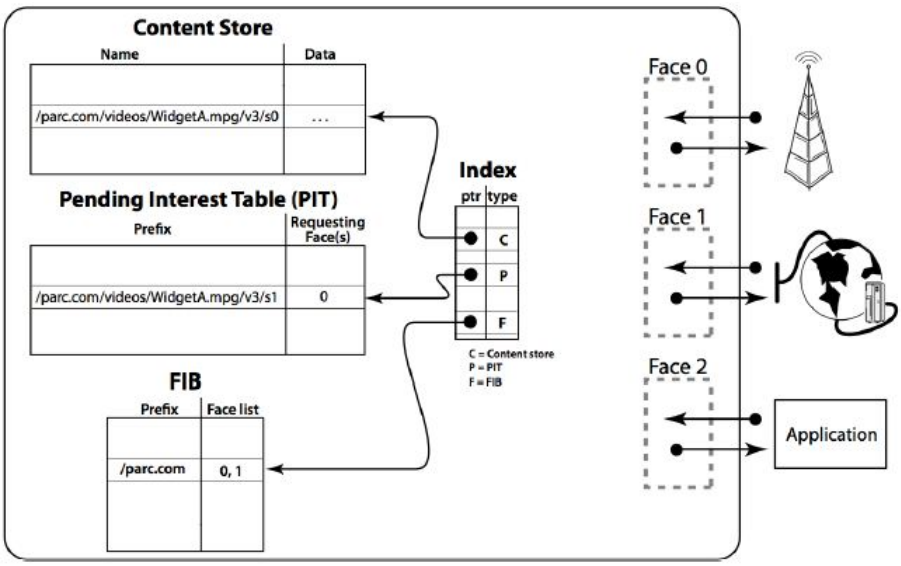

FIGURE 1.6 – PIT, FIB et CS dans NDN [13]

Rôle de la PIT

La PIT contient les Interests transmis en attente de réponse et les interfaces par lesquelles ils sont arrivés. Elle est utilisée pour acheminer les paquets Data vers leurs consommateurs. L'efficacité et la rapidité de l'acheminement dépendent de la durée de vie des entrées de la PIT. Si un Interest est supprimé de la PIT avant qu'il n'ait reçu de réponse, le Data correspondant sera supprimé à son arrivée et c'est au consommateur de renouveler son Interest.

La PIT peut faire correspondre à chaque Interest plusieurs interfaces, c'est donc elle qui permet de faire l'agrégation des Interests et le multicast. Le routeur peut également contrôler le flux en contrôlant la taille de sa PIT, ce qui permet de se passer des protocoles de couche transport qui offrent traditionnellement cette fonctionnalité. Mais surtout, l'analyse du contenu de la PIT offre un moyen simple pour détecter et/ou éviter des attaques de dénis de service : limiter le nombre d'entrées, limiter la durée de vie des entrées, étudier les interfaces ayant reçu des paquets suspects, etc.

Transport

L'architecture NDN n'a pas de couche transport explicite. Les fonctionnalités fournies par cette couche sont distribuées entre la couche réseau et la couche application : Le multiplexage/démultiplexage des données des applications est fait directement en se basant sur les noms des données, et la vérification des données est gérée par l'application en utilisant les fonctionnalités offertes par la couche réseau (signatures, certificats,

etc.). Le contrôle de flux et la sécurité sont implémentés dans la couche réseau NDN.

Dans NDN, le fournisseur et le consommateur exécutent des fonctionnalités différentes : Le fournisseur s'occupe de démultipléxer les Interests, segmenter les données à envoyer, signer et chiffrer les paquets Data et vérifier la validité de ses données. Le consommateur s'occupe de générer les Interests, de rassembler tous les segments de données reçus et de vérifier l'authenticité et la validité des données.

Dans le transport et notamment la segmentation des données, NDN et TCP/IP différent sur deux points : premièrement, la segmentation dans TCP/IP ne tient pas compte des unités de données de l'application (ADU) alors que dans NDN, la segmentation d'une donnée se fait suivant ses ADU et les noms donnés aux paquets expriment bien les différents ADU (Figure 1.7). Deuxièmement, dans TCP/IP, si le segment d'une ADU n'est pas reçu alors les segments des ADU suivants sont inutilisables. Dans NDN, une application produit et transmet ses ADU indépendamment les uns des autres. Ceci offre une plus grande flexibilité dans la manipulation des données.

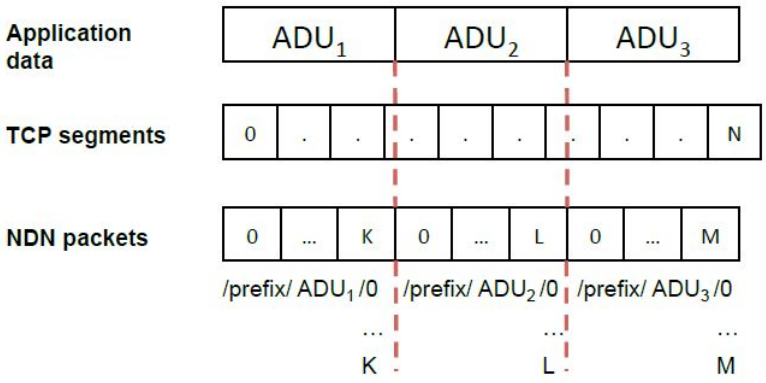

FIGURE 1.7 – Segmentation NDN et segmentation TCP [14]

1.3.4 Les paquets

Actuellement, les paquets NDN sont sous le format TLV. NDN utilise deux paquets différents [17] [18] (Figure 1.8) : Les paquets Interest et les paquets Data. Les paquets Interest sont définis en TLV de la manière suivante :

```
Interest ::= INTEREST-TYPE TLV-LENGTH
               Name
               Selectors?
               Nonce
               InterestLifetime?
```

- Name : Champ obligatoire. Représente le nom de la donnée.
- Selectors : Permet d'affiner le contrôle sur la donnée demandée par l'application. Il est placé juste après le champ *Name* pour faciliter l'implémentation en adressant des blocs mémoires successifs (utile pour les systèmes embarqués) car ces deux champs sont utilisés ensemble dans les PIT.
  D'autre part, le champ *Selectors* est lui-même un groupe TLV regroupant différents sélecteurs, ce qui permet au système de facilement l'ignorer et passer directement au champ *Nonce* utilisé avec *Name* pour détecter le bouclage des Interests.
  Selectors est facultatif, mais s'il existe il doit contenir au moins un sélecteur. Voici quelques sélecteurs possibles :

- PublisherPublicKeyLocator : Spécifie le nom de la clé utilisée pour signer le paquet Data demandé. C'est une façon de contrôler la provenance des données.
- Exclude : Permet au consommateur de spécifier une liste et/ou une suite de noms qui ne doivent PAS suivre le préfixe du nom de la donnée demandée. Par exemple, si le nom d'une donnée a le préfixe *'/ndn/edu'* et *Exclude* est *'ucla'*, alors ni le fournisseur de la donnée ni les routeurs ne doivent acheminer des paquets Data ayant le préfixe *'/ndn/edu/ucla'*.

- Nonce : Champ obligatoire. *Nonce* est une valeur aléatoire. La combinaison de *Name* et *Nonce* identifie un paquet Interest de manière unique. Cette combinaison est utilisée pour détecter la redondance d'un Interest.

- InterestLifetime : Champ optionnel. Il spécifie la durée de vie d'un Interest (le temps maximal que peut rester un Interest dans la PIT) en millisecondes. Ce temps est relatif au moment de l'arrivée de l'Interest dans un routeur. Un routeur peut diminuer cette valeur en soustrayant le temps que le paquet a passé dans le nœud avant d'être routé, mais n'est pas obligé de le faire. Cette valeur est spécifiée par l'application. Si aucune valeur n'est donnée, la valeur par défaut est 4000 ms (4 secondes).

Les paquets Data sont définis en TLV de la manière suivante :

```
Data ::= DATA-TLV TLV-LENGTH
           Name
           MetaInfo
           Content
           Signature
```

Le paquet Data représente une quelconque donnée (champ *Content*) ainsi que son nom (*Name*), un champ d'informations (*MetaInfo*) et la signature numérique (*Signature*) des trois champs précédents. La signature est le dernier élément du paquet, ceci facilite l'implémentation et la vérification car le calcul de la signature est basé sur les trois premiers champs.

- Name : Même sens que pour les paquets Interest.

- MetaInfo : Contient des informations sur la donnée. Il est définit comme suit :

  ```
  MetaInfo ::= META-INFO-TYPE TLV-LENGTH
  ContentType?
  FreshnessPeriod?
  FinalBlockId?
  ```

  - ContentType : Optionnel. Indique la nature du contenu (Contenu ordinaire, LINK, Clé publique, NACK).
  - FreshnessPeriod : Optionnel. Ce champ indique au nœud le temps (en millisecondes) à attendre après l'arrivée de cette donnée pour la marquer obsolète. Toutefois, une donnée obsolète reste valide ; l'expiration du *FreshnessPeriod* indique uniquement que le fournisseur a pu produire une autre donnée plus récente. Si aucune valeur n'est spécifiée, la donnée ne sera jamais marquée obsolète (i.e. *FreshnessPeriod* infini). Le Content Store associe à chaque paquet Data une valeur sur sa fraicheur « *staleness bit* ». La valeur initiale pour un paquet Data fraichement reçu est « *not stale* ». Si le paquet indique une valeur *FreshnessPeriod*, il sera conservé dans le Content Store pour cette durée et sera marqué comme obsolète après. Si un Interest contient un sélecteur *MustBeFresh*, signifiant que la donnée doit être récente, un paquet Data ayant une quelconque valeur du *staleness bit* (autre que la valeur initiale) dans le Content Store ne peut être envoyé comme réponse à cet Interest. Dans ce cas, le nœud agit somme si cette donnée n'était pas disponible et route l'Interest vers d'autres nœuds. Les données marquées obsolètes font partie des premières données à éliminer lorsque le Content Store atteint son quota maximal.

- FinalBlockId : Optionnel. Identifie le dernier bloc dans une suite de fragments. Il apparait dans le dernier bloc ou dans d'autres fragments afin de prévenir le consommateur que la suite de donnée arrive à la fin. Sa valeur est égale au dernier nom de composant du bloc final.

- Content : C'est le contenu du paquet.
- Signature : La signature est définie en deux blocs TLV consécutifs :
  - SignatureInfo : Contient tout ce qui est inclus dans le calcul de la signature. Il décrit la signature, l'algorithme de signature et toute autre information permettant d'obtenir le certificat du signataire.
  - SignatureValue : Représente la valeur de la signature.

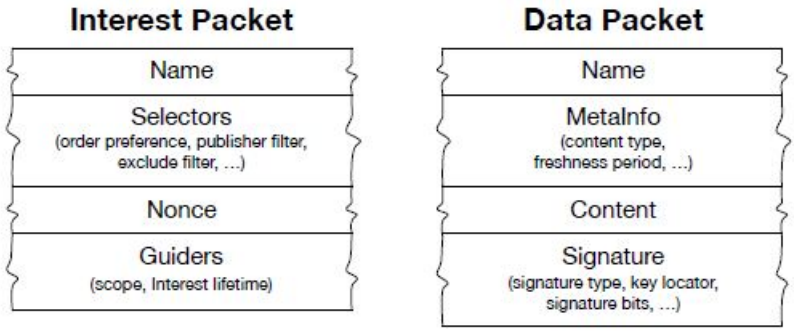

FIGURE 1.8 – Représentation des paquets Interest et Data [14]

1.3.5 Le nommage

Dans NDN, les contenus sont identifiés par des noms hiérarchiques. C'est une suite de noms de composants séparés par des *'/'* [19] : Un nom dans NDN a la définition TLV suivante :

```
Name ::= NAME-TYPE TLV-LENGTH NameComponent*
NameComponent ::= GenericNameComponent | ImplicitSha256DigestComponent
GenericNameComponent ::= NAME-COMPONENT-TYPE TLV-LENGTH BYTE*
ImplicitSha256DigestComponent ::= IMPLICIT-SHA256-DIGEST-COMPONENT-TYPE
TLV-LENGTH(=32) BYTE{32}
```

- GenericNameComponent : C'est le nom du composant. C'est un nom naturel sans restriction sur la valeur.
- ImplicitSha256DigestComponent : C'est un hash (SHA256) implicite sur 32 octets.

Le nom complet de chaque paquet Data comporte à la fin (comme dernier composant) un hash (*ImplicitSha256DigestComponent*) qui rend le paquet unique. Ce hash peut également apparaitre dans le nom d'un Interest (toujours à la fin) afin de demander un paquet Data spécifique, ou bien dans le sélecteur *Exclude* pour exclure un paquet Data. Il est dit implicite car il n'est pas inclus dans la transmission des paquets Data, en cas de besoin, il est recalculé par les nœuds à partir du contenu du paquet (Data). Ce hash permet d'identifier un paquet Data de manière univoque, et pour l'Interest de cibler un paquet Data spécifique et aucun autre.

Les noms NDN son inconnus du réseau (i.e. les routeurs n'interprètent pas le nom global). Ceci donne la possibilité aux applications d'utiliser leur propre schéma de nommage et de l'adapter indépendamment du réseau. Nous avons vu plus haut que les noms étaient hiérarchiques ; une vidéo produite par *Youtube* peut

avoir le nom : *'youtube/videos/uneVideo.mpg'*. Cette structure hiérarchique offre la possibilité de classer des ensembles et/ou des séries de données puis de les retrouver très facilement ; dans l'exemple précédent, le segment *S* de la version *1* de la vidéo pourra être nommé : *'youtube/videos/uneVideo.mpg/1/S'*.

Mais le plus important c'est qu'un nommage hiérarchique permet de faire l'agrégation des réseaux et donc de réaliser un routage extensible supportant la mise à l'échelle (i.e. un très grand nombre de nœuds et de données).

Les noms n'ont pas besoin d'être absolument uniques, mais doivent assurer une certaine unicité globale permettant de réaliser le routage des Interests sur de grandes portées à travers le réseau.

Pour les données générées dynamiquement, le consommateur doit pouvoir trouver les noms de ces données de manière exacte. Pour cela, deux possibilités existent :

- Utiliser des algorithmes déterministes partagés entre le fournisseur et le consommateur, qui permettront à tous deux d'arriver aux mêmes noms.
- Le consommateur peut se baser sur un nom partiel pour retrouver d'autres noms, notamment pour des données organisées de manière séquentielle. Par exemple, si on demande le nom *'/parc/videos/WidgetA.mpg'* et qu'on reçoit une donnée portant le nom *'/parc/videos/WidgetA.mpg/1/1'*, on sait qu'on pourra spécifier d'autre segments lors des prochaines requêtes en se basant sur la convention de nommage établie au préalable.

Le nommage est aujourd'hui l'aspect le plus important de NDN car toutes les autres fonctionnalités de NDN (architecture, routage, découverte des noms, etc.) dépendent étroitement de la façon dont les données et les domaines sont nommés.

1.3.6 Le routage

Le routage NDN se base sur les noms, ceci libère NDN des problèmes posés traditionnellement par l'architecture IP : la limitation du nombre d'adresses, le NAT et la mobilité des nœuds. Il n'y a pas de limitation des adresses disponibles car l'espace de noms NDN est hiérarchique et n'a aucune restriction (les noms ne sont pas dans des intervalles). Le NAT n'est pas nécessaire car les hôtes n'ont pas à communiquer leurs adresses pour échanger des données. La mobilité des nœuds n'affecte pas une communication tant qu'elle utilise des noms de données cohérents [20]. L'affectation et la gestion des adresses logiques n'est pas nécessaire dans les réseaux, ce qui est utile dans le cas des objets connectés dans une même maison par exemple.

Le protocole de routage peut fonctionner de manière similaire à celui de l'IP. Au lieu d'échanger des préfixes d'adresses IP, les routeurs NDN échangent des préfixes de noms de données. Les annonces propagées dans le réseau grâce à ces protocoles permettent à chaque routeur de construire sa FIB. Donc, les protocoles traditionnels de l'IP (OSPF, BGP, etc.) peuvent être adaptés à NDN pour manipuler des noms au lieu de manipuler des adresses. Le principal protocole de routage NDN est NLSR (Named-data Link State Routing Protocol) [21].

Les routeurs manipulent les noms comme une suite de composants. Pour router un Interest, ils cherchent dans la FIB le plus long préfixe correspondant au nom (champ *Name*) du paquet. Par exemple : *'/parc/videos/WidgetA.mpg'* correspond à *'/parc/videos'* et *'/parc'* dans la FIB, mais *'/parc/videos'* est le plus long préfixe.

NDN supporte intrinsèquement le multipath. L'IP n'aiguille un paquet que sur une seule interface pour éviter les redondances et le bouclage des paquets. Dans NDN, les paquets ne peuvent pas boucler continuellement, les redondances sont facilement repérables car les champs *Name* et *Nonce* ensemble forment un

identifiant unique de l'Interest. Quand aux paquets Data, pas de risque non plus car ils ne font que suivre le chemin inverse de l'Interest. Donc, un routeur peut aiguiller un Interest sur plusieurs interfaces sans se soucier des redondances. Le premier paquet Data correspondant à l'Interest est acheminé et gardé en cache. Si des copies du même Data arrivent plus tard, elles seront supprimées.
Le multipath offre une bonne solution à la gestion du trafic dans le réseau et la qualité de service. Par exemple, un routeur peut aiguiller un Interest sur toutes les interfaces possibles (d'après la FIB) au début, puis en se basant sur la façon dont les paquets Data arrivent, il pourra choisir l'interface la plus rentable pour les prochains Interests.

À grande échelle, les espaces de noms illimités donnent lieu à des tables de routage très volumineuses et difficiles à gérer. Des systèmes tels que SNAMP (Secure NAmespace MaPping) [22] sont proposés et étudiés pour prendre en charge la mise à l'échelle du routage NDN.

1.3.7 La sécurité

La sécurité du routage est également améliorée. La signature de toutes les données, y compris les informations de routage échangées, empêche leur falsification et leur détournement. Le multipath atténue l'effet des routes détournés ; en employant d'autres routes pour avoir la donnée, un routeur peut détecter les « mauvaises » routes. En plus d'être signés, les paquets Data ne contiennent pas d'adresse destination, ce qui rend difficile l'envoie de code malicieux à un hôte spécifique. Dans NDN, les seules menaces possibles sont les attaques de dénis de service.

La signature des données et sa vérification sont implémentées dans l'architecture NDN. Chaque donnée est liée à son nom et signée. La signature des données est obligatoire. La combinaison de la signature et des informations sur le signataire permet de déterminer l'origine de la donnée. Par conséquent, la confiance accordée ou non à une donnée est indépendante de l'endroit et de la manière dont cette donnée a été obtenue (important dans la mobilité).

La distribution des clés est facilitée car une clé est envoyée sous forme d'un paquet Data lui-même signé [23]. Par exemple, si un paquet Data signé par une autorité contient une clé publique, alors la clé et sa signature représentent un certificat de clé publique. Aussi, NDN ayant été conçu sur le principe de bout-en-bout, il facilite la mise en place de modèles de vérifications entre consommateurs et fournisseurs au niveau application.

L'authenticité d'une donnée se vérifie chez le consommateur, mais avant de vérifier que la signature est valide et donc que la donnée n'a pas été modifiée en cours d'acheminement, le consommateur doit vérifier que la clé qui a signé la donnée est authentique et autorisée à fournir cette donnée. Pour cela, on vérifie que le nom de la donnée correspond bien au nom de la clé qui a signé cette donnée. Un modèle de sécurité est utilisé. Il consiste en un ensemble de règles (*rules*) liant de manière hiérarchique le nom de la donnée à la clé qui doit la signer. À ces règles s'ajoutent les clés de confiance auto-signées (*trust anchor*) correspondant généralement à la clé ayant la plus grande portée (administrateur du domaine, autorité de confiance, etc.). Ce modèle permet de mettre en place des règles permettant de retrouver les clés de confiance à partir des noms de données et des noms de données possibles à partir d'un nom de clé.

À la réception d'un paquet Data, la vérification se fait en commençant par le nom de la clé qui a signé le paquet, puis en suivant de manière stricte l'enchainement des règles, on vérifie que chaque clé est bien signée par la clé attendue, jusqu'à atteindre une clé de confiance (*trust anchor*). Si avant d'atteindre un *trust anchor*, un nom de clé attendu par une règle ne correspond pas au nom trouvé, la validation échoue (Figure 1.9).

Avec un tel modèle, le producteur et le consommateur des données d'une même application peuvent

adopter et implémenter le même modèle de sécurité leur permettant de vérifier réciproquement leurs données.

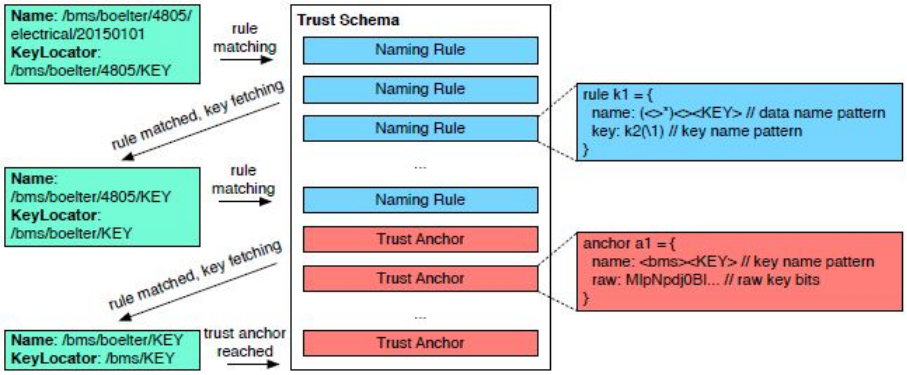

FIGURE 1.9 – Exemple de modèle de sécurité (Trust Model) [14]

Le modèle de sécurité (*Trust Model*) est étudié en détails dans un document plus récent [24]. On y présente notamment un outil et une méthodologie permettant de construire un modèle de sécurité : Cet outil supporte une syntaxe (expérimentale) semblable eux expressions régulières, avec laquelle on peut exprimer les règles de sécurité (*rules*) et les clés de confiance auto-signées (*anchor*) avec une grande simplicité, ce qui nous offre la possibilité de mettre en place des modèles de sécurité souples et réutilisables dans des applications ayant les mêmes besoins en sécurité. Les Figures 1.10 et 1.11 montrent respectivement les symboles de cette syntaxe et un exemple de *Trust Model* avec cette syntaxe.

`<name>`	Match name component `name`
`<>`	Match any single name component, i.e., wildcard
`<name><>`	Match name component `name` followed by any single name component
`<>*`	Match any sequence of name components
`(...)`	Match pattern inside the brakets and assign it as an indexed sub-pattern
`\n`	Reference to the n-th indexed sub-pattern
`[func]`	Match (for authentication) or specialize (for signing) name component according to function `func` defined pattern, i.e., wildcard specializer
`rule(arg1, ...)`	Derive a more specific name pattern from `rule`'s data name pattern with arguments `arg1`, ...

FIGURE 1.10 – Syntaxe des Trust Model [24]

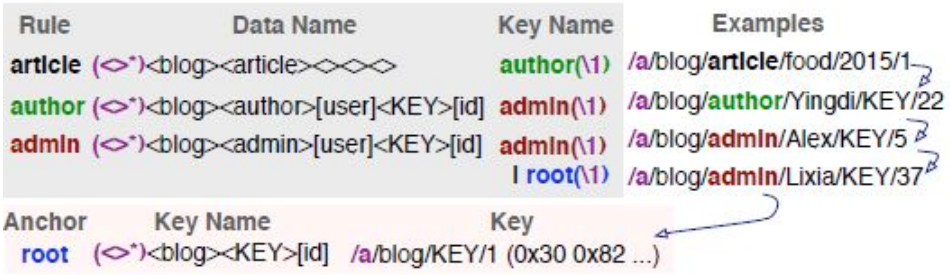

FIGURE 1.11 – Exemple de Trust Model [24]

Dans cet exemple, la règle « *article* » dit que tous les noms de données formés de *'blog'* suivi de *'article'* avec 0 ou plusieurs composants avant (exprimé avec (<>*)) et exactement 3 composants après, sont associés

aux noms définis dans la règle « *author* » qui prend comme paramètre le premier composant de la règle « *article* », c'est-à-dire les composants avant *'blog'* ; les éléments entre parenthèses sont considérés comme un seul composant. Même principe pour les autres règles jusqu'à atteindre « *root* » qui associe le nom de la clé à une valeur de clé certifiée de confiance.

1.4 Outils pour NDN

1.4.1 NDN-CCL

NDN Common Client Libraries (CCL) [25] [26] contient un lot d'API, dans différents langages, permettant d'implémenter des applications communicant avec NDN. Actuellement, les langages supportés sont C++, Python, JavaScript et Java. Toutes les entités présentes dans NDN telles que l'Interest, le Data et les noms sont modélisées dans des objets. Les noms des classes, des méthodes et des paramètres sont les mêmes dans tous les langages. La Figure 1.12 montre les entités qui composent cette bibliothèque.

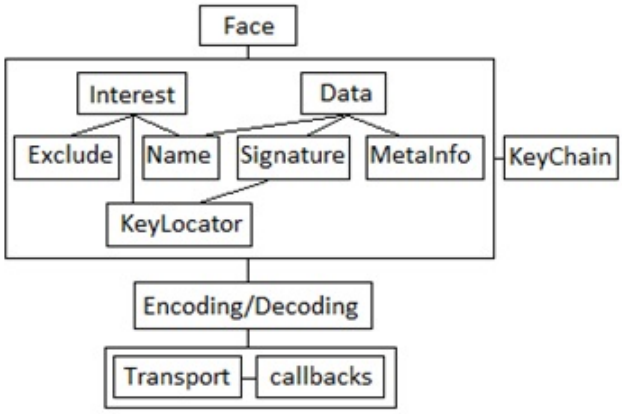

FIGURE 1.12 – Architecture de NDN-CCL [25]

NDN peut également être testé sur de simples microcontrôleurs. Pour cela, une bibliothèque a été développée pour la plateforme Arduino, offrant la possibilité d'adresser une requête à un capteur local, encapsuler le résultat dans le format TLV des paquets, ajouter la signature, former le paquet Data et répondre aux Interests.

1.4.2 NDN-CXX

L'implémentation de NDN-CCL pour le langage C++ est appelée NDN-CPP. En plus de cette bibliothèque, l'équipe du projet NDN développe et maintient la bibliothèque ndn-cxx (C++ with eXperimental eXtensions). Elle est utilisée dans l'implémentation de NFD, ndnSIM et pour tester de nouvelles fonctionnalités de l'architecture NDN qui seront ajoutées plus tard dans les bibliothèques NDN-CCL.

1.4.3 NDN Forwarding Deamon (NFD)

NFD [27] [28] a été développé pour faciliter l'expérimentation de l'architecture NDN dans un contexte réel et rendre plus accessible son utilisation et son amélioration. C'est l'un des composants les plus importants de NDN car c'est lui qui implémente les traitements de base des paquets (il supporte le format TLV) ; sa fonction est de router les paquets Interest et d'acheminer les Data.

La première version est apparue en Aout 2014. Il est open source et disponible également sous forme de packages. Il fonctionne sur Ubuntu, Mac OSX (avec MacPorts), Fedora, Gentoo, FreeBSD mais aussi sur

des plateformes embarquées telles que Raspberry Pi et Galileo.
La Figure 1.13 montre les composants de NFD :

- ndn-cxx Library, Core, et Tools : Contiennent les fonctionnalités utilisées communément par les composants NFD (manipulation des structures de données, fonctions de hashage, manipulation des paquets, etc.).
- Faces : Permet l'envoie/réception des paquets en faisant abstraction des supports ou des protocoles de couches inférieures utilisés. Supporte Ethernet, UDP, TCP et WebSocket.
- Tables : Implémente le CS, la PIT, la FIB et d'autres structures de données contenant les paramètres des stratégies d'acheminement des paquets.
- Forwarding : Implémente la stratégie de routage. C'est un ensemble de cellules (pipelines) qui effectuent les traitements de base sur les paquets en interagissant avec Faces, Tables et Strategies. On peut définir une stratégie de routage différente dans chaque nœud et pour chaque espace de noms.
- Management : Protocole permettant d'effectuer la configuration du NFD et de vérifier son état.
- RIB Management : Permet de gérer la RIB (Routing Information Base) et la façon dont la FIB est mise à jour grâce au contenu de celle-ci. La RIB est peuplé par les informations provenant des applications et des protocoles de routage.

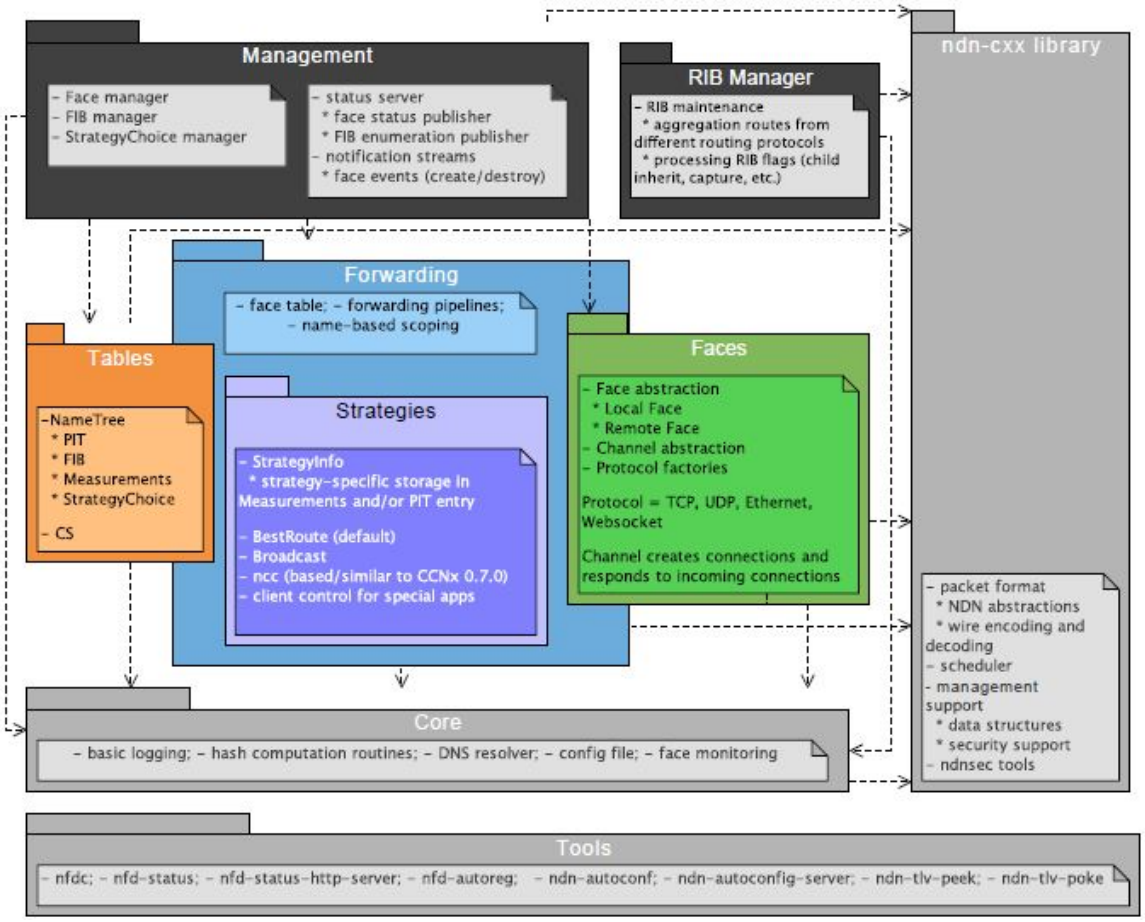

FIGURE 1.13 – Composants de NFD [27]

1.4.4 ndnSIM

Les nombreuses et différentes fonctionnalités de NDN ont besoin d'être testées à grande échelle ; notamment les stratégies de routage et les politiques de remplacement dans le cache. Ainsi, des simulations sont nécessaires afin d'avoir une évaluation précise des techniques et fonctionnalités développées.

L'outil ndnSIM [29] a été réalisé dans le but d'offrir une plateforme de simulation open source et facile à utiliser. Le développement de ndnSIM a commencé en 2011 et la première version fut disponible à partir de Juin 2012. ndnSIM est un outil open source basé sur le simulateur NS-3 [30], capable de simuler fidèlement les protocoles NDN et ses principes de base (routage, caching, etc.) même à grande échelle. Il peut être utilisé directement sur un protocole de couche liaison, ou sur un protocole de couche réseau (IP) ou encore de couche transport (TCP, UDP).

L'outil est modulaire, structuré en plusieurs ensembles de classes C++, chaque ensemble (module) représente une entité de NDN (PIT, FIB, CS, etc.). Cette modularité fait que la modification d'une entité ait un minimum (ou pas) d'effet sur les autres.

Depuis la version 2.0, le NFD est supporté. La version actuelle de ndnSIM est la 2.1 [31]. Ses composants sont listés ci-dessous (voir Figure 1.14) :

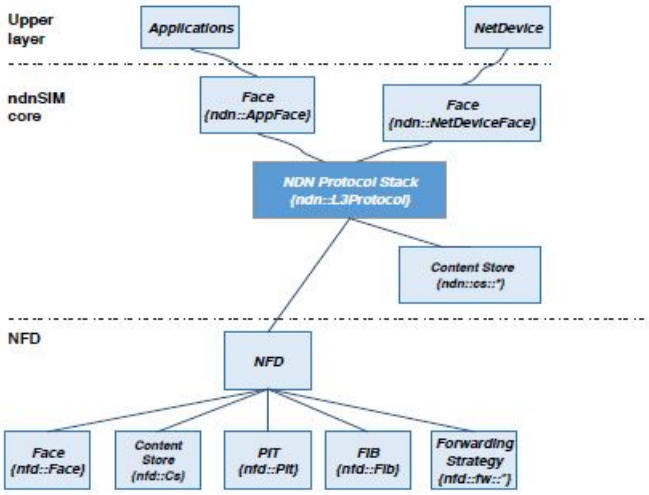

FIGURE 1.14 – Composants de ndnSIM [31]

- ndn : :L3Protocol : Abstraction de la pile NDN. Sa tâche principale est l'initialisation du NFD de chaque nœud de la simulation, et permet la mesure des performances (paquets Interest et Data reçus/envoyés, satisfaits/non satisfaits).
- NFD : Implémentation de NDN Forwarding Deamon, il inclut :
  - nfd : :Forwarder : Classe principale de NFD, contient les interfaces et les tables d'un routeur NDN.
  - nfd : :Face : Classe abstraite. Contient les méthodes nécessaires à l'envoie/réception des paquets.
  - nfd : :Cs : Content Store de NFD. C'est le cache des paquets Data gardés momentanément en mémoire.
  - nfd : :Pit : PIT utilisée par NFD dans le routage.
  - nfd : :Fib : FIB utilisée par NFD dans le routage.

  - nfd : :fw : :Strategy : Classe abstraite de NFD qui doit être implémentée pour exécuter une stratégie de routage des paquets.

- ndn : :AppFace : Concrétisation (implémentation) de *nfd : :Face* afin de communiquer avec les applications.
- ndn : :NetDeviceFace : Concrétisation (implémentation) de *nfd : :Face* afin de communiquer avec les autres nœuds de la simulation.
- Applications NDN : Le simulateur implémente des applications basiques permettant de générer du trafic NDN (Interest, Data) selon différents paramètres afin de réaliser la simulation. Voici quelques applications disponibles :
  - ConsumerCbr : Génère des Interests en fonction des paramètres donnés (e.g. fréquence).
  - ConsumerBatches : Génère un nombre spécifique d'Interests à des instants précis de la simulation.
  - ConsumerWindow : Génère des Interests avec des débits différents.
  - Producer : Répond à chaque Interest reçu en générant un paquet Data ayant le même nom.

  L'interaction des applications avec le simulateur se fait en utilisant *ndn : :AppFace*. La classe *ndn : :App* s'occupe de la création/destruction des instances de la classe *ndn : :AppFace*.
- Trace helpers : Donnent les mesures et les statistiques sur la simulation comme le traçage des paquets de la couche réseau et liaison (dans un fichier texte).

Utilisation de ndnSIM

ndnSIM permet un paramétrage très précis de la simulation :

- La topologie, en la définissant manuellement ou à l'aide des topologies prédéfinies disponibles.
- Le choix de la taille du Content Store et la politique de remplacement de son contenu (freshness, fifo, etc.).
- Les stratégies d'acheminement du trafic (multicast, best route, etc.).
- Définition des routes entre les nœuds (manuelle, semi-automatique).
- Utilisations d'applications par défaut ou personnalisées.
- Collecte des résultats (débit, vitesse d'acheminement, nombre de paquets, cache hits/misses etc.).

Des tutoriels sont disponibles dans [32].

1.4.5 Mini-NDN

Afin de tester les algorithmes de routage, un autre outil a été développé : Mini-NDN. Son but est de rendre les tests plus rapides et moins contraignants (i.e. éviter le paramétrage manuel de chaque nœud à chaque mise à jour, etc.). Dans Mini-NDN chaque nœud du réseau et ses ressources sont gérés par un conteneur (Figure 1.15). Il dispose également d'une interface graphique facilitant la mise en place de la topologie et des paramètres (Figure 1.16). L'avantage de cet outil adapté spécialement à NDN est qu'il permet de tester immédiatement les mises à jour des protocoles (notamment NLSR), plutôt que d'utiliser un outil à usage général comme Emulab (utilisé initialement). Le code source de Mini-NDN est disponible sur GitHub [33].

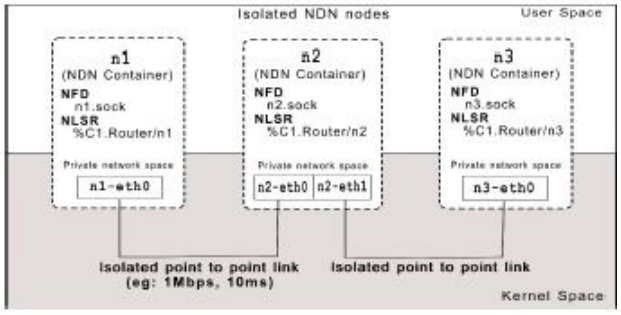

FIGURE 1.15 – Représentation des nœuds avec MiniNDN [14]

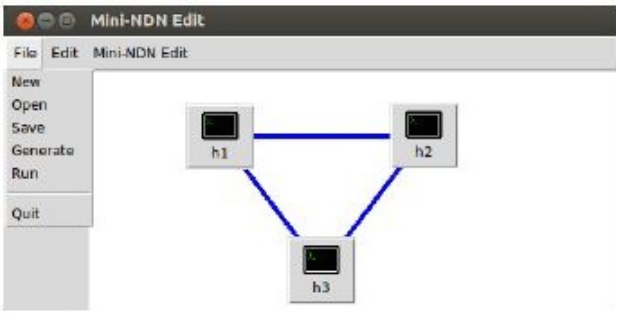

FIGURE 1.16 – Interface graphique de MiniNDN [14]

1.4.6 NDN Testbed

Les simulations et les émulations de réseaux ne peuvent pas assurer fidèlement certains types de tests. C'est pour cela qu'un vrai réseau NDN a été mis en place à travers certains pays tels que les États Unis, la Chine, l'Espagne, la Suisse et d'autres. Il est utilisé pour tester des applications et des protocoles de routage dans des conditions réelles. L'état de ce réseau est visible en temps réel sur la page : `http://ndnmap.arl.wustl.edu/`

1.4.7 Autres outils

D'autres outils, bibliothèques et plateformes NDN existent encore ; pour Android, pour le temps réel, le routage, etc. Nous avons cité ici les plus importantes et les plus indispensables pour la compréhension et l'utilisation de l'architecture NDN. Le reste des outils et applications NDN est disponible sur [34].

1.5 Conclusion

NDN est défini comme un protocole réseau basé sur des requêtes de données nommées . S'il développe les mécanismes importants de son architecture et s'enrichit de bibliothèques et de techniques facilitant la mise en place des applications émergentes, il va sans doute bouleverser l'Internet qui s'avère limité face aux défis lancés par ces nouvelles applications ; tout comme la mécanique quantique a bouleversé, au début du siècle dernier, la mécanique classique qui fut inapte à répondre aux nouvelles questions de la physique.

Nous avons vu dans ce chapitre le paradigme sur lequel repose l'architecture NDN ainsi que ces principes fondateurs et les fonctionnalités réseau qu'il propose. L'étude des mécanismes NDN et leur fonctionnement est une étape importante pour la réalisation d'une application IoT basée sur NDN. En effet, les propriétés des noms, les paquets et les détails des communications NDN étudiées ici permettront d'exploiter au mieux

les avantages de NDN dans notre conception.

Il nous faut maintenant comprendre ce qu'est l'IoT, quelles sont ses contraintes et ce qui fait que NDN soit adapté à l'IoT. Il nous faut également savoir comment les mécanismes NDN sont utilisés dans l'IoT en étudiant les techniques et solutions NDN proposées pour gérer les contraintes de l'IoT. C'est tout ceci que nous allons voir dans le prochain chapitre.

Chapitre 2

L'Internet of Things et NDN

2.1 Introduction

L'Internet des objets (IoT) est probablement le changement qui a le plus mis la lumière sur les limites de l'Internet actuel, c'est pour cela que toute nouvelle architecture voulant s'imposer comme l'Internet du futur devra inévitablement répondre aux contraintes de l'IoT, et NDN n'échappe pas à cette règle. Dans ce chapitre, nous allons définir l'IoT et énoncer ses contraintes les plus importantes. Nous étudierons la compatibilité de NDN avec l'IoT grâce à ses principes ainsi qu'à travers quelques travaux de recherche NDN sur l'IoT.

2.2 L'Internet of Things

Les dernières avancées en électroniques et en informatique ainsi que les divers supports de communication, notamment sans fil (Wifi, ZigBee, etc.) ont permis l'émergence d'objets connectés offrant des services allant de la montre ou la lampe connectée, aux maisons connectées et même aux villes intelligentes (Smart Cities) dans lesquelles tout ou presque est connecté. Le système censé connecter tous ces objets à travers Internet est appelé Internet of Things (Internet des objets).

L'importance de l'IoT devient considérable, puisqu'il s'agit de la première véritable évolution de l'Internet. Celle-ci donnera lieu à des applications révolutionnaires capables de transformer profondément notre mode de vie, et notre façon d'apprendre, de travailler et de nous divertir.

2.2.1 Origines du concept

La plus ancienne utilisation connue de cette expression (IoT) remonte à 1999 par Kevin Ashton [35]. Il disait qu'en ajoutant des puces RFID (Radio Frequency IDentification) aux objets du quotidien on créerait un « Internet of Things »(Internet des Objets). Le concept initial était donc d'ajouter à n'importe quel objet, une puce RFID connectée à une base de données à travers l'Internet afin d'avoir des informations sur cet objet. Aujourd'hui le concept a beaucoup évolué en dotant l'Internet de capacités sensorielles (température, pression, vibration, luminosité, humidité, tension, etc.), nous permettant d'anticiper et d'interagir plutôt que de simplement observer ou surveiller.

2.2.2 Définitions selon des organismes de standardisation

Avec l'intérêt que portent les entreprises et les projets de recherche à l'IoT, ce dernier devra obligatoirement connaitre des normes et standards dans un avenir proche. C'est pour cela que l'avis des organismes de technologies et standardisations concernant l'IoT doit être étudié et suivi avec intérêt.

IEEE

Sa définition de l'IoT est plutôt une description axée sur l'aspect physique du concept [36] : « *A network of items-each embedded with sensors-which are connected to the Internet.* »

NIST

Pour le NIST, l'IoT est aussi désigné sous l'expression « *Cyber-physical systems* »et deux descriptions de l'IoT sont proposées par le NIST. La première est celle de l'équipe *Smart America/Global Cities Challenge* : « *Cyber-physical systems (CPS) – sometimes referred to as the Internet of Things (IoT) – involves connecting smart devices and systems in diverse sectors like transportation, energy, manufacturing and healthcare in fundamentally new ways. Smart Cities/Communities are increasingly adopting CPS/IoT technologies to enhance the efficiency and sustainability of their operation and improve the quality of life.* »[36]. La deuxième est celle de Chris Greer (NIST) : « *Cyber-physical systems, also called the Internet of Things, are the next big advance for our use of the web. They allow complex systems of feedback and control that can help a robot coordinate with a dog or human in a search-and rescue operation or help health care providers evaluate the recovery of patients after they leave the hospital.* »[37]

W3C

Sans surprises, W3C définit l'IoT d'un point de vue Web avec l'expression « Web-of-Things ». Le Web-of-Things représente l'IoT tel qu'il est vu par les applications et technologies Web. Il est définit comme suit : « *The Web of Things is essentially about the role of Web technologies to facilitate the development of applications and services for the Internet of Things, i.e., physical objects and their virtual representation. This includes sensors and actuators, as well as physical objects tagged with a bar code or NFC. Some relevant Web technologies include HTTP (...), and JavaScript APIs for virtual objects acting as proxies for real-world objects.* »[36][38]

2.2.3 Définitions selon des entreprises et projets

Casagras Project

« *A global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities. This infrastructure includes existing and evolving Internet and network developments. It will offer specific object-identification, sensor and connection capability as the basis for the development of independent cooperative services and applications. These will be characterized by a high degree of autonomous data capture, event transfer, network connectivity and interoperability.* »[39]

IoT-A Project

Internet of Things Architecture (IoT-A [40]) est un projet Européen travaillant sur un modèle et une architecture globale de l'IoT. Sa description de l'IoT est la suivante [41] : « *It can be seen as an umbrella term for interconnected technologies, devices, objects and services.* »
La Figure 2.1 montre les relations entre services, ressources, devices, cloud et utilisateurs dans l'IoT.

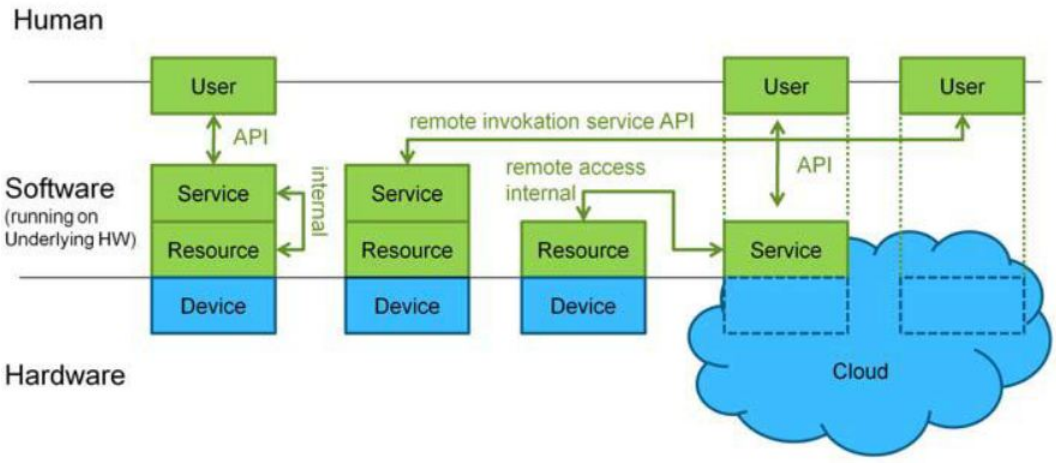

FIGURE 2.1 – Ressources, devices et services dans l'IoT [42]

Cisco

Pour l'aspect temporel, Cisco Internet Business Solutions Group (Cisco IBSG) situe l'apparition de l'IoT au moment où le nombre d'objets connectés à Internet a dépassé le nombre de personnes dans le monde [43]. Il a situé ce moment entre 2008 et 2009 (Figure 2.2).

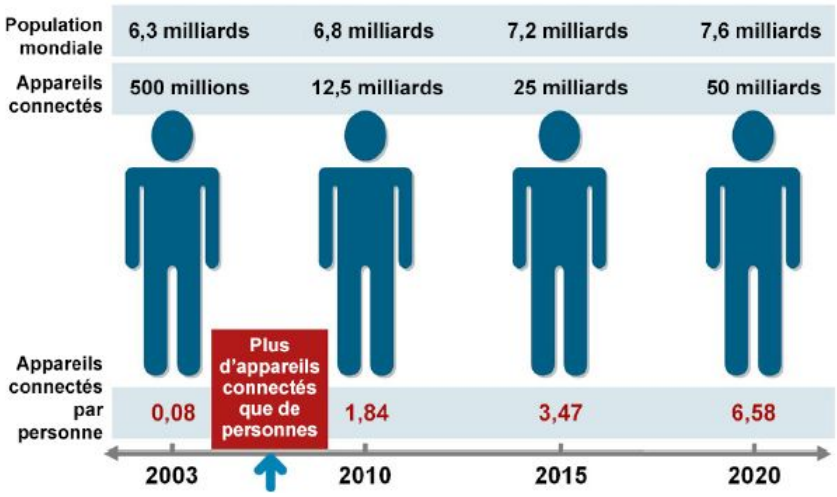

FIGURE 2.2 – Evolution du nombre d'objets connectés selon Cisco [43]

Nous avons cité quelques définitions de l'IoT, mais elles sont encore nombreuses. Il y a presque autant de définitions de l'IoT que de projets le concernant, car chaque entreprise, organisme ou projet de recherche décrit l'IoT selon son domaine de compétences et son champ d'action (hardware, software, standards, Web, etc.). Néanmoins, toutes ces visions de l'IoT convergent vers l'idée de construire un système global, doté d'une architecture pouvant contenir les 50 à 100 milliards d'objets connectés prévus d'ici 2020 qui seront pour la plus part mobiles, fait par des constructeurs différents et auront différentes façons de communiquer. Ce système global devra par conséquent gérer efficacement des ressources dynamiques et hétérogènes. Pour donner une image simple, imaginons qu'on doive rassembler tous les peuples du monde avec leurs différences dans un même pays et sous un même gouvernement.

Il y a globalement quatre groupes d'appareils dans l'IoT :

- Les PC, serveurs, routeurs, switches et autres équipements primaires utilisant principalement des connexions filaires.
- Matériels médicaux et autres systèmes de monitoring et de contrôle utilisant des connexions filaires ou sans fil.
- Smartphones et tablettes utilisés par les particuliers pour interagir avec les objets et les services, utilisant parfois plusieurs connexions sans fils de différents types (wifi, 3G, 4G, etc.).
- Appareils qui réalisent une tache spécifique ou qui offrent un service spécifique (capteurs, actionneurs, etc.) utilisant exclusivement des connexions sans fil.

Actuellement, l'IoT se compose d'un ensemble hétérogène de réseaux spécialisés disparates tel que le montre la Figure 2.3 :

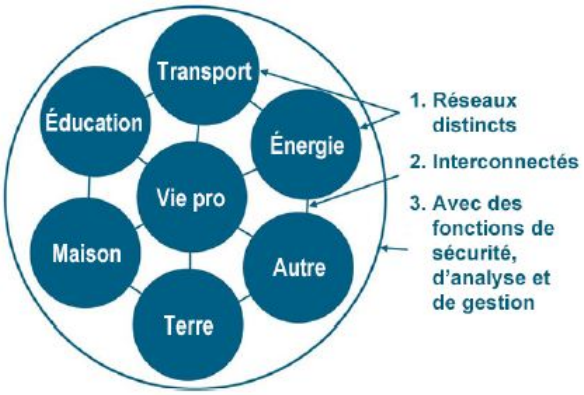

FIGURE 2.3 – Les objets de l'IoT sont séparés en réseaux spécialisés [43]

2.3 NDN et les contraintes de l'IoT

Dans ce qui suit, nous allons voir les contraintes les plus importantes qu'une plateforme de l'IoT doit prendre en compte conformément à ce qui a été énoncé dans [44]. Nous verrons également comment NDN peut prendre en charge chacune de ces contraintes.

2.3.1 Applications basées contenu

La plupart du temps, les applications de l'IoT ont besoins de données concernant leur environnement sans se soucier du capteur qui a collecté cette donnée ou de son fournisseur en général. Tout ce qui compte c'est le contenu, non la façon de l'obtenir ou l'endroit d'où il vient.

NDN répond à cette contrainte majeur par la philosophie même de son architecture ; en effet, NDN est basé contenu et ne manipule aucune adresse dans ses paquets ou dans ses algorithmes de routage. C'est le contenu qui est directement identifié dans NDN.

2.3.2 Le nommage

Il est primordial de pouvoir identifier de manière univoque et durable les objets, les données, ou des groupes d'objets de l'IoT. Les noms doivent supporter la nature dynamique des objets due à la mobilité et au changement de contexte d'utilisation. Ils doivent également être sécurisés et adaptés à l'application.

Les noms dans NDN ne sont pas des adresses. Ils sont naturels, ont une structure hiérarchique et pas d'intervalle limitant les possibilités. Ils peuvent identifier une donnée, un capteur, une application, une commande pour un objet connecté, etc. De plus, les développeurs d'applications et les constructeurs peuvent choisir leur propre convention de nommage à utiliser par l'application et les objets.

2.3.3 La mise à l'échelle (scalabilité)

Le nombre astronomique d'objets connectés doit être géré sans altérer la qualité de service. Pour cela, l'attribution des noms et leur découverte, le routage et l'acheminement doivent être adaptés à la manipulation d'un très grand nombre de noms, de préfixes, d'interfaces, etc. Mais surtout, la décentralisation des services peut s'avérer nécessaire.

Pour faire face à la scalabilité, NDN se base sur la structure hiérarchique des noms, permettant l'agrégation des réseaux (comme avec l'IP) ce qui permet par exemple, de séparer le routage global du routage local. De plus, des solutions plus complexes telles que le SNAMP [22] sont proposées pour un meilleur support de la scalabilité.

2.3.4 Les contraintes de ressources

Les objets connectés sont généralement des systèmes embarqués, ils disposent de ressources assez limitées les empêchant d'effectuer des calculs complexes et de communiquer longtemps. On doit donc ménager ces objets en trouvant les solutions les moins couteuses en temps et en complexité pour manipuler et échanger les données. En général, les traitements doivent êtres répartis intelligemment entre les nœuds, et les communications (qui sont les plus couteuses en énergie) ne doivent être effectuées qu'au moment opportun. Il s'agit également de bien gérer la bande passante et le trafic dans le réseau.

Étant dépourvu d'adressage, NDN permet d'agréger facilement des nœuds et/ou de mettre en place des modèles à plusieurs entités afin d'alléger les traitements. Nous avons vu que le contrôle de flux est implémenté dans l'architecture NDN ; grâce à la manipulation de la PIT (taille de la table et durée de vie des entrées) et du Content Store (taille et politique de remplacement). Le principe du caching offre également un moyen de stocker des données dans le réseau (à travers les nœuds), ce qui ménage considérablement les objets dotés d'une petite capacité de stockage. Enfin, le principe du bout-en-bout sur lequel repose l'architecture NDN dirige les traitements critiques, délicats et couteux vers les frontières du réseau dont les nœuds sont généralement des appareils plus puissants.

2.3.5 La caractéristique du trafic

On distingue deux types de trafic dans l'IoT : le trafic à portée locale et le trafic global (ou longue distance). Le trafic à portée locale est généré entre des nœuds proches. Par exemple, les objets contenus dans une maison intelligente communiquent entre eux pour donner les valeurs adaptées au chauffage, à la lumière, etc. Le trafic longue distance est présent à plus grande échelle entre des nœuds plus nombreux et plus éloignés, et nécessite une découverte des ressources et des services, et un acheminement efficaces des données.

Grâce au principe du caching, NDN peut gérer correctement le trafic à portée locale car ce principe permet de réutiliser une donnée tant qu'elle est valide sans avoir à remonter à son fournisseur. NDN supporte le multicast qui peut être très utile pour partager les données entre plusieurs nœud proches à la fois. Pour le trafic à longue distance, le multipath augmente la sureté de l'acheminement des données en aiguillant les paquets sur plusieurs chemins à la fois.

2.3.6 La mobilité

La mobilité est l'une des contraintes les plus importantes et les plus difficiles à gérer. Elle peut avoir plusieurs sens : La mobilité du fournisseur de la donnée, la mobilité du consommateur de la donnée, la mobilité du réseau entier ou encore que des coupures de connexion arrivent souvent. Une plateforme IoT doit être capable d'acheminer les données dans tous ces précédents cas.

NDN n'identifie pas les entités par des adresses. La communication se fait indépendamment de la localisation du nœud ou de son réseau. Même en étant mobile, le consommateur pourra recevoir les données soit depuis le cache d'un nœud voisin, soit acheminées depuis le fournisseur ; il n'a qu'à régénérer le paquet Interest correspondant.

2.3.7 Sécurité et vie privée

Les objets connectés se rapprochent de plus en plus de l'utilisateur, et donc de sa vie privée. Les informations collectées par ces objets puis transmises aux applications doivent rester confidentielles, et les informations provenant des serveurs et autres compagnies doivent êtres authentiques. Ceci impose à toute plateforme de l'IoT de gérer l'intégrité, l'authentification et le contrôle d'accès aux données à tous les niveaux.

Dans NDN, la donnée est l'entité de base. C'est donc autour d'elle que la sécurité est construite. La sécurité est implémentée dans l'architecture et n'est pas une super-couche du réseau. Nous avons cité plus haut les principes de NDN relatifs à la sécurité : entre autre, toute donnée doit être signée et liée avec son nom, et doit contenir des informations permettant de vérifier l'identité du signataire. Les développeurs ont la possibilité de choisir et de personnaliser leur propre modèles de sécurité et de vérification des données.

2.4 Travaux NDN relatifs à l'IoT

Plusieurs travaux de recherche ont été consacrés à NDN pour l'IoT. Cette section présente quelques solutions et implémentations proposées par ces travaux, elles sont classées selon leur domaine d'application.

2.4.1 Domotique et monitoring

Dans [45], une architecture globale de NDN pour l'IoT est proposée et discutée. On y explique comment une plateforme IoT peut bénéficier des fonctionnalités de NDN et on y propose un découpage de toutes ces fonctionnalités tout en décrivant leurs interactions avec la couche application et la couche « Thing » censée représenter les appareils connectés.

NDNoT [46] s'intéresse aux objets connectés dans une maison. C'est un protocole qui prend en charge des fonctionnalités telles que : (i) Gérer l'ajout et la suppression des appareils et des services. (ii) Gérer l'accès aux données. (iii) Servir à collecter les données de la maison. Le protocole propose un mécanisme d'authentification auprès de la station pour les nouveaux appareils. On y exploite efficacement la souplesse offerte par les noms hiérarchiques et illimités de NDN ; par exemple, des composants (suffixes) sont ajoutés après le nom d'un Interest pour contenir une estampille, exprimer la version, la signature, etc.

Dans le même domaine, NDP (Neighbor Discovery Protocol) et SPDP (Service Publish and Discovery Protocol) [47] sont deux protocoles très utiles. NDP permet la découverte des nœuds voisins disponibles. SPDP permet la publication et la découverte de services en reposant sur les résultats fournis par NDP.

NDN-BMS [48] est un système sécurisé de surveillance de bâtiments. Il a été déployé à l'UCLA et exploite son BMS déjà installé et composé d'environ 150000 capteurs en tous genres. Le système assure la collecte et la publication des données fournies par les capteurs vers les applications BMS ainsi que la gestion du contrôle d'accès à ces données. Les données collectées sont nommées de manière à exprimer leur type (voltage, lumière, etc.), leur provenance géographique (étage, pièce, etc.) et temporelle. La sécurité est basée sur le chiffrement des données et la gestion des clés des groupes/utilisateurs. Le prototype se distingue par les aspects suivants :

- Le découpage et les rôles des entités.
- Le schéma de nommage et la sécurité.
- La gestion des accès et privilèges utilisateurs.

NDN-ACE [49] s'intéresse au contrôle d'accès dans NDN et décrit un protocole basé sur les Interests signés. Ce protocole ne gère que le cas de l'envoie d'une commande pour contrôler un appareil (lumière, verrouillage, etc.) mais il est particulièrement intéressant car il permet à des appareils avec de faibles ressources de déléguer le contrôle d'accès à des stations plus puissantes et toujours actives.

2.4.2 Santé

SmartHealth-NDNoT [50] décrit un système de surveillance de l'état de santé d'un patient, l'accès aux services de soins et d'urgences et la possibilité de déclencher des alertes en cas de besoin. La fonction principale de l'architecture proposée est de collecter, transférer et analyser les données personnelles du patient en utilisant NDN. L'architecture se compose de quatre couches :

- La couche capteurs et systèmes embarqués, se trouve à proximité du patient et collecte les données.
- La couche passerelle NDN (appelée *NDNoT gateway*), transfère les données du patient vers les différents services.
- La couche des services réseaux et stockage, s'occupe du traitement, de l'analyse et du stockage des données et des requêtes de l'utilisateur. Elle gère la surveillance de l'état de santé du patient, génère des rapports et peut déclencher des alarmes si nécessaire.
- Le serveur central, contient les informations nécessaires pour la découverte des services.

2.4.3 VANETs

Le document [51] explore l'adaptabilité de NDN pour les réseaux de véhicules, notamment les communications V2V et V2R (Vehicle to vehicle et Vehicle to Road-side unit). Ces types de communications sont très difficiles à gérer avec les réseaux IP à cause de l'absence d'infrastructure pour le réseau et de la grande mobilité des nœuds.

NDN n'étant pas basé sur les adresses et l'établissement de sessions, ses communications sont plus faciles à renouveler même en cas de coupures courantes. Il peut donc représenter une solution aux communications V2V et V2R, grâce notamment aux technologies DSRC (IEEE 802.11 p) et WAVE (IEEE 1609) prévues pour les communications sans fil des véhicules.

Le document explique comment tirer avantage des propriétés des véhicules. En effet, les systèmes embarqués dans les véhicules disposent de plus de capacité de stockage et de puissance de communications que la plus part des appareils de l'IoT, ce qui permet d'utiliser chaque véhicule pour stocker un maximum de données qu'il pourra mettre à disposition des autres véhicules.

Une autre propriété des véhicules est leur grande vitesse de déplacement, on peut donc les utiliser comme « data mule » pour transporter physiquement les données qu'ils stockent d'un point à un autre de la route. Le concept de « data mule » est intéressant car il permet de retrouver des données provenant de points éloignés de plusieurs sauts, en utilisant simplement un broadcast en un seul saut (qui est beaucoup plus facile à gérer car il ne nécessite pas de routage). On y présente également les adaptations à faire pour NDN afin de prendre en charge ces communications :

- La gestion de la PIT car les Interests ont beaucoup de risques d'être perdus ou de ne pas être satisfait à cause de la grande mobilité des véhicules.
- Accepter les données non sollicitées (i.e. Data reçu sans avoir envoyé d'Interest) pour servir de data mule.
- La taille du Content Store et les politiques de remplacement pour offrir les données les plus pertinentes aux autres véhicules.

Un schéma de nommage a été proposé en se basant sur les contraintes suivantes :

- Exprimer les informations géographiques.
- Exprimer les informations temporelles.
- Détecter les données dupliquées.
- Être souple et s'adapter à tout type d'application destinée aux véhicules.
- Assurer la fiabilité des données reçues.

Son espace de noms a la forme suivante : **/traffic/geolocation/timestamp/dataType/nonce**.

- */traffic* : est le préfixe de l'application (ici l'application concerne le trafic).
- */geolocation* : peut être sous forme de coordonnées GPS ou bien sous forme de nom de route, direction et numéro de section. */geolocation* doit pouvoir identifier un tronçon de route de manière univoque.
- */timestamp* : peut être une seule valeur de temps ou une période.
- */dataType* : exprime la nature de la donnée.
- */nonce* : valeur aléatoire pour distinguer les données.

Exemple : /traffic/Highway101/north/{400,410}/{1323201600,1323205200}/speed/19375887.
Ce nom permet de nommer les données concernant la vitesse, générées géographiquement entre les sections 400 et 410 de l'autoroute 101 direction nord, et temporellement entre 12 :00 et 13 :00 heures du 6 Décembre 2011.

Les contributions NDN pour l'IoT ainsi que certains problèmes qui doivent être gérés dans le contexte de l'IoT sont discutées plus longuement dans [16] et [52].

2.5 Travaux NDN relatifs aux types de trafic dans l'IoT

Concernant les échanges présents dans l'IoT, les documents [53][54][55][56][57] s'intéressent à différents types de trafic et proposent des méthodes et des techniques pour les gérer. Du point de vue application, on distingue globalement deux types de trafic dans l'IoT :

2.5.1 Le trafic « pull »

C'est le trafic qu'on « retire » ; le consommateur demande (avec un Interest) une certaine donnée (paquet Data) qui lui est renvoyée par le(s) fournisseur(s). Trois cas sont étudiés dans ce type de trafic : (i) Le cas d'un seul Data depuis une seule source (e.g. une station domotique demande l'état d'une lampe connectée), (ii) le cas de Data multiples depuis une source unique (e.g. surveiller l'état d'une alarme) et (iii) le cas de Data multiples depuis des sources multiples (e.g. une station domotique collecte la température de toutes les pièces) ; on parle alors de collecte de données.

Le cas (i) est supporté par défaut par NDN ; un échange de donnée NDN est initié par le consommateur et un Data satisfait un Interest.

Pour le cas (ii), le principe d'abonnement (*Publish/Subscribe*) a été proposé dans [57] : Dans ce mécanisme, le consommateur s'abonne en enregistrant le préfixe sous lequel son fournisseur publie ses informations (appelées notifications). Le préfixe est propagé à travers les FIB des nœuds du réseau, le fournisseur pourra par la suite envoyer des notifications sous forme d'Interests. Cependant, cette solution ne permet de recevoir que des petites informations.

Pour le cas (iii), le document [55] propose un mécanisme basé sur un Interest multi-source (appelé *msINT*) pouvant atteindre *N* fournisseurs différents partageant le même préfixe, et envoyer après d'autres Interests en utilisant le champ *Exclude*, pour exclure les fournisseurs qui ont déjà transmis leurs données et ne cibler que ceux qui n'ont pas encore répondu. On y discute la valeur de la durée de vie d'un *msINT*, les conditions de sa retransmission et l'importance du délai (fenêtre) que les fournisseurs doivent respecter avant de répondre afin d'éviter les collisions dans un réseau local.

L'utilisation d'un *msINT* a été proposée dans le cas d'un réseau local. Dans le cas d'une collecte depuis un emplacement distant (e.g. un propriétaire surveillant l'état de sa maison), on peut combiner l'abonnement et la collecte locale : l'utilisateur distant s'abonne à la station domotique et la station collecte les données localement et les retransmet (paquet par paquet ou agrégées dans un seul paquet si possible).

Dans [54], on étudie les possibilités à exploiter et les contraintes à respecter pour un système de collecte de données par une station depuis des sources mobiles (cas de véhicules roulant sur l'autoroute). On défini pour cela trois étapes :

1. L'entreprise voulant recevoir ces données doit contacter l'opérateur réseau pour que les stations sur les bords des routes (*Road-Side Unit RSU*) puissent annoncer son préfixe (e.g. la RSU N°1 de la ville de Los Angeles annoncera pour *Toyota* le préfixe *'ndnx :/toyota/diagnosis/us/ca/los-angeles/bs-1/'*).

2. Pour collecter les données, l'entreprise enverra un Interest exprimant la zone géographique concernée par la collecte et/ou le modèle de véhicules ciblé, etc. Par exemple, l'Interest *'ndnx :/toyota/diagnosis/us/ca/*/prius/2009'* demande les données des véhicules de type *'Prius'* de l'année 2009 dans tout l'état de Californie (USA). On remarque l'utilisation du caractère '*' qui veut dire ici, que toutes les villes de cet état sont concernées.

3. L'Interest envoyé a une durée de vie relativement longue, il est appelé *Long-Lived Interest*, et il doit être rediffusé par les stations plusieurs fois durant cette durée de vie car les véhicules se déplacent vite et les risques de perte sont élevés.

Cette solution repose sur un schéma de nommage souple qui exprime toutes les informations nécessaires à l'application utilisée.

Un autre échange considéré comme un trafic de type « *pull* » est l'envoie d'une commande à un appareil. Une commande représente un contenu simple et petit (lecture/écriture), une solution proposée dans divers documents est donc d'exprimer cette commande directement dans le nom d'un Interest, et de se servir du Data retourné comme acquittement ou réponse (succès, écher, code erreur, etc.). Toutefois un tel Interest doit être signé afin de vérifier son origine, c'est ce qui est proposé dans [46].

2.5.2 Le trafic « push »

C'est le trafic qu'on envoie directement soit de manière périodique (e.g. un capteur envoie une donnée toutes les 30 secondes), soit à l'arrivée d'un évènement (e.g. un capteur déclenche une alarme si une certaine température est atteinte). Dans [53], trois solutions sont proposées pour gérer ce type de trafic :

- Les « *Notification Interest* » : De même que pour les commandes, les notifications contiennent de petites informations qui peuvent tenir dans une chaine de caractères. On peut donc mettre cette notification dans le nom d'un Interest (appelé *nINT*). Un paquet Data (appelé *aData*) peut être retourné pour servir d'acquittement. Les *nINT* doivent également être signés afin de vérifier leur authenticité.

- *Unsolicited Data* : Cette solution consiste à diffuser des paquets Data (appelés *uData*) que certains nœuds peuvent recevoir même s'ils n'ont pas généré d'Interest pour eux. Ils pourront ensuite envoyer un *aData* pour acquittement. Toutefois, du fait qu'un *uData* ne peut faire qu'un seul saut et sur un support en diffusion (wifi, ZigBee, etc.) cette solution n'est envisageable que dans un réseau local et à support en diffusion comme des objets connectés dans une maison.

- *Virtual Interest Polling* : Se base sur l'utilisation d'un Interest ayant une longue durée de vie (appelé *Long-lived-Interest*) et qui n'est pas supprimé de la PIT, même si le Data correspondant est reçu et acheminé. Cette solution permet à un consommateur de s'abonner à un fournisseur, lui donnant ainsi la possibilité de lui envoyer des données sans les solliciter à chaque fois. Cette solution est basée sur un délai maximal (τ) entre la génération de deux Data successifs et un délai de retransmission de l'Interest (légèrement supérieur à τ), elle n'est donc applicable que dans le cas du trafic de type « *push* » périodique.

Remarques

1. L'utilisation des *Long-Lived-Interest* occupe la PIT pendant de longues durées, ce qui peut altérer la performance de l'acheminement si la PIT est petite. Il n'est donc envisageable que pour des nœuds disposant de ressources suffisantes (e.g. stations domotiques, passerelles, etc.).

2. Le cas (ii) dans le trafic de type « *pull* » peut être vu également comme un trafic de type « *push* » si on se place du point de vue du fournisseur (i.e. le fournisseur veut fournir des données périodiquement ou suite à un évènement). Les solutions du cas (ii) peuvent donc être utilisées pour gérer le trafic de type « *push* » et vice-versa. Elles se basent d'ailleurs globalement sur les mêmes principes.

La Figure 2.4 donne un résumé des différents types de trafic présents dans les applications de l'IoT.

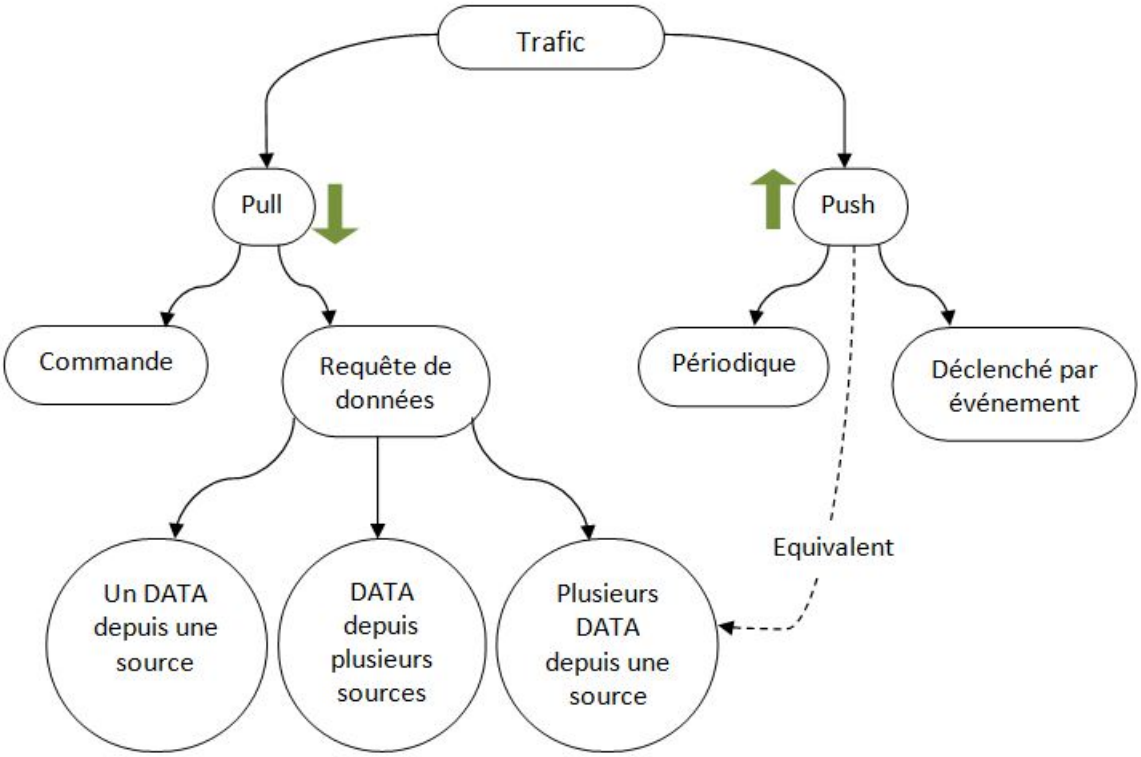

FIGURE 2.4 – Types de trafic dans l'IoT

2.6 Conclusion

Nous avons vu dans ce chapitre quelques descriptions de l'IoT et ses caractéristiques, et comment NDN répond à ses principales contraintes. Même si les travaux NDN sur l'IoT n'en sont qu'à leurs débuts, les solutions étudiées et présentées précédemment montrent la force et les avantages que peut avoir NDN dans l'IoT. NDN bénéficie d'un grand potentiel pour gérer les exigences de l'IoT, notamment par le fait qu'il soit conçu sur des principes complètement compatibles avec l'esprit et la nature des applications de l'IoT. En outre, sa simplicité combinée avec le fruit de la recherche et les efforts nécessaires lui permettront à terme de proposer une architecture complète et robuste pour l'IoT et l'Internet du futur en général.

Les travaux réalisés sur l'application de NDN pour l'IoT nous ont inspirés et nous ont servis de base dans la conception et l'implémentation de notre système de parking intelligent. Les détails du prototype du parking intelligent que nous avons réalisé, son architecture, ainsi que les travaux qui nous ont inspirés dans notre conception sont présentés dans le chapitre suivant.

Chapitre 3

Conception de NDN Smart Parking

3.1 Introduction

Récemment, les objets connectés ont investi les villes intelligentes. Mais pour être intelligente, une ville doit développer de nouveaux services performants dans tous les domaines : environnement durable, urbanisation responsable et habitat intelligent, transport et mobilité intelligente, etc. Les experts prévoient que le nombre de véhicules motorisés dans les zones densément peuplées augmentera sensiblement au cours des 30 prochaines années, entrainant ainsi une hausse de la pollution et de la congestion du trafic. L'une des solutions à ces problèmes est d'optimiser le stationnement afin de rendre la ville plus accueillante. En effet, dans les villes, pour trouver une place de stationnement, les automobilistes doivent sillonner les rues dans une recherche longue et frustrante, et qui peut les conduire parfois très loin de leur destination, sans parler du stress causé. Ainsi, pour les urbanistes [58], 30% de l'ensemble du trafic dans un centre urbain moyen est lié à la recherche d'une place de stationnement disponible. Un des services les plus importants pour les villes modernes (et celles du futur) est donc le stationnement intelligent. C'est dans ce contexte que nous proposons un système de parking intelligent basé sur l'architecture NDN que nous allons présenter dans ce chapitre.

3.2 Les parkings intelligents

3.2.1 Les solutions existantes

De nombreux constructeurs se sont intéressés aux parkings intelligents, donnant ainsi naissance à des prototypes plus au moins complets ou encore à de vrais déploiements dans certaines grandes villes d'Europe, d'Amérique ou d'Asie. Par exemple, à San Francisco, des parcmètres intelligents (système SFpark) [59] calculent le taux d'occupation des voitures garées dans l'espace public pour adapter leur tarif : si la rue est pleine, le prix de l'heure augmente, et si elle est vide, le prix baisse. Le but est de mieux distribuer les places de parking et réduire le temps passé à chercher une place. SENSIT [58] est une infrastructure complète pour les parkings intelligents ; c'est un réseau de capteurs sans fil reliés à toute une panoplie de collecteurs, serveurs et panneaux d'affichage permettant une vue d'ensemble, en temps réel, de l'occupation des places de stationnement ainsi que leur gestion. BANNER [60] propose un système similaire, à une échelle un peu plus petite puisqu'il permet la supervision d'un parking depuis une salle de contrôle. BOSCH [61] va plus loin en développant un réseau de capteurs sans fil pour la surveillance des places de stationnement libres dans les rues ou les parkings. L'information est centralisée sur des serveurs et accessible au grand public via une application mobile. L'application permet aux automobilistes de consulter en temps réel sur une carte la disponibilité des places de parking dans la zone où ils veulent se rendre. D'autres systèmes tels que [62]et [63] proposent des solutions plus au moins similaires au premières.

En approfondissant l'étude de ces systèmes, on constate que leur principe de fonctionnement et les équipements qu'ils utilisent sont globalement les mêmes. Les installations sont basées sur les technologies sans fil qui permettent la mise en place de capteurs autonomes, basse consommation, dotés d'une batterie capable d'assurer leur fonctionnement pendant plusieurs années. Ils sont installés sur la chaussé ; encastrés (Figure 3.2) ou apparent (Figure 3.3), pour les places de stationnement en extérieur, et généralement au-dessus des places de stationnement (Figure 3.1) dans le cas de parkings couverts et souterrains.

FIGURE 3.1 – Capteur fixé au plafond [63]

FIGURE 3.2 – Capteur enterré dans le sol [59]

FIGURE 3.3 – Capteur apparent installé sur le sol (28mm de hauteur et 190mm de diamètre) [64]

Quelque soit le type d'installation, chaque place de stationnement est dotée de son propre capteur afin de connaitre son état à tout instant. Les types de capteurs les plus présents dans ces systèmes sont les suivants :

- Capteur à ultrasons : Utilisé dans les systèmes BANNER par exemple. Il se base sur la distance détectée pour déduire si une place est libre ou occupée. Son inconvénient est qu'il peut être facilement trompé par l'entourage.
- Capteur de pression : Si la pression exercée correspond à celle que peut exercer un véhicule, il en déduit que la place est occupée. Plus difficile à tromper mais nécessite un contacte physique direct afin de détecter la présence d'un véhicule et donc une installation plus complexe.
- Capteur magnétique (ou technologie magnéto-résistive) : Analyse les variations du champ magnétique environnant. Ce champ est stable lorsque la place est libre et il se déforme d'une manière particulière avec la présence d'un véhicule. C'est à priori le capteur le plus fiable pour la détection de véhicules. Il est utilisé par exemple dans les systèmes de BANNER (disque de 70mm de diamètre et 25mm de hauteur).

Chaque capteur transmet ses informations à un serveur via des bornes d'accès (Figure 3.4). Les bornes peuvent être intégrées dans les parcmètres ou installées dans des infrastructures publiques telles que des lampadaires. Dans certains endroits, les bornes peuvent même être alimentées par des panneaux solaires.

FIGURE 3.4 – Exemple de borne d'accès sans fil (Systèmes BANNER) [60]

Les données centralisées peuvent ensuite être exploitées pour une gestion plus ou moins intéressante des parkings ; allant de la simple consultation des disponibilités des places via une application mobile, à des systèmes plus complexes de guidage vers les places disponibles.

Concernant la communication, ces systèmes, qu'ils soient expérimentaux ou déployés sur le terrain, reposent sur les protocoles classiques TCP/IP ou GPRS souvent associés à des protocoles propriétaires. Il est à noter également qu'actuellement, aucun de ces systèmes ne prend en charge la réservation de places de stationnement ; ils se contentent de fournir les informations et c'est le principe du premier arrivé premier servi.

3.2.2 Efficacité du stationnement intelligent

Pour montrer l'avantage des parkings intelligents, nous donnerons dans ce qui suit quelques statistiques concernant le système SFpark déployé à San Francisco [65]. Dans ces statistiques, il faut considérer la partie *Pilot* car elle représente les résultats fournis par les places de stationnement gérées par le système. La partie *Control* représente le résultat des places où le système n'est pas installé. Des améliorations sont observées à la fois sur la partie *Pilot* et la partie *Control*. Les améliorations observées sur la partie *Control* montrent l'effet global que peut avoir un tel système même s'il ne prend pas en charge toutes les places de stationnement (déploiement partiel). La Figure 3.5 montre le temps que mettent les conducteurs à chercher une place de stationnement avant et après l'installation du système SFpark. La Figure 3.6 montre la distance parcourue par les véhicules pour trouver une place, avant et après l'installation du système. La Figure 3.7 montre le taux d'émission de gazes à effet de serre avant et après l'installation du système.

FIGURE 3.5 – Temps de recherche d'une place avant et après l'installation du système [65]

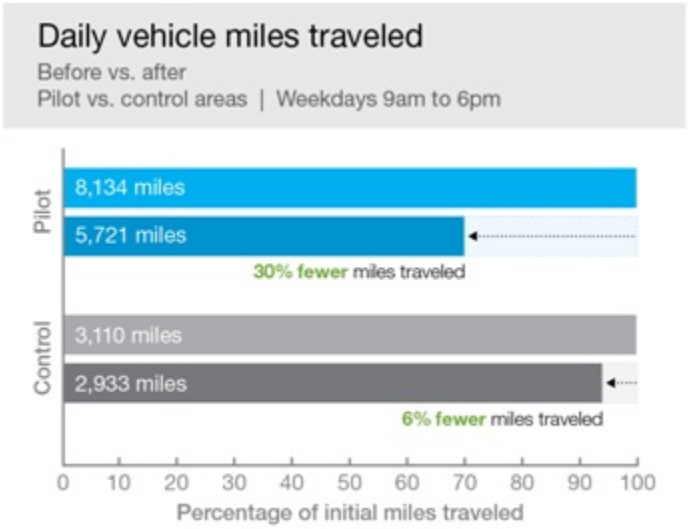

FIGURE 3.6 – Distance parcourue pour trouver une place avant et après l'installation du système [65]

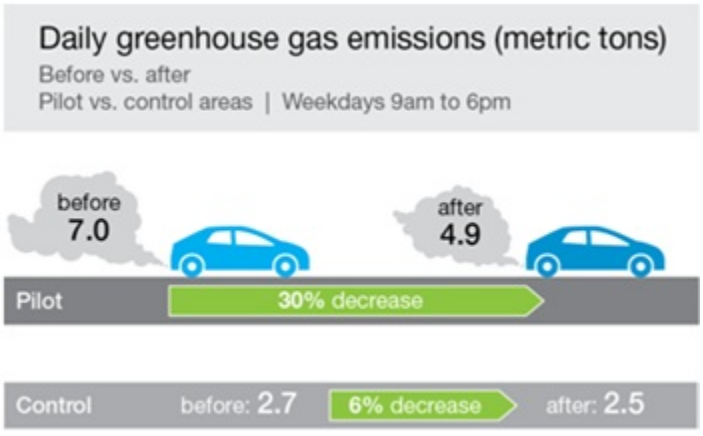

FIGURE 3.7 – Taux d'émission de gazes à effet de serre avant et après l'installation du système [65]

3.3 NDN Smart Parking

3.3.1 Objectifs

Le système que nous proposons permet la recherche, la réservation et la gestion des places de stationnement dans une ville. Le tout, conçu comme une plateforme IoT entièrement NDN. Notre objectif est de proposer un prototype de parking intelligent mais également de montrer qu'une plateforme basée sur NDN est très prometteuse pour des systèmes hétérogènes qui englobent des véhicules, des passerelles et des serveurs, car il fourni une prise en charge facile des fonctions de base telles que :

- Le routage effectué directement sur les noms des contenus échangés par les applications au lieu des adresses, acceptant ainsi un grand nombre de nœuds.
- Une sécurité des échanges construite autour des données plutôt que des connexions qui sont ici très brèves, très instables et très exposées aux menaces.
- Une communication requête/réponse implantée et gérée directement dans le protocole réseau, au lieu de passer par un protocole applicatif (https).
- La mise en cache des données dans le réseau permettant la récupération en cas de perte locale de l'information, mais aussi la réutilisation des mêmes données à d'autres fins (statistiques, information, etc.) sans encombrer plus le réseau.

3.3.2 Description fonctionnelle et architecture

NDN-SP est notre plateforme IoT entièrement basée sur NDN. Il permet de savoir en temps réel les places de stationnement disponibles dans les parkings ou sur les bords des routes, et de réserver une place. Nous nous sommes inspiré dans notre conception, des travaux [48] et [50] cités plus haut. La Figure 3.8 montre l'architecture de notre système.

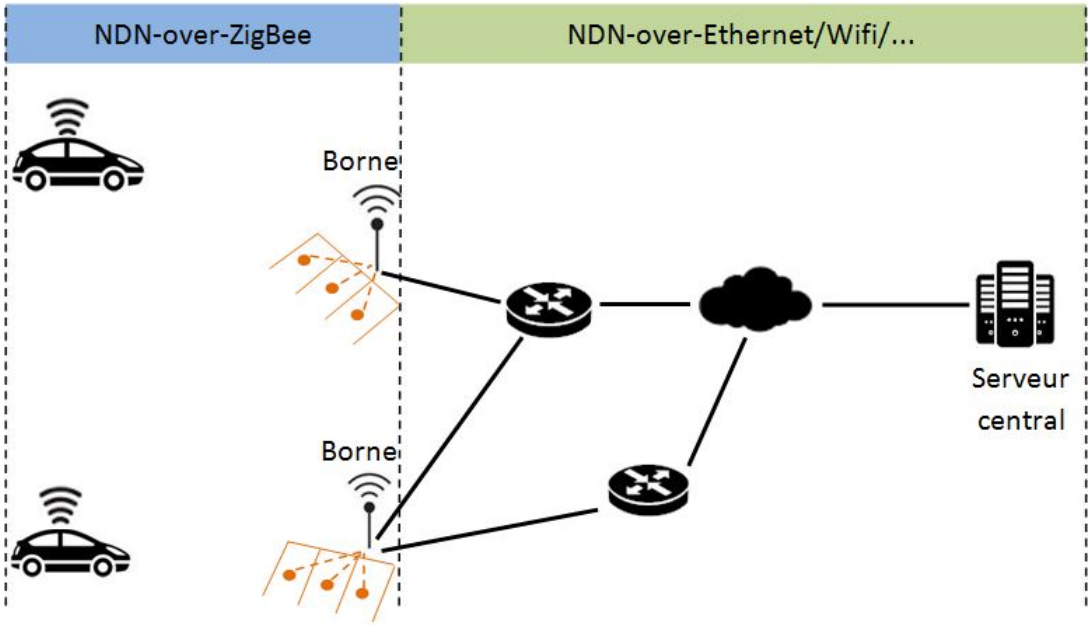

FIGURE 3.8 – Architecture globale

Les entités intervenant dans notre système sont les suivantes :

- Le serveur central : connait en temps réel l'état des places de stationnement de tous les parkings. Il est doté d'une application NDN de type *producer* dont le rôle est de traiter les requêtes (Interests) des bornes et des véhicules et d'y répondre.

- La borne/passerelle : elle est connectée au serveur central distant à travers le réseau, la communication entre elle et le serveur s'effectue en *NDN-over-Ethernet* (supporté nativement par NDN). La communication entre elle et les véhicules s'effectue en *NDN-over-ZigBee* (non supporté nativement par NDN). Chaque place de parking est dotée d'un capteur de détection de véhicules relié à une borne. La borne surveille continuellement l'état de ses capteurs pour détecter les changements d'état des places de stationnement et informer le serveur. La borne sert également de point d'accès ou passerelle (borne/passerelle) pour les véhicules afin de communiquer avec le serveur central distant. C'est pour cela qu'elle est équipée d'un module XBee.

- Le véhicule : équipé d'un module XBee, il communique avec le serveur central à travers les bornes/-passerelles qui sont à sa portée. La communication entre le véhicule et le serveur se fait d'abord par *NDN-over-ZigBee* du véhicule à la passerelle, puis *NDN-over-Ethernet* de la passerelle au serveur. Il est important de noter que dans ces échanges, c'est le même paquet Interest qui traverse ces deux réseaux différents en subissant le routage NDN.

3.3.3 NDN-over-ZigBee

Pour gérer la partie mobile du système, nous avons choisi d'utiliser le protocole ZigBee [66]. Le choix du ZigBee pour la communication véhicule-passerelle est motivé par le fait que ce protocole offre beaucoup d'avantages en termes de nombre de nœuds supportés, de bande passante et d'économie d'énergie, ce qui le rend très avantageux aux applications de l'IoT.

Le ZigBee n'étant pas supporté actuellement par NDN, nous avons développé une couche logicielle greffée à l'implémentation NDN. Son rôle est d'encapsuler les paquets NDN dans des trames du protocole ZigBee. Cette couche logicielle récupère des Interests et les envoie par ZigBee. Nous avons choisi de ne pas intégrer cette couche dans le corps du module NDN pour deux raisons : d'une part, l'implémentation NDN actuelle (NFD) est encore en cours de développement, il est donc sujet à des changements fréquents qui peuvent engendrer des maintenances sur la couche ZigBee, voir provoquer des instabilités du système. D'autre part, les modules XBee utilisés pour les communications sont dotés d'entrées/sorties qui peuvent être utilisées comme capteurs ou actionneurs pour d'éventuelles applications intéressantes, et la gestion de ces entrées/sorties ne peut pas, et ne doit pas être intégrée dans le module NDN qui est un protocole réseau.

La Figure 3.9 montre les couches logicielles et matérielles présentes dans le système du véhicule et celui de la borne/passerelle après l'intégration de la couche ZigBee.

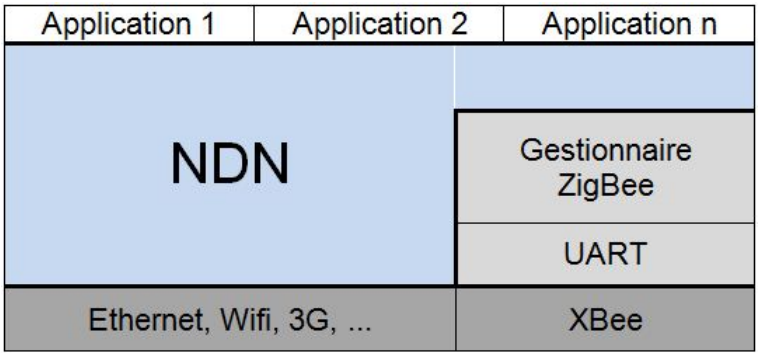

FIGURE 3.9 – Couches matérielles et logicielles du véhicule et de la borne/passerelle

En plus de gérer la communication ZigBee, le gestionnaire est capable de lire l'état des capteurs reliés au module XBee, d'effectuer un traitement personnalisé sur chaque valeur lue et d'envoyer le résultat du traitement sous forme d'un Interest également personnalisé. Un exemple de ce traitement a été implémenté dans notre prototype grâce à un capteur de CO_2 branché au module XBee permettant ainsi d'informer le serveur sur la qualité de l'air. La personnalisation des traitements effectués sur les valeurs lues ainsi que les Interests envoyés se fait à travers un fichier de configuration.

Le gestionnaire ZigBee est composé de deux processus ayant accès au module XBee du système à travers son UART. Le premier, appelé gestionnaire *expéditeur* (ou mode émission) reçoit un Interest du module NDN local, l'expédie via ZigBee et attend la réponse qu'il transmettra au module NDN local. Le second, appelé gestionnaire *receveur* (ou mode réception) reçoit les Interests via ZigBee et les transmet au module NDN pour les router, et renvoie ensuite par ZigBee les réponses retournées par le module NDN local.

Ainsi, un Interest destiné à être transmis via ZigBee est routé localement vers le gestionnaire expéditeur qui s'occupe de l'encapsuler dans des trames ZigBee, de l'envoyer et d'attendre une réponse. Cette réponse est soit un paquet Data, soit la réponse d'un module XBee distant indiquant un problème, soit un *Timeout* si aucune réponse n'est reçue. De l'autre coté, le gestionnaire ZigBee receveur du nœud distant reconstitue l'Interest qu'il reçoit, l'envoie au module NDN local qui le route vers une application ou vers une autre interface réseau ; Ethernet dans notre cas, mais aussi Wifi, 3G, etc. Une fois le Data correspondant reçu par le gestionnaire receveur, il est encapsulé et envoyé à l'adresse du module XBee qui a envoyé l'Interest.

Un Interest reçu par le gestionnaire ZigBee receveur peut également être routé vers le gestionnaire ZigBee expéditeur du même nœud. Ceci permet donc un routage NDN sur plusieurs sauts via ZigBee.

Notons que ce qui fait la force du *NDN-over-ZigBee* proposé est qu'il est ***personnalisable*** et ***peut être utilisée*** par toute autre application NDN, car le gestionnaire est indépendant de l'application. Les Figures 3.10 et 3.11 montrent les traitements effectués respectivement par le gestionnaire *expéditeur* et le gestionnaire *receveur* :

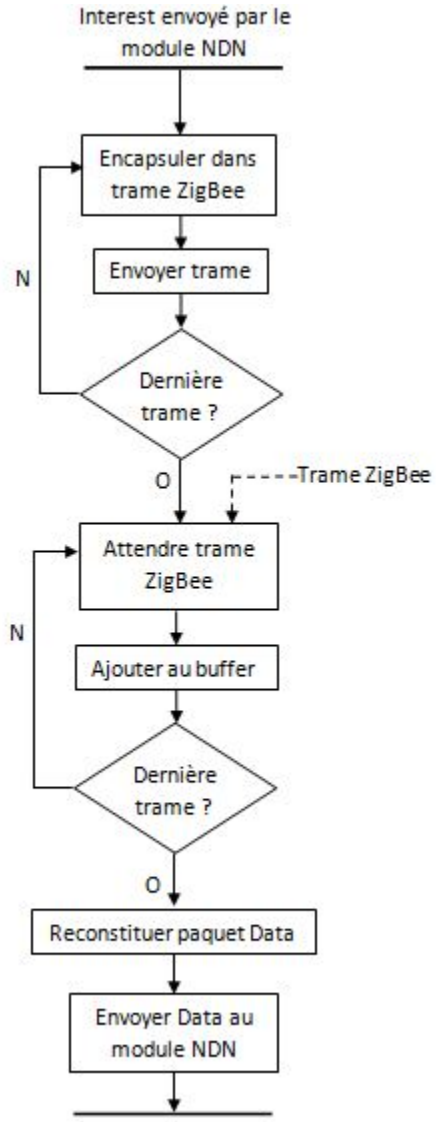

FIGURE 3.10 – Traitement expéditeur

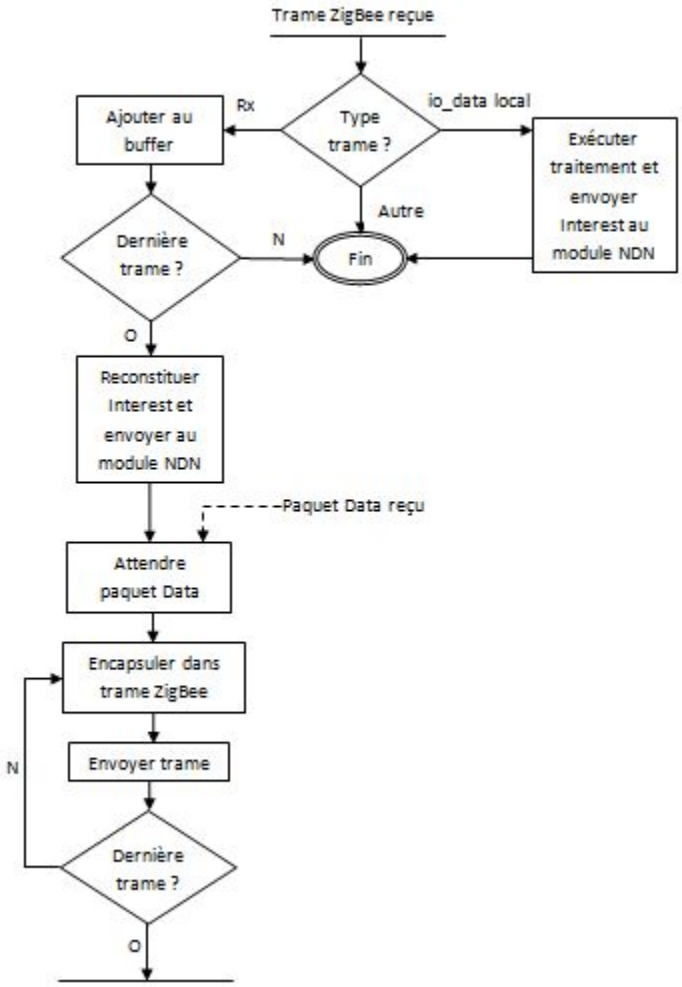

FIGURE 3.11 – Traitement receveur

3.3.4 Composants logiciels

Le système de parking intelligent proposé contient les applications NDN suivantes :

Application du serveur

Elle permet de traiter quatre types de requêtes, correspondant à quatre noms d'Interests : (i) l'ajout d'une nouvelle place de stationnement. Envoyée par une borne pour ajouter une nouvelle place de stationnement. (ii) le marquage d'une place comme étant occupée ou libre. Envoyée par la borne dès qu'une place change d'état. (iii) la demande de réservation d'une place. Envoyée par le véhicule qui cherche une place de stationnement. (iv) information sur la concentration de CO_2, envoyée périodiquement par le gestionnaire ZigBee de la borne.

Ces requêtes sont traitées par le serveur qui renvoie un paquet Data indiquant un acquittement ou une erreur pour les requêtes *i*, *ii* et *iv*. Pour la requête *iii*, si une place est trouvée, il envoie les coordonnées de cette place, le numéro de la place et la durée *t* avant expiration de la réservation, sinon il renvoie une réponse indiquant qu'aucune place n'est disponible ainsi qu'une durée d'attente à observer avant de renouveler la demande. La Figure 3.12 montre les quatre types de requêtes possibles et les entités impliquées dans leur utilisation :

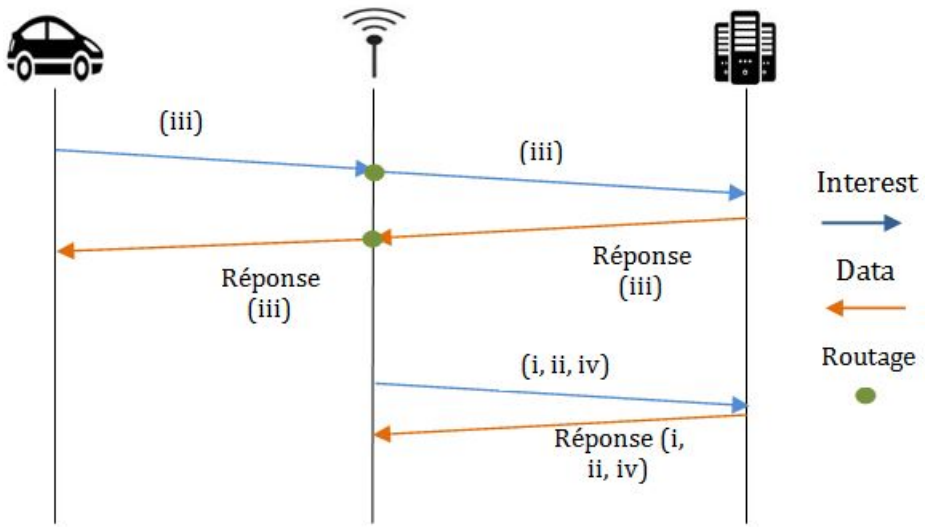

FIGURE 3.12 – Les quatre types de requêtes possibles

Le mécanisme de réservation fonctionne comme suit : la place proposée au véhicule est marquée provisoirement pour une durée limitée. Si après cette durée, la place est toujours libre, elle pourra être attribuée à un autre véhicule. Ainsi, si un conducteur change d'avis après avoir réservé une place ou s'il tarde à arriver, la borne n'enverra pas de mise à jour et cette place sera réutilisée. Idéalement, la durée de validité doit être calculée selon des informations sur le trafic et la vitesse, et doit être proche du temps réel que mettra le véhicule pour atteindre la place. La Figure 3.13 montre un échange dans lequel le véhicule réserve une place et occupe cette place avant expiration du délai de validité.

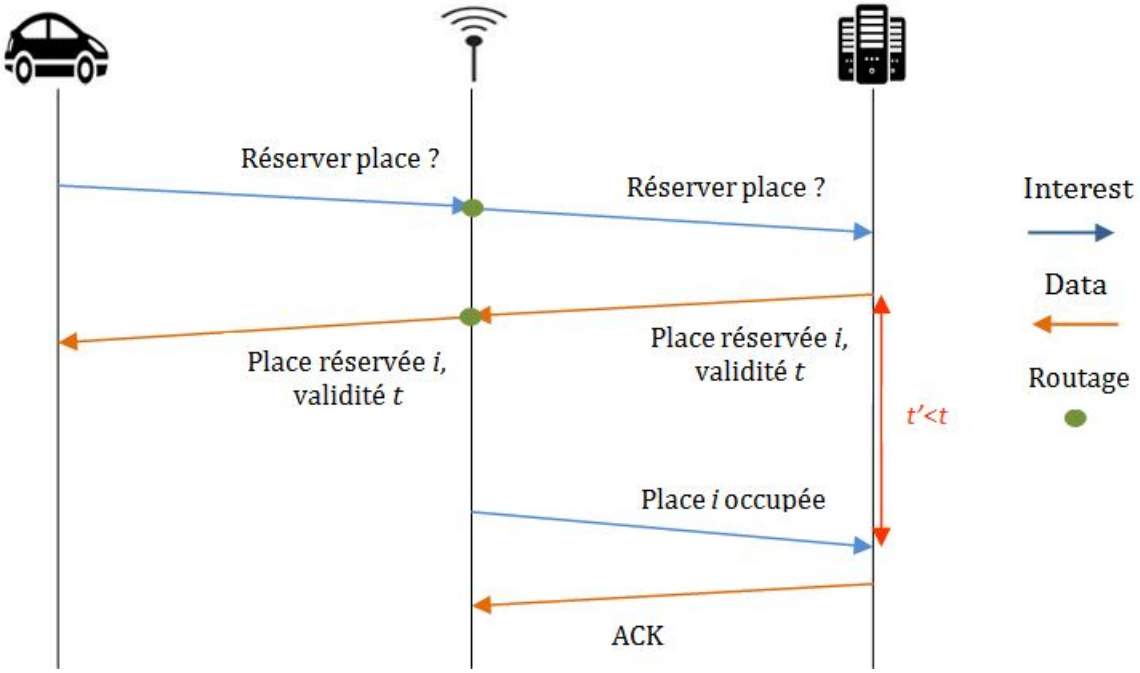

FIGURE 3.13 – Réservation d'une place avec confirmation

Pour le serveur, le choix de la place à proposer à un véhicule est très important. Il permettra de rendre le meilleur service possible en proposant la place la plus proche et la plus adaptée au véhicule, tout en régulant le trafic global et en assurant la fluidité de la circulation. Dans l'implémentation actuelle, le choix de la place

se fait sur la base de la plus petite distance Euclidienne entre le véhicule et les places disponibles. Mais ce traitement peut être étendu en croisant les informations du serveur avec d'autres informations concernant l'itinéraire, l'état de la route, la météo, le trafic, etc. Le serveur gère également des types de places différents tels que les places handicapés, espace familial, places de camions, de bus, etc. L'indice de CO_2 fourni par les bornes permet également au serveur de limiter ou d'interdire l'usage de tel ou tel parking en raison de son fort taux de pollution.

Application embarquée dans la borne/passerelle

Son traitement principal consiste à surveiller constamment l'état de ses capteurs de détection de véhicules. Si un changement d'état est détecté, une requête est envoyée au serveur sous forme d'Interest dont le nom porte les informations nécessaires. Lors de son premier démarrage, l'application envoie une série de requêtes d'ajout de nouvelles places au serveur.

Si une requête de mise à jour de l'état d'une place donne comme réponse « *Aucune place modifiée* », la borne en déduit que la place est inconnue du serveur et envoie une requête d'insertion de cette place avec son état correspondant. S'il n'y a aucune réponse ou si la réponse est « *Erreur* », la borne observe une période d'attente puis renouvèle sa requête. La Figure 3.14 montre ce traitement :

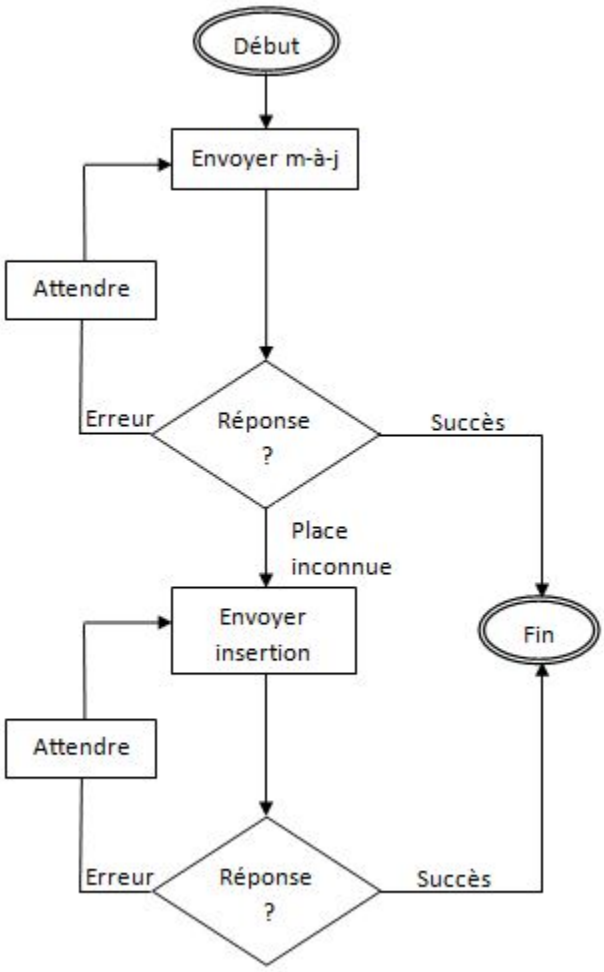

FIGURE 3.14 – Mise à jour de l'état d'une place

Pour être sûr qu'une place est bien occupée et que l'obstacle détecté n'est pas une perturbation momen-

tanée, le processus de surveillance de la borne effectue une vérification quand il détecte qu'une place vient d'être occupée. Cette vérification se déclenche en parallèle quelques secondes après avoir détecté qu'une place est occupée. Le processus de surveillance est illustré dans la Figure 3.15.

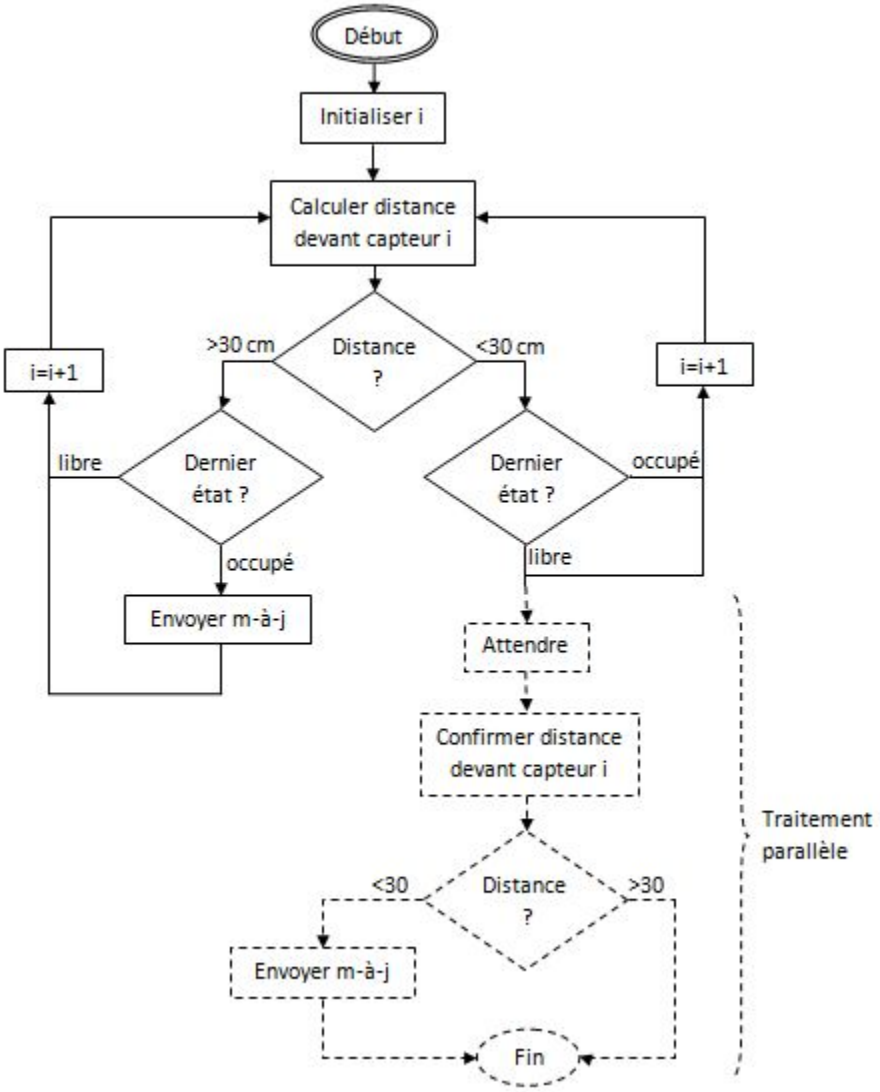

FIGURE 3.15 – Surveillance de l'état des capteurs et envoie des mises à jour

Pour servir de point d'accès aux véhicules, la borne doit router les Interests reçus via ZigBee vers le réseau global. L'application n'intervient aucunement dans cette opération, c'est le module NDN qui s'occupe de router les Interests. L'indépendance entre le routage des Interests du véhicule et l'application de la borne nous permettra d'utiliser la borne/passerelle comme point d'accès ZigBee pour n'importe quelle autre application future (autres types de réservations, collecte d'informations, etc.).

Application embarquée dans le véhicule

Elle permet de réserver une place de stationnement. Si la réponse est positive, une carte affiche la position actuelle du véhicule, la localisation de la place réservée, le numéro exact de la place ainsi que le temps restant avant expiration de cette réservation. La durée de validité de la réservation est contenue dans le champ *FreshnessPeriod* du Data retourné, ce qui permet à la fois d'avoir des paquets ***plus petits*** et ***réutilisables*** tant qu'ils sont valides.

Les échanges entre le véhicule et le serveur sont de type *pull*. En effet, le véhicule envoie un Interest pour « récupérer » une information du serveur. L'échange de type *pull* est l'échange par défaut dans NDN. Les échanges entre la borne et le serveur sont de type *push* ; la borne décide de « l'envoie » d'informations à des instants inconnues au préalable. Nous avons géré ces échanges *push* en nous inspirant des « *notification Interst* » proposés dans [53], ce mécanisme consiste à exprimer l'information dans le nom même de l'Interest et d'utiliser le Data retourné comme acquittement. Les types d'échanges dans l'IoT et leur gestion par NDN sont discutés plus en détails dans la Section 2.5.

3.3.5 Espace de noms et routage

L'espace de noms que nous avons adopté s'inspire de celui proposé dans [51] concernant les véhicules ; notamment en exprimant les coordonnées GPS dans les noms NDN. Les trois applications de notre système fonctionnent avec le même préfixe *'/parking'*. Ce préfixe commun représente l'espace de noms du système de parking dans sa globalité. Après ce préfixe, un autre composant détermine le type de requête dont il est question dans un Interest, suivi par les informations à exploiter pour exécuter cette requête. Ainsi, les Interests envoyés peuvent être nommés (les valeurs entre <> sont complétées selon l'état) :

- *'/parking/insert/<IdParking>/<NumSpace>/<SpaceType>/<xLocParking>/<yLocParking>/ <SpaceState>/<Timestamp>'* : requête d'ajout d'une nouvelle place de stationnement.
- *'/parking/update/<IdParking>/<NumSpace>/<NewState>/<Timestamp>'* : requête de mise à jour de l'état d'une place.
- *'/parking/getSpace/<SpaceType>/<xLocVehicle>/<yLocVehicle>/<IdVehicle>'* : requête de réservation d'une place de stationnement.
- *'/parking/co2/<IdParking>/<Value>/<Timestamp>'* : indice de CO_2 à proximité d'une borne de parking. Envoyé par son gestionnaire ZigBee. L'information peut également concerner un autre capteur (CO, température, etc.) selon l'installation.

Où :

- <*IdParking*> représente l'identifiant du parking, il doit être unique pour chaque parking.
- <*NumSpace*> est le numéro de la place dans le parking.
- <*SpaceType*> et <*SpaceState*> sont respectivement le type de la place (normal, handicapé, familial, etc.) et son état (libre ou occupée).
- <*xLocParking*> et<*yLocParking*> représentent la localisation du parking et donc de la place.
- <*NewState*> est le nouvel état de la place.
- <*Timestamp*> est la date d'envoie de l'Interest.
- <*IdVehicle*> représente l'identifiant du véhicule, il doit être unique pour chaque véhicule.
- <*xLocVehicle*> et <*yLocVehicle*> représentent la localisation du véhicule.
- <*Value*> est l'indice CO_2 (en ppm) calculé à partir du capteur de CO_2.

L'espace de noms actuel peut être enrichi de manière à permettre une hiérarchisation ou une agrégation des parkings selon des zones de la ville par exemple, dont les requêtes seraient routées vers le serveur de chaque zone. Ou encore, ajouter d'autres types de services avec d'autres types de requêtes sans avoir à modifier en

profondeur l'architecture actuelle.

Une fois ces noms posés, le routage NDN de base s'applique sur les deux premiers composants pour router chaque Interest vers l'application ciblée.

Dans le véhicule, le gestionnaire ZigBee expéditeur fonctionne sur le préfixe *'/parking'*. Lorsqu'un Interest *'/parking/getSpace/...'* est envoyé par l'application du véhicule, il est localement routé vers ce gestionnaire qui l'envoie via ZigBee. Il est ensuite reçu par le gestionnaire ZigBee receveur d'une borne/passerelle qui le transmet au module NDN local afin d'être routé à travers le réseau jusqu'au serveur central.

La Figure 3.16 donne les tables FIB minimales des trois entités du système permettant de réaliser le routage de nos Interests :

Véhicule			**Borne/passerelle**			**Serveur central**		
Préfix	*Face*	*Cout*	*Préfix*	*Face*	*Cout*	*Préfix*	*Face*	*Cout*
/parking	{Gestionnaire ZigBee (mode émission)}	0 (local)	/parking	{eth0}	1 (ou >)	/parking	{Application du serveur}	0 (local)

FIGURE 3.16 – Tables FIB du système

3.3.6 Avantages de NDN dans notre système

L'architecture NDN a grandement facilité la réalisation de notre système. Aussi, les mécanismes NDN nous permettent de gérer plusieurs cas intéressants tels que :

Cas où plusieurs bornes reçoivent l'Interest du véhicule

Comme vu plus haut, le véhicule envoie son Interest à la borne/passerelle de son réseau ZigBee, mais dans d'autres applications il se peut qu'il l'envoie en diffusion, et donc il est possible que l'Interest soit reçu par plusieurs bornes/passerelles qui toutes vont le router. Ceci ne pose pas de problèmes car les multiples Interests envoyés par les bornes/passerelles sont exactement les mêmes que l'original. Dans le routage NDN, seul le premier Interest sera routé jusqu'au serveur, les autres seront repérés comme des redondances et seront supprimés par un routeur dès qu'il les détecte. Ainsi la duplication d'Interests causée par la diffusion ZigBee ne saturera pas le réseau.

Cas de perte d'information au niveau du véhicule

Si l'application du véhicule plante ou même si son système redémarre après avoir réservé une place de stationnement, il existe un moyen de récupérer cette information sans remonter jusqu'au serveur central ; c'est donc un moyen rapide, qui n'encombre pas le réseau et qui n'engendre pas un traitement supplémentaire au serveur. Ceci est possible grâce au *in-network storage* de NDN combiné avec les indicateurs optionnels dans les paquets : La réponse du serveur à la demande de réservation est un paquet Data ayant le même nom que l'Interest et un FreshnessPeriod égal à la durée de validité de la place proposée. Donc, tant que la place est valide, la réponse (i.e le Data) sera aussi valide. Comme les Data qui transitent par les nœuds sont mis en cache durant une certaine période, si un Data est encore valide dans un nœud, il sera renvoyé comme réponse à l'Interest sans aller plus loin. Après une perte d'information, il suffira de renvoyer le dernier Interest pour retrouver la réponse correspondante. Et si la réponse a expiré, alors l'Interest arrivera jusqu'au serveur central et sera traité.

Cas où la passerelle ayant transmis l'Interest n'est plus à la portée du véhicule

Bien que les véhicules ne se déplacent pas à très grande vitesse, il se peut qu'un véhicule transmette un Interest par une passerelle puis celui-ci sort de la portée de cette passerelle avant d'avoir la réponse. Dans ce cas, l'application du véhicule constatera un *Timeout*, il suffira de renvoyer l'Interest qui passera à travers une autre passerelle : Si le serveur n'a pas déjà traité l'Interest ou si il l'a traité et que la réponse a expiré, l'Interest arrivera au serveur qui le traitera et la réponse sera renvoyée à travers la nouvelle passerelle (échange normal). Si le serveur a traité le premier Interest, le Data correspondant, mis en cache dans le réseau, sera renvoyé comme réponse s'il est toujours valide.

Cas de la surcharge du réseau

Grâce à la PIT qui contient les Interests en attente de réponse et qui est contenue dans chaque nœud NDN, les passerelles ont la possibilité de limiter le nombre de requêtes venant des véhicules en limitant la taille de leur PIT. Avec ce même mécanisme, le serveur peut également limiter le nombre de requêtes qu'il traite. Plus encore, en étudiant la taille moyenne de la PIT, on aura une idée sur le flux de requêtes entrant. En d'autres termes, de précieuses informations statistiques seront fournies par le système, ainsi qu'un moyen de gérer les requêtes, sans générer plus de trafic.

Cas de la sécurité

La sécurité est implémentée et gérée dans le protocole réseau NDN. Tous les paquets Data envoyés par le serveur aux véhicules sont signés, le véhicule peut à tout moment vérifier l'authenticité des données reçues en recalculant la signature et en comparant la clé attendue et la clé utilisée pour signer les paquets Data en appliquant le modèle de sécurité proposé dans [24].

Comme les bornes/passerelles envoient des informations au serveur en utilisant les Interests, ces Interests doivent être authentifiables. Pour cela, ils sont signés. La procédure est implémentée dans NDN : les informations sur la signature (certificat et clé) appelées *SignatureInfo* sont ajoutées au nom de l'Interest, puis le nom obtenu est signé et la signature est ajoutée à la fin du nom.

Cas d'une implémentation avec TCP/IP

La Figure 3.17 représente la pile de protocoles IoT typique pour l'IP (à gauche) et pour NDN (à droite). Les points en bleu représentent les endroits dans la pile où nous sommes intervenus pour réaliser notre prototype avec NDN. Les points en orange représentent les endroits où on serait obligés d'intervenir si on réalisait le même prototype avec l'architecture TCP/IP classique. Ceci résume bien la simplicité de mise en œuvre avec NDN et sa compatibilité avec l'IoT.

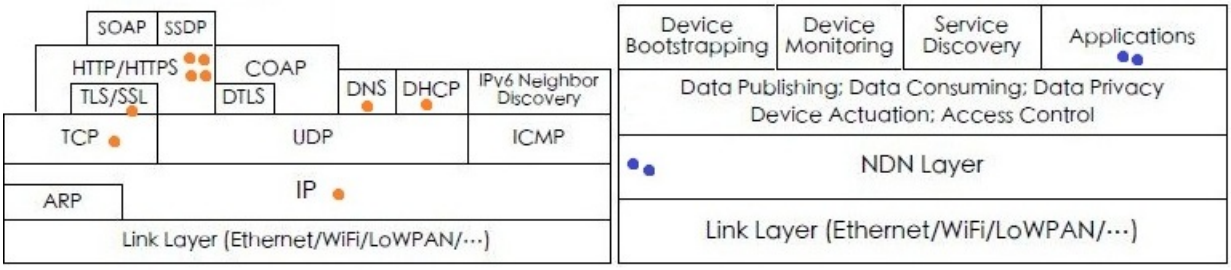

FIGURE 3.17 – Comparaison pile IoT IP et NDN [14](modifiée)

3.4 Conclusion

Les détails de conception vus dans ce chapitre illustrent bien les avantages de NDN par rapport à l'architecture IP. Ceci montre que cette nouvelle architecture est parfaitement compatible avec les contraintes de l'IoT. Bien que notre solution s'inspire de certains travaux NDN sur l'IoT concernant l'architecture, l'espace des noms et la gestion du trafic de type « *push* », elle est l'une des premières (si ce n'est la première) à s'intéresser aux parkings intelligents en utilisant NDN. Elle est également la première à utiliser la norme ZigBee comme moyen de communication sans fil pour NDN, élargissant ainsi les possibilités concernant les objets mobiles.

Dans le chapitre suivant, nous verrons quels sont les composants matériels utilisés dans la réalisation de notre prototype et les traitements les plus importants. Nous donnerons également les configurations et les étapes de lancement de notre système, ainsi que les tests de toutes les fonctionnalités décrites précédemment.

Chapitre 4

Réalisation et test du prototype

4.1 Introduction

Nous allons voir de ce chapitre les composants matériels intervenant dans la réalisation de notre prototype et les traitements les plus importants du système. Nous citerons les bibliothèques utilisées pour la manipulation de ces équipements et l'implémentation des différentes applications. Nous donnerons également les configurations dont le système a besoin pour fonctionner et les étapes de son lancement. Ensuite, une série de figures illustreront les tests de toutes les fonctionnalités fournies par notre système. Enfin, nous donnerons quelques points concernant la fiabilité du système.

4.2 Réalisation et implémentation du prototype

Deux applications sont nécessaires au fonctionnement du serveur : le module NDN (NFD v. 0.4.0) et l'application du serveur. Dans notre prototype, le serveur est un PC doté d'un processeur Intel® Core™ i3 avec une distribution Linux Ubuntu 14.04 LTS 64 bit.

Pour la borne/passerelle, trois applications sont nécessaires : le module NDN (NFD v. 0.4.0), le gestionnaire ZigBee (mode réception) et l'application de surveillance des places de stationnement. La borne/passerelle est une carte Raspberry Pi 1 B+ (Annexe A). Les capteurs de détection de véhicules sont des radars à ultrasons HC-SR04 (Annexe C) reliés au port GPIO du Raspberry Pi.

Le véhicule nécessite également trois applications : le module NDN (NFD v. 0.4.0), le gestionnaire ZigBee (mode émission) et l'application principale du véhicule. Ces applications sont embarquées dans un Raspberry Pi 2 B (Annexe A).

Les communications ZigBee sont effectuées avec des modules Xbee-Pro (Annexe B) reliés aux Raspberry Pi via l'UART. L'Xbee-Pro est l'un des modules matériels les plus utilisés dans le prototypage et les applications finales. Sa portée est de 1,5 Km en environnement extérieur.

L'indice de CO_2 dans l'air est obtenu à l'aide d'un capteur de gaz MQ135 (Annexe D) relié à la broche *AD0* du module XBee de la borne/passerelle. Le gestionnaire ZigBee s'exécutant sur la borne/passerelle s'occupe de lire périodiquement l'état de la broche *AD0*, de calculer la concentration de CO_2 dans l'air (en ppm) à partir de la valeur lue et d'envoyer un Interest qui sera reçu par le serveur central.

Les applications et les gestionnaires ZigBee ont été implémentés en langage Python et les bibliothèques : PyNDN2 [67], bibliothèque officielle NDN pour le langage Python. PyXBee [68], bibliothèque open source pour l'utilisation des modules XBee, légèrement modifiée pour gérer les délais d'attente. PySerial [69], bi-

bliothèque open source pour les communications série. *RPi.GPIO*, bibliothèque Python pour la gestion des ports GPIO des Raspberry Pi.

La Figure 4.1 montre une illustration du montage représentant la borne et le serveur (voir en Annexe les schémas de connexion des différents composants matériels et leurs configurations) :

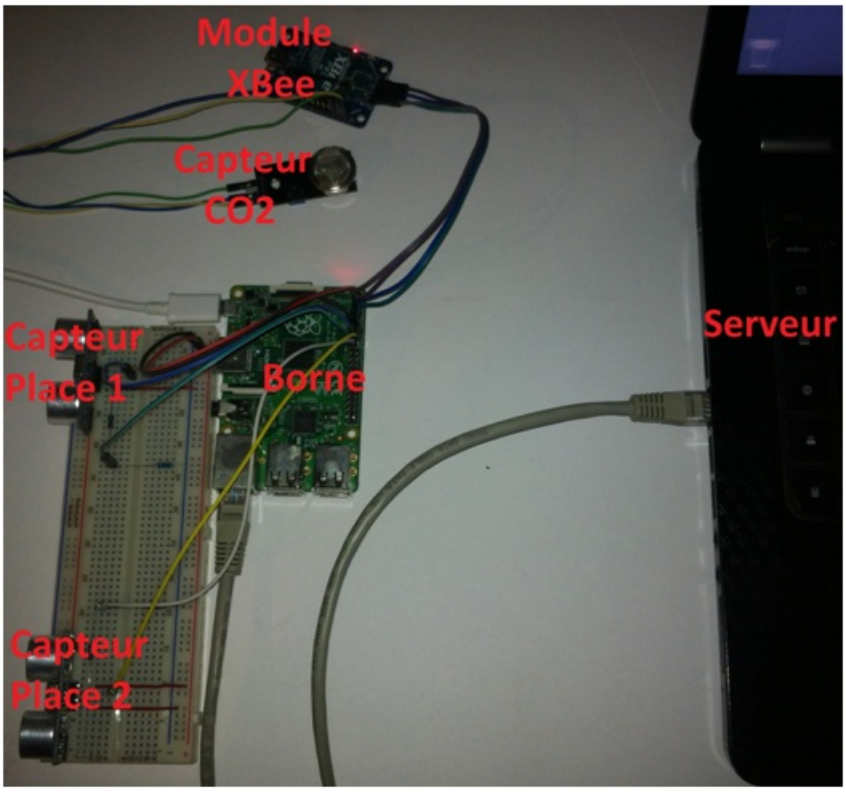

FIGURE 4.1 – Montage représentant la borne et le serveur

La Figure 4.2 montre une illustration du montage embarqué dans le véhicule (voir en Annexe les schémas de connexion des différents composants matériels et leurs configurations) :

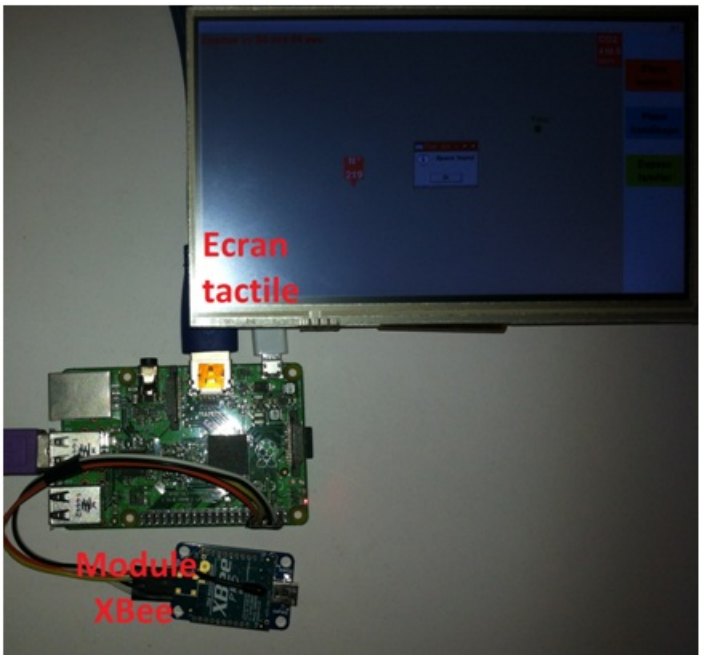

FIGURE 4.2 – Montage embarqué dans le véhicule

4.3 Configuration et lancement du système

Voici les étapes nécessaires au premier lancement du système et leurs commandes :

4.3.1 Serveur

1. Démarrer le module NDN : **nfd-start**
2. Démarrer l'application serveur avec le paramètre *first* pour réinitialiser la Base de données (uniquement au premier démarrage) : **python server.py f**

4.3.2 Borne de parking

1. Démarrer le module NDN : **nfd-start**
2. Ajouter l'information de routage pour le préfixe *'/parking'* : **nfdc register /parking udp ://<@-next-hop>**
   Comme les systèmes et les équipements actuels ne supportent pas encore le protocole NDN, le module NDN encapsule les paquets dans des protocoles traditionnellement supportés ; UDP/IP dans notre cas. Toutefois, NDN est destiné à fonctionner directement sur des protocoles de couche liaison.
3. Personnalisation du gestionnaire ZigBee : Écrire dans le fichier de configuration du gestionnaire ZigBee les traitements correspondant aux broches *Input* actives du module XBee (dans notre prototype, la broche *AD0* est reliée au capteur de CO_2), ainsi que le nom de l'Interest qui portera le résultat du traitement. La configuration dans notre cas est la suivante :

```
functions={
'adc-0':{'name':'parking/co2/1','function':getPPM},
'adc-1':{'name':None,'function':None},
'adc-2':{'name':None,'function':None},
'adc-3':{'name':None,'function':None},
```

```
'adc-4':{'name':None,'function':None},
'adc-5':{'name':None,'function':None},
'adc-6':{'name':None,'function':None}
}
```

4. Démarrer le gestionnaire ZigBee receveur : **python XB_producer.py**
5. Démarrer l'application principale avec le paramètre *first* pour envoyer les requêtes d'insertion des places (uniquement au premier démarrage) : **sudo python borne.py f**

4.3.3 Véhicule

1. Démarrer le module NDN : **nfd-start**
2. Démarrer le gestionnaire ZigBee expéditeur avec le préfixe *'/parking'* : **python XB_consumer.py /parking**
3. Lancer l'application principale : **python vehicle.py**

4.4 Tests

4.4.1 Test du serveur

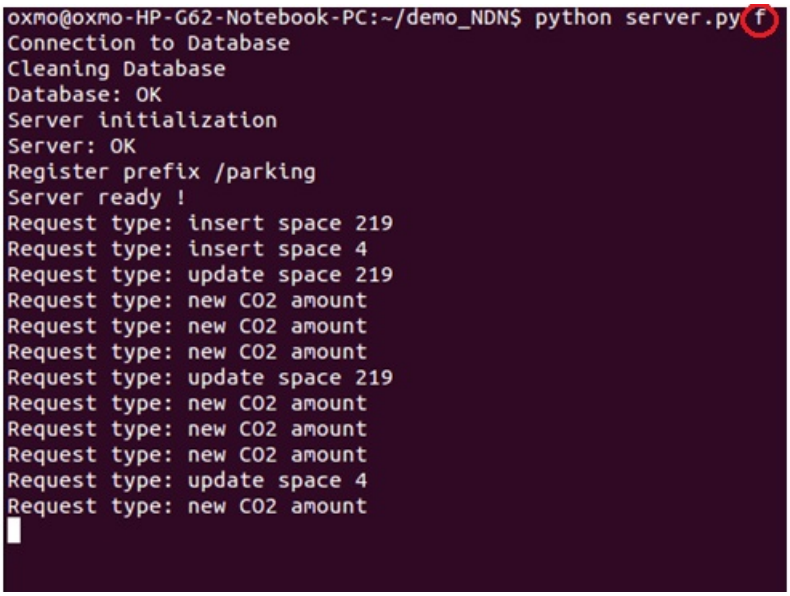

FIGURE 4.3 – Premier démarrage du serveur et réception de différentes requêtes (insertion, m-à-j, indice CO2)

4.4.2 Test de la borne de parking

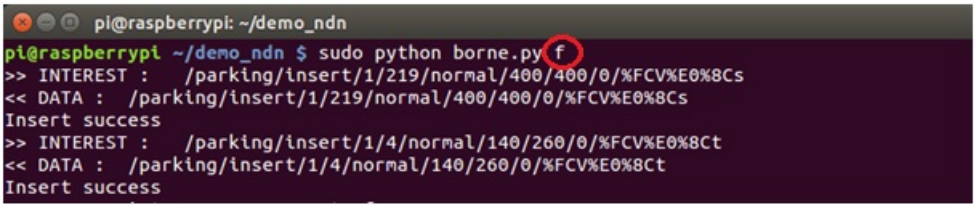

FIGURE 4.4 – Premier démarrage et envoie des requêtes d'insertion des places

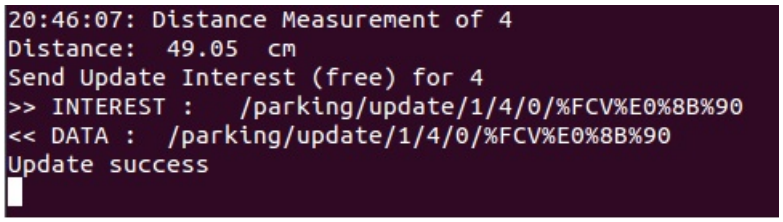

FIGURE 4.5 – Une place vient de se libérer

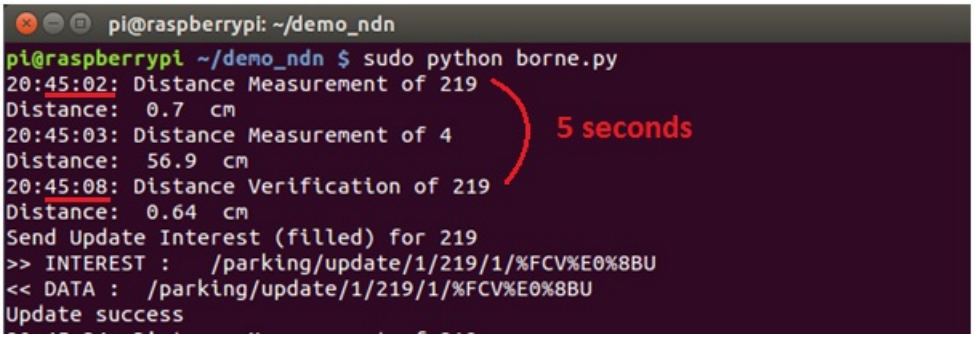

FIGURE 4.6 – Une place vient d'être occupée

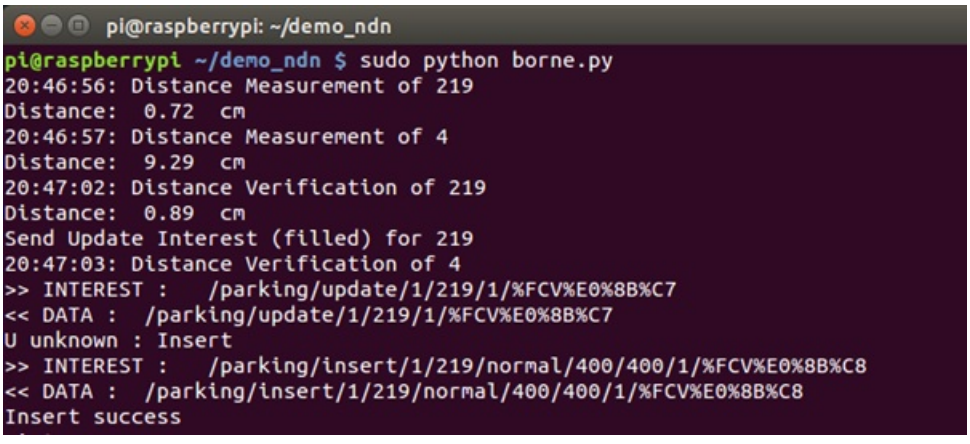

FIGURE 4.7 – Mise à jour de l'état d'une place inconnue du serveur

```
pi@raspberrypi ~/demo_ndn $ sudo python borne.py
23:32:40: Distance Measurement of 219
Distance:  0.67  cm
23:32:41: Distance Measurement of 4
Distance:  64.93  cm
Send Update Interest (free) for 4
>> INTEREST :   /parking/update/1/4/0/%FCV%E0%B2%9A
*TIMEOUT :  /parking/update/1/4/0/%FCV%E0%B2%9A
U timeout : resend
>> INTEREST :   /parking/update/1/4/0/%FCV%E0%B2%9E
*TIMEOUT :  /parking/update/1/4/0/%FCV%E0%B2%9E
```

FIGURE 4.8 – Le serveur ne répond pas à une mise à jour

```
23:34:46: Distance Verification of 219
Distance:  0.77  cm
Send Update Interest (filled) for 219
>> INTEREST :   /parking/update/1/219/1/%FCV%E0%B3%17
<< DATA :  /parking/update/1/219/1/%FCV%E0%B3%17
U error : retry after 10s
>> INTEREST :   /parking/update/1/219/1/%FCV%E0%B3%21
<< DATA :  /parking/update/1/219/1/%FCV%E0%B3%21
```

FIGURE 4.9 – Erreur serveur lors d'une mise à jour

Lors d'une mise à jour, si le serveur ne répond pas, la borne renvoie tout de suite la mise à jour. Mais si le serveur répond en indiquant une erreur inconnue, la mise à jour est renvoyée après une période d'attente (ici 10 secondes).

4.4.3 Test du véhicule

L'application embarquée du véhicule dispose d'une interface graphique destinée à un écran tactile. Elle se présente sous forme de carte indiquant la localisation du véhicule, avec des boutons à droite permettant d'envoyer une réservation pour différents types de places de stationnement (Figure 4.10). Si la réservation d'une place de stationnement donne un résultat positif, la carte affiche la localisation du parking, le temps restant avant expiration de cette réservation ainsi que l'indice de CO_2 aux alentours du parking destination (Figure 4.11). Dans le cas contraire, un message indiquant qu'aucune place n'est disponible est affiché ainsi que le temps restant avant de pouvoir renouveler la demande (Figure 4.12). Si aucune réponse n'est reçue, un message indique que l'Interest envoyé n'a pas reçu de réponse (Figure 4.13).

FIGURE 4.10 – Interface graphique initiale

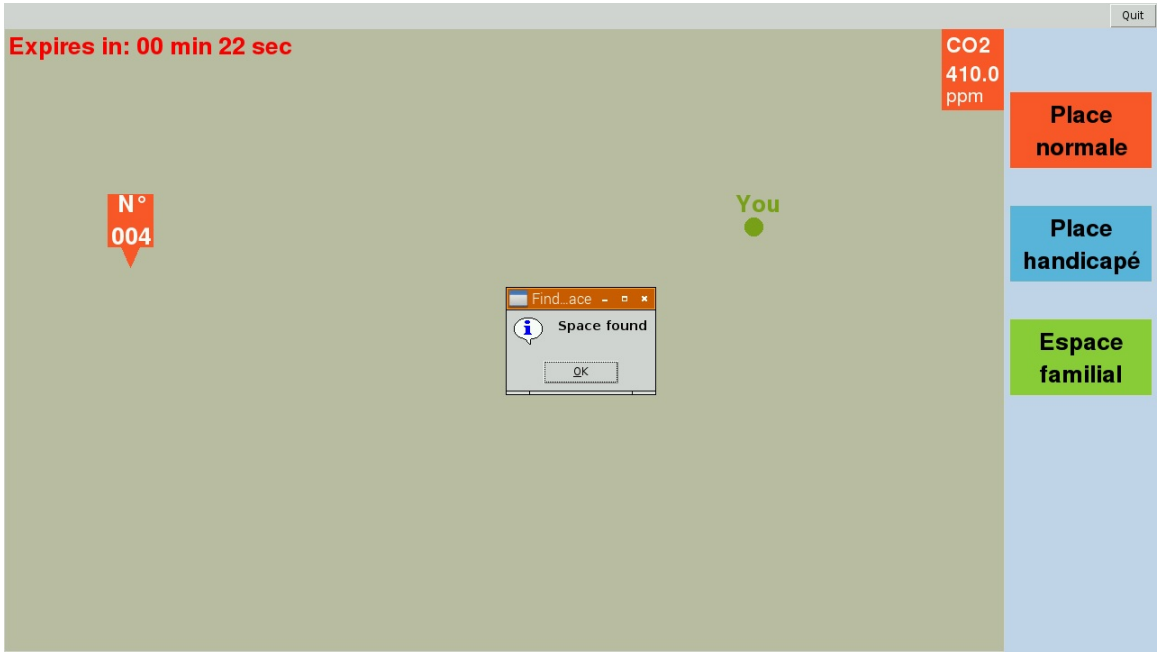

FIGURE 4.11 – Réservation d'une place (cas 1 : Place trouvée)

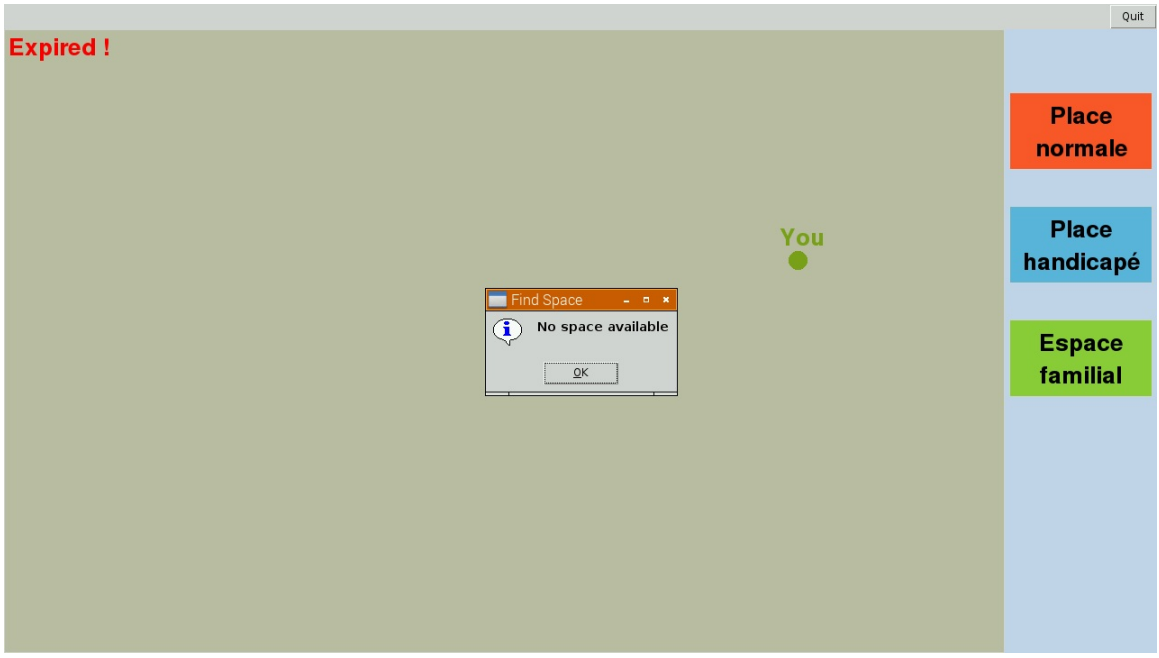

FIGURE 4.12 – Réservation d'une place (cas 2 : Aucune place n'est disponible)

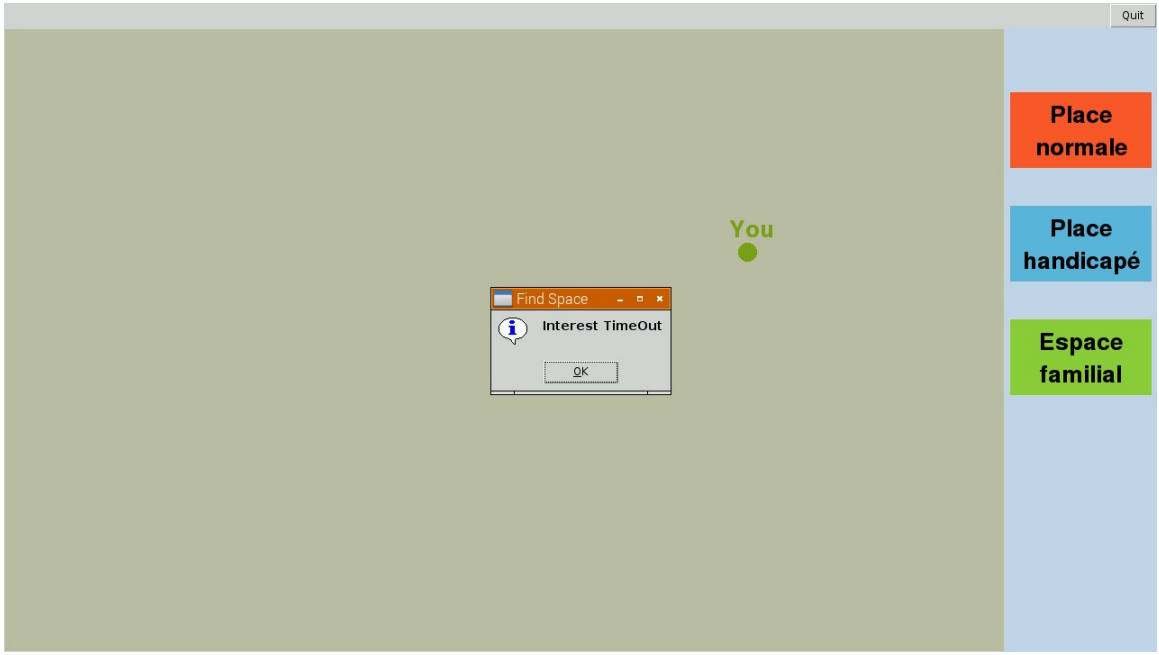

FIGURE 4.13 – Aucune réponse reçue

Si l'application ou le système du véhicule s'arrête pendant qu'une place est réservée, une boite de dialogue est affichée au redémarrage de l'application pour proposer à l'utilisateur de renvoyer la dernière requête (Figure 4.14).

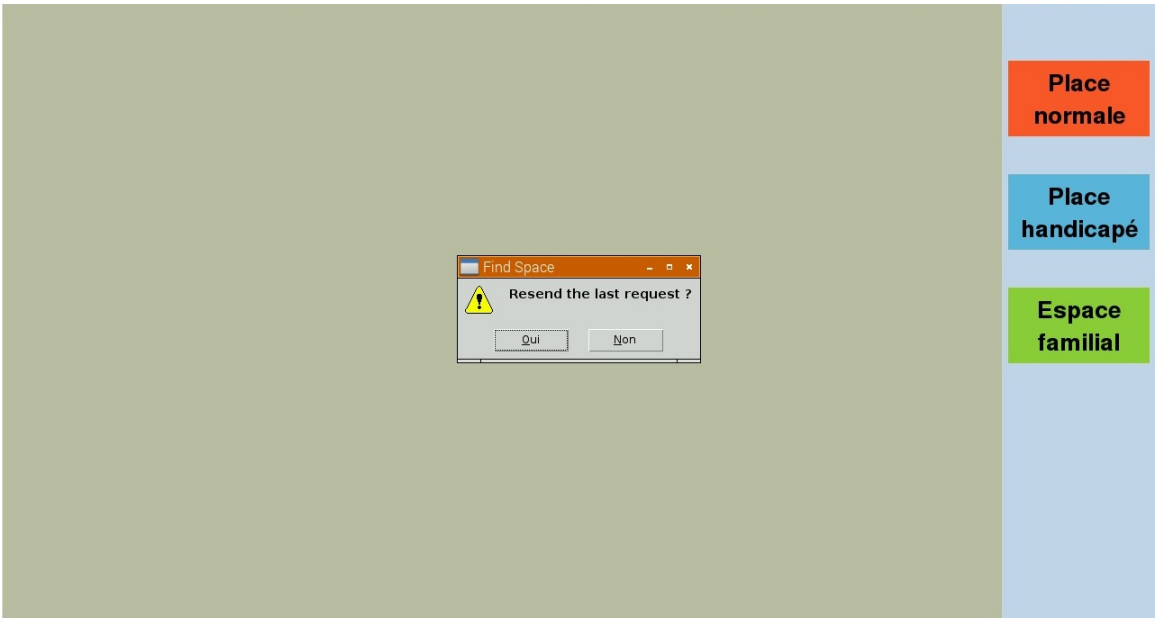

FIGURE 4.14 – Reprise après une panne (demande de renvoie de la dernière requête)

4.4.4 Test du gestionnaire ZigBee expéditeur

La Figure 4.15 montre trois cas de communications pris en charge par le gestionnaire ZigBee : (1) le cas où le module XBee distant ne répond pas, (2) le cas où l'application distante ne répond pas, et (3) le cas où l'échange ZigBee se déroule normalement. Le gestionnaire prend également en charge le cas où une trame inattendue est reçue et le cas où les données reçues ne constituent pas un paquet Data NDN. Dans ces deux derniers cas, les trames reçues sont ignorées.

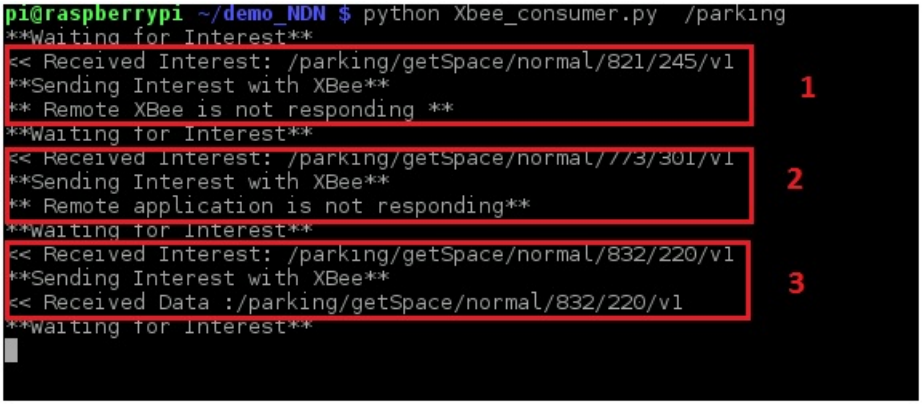

FIGURE 4.15 – Gestionnaire ZigBee expéditeur

4.4.5 Test du gestionnaire ZigBee receveur

La Figure 4.16 montre une utilisation normale du gestionnaire ZigBee receveur. Il envoie périodiquement des Interests indiquant l'indice de CO_2 calculé à partir du capteur de CO_2 relié au module XBee (adc-0), tout en gérant les Interests reçu et en envoyant le Data correspondant retourné par le module NDN.

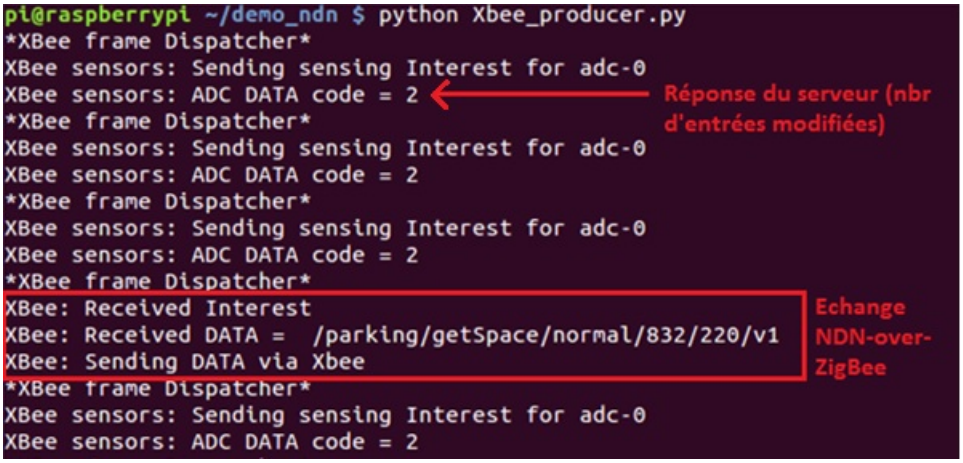

FIGURE 4.16 – NDN-over-ZigBee et envoies périodiques de l'indice CO2

4.5 Fiabilité et tolérance aux pannes

Le mécanisme de réservation d'une place de stationnement est un mécanisme distribué ; il consiste en trois étapes faisant intervenir deux à deux toutes les entités du système (véhicule, borne, serveur). Les trois étapes doivent être complétées dans l'ordre afin de réserver correctement une place (Figure 4.17) :

1. Le véhicule demande une place au serveur. C'est un échange Interest-Data entre le véhicule et le serveur. Si une place est disponible elle est proposée et pré-réservée.

2. S'accomplit lorsque le véhicule stationne dans la place proposée. C'est une action physique qui fait intervenir le véhicule et la borne du parking et non une communication entre applications.

3. La borne envoie une mise à jour de l'état de la place. C'est un échange Interest-Data entre la borne et le serveur.

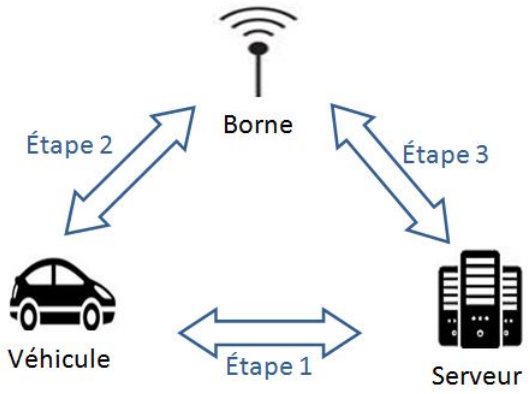

FIGURE 4.17 – Étapes du mécanisme de réservation

Nous avons choisi d'impliquer les trois entités dans le processus de réservation et de confirmation pour éviter que des usagers mal intentionnés ne réservent plusieurs places à la fois ou qu'ils réservent une place sans l'utiliser réellement. De plus, la deuxième étape du mécanisme nécessite que le véhicule soit stationné sur la place pré-réservée pour être complétée. Le fait qu'une action physique (i.e. se garer) soit nécessaire, associé au fait que les entités soient en général géographiquement éloignées rend le système plus difficile à tromper à distance de façon purement logicielle (Figure 4.18).

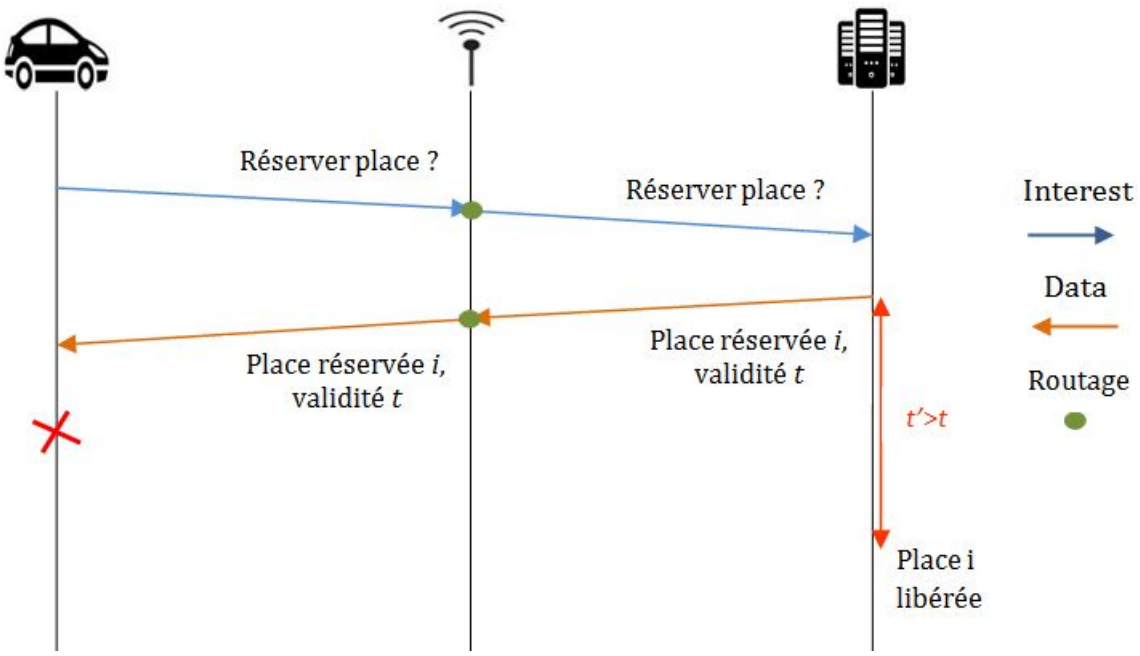

FIGURE 4.18 – Pré-réservation d'une place sans l'occuper

La Figure 4.19 montre l'affichage au niveau du serveur et l'affichage au niveau et de la borne lorsqu'un véhicule réserve une place et occupe cette place quelques instants après.

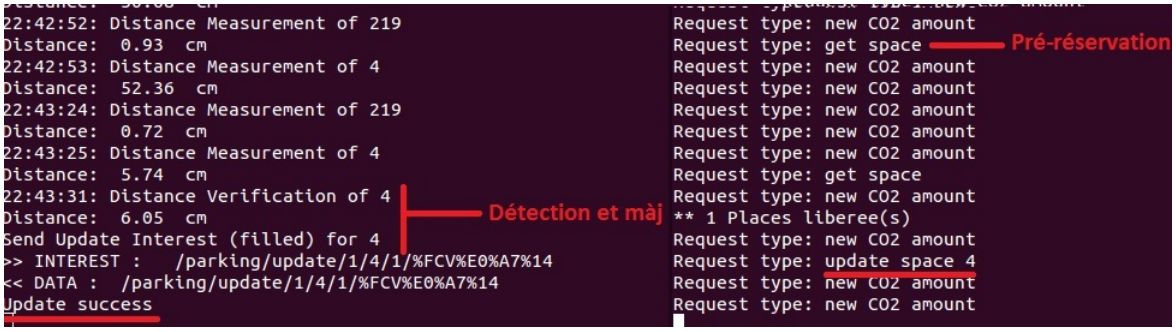

FIGURE 4.19 – M-à-j d'une place pré-réservée (à gauche la borne, à droite le serveur)

La tolérance aux pannes d'un système est sa capacité à continuer de fonctionner même si une partie du système n'est plus opérationnelle. Notre système peut globalement être divisé en deux parties : la partie permettant la communication des véhicules avec le serveur, et la partie permettant la communication entre les bornes/passerelles et le serveur. Ces deux parties sont indépendantes et chacune peut fonctionner si l'autre est défaillante ou même absente. En effet, un véhicule peut communiquer avec le serveur, et globalement tout le réseau, à travers une borne/passerelle même si l'application embarquée qui gère les capteurs ne fonctionne pas ; il suffit que l'antenne XBee et le gestionnaire ZigBee soient opérationnels. De même, si l'antenne XBee et le gestionnaire ZigBee d'une borne/passerelle sont absents, celle-ci pourra toujours communiquer avec le serveur car elle communique à travers le réseau global et non avec du ZigBee. Par ailleurs, cette indépendance nous permet d'assurer aux véhicules l'accès au serveur en équipant uniquement une partie des bornes/passerelles avec des antennes XBee.

4.6 Conclusion

Dans ce chapitre, nous avons présenté notre système de parking intelligent basé sur NDN. Les éléments et équipements intervenant, les détails de réalisation, ainsi que des tests y sont présentés. Notre système peut facilement être adapté pour prendre en charge d'autres applications destinées au villes intelligentes telles que l'éclairage publique et les panneaux de signalisation, mais il est aussi adapté à des applications plus fréquentes comme celles de la domotique ou du monitoring. C'est une véritable plateforme polyvalente de l'IoT.

Conclusion générale et perspectives

Conclusion générale

Il y a encore quelques années, les termes « Internet of Things » ou « Internet des objets » étaient presque inconnus du grand public et même des utilisateurs d'Internet. Seuls quelques passionnés et quelques projets de recherche au sein d'entreprises et d'universités s'intéressaient à ce nouveau concept, et y voyaient le futur inévitable d'Internet et même le futur de notre vie quotidienne, car cette nouvelle évolution permet désormais d'interagir avec le monde réel grâce à un ensemble de capteurs et d'actionneurs connectés à Internet, offrant des services de plus en plus pertinents et indispensables.

Ce concept est aujourd'hui devenu en partie une réalité. En partie seulement car des obstacles doivent encore être franchis pour faire de l'IoT une véritable révolution technologique. Ces obstacles sont pour la plupart techniques. Pour y faire face, certains tentent de développer des protocoles et des couches logicielles qui viennent se greffer à l'architecture actuelle d'Internet. De telles approches rencontrent de nombreux succès. La preuve est qu'elles nous ont permis d'avoir une première esquisse de l'IoT avec les services intéressants et très utiles qu'on connait aujourd'hui. Toutefois, ces middlewares éprouvent des difficultés à prendre en charge certaines contraintes, comme nous l'avons vu tout au long de ce document. C'est pour cela que d'autres approches essayent de proposer de nouvelles architectures entièrement étudiées pour les applications émergentes. Parmi ces architectures figure NDN qui, comme nous l'avons expliqué au début du document, englobe les forces de l'Internet actuel tout en évitant ses faiblesses.

C'est dans ce contexte que nous avons apporté notre pierre à cet édifice qu'est l'Internet du futur ; en utilisant l'architecture NDN dans une application de l'IoT tournée vers les Smart Cities. Nous avons choisi le cas des parkings intelligents car il synthétise les caractéristiques les plus importantes de l'IoT telles que la mobilité, la mise à l'échelle, l'utilité des noms, etc. Nous avons essayé de montrer, par l'implémentation d'un prototype, la capacité de NDN à supporter la mise en place d'applications de l'IoT, plus efficacement et plus simplement que les protocoles utilisés jusqu'ici.

Perspectives

Améliorations à court terme

Les coordonnées de localisation utilisées dans notre prototype sont des coordonnées fictives, non issues d'un GPS. Une des améliorations immédiates consisterait à utiliser un module GPS pour avoir la localisation, et par la suite utiliser la réponse du serveur pour envoyer un POI (Point Of Interest) à une application de navigation pour avoir l'itinéraire menant à cette place de stationnement.

Une autre amélioration importante serait de remplacer les liens filaires entre la borne/passerelle et les radars à ultrasons par un lien sans fil, infrarouge, Bluetooth ou même ZigBee afin de réduire le coût et le temps d'installation du système.

Perspectives à moyen et long terme

Afin d'améliorer notre prototype, nous devons le tester à grande échelle. Il faudrait pour cela effectuer des simulations pour étudier les communications NDN engendrées par un grand nombre de véhicules et de bornes/passerelles. Ces simulations nous permettront d'avoir une idée sur le comportement de notre système, sa fiabilité et ses faiblesses. Elles nous permettront également de mettre en évidence l'avantage de NDN dans l'IoT comparé à l'architecture TCP/IP. Pour effectuer de telles simulations, des données statistiques sur le trafic et les véhicules dans les villes doivent être disponibles. Aussi, le simulateur de NDN (ndnSIM) ne prenant pas en charge le protocole ZigBee, il est difficile actuellement de réaliser des simulations fiables.

Des traitements plus complexes peuvent être envisagés du coté serveur en combinant les informations sur les places de stationnement avec d'autres données concernant l'état du trafic, de la route, du climat, etc., pour choisir les places de stationnement de manière à réguler le trafic et fluidifier la circulation. D'autres systèmes tels que la durée de stationnement limitée ou même le payement électronique du parking peuvent facilement être pris en charge par notre plateforme grâce à NDN. Mais cela nécessite d'avoir des modèles théoriques permettant de traiter les informations recueillies et des compétences relevant d'autres domaines tels que l'Analyse de données.

L'amélioration la plus importante à long terme est d'enrichir le prototype actuel pour mettre en place une plateforme NDN complète, renfermant tous les mécanismes nécessaires à une ville intelligente. On pourrait dans un premier temps étudier et recenser ces mécanismes. Par la suite, on intégrera dans le gestionnaire ZigBee actuel des solutions pour gérer ces mécanismes, offrant ainsi un moyen de communication sans fil prometteur (ZigBee) pour NDN, et assurant une gestion sûr et personnalisée des équipements embarqués indispensables dans de telles applications.

Annexe A

La carte Raspberry Pi

A.1 Généralités sur le Raspberry Pi

Le Raspberry Pi (Figure A.1) est un nano-ordinateur monocarte à processeur ARM crée par David Braben, dans le cadre de sa fondation Raspberry Pi. Cet ordinateur, est destiné aussi bien à l'enseignement de la programmation et des systèmes embarqués qu'au prototypage et réalisation de plateformes IoT. Il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni sous forme d'une carte mère seule, sans boîtier, alimentation, clavier, souris ni écran ; dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération. Le but initial du projet Raspberry Pi était de permettre aux étudiants de s'équiper du même matériel qu'en cours, et ce à moindres frais. Malgré son faible coût, la carte dispose d'un panel d'entrées-sorties intéressant et se prête aussi bien à des applications statiques qu'embarquées.

Bien que consistant en une cible embarquée, le Raspberry Pi dispose néanmoins d'un OS officiel appréciable, Linux Debian. Cet OS a été surnommé Raspbian. Le système d'exploitation Windows IoT fait également partie des systèmes utilisables. La communauté étant très active sur cette plateforme, il est également possible d'y retrouver, entre autres, des systèmes tels que Fedora, ou encore XBMC/Kodi.

L'alimentation s'effectue par un connecteur micro-USB (5V, 1500-2000mA). Les premières générations pouvaient être alimentées par un port USB, via un HUB. Ce n'est plus le cas des nouvelles versions (A+/B+/B2).

A.2 Le port GPIO

Les modèles A et B possédaient un port GPIO à 26 points. Les modèles A+, B+ et B2 possèdent un port GPIO à 40 points, correspondant aux 26 points des modèles A/B plus 14 points supplémentaires. Un circuit conçu pour les versions A/B est donc compatible A+/B+/B2. Les entrées-sorties sont de type CMOS, et fonctionnent donc en +3.3V. Brancher des circuits TTL (+5V) sur les entrées-sorties nécessite une adaptation (pont diviseur de tension) comme nous allons le voir avec le capteur à ultrasons.

Le connecteur GPIO supporte les GPIO (entrées/sorties binaires) mais également les sorties PWM, les périphériques de communication (UART, I2C, SPI) et les alimentations 5V et 3.3V. Les broches peuvent avoir des fonctions différentes suivant qu'elles soient activées en tant que GPIO ou périphérique de communication. Le port GPIO n'est accessible qu'en mode *root*. Le Raspberry Pi ne possède pas de convertisseur analogique numérique pour son port GPIO.

L'ensemble des fonctionnalités peuvent être activées une à une selon les besoins. Il existe de nombreuses

bibliothèques dédiées au Raspberry Pi notamment en langage Python. Une des bibliothèques les plus utilisées du langage Python pour le port GPIO est la RPi.GPIO. Raspberry Pi autorise deux numérotations : celle de la sérigraphie du connecteur de la carte (GPIO.BOARD), ou la numérotation électronique de la puce (GPIO.BCM).

A.3 Modèles utilisés

Il existe plusieurs modèles de Raspberry Pi, tels que le modèle 1A, 1A+, 1B+, 2B, etc. Voici les caractéristiques des deux modèles utilisés dans notre prototype [70] :

- Raspberry Pi 1 Modèle B+
  Annoncé en juillet 2014. Ses caractéristiques sont les suivantes :
  - Dimension : 85,60 mm x 53,98 mm x 17 mm
  - Poids : 45g
  - GPIO 40 broches
  - 4 ports USB 2.0 et meilleur comportement en cas de surcharge
  - Micro SD
  - Meilleur circuit audio
  - Sortie HDMI
- Raspberry Pi 2 Modèle B
  Sorti le 2 février 2015. Plus puissant, il est équipé d'un processeur Broadcom BCM2836 quatre cœurs ARMv7 à 900 MHz, et de 1 Go de RAM. Il possède les mêmes dimensions et connectiques que le modèle B+

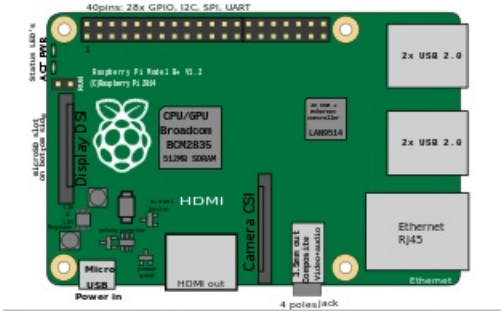

FIGURE A.1 – Raspberry Pi modèle 1B+ et modèle 2B [70]

La Figure A.2 montre les fonctionnalités du port GPIO des modèles utilisés (1B+ et 2B) :

GPIO#	2nd func.	Pin#	Pin#	2nd func.	GPIO#
	+3.3 V	1	2	+5 V	
2	SDA1 (I2C)	3	4	+5 V	
3	SCL1 (I2C)	5	6	GND	
4	GCLK	7	8	TXD0 (UART)	14
	GND	9	10	RXD0 (UART)	15
17	GEN0	11	12	GEN1	18
27	GEN2	13	14	GND	
22	GEN3	15	16	GEN4	23
	+3.3 V	17	18	GEN5	24
10	MOSI (SPI)	19	20	GND	
9	MISO (SPI)	21	22	GEN6	25
11	SCLK (SPI)	23	24	CE0_N (SPI)	8
	GND	25	26	CE1_N (SPI)	7
(RPi 1 Models A and B stop here)					
EEPROM	ID_SD	27	28	ID_SC	EEPROM
5	N/A	29	30	GND	
6	N/A	31	32		12
13	N/A	33	34	GND	
19	N/A	35	36	N/A	16
26	N/A	37	38	Digital IN	20
	GND	39	40	Digital OUT	21

FIGURE A.2 – Fonctionnalités du port GPIO des modèles 1B+ et 2B [70]

Annexe B

La norme ZigBee et la configuration des modules XBee

B.1 Le ZigBee et les modules XBee

Le ZigBee est un protocole de communication qui s'appuie sur la norme IEEE 802.15.4 et est défini par le groupe ZigBee Alliance. Le XBee est une marque, un produit qui utilise le protocole ZigBee. Les XBee sont des modules de communication sans fil très populaires, fabriqués par l'entreprise Digi International [71][72]. Ils ont été certifiés par la communauté industrielle ZigBee Alliance. La certification Zigbee se base sur le standard IEEE 802.15.4 qui définit les fonctionnalités et spécifications des réseaux sans fil à dimension personnelle (Wireless Personal Area Networks : WPANs). Les principales caractéristiques du XBee sont :

- Fréquence porteuse : 2.4 Ghz
- Portées variées : 10 à 100m pour les XBee 1 et 2, 1500m pour le XBee-Pro (Figure B.1)
- Débit : 250kbps
- Faible consommation : 3.3V @ 50mA
- Entrées/sorties : 6 x 10-bit ADC input pins, 8 digital IO pins
- Sécurité : communication fiable avec des clés de chiffrement de 128-bits
- Faible coût : environ 25 euro
- Simplicité d'utilisation : communication via le port série
- Ensemble de commandes AT et API
- Flexibilité du réseau : sa capacité à faire face à un nœud hors service ou à intégrer de nouveaux nœuds rapidement
- Grand nombre de nœuds dans le réseau : 65000
- Topologies de réseaux variées : maillé, point à point, point à multipoint, en arbre

FIGURE B.1 – Module XBeePro

B.2 Les rôles possibles des modules XBee

- Le coordinateur : Ce module gère les fonctions de haut niveau du réseau comme l'authentification, la sécurité, etc. Un seul coordinateur doit être présent pour le même réseau (même identifiant de réseau PAN). Il est obligatoire pour la mise en place du réseau. Ce dernier se doit d'être présent durant toute la durée de vie du réseau. Il doit donc être alimenté en permanence.

- Les routeurs : Ces modules disposent de toutes les fonctions d'un module End-device (dispositif terminal du réseau) avec en plus des fonctions de haut niveau utiles pour étendre le réseau. Ils sont disposés généralement à des points clef de maillage. Ils permettent d'étendre la taille du réseau en permettant aux autres modules de s'enregistrer auprès d'eux et non exclusivement auprès du coordinateur. Cela évite de saturer le coordinateur en nombre de modules inscrits. La taille maximum d'un réseau peut ainsi atteindre environ 65000 modules. Ils permettent aussi d'étendre la portée du réseau. Chaque module routeur répète les signaux reçus aux autres modules qui lui sont enregistrés. Le signal se répercute ainsi de module en module pour atteindre le End-device concerné. Mais ces fonctions augmentent fortement la consommation de ces modules. De plus, en cas d'indisponibilité d'un routeur, tous les modules enregistrés auprès de lui deviennent invisibles. Il est donc conseillé de choisir une alimentation continue. La plupart du temps la fonction routeur est dédié aux module sur prise de courant (actionneur de lampe, chauffage, etc.).

- Les End-devices : Ce sont tous les modules terminaux comme les capteurs, actionneurs, etc. Ils ne sont actifs que sur changement de leurs états ou sur réponse à une trame, leur consommation est donc très faible et ils peuvent être alimentés par des piles ou des batteries.

B.3 Les modes de fonctionnement des XBee

Le XBee supporte trois modes de fonctionnement : Transparent, Command (ou AT) et API. Le mode Transparent est le mode sélectionné par défaut à la mise en marche du module. Dans ce mode, le module émet par son antenne toutes les données qu'il reçoit par l'UART. Le mode Command permet de configurer le module : ses entrées, ses sorties, son adresse, l'adresse de destination de ses messages, etc.

Le mode API (Application Programming Interface) est utilisé pour communiquer grâce à des messages (trames) ZigBee. Dans le mode API, les messages échangés ont une structure particulière (Figure B.2) :

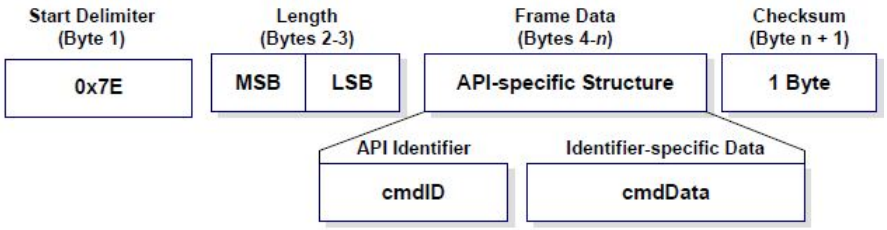

FIGURE B.2 – Structure d'une trame ZigBee (API) [72]

Il existe deux modes API, API 1 et API 2. Dans le premier, les caractères spéciaux ne sont pas échappés contrairement au second. Les caractères échappés sont : 0x7E (Frame Delimiter), 0x7D(Escape), 0x11 (XON), 0x13 (XOFF).

Dans une trame API, le champ *cmdID* indique quel message API est contenu dans *cmdData*. Les types de trames API les plus courants sont :

- AT Command 0x08 : Positionner ou lire un paramètre du module local.
- AT Command Response 0x88 : Réponse à la dernière commande AT locale.
- Remote Command Request 0x17 : Positionner ou lire un paramètre du module distant.
- Remote Command Response 0x97 : Réponse à la dernière commande AT distante.
- Transmit Request 0x10 : Envoyer des données dans une trame API.
- Transmit Status 0x8B : Indique le résultat de la transmission d'une trame de données.
- Receive Packet (AO=0) 0x90 : Trame reçue par le module local.

B.4 L'adressage

La Figure B.3 montre les différents types d'adressage impliqués dans la norme ZigBee.

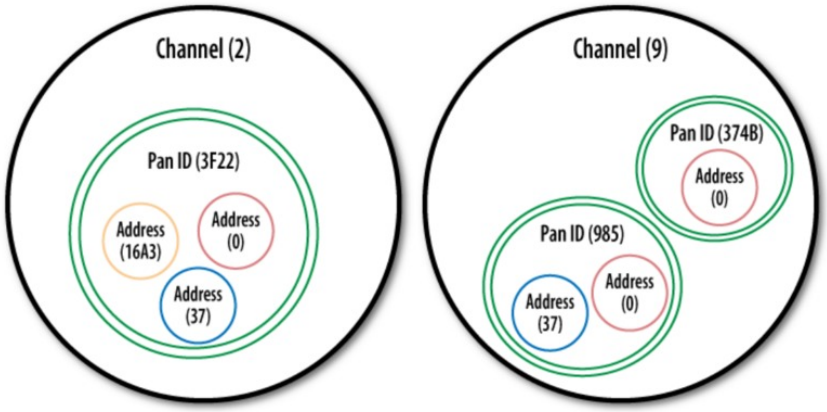

FIGURE B.3 – Adressages ZigBee [73]

La valeur du Channel identifie un canal parmi les 16 existants sur la bande de fréquences de 2.4 GHz, et sur lequel s'effectue la communication. Les modules XBee supportent les 16 canaux et les XBee-Pro ne

supportent que 14 canaux parmi les 16.

Les réseaux ZigBee sont appelés PANs (Personal Area Networks). Chaque réseau est identifié par un PAN ID unique et commun à tous ses nœuds. Les modules sont soit configurés pour rejoindre un PAN ID particulier, soit ils effectuent une découverte des réseaux proches et choisissent un réseau (un PAN ID) à rejoindre. Les PAN ID peuvent être sur 64-bit ou 16-bit et identifient de manière univoque le réseau. Les nœuds d'un même réseau doivent avoir le même PAN ID 64-bit et 16-bit. Le PAN ID 64-bit est utilisé pour résoudre les conflits d'identification des réseaux causés par les PAN ID 16-bit. Si un module est pré-configuré pour rejoindre un PAN ID particulier, il ne pourra rejoindre que le réseau ayant ce PAN ID. Le PAN ID correspond à un adressage MAC.

Les modules XBee supportent des adresses sur 16-bit ou 64-bit. L'adressage 64-bit dote chaque module d'une adresse constructeur unique, alors que l'adresse 16-bit peut changer et n'est pas définitive. Lorsqu'un module rejoint un réseau, une adresse 16-bit lui est affectée. Cette adresse est générée aléatoirement par le routeur ou le coordinateur qui accepte le module. L'adresse '0x0000' est réservée au coordinateur du réseau. Pour qu'un module utilise son adresse unique 64-bit dans ses échanges, il faut mettre l'adresse 16-bit (paramètre MY) à '0xFFFF' ou '0xFFFE'. Les adresses des modules correspondent à un adressage réseau.

B.5 Configuration des modules XBee

Pour configurer un module XBee, il faut le relier à l'UART de l'ordinateur. La communication entre l'ordinateur et le module XBee se fait via une liaison série grâce à un adaptateur USB. Pour configurer correctement nos XBee, nous avons utilisé l'outil XCTU [74] de Digi International.
La Figure B.4 montre la configuration de nos deux modules XBee-Pro.

	Rôle	**Adresse module**	**PAN ID**	**Adresse destination**	**Entrées/Sorties**
XBee 1 (borne)	Coordinateur	0x0000	64-bit (unique)	'Dynamique'	AD0 mode Analog Input
XBee 2 (véhicule)	End-device	64-bit (unique)	'Dynamique'	0x0000	Aucune

FIGURE B.4 – Configuration des modules XBee du véhicule et de la borne

B.6 Montage des modules XBee avec les Raspberry Pi

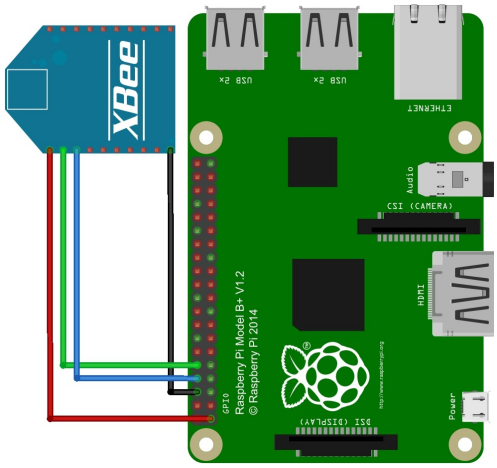

FIGURE B.5 – Schéma de montage XBeePro-RPi (le même pour le véhicule et la borne)

Annexe C

Le capteur à ultrasons HC-SR04

C.1 Présentation et utilisation

Les capteurs à ultrasons (Figure C.1) sont conçus pour détecter la proximité des objets en utilisant les ultrasons. Le principe consiste à calculer à quelle distance se trouve l'obstacle à partir du temps mis par l'onde sonore pour faire le trajet aller et retour capteur-obstacle-capteur. Les ultrasons sont utilisés, car ils sont inaudibles à l'oreille humaine et sont relativement précis sur de courtes distances.

Les capteurs sont constitués d'un émetteur à ultrasons, d'un récepteur et d'un circuit de contrôle. Le transmetteur émet un signal ultrason à haute fréquence qui sera réfléchi par tout objet solide se trouvant à proximité. Une partie du signal réfléchi est détectée par le récepteur situé sur le capteur. Le signal est alors traité par un circuit de contrôle qui va permettre de calculer la durée entre son émission et sa réception. Cette durée est utilisée, pour calculer la distance entre le capteur à ultrasons et l'objet détecté. Le module à ultrasons HC-SR04 [75][76], utilisé pour notre prototype possède quatre broches :

- Une broche de sortie (Echo), utilisée pour informer de la fin de l'émission du train d'ultrasons et de son retour après réflexion sur l'obstacle.
- Une broche d'entrée (Trigger), utilisée pour déclencher l'émission du train d'ultrasons.
- Une broche (Vcc), utilisée pour alimenter le capteur (5V).
- Une broche (Gnd).

FIGURE C.1 – Le capteur à ultrasons HC-SR04 [76]

Pour son utilisation, une impulsion d'une durée de $10\mu s$ envoyée sur la broche Trigger déclenche l'émission d'un train d'ultrasons. La broche Echo bascule alors à l'état haut (5V) indiquant la fin de l'émission des

ultrasons. La broche Echo bascule à l'état bas (0V) lorsque l'onde réfléchie par l'obstacle est détectée par le récepteur du module (Figure C.2).

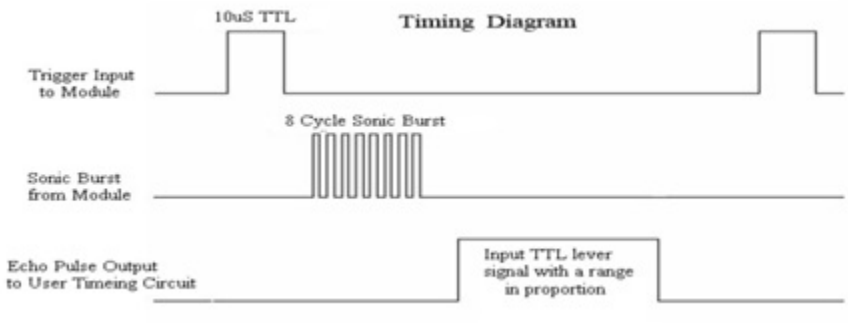

FIGURE C.2 – Diagramme des signaux impliqués dans le fonctionnement du capteur [75]

C.2 Utilisation dans notre parking

Pour chaque capteur, le programme principal de la borne doit générer un signal d'une durée de $10\mu s$ sur la broche correspondant au Trigger, puis mesurer la durée du signal Echo à l'état haut et calculer la distance avec l'obstacle. Si la distance est inférieure à 30cm, alors il en déduit qu'un véhicule occupe cette place, sinon il en déduit que la place est libre. En Python, l'impulsion de $10\mu s$ du Trigger est générée comme suit :

```
GPIO.output(TRIG, False)        #initialiser le tigger à 0
time.sleep(1)   #attendre que le capteur soit prêt
GPIO.output(TRIG, True) #positionner le trigger à 1
time.sleep(0.00001)     #attendre 10us
GPIO.output(TRIG, False)        #remettre le trigger à 0
```

Après ça, la broche Echo du capteur restera à un potentiel haut (5V) pendant tout le temps que prend le train d'ultrasons pour aller et venir entre le capteur et l'objet. On doit donc mesurer la durée pendant laquelle la broche Echo reste à un potentiel haut.
La première étape consiste à détecter l'instant du front montant du signal Echo (changement d'état de l'état bas à l'état haut). Cet instant sera celui de la fin de l'émission du train d'ultrasons par le capteur :

```
while GPIO.input(ECHO) == 0:
                pulse_start = time.time()
```

Une fois le train d'ultrasons émis, la broche Echo restera à l'état haut jusqu'au retour des ultrasons réfléchis par l'obstacle. On doit alors détecter le front descendant du signal Echo (basculement à l'état bas). Cet instant sera celui de la détection du retour des ultrasons :

```
while GPIO.input(ECHO) == 1:
                pulse_end = time.time()
```

La durée de l'impulsion est la différence entre les deux instants enregistrés :

```
pulse_duration = pulse_end - pulse_start
```

Connaissant la durée que le signal ultrason met pour parcourir la distance émetteur-obstacle-récepteur, nous pouvons calculer cette distance :

$$Vitesse = \frac{Distance}{Temps}$$

La vitesse du son dans l'air et au niveau de la mer est de 343 m/s (34300 cm/s), nous allons donc utiliser cette valeur comme référence. Nous devons également diviser la durée par deux, car la durée calculée est celle

que le signal ultrason met pour parcourir la distance émetteur-obstacle-récepteur, soit deux fois la distance recherchée.
Nous pouvons donc simplifier le calcul de la façon suivante :

$$34300 = \frac{Distance}{\frac{Temps}{2}}$$

Donc,

$$Distance = Temps \times 17150$$

Remarque
Nous avons tenté de gérer ces capteurs avec des mécanismes d'interruptions, mais après les tests, nous n'avons pas retenu cette solution pour les raisons suivantes :

1. Les capteurs à ultrasons ne déclenchent pas d'interruptions instantanément, on est donc obligé de générer pour chaque capteur le signal Trigger, puis de mesurer la longueur du signal Echo pour connaître la distance. Une boucle est donc nécessaire même en utilisant les interruptions.

2. Comme la mesure du signal Echo est liée à l'envoie de paquets NDN de mises à jour, sa gestion par interruption a généré des communications imprévisibles rendant le contrôle sur les mises à jour moins fiable qu'avec une gestion en attente active.

3. Le temps d'attente entre l'envoie du signal Trigger et le calcul du signal Echo est très court et ne ralentit pas beaucoup le traitement de l'application.

C.3 Montage des capteurs avec le Raspberry Pi

La tension délivrée par la broche Echo du module HC-SR04 est de 5V. Or, la broche d'entrée du Raspsberry Pi est conçue pour une tension de 3.3V au maximum. Un pont diviseur de tension est donc nécessaire. Il est constitué de deux résistances (R1 et R2) montées en série et connectées à une tension d'entrée (V_in), la tension de sortie (V_out) sera abaissée en fonction de la valeur des résistances utilisées. Dans notre circuit, la tension d'entrée (V_in) sera la tension de 5V générée par la broche Echo de notre capteur et devra être abaissée à une tension de 3.3V (V_out). Le schéma de la Figure C.3 ainsi que les équations qui suivent sont utilisés dans de nombreuses applications où une réduction de tension est nécessaire.

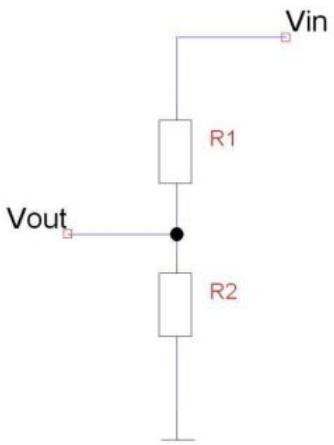

FIGURE C.3 – Schéma du diviseur de tension [76]

$$V_{out} = V_{in} \times \frac{R2}{(R1 + R2)}$$
$$\Longrightarrow \frac{V_{out}}{V_{in}} = \frac{R2}{(R1 + R2)}$$

Avec une résistance de 100Ω pour R1. Nous obtenons le résultat suivant : $\frac{3.3}{5} = \frac{R2}{(100+R2)}$
Soit, $R2 = 200\Omega$. Donc une résistance de 100Ω pour R1 et une résistance de 200Ω pour R2.

Nous utilisons quatre broches du Rapsberry Pi pour chaque capteur à ultrasons. Par exemple :

- La broche GPIO 5V (pin 2) sera reliée à la broche Vcc du capteur pour l'alimentation en 5V.
- La broche GPIO GND (pin 6) sera reliée à la broche Gnd du capteur pour la mise à la masse.
- La broche GPIO 23 (pin 16), broche de sortie du GPIO qui sera reliée à l'entrée Trigger du capteur.
- La broche GPIO 24 (pin 18), broche d'entrée du GPIO qui sera reliée à la sortie Echo du capteur via le pont diviseur de tension.

Dans le câblage réel, les broches d'alimentation (5V et GND) sont communes à tous les capteurs. Chaque capteur ne nécessite donc que deux broches spécifiques (Trigger et Echo). La Figure C.4 représente le circuit de montage d'un des capteurs sur le Raspberry Pi :

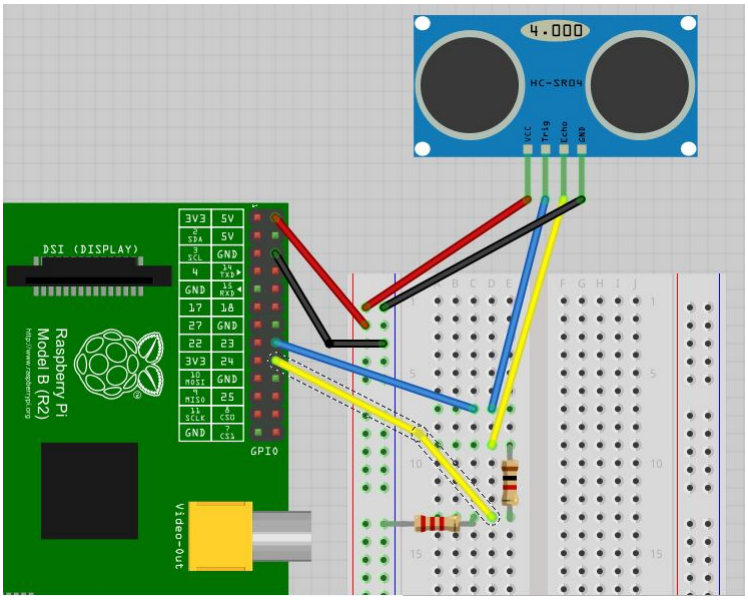

FIGURE C.4 – Schéma de montage HC-SR04-Raspberry Pi [76]

Annexe D

Le capteur de gaz MQ135

D.1 Présentation et utilisation

MQ135 [77] est un capteur de qualité de l'air utilisé pour détecter des gaz tels que NH_4, Benzène, CO, CO_2, etc. Son fonctionnement consiste en une plaque de Dioxyde d'étain (mélangé à d'autres matières et préchauffée par le courant) dont la résistance change au contact de certains gaz. Il existe plusieurs modèles de ce capteur, chacun permettant de détecter un ensemble particulier de gaz.
La Figure D.1 représente le capteur MQ135 et ses broches de connexion.

Pin No.	Symbol	Descriptions
1	DOUT	Digital output
2	AOUT	Analog output
3	GND	Power ground
4	VCC	Positive power supply (2.5V-5.0V)

FIGURE D.1 – Le capteur MQ135 et ses borches [77]

Pour utiliser le capteur, il faut l'alimenter avec une tension de 2.5V à 5V, la tension en sortie dans la broche A_{out} varie en fonction de la réaction qui se produit entre la plaque de Dioxyde d'étain et les gaz. C'est cette tension qui est reliée à la broche d'entrée du microcontrôleur (module XBee, Arduino, etc.).
La Figure D.2 montre les caractéristiques sensitives du capteur par rapport aux gaz qu'il détecte, dans les conditions suivantes : Température 20°C, Humidité 65%, Concentration O_2 : 21%, RL(résistance interne) = $20K\Omega$.

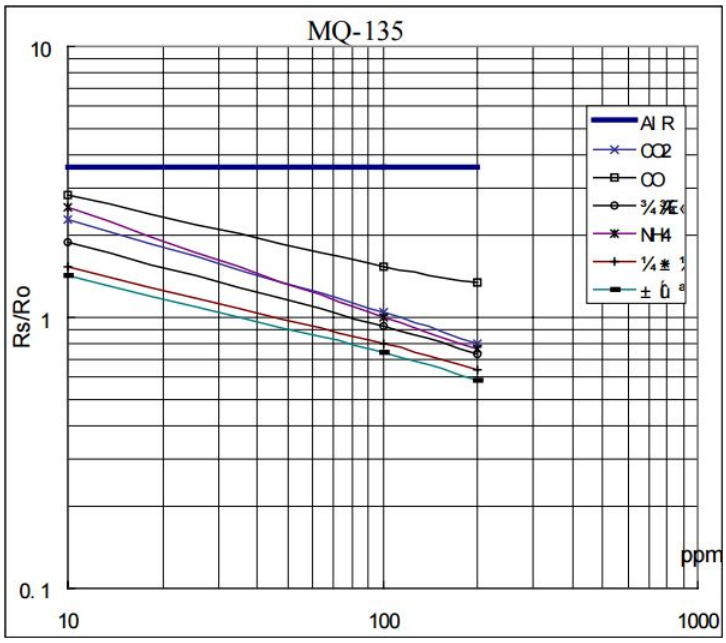

FIGURE D.2 – Capacités sensorielles du capteur MQ135 [77]

Où :

- R_0 : Résistance du capteur dans les conditions initiales (100ppm de NH_4 dans l'air). Cette valeur doit être calibrée pour chaque capteur.
- R_s : Résistance du capteur, variable selon les conditions. Cette valeur est calculée à partir de la tension donnée par A_{out}.

Cette figure est utilisée pour convertir la résistance de sortie (R_s) en une grandeur physique (en ppm, partie par million) qui représente la concertation du gaz étudié.

D.2 Calcul de l'indice de CO2

Les courbes de la figure précédente sont des fonctions de type : $y = a \times x^b$. Donc :

$$ppm = a \times (\frac{R_s}{R_0})^b \tag{D.1}$$

Pour le CO_2, le calcul des paramètres a et b donne la fonction de la courbe :

$$ppm = 116.602 \times (\frac{R_s}{R_0})^{-2.769}$$

À la lecture, la valeur analogique A_{out} est convertie par le convertisseur A/N du module XBee en un entier *Val* tel que $0 \leq Val \leq 1023$ (convertisseur A/N de 10-bit). R_s se calcul alors à partir de *Val* avec la formule suivante :

$$((\frac{1023}{Val}) \times VREF - 1) \times RL$$

Où :

- RL : Résistance interne du capteur, $20K\Omega$

- VREF = Tension de référence du convertisseur A/N (2V d'après le manuel XBee)

En ayant R_s et les paramètres a et b, il reste à définir la valeur de R_0 (calibration). La calibration est importante car chaque capteur est quelque peu différent des autres. Donc, pour définir R_0 pour un certain gaz, il faut connaitre la concentration de ce gaz dans des conditions normales, puis lire A_{out} et déduire la valeur de la résistance de sortie (R_s), et calculer R_0 avec la formule suivante (tirée de la formule D.1) :

$$R_0 = R_s \times \sqrt[b]{(\frac{ppm}{a})}$$

On sait que dans des conditions normales, la concertation de CO_2 dans l'atmosphère est de 392ppm. Il suffit donc de préchauffer le capteur pendant 24 heures (la chaleur accélère la réaction chimique et augment la précision), le laisser à l'air libre, et mesurer la résistance de sortie R_s. Si par exemple $R_s = 26954\Omega$, alors $R_0 = 41763\Omega$.

D.3 Montage du capteur avec le module XBee

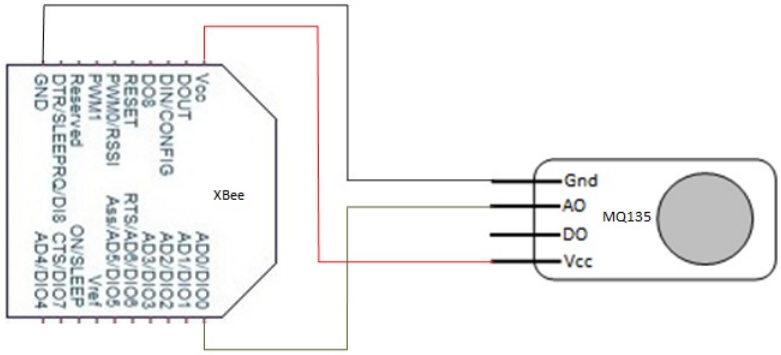

FIGURE D.3 – Schéma de montage du capteur au module XBeePro

Bibliographie

[1] Cisco System Inc. Cisco visual networking index : Global mobile data traffic forecast update. Technical report, Cisco, February 2015.

[2] SmartSantander. Smartsantander. [Online] ; http://www.smartsantander.eu/.

[3] Wikipedia. Project xanadu. [Online] ; https://en.wikipedia.org/wiki/Project_Xanadu.

[4] Brent Baccala. Data-oriented networking, 2002. [Online] ; https://tools.ietf.org/html/draft-baccala-data-networking-00.

[5] Ali Ghodsi, Teemu Koponen, Barath Raghavan, Scott Shenker, Ankit Singla, and James Wilcox. Information-centric networking : Seeing the forest for the trees. Technical report, ACM, November 2011.

[6] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM*, 2007.

[7] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Börje Ohlman. A survey of information-centric networking. IEEE communications magazine, IEEE, July 2012.

[8] PARC. The CCNx Project. [Online] ; http://blogs.parc.com/ccnx/.

[9] Fabian Oehlmann. Content-centric networking. Technical report, Technische Universität München, February 2013.

[10] Marc Mosko, Ignacio Solis, Ersin Uzun, and Christopher Wood. CCNx 1.0 protocol architecture. Technical report, PARC, 2015.

[11] NDN Project Team. NDN project overview. [Online] ; http://named-data.net/project/.

[12] FIA. NSF future internet architecture project. [Online] ; http://www.nets-fia.net/.

[13] Deborah Estrin Lixia Zhang and Jeffrey Burke. Named data networking NDN project. Technical Report NDN-0001, NDN, October 2010.

[14] Van Jacobson, Jeffrey Burke, and Lixia Zhang. Named data networking next phase NDN-NP project - annual report. Technical report, NDN, April 2015.

[15] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. Named data networking. *ACM*, 2014.

[16] Wentao Shang, Adeola Bannisy, Teng Liangz, Zhehao Wangx, Yingdi Yu, Alexander Afanasyev, Jeff Thompsonx, Jeff Burkex, Beichuan Zhangz, and Lixia Zhang. Named data networking of things (invited paper). 2016.

[17] NDN Project Team. NDN specification documentation. Technical Report 0.1a2, NDN, March 2014.

[18] Alexander Afanasyev, Junxiao Shi, Lan Wang, Beichuan Zhang, and Lixia Zhang. Packet fragmentation in NDN : Why NDN uses hop-by-hop fragmentation. Technical report, NDN, 2015.

[19] NDN Project Team. NDN technical memo : Naming conventions. Technical report, NDN, July 2014.

[20] Yu Zhang, Alexander Afanasyev, Jeff Burke, and Lixia Zhang. A survey of mobility support in named data networking. *IEEE*, 2016.

[21] A K M Mahmudul Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. NLSR : Named-data link state routing protocol. Technical report, NDN, August 2013.

[22] Alexander Afanasyev, Cheng Yi, Lan Wang, Beichuan Zhang, and Lixia Zhang. SNAMP : Secure namespace mapping to scale NDN forwarding. *IEEE*, 2015.

[23] Yingdi Yu. Public key management in named data networking. Technical report, NDN, 2015.

[24] Yingdi Yu, Alexander Afanasyev, David Clark, kc claffy, Van Jacobson, and Lixia Zhang. Schematizing trust in named data networking. *ACM*, 2015.

[25] Jeff Thompson and Jeff Burke. NDN common client libraries. Technical report, NDN, September 2014.

[26] NDN Project Team. NDN common client libraries API. [Online] ; `http://named-data.net/doc/ndn-ccl-api/#`.

[27] Beichuan Zhang Alexander Afanasyev, Junxiao Shi and NFD Team. NFD developer's guide. Technical Report NDN-0021, NDN, February 2015.

[28] NDN Project Team. NFD overview. [Online] ; `http://named-data.net/doc/NFD/current/overview.html`.

[29] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM : NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

[30] ns 3 project. ns-3tutorial. Technical report, ns-3 project, March 2016.

[31] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2.0 : A new version of the NDN simulator for NS-3. Technical Report NDN-0028, NDN, January 2015.

[32] Alexander Afanasyev, Spyridon Mastorakis, Ilya Moiseenko, and Lixia Zhang. ndnSIM documentation. [Online] ; `http://ndnsim.net/2.1/getting-started.html`.

[33] Gitub. MiniNDN. [Online] ; `https://github.com/named-data/mini-ndn`.

[34] NDN Project Team. Libraries/NDN Platform. [Online] ; `http://named-data.net/codebase/platform/`.

[35] Wikipedia. Internet of things. [Online] ; `https://en.wikipedia.org/wiki/Internet_of_things`.

[36] IEEE Internet Initiative. Towards a definition of the internet of things (IoT). Technical report, IEEE, 2015.

[37] Chris Greer. The internet's next big idea : Connecting people, information, and things, 2014. [Online] ; `http://www.nist.gov/el/20140611_internets_next_big_idea.cfm`.

[38] W3C. Web of things at W3C. [Online] ; https://www.w3.org/WoT/.

[39] Casagras Project. RFID and the inclusive model for the internet of things. Technical report, Casagras Project.

[40] IoT-A Project. IoT-A project. [Online] ; http://www.iot-a.eu/public.

[41] Alessandro Bassi, Martin Bauer, Martin Fiedler, Thorsten Kramp, Rob van Kranenburg, Sebastian Lange, and Stefan Meissner. *Enabling Things to Talk*. Springer, 2013.

[42] IoT-A Project. Initial architectural reference model for IoT. Technical report, IoT-A, 2011.

[43] Dave Evans. L'internet des objets. Technical report, Cisco Internet Business Solutions Group, April 2011.

[44] L. Grieco Y. Zhang, D. Raychadhuri and ICN Research Group. ICN based architecture for iot - requirements and challenges. Technical report, ICN Research Group, December 2014.

[45] Marica Amadeo, Claudia Campolo, Antonio Iera, and Antonella Molinaro. Named data networking for iot : an architectural perspective. *IEEE*, 2014.

[46] Adeola Bannis and Jeff Burke. Creating a secure, integrated home network of things with named data networking. Technical report, NDN, November 2015.

[47] Ravindran R., Biswas T., Xinwen Zhang, Chakraborti A., and Guoqiang Wang. Information-centric networking based homenet. *IFIP/IEEE*, 2013.

[48] Wentao Shang, Qiuhan Ding, Alessandro Marianantoni, Jeff Burke, and Lixia Zhang. Securing building management systems using named data networking. *IEEE*, 2014.

[49] Wentao Shang, Yingdi Yu, Teng Liangy, Beichuan Zhangy, and Lixia Zhang. NDN-ACE : Access control for constrained environments over named data networking. Technical report, NDN, December 2015.

[50] Divya Saxena, Vaskar Raychoudhury, and Nalluri SriMahathi. SmartHealth-NDNoT : Named data network of things for healthcare services. *ACM*, 2015.

[51] Lucas Wang, Ryuji Wakikawa, Romain Kuntz, Rama Vuyyuru, and Lixia Zhang. Data naming in vehicle-to-vehicle communications. *IEEE INFOCOM*, 2012.

[52] Wentao Shang, Yingdi Yu, Ralph Droms, and Lixia Zhang. Challenges in IoT networking via TCP/IP architecture. Technical report, NDN, February 2016.

[53] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Internet of things via named data networking : The support of push traffic. *IEEE*, 2014.

[54] Jiangzhe Wang, Ryuji Wakikawa, and Lixia Zhang. DMND : Collecting data from mobiles using named data. *IEEE Vehicular Networking Conference*, 2010.

[55] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Multi-source data retrieval in iot via named data networking. *ACM*, 2014.

[56] Lucas Wang, Alexander Afanasyev, Romain Kuntz, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. Rapid traffic information dissemination using named data. *ACM*, 2012.

[57] Antonio Carzaniga, Michele Papalini, and Alexander L. Wolf. Content-based publish/subscribe networking and information-centric networking. *ACM*, 2011.

[58] NEDAP identification systems. Le stationnement intelligent. Technical report, NEDAP.

[59] SFMTA. SFpark technical manual, 2014. [Online]; http://sfpark.org/wp-content/uploads/2014/07/SFpark_Tech_Manual_web1.pdf.

[60] Banner Engineering Corp. Wireless solutions for parking. [Online]; http://info.bannerengineering.com/cs/groups/public/documents/literature/179699.pdf.

[61] Marc Zaffagni. Une appli pour trouver une place de parking libre, 2015. [Online]; http://www.futura-sciences.com/magazines/high-tech/infos/actu/d/voiture-appli-trouver-place-parking-libre-59952/.

[62] Benoît Solivellas. Ce capteur pourrait révolutionner le stationnement en ville, 2012. [Online]; http://www.cnetfrance.fr/cartech/ce-capteur-pourrait-revolutionner-le-stationnement-en-ville-39785210.htm.

[63] Siemens AG. Intelligent "parking" solutions. Technical report, Siemens, 2011.

[64] Tinynode SA. A4 - S / M / L fiche produit. Technical report, Tinynode SA, 2013.

[65] SFMTA. SFpark pilot project evaluation, 2014. [Online]; http://direct.sfpark.org/wp-content/uploads/eval/SFpark_Pilot_Project_Evaluation.pdf.

[66] ZigBee Alliance. ZigBee PRO with green power. [Online]; http://www.zigbee.org/zigbee-for-developers/network-specifications/zigbeepro/.

[67] Github. NDN client library with TLV wire format support in native Python. [Online]; https://github.com/named-data/PyNDN2.

[68] Github. Python tools for working with XBee radios. [Online]; https://github.com/nioinnovation/python-xbee.

[69] Github. Python serial port access library. [Online]; https://github.com/pyserial/pyserial.

[70] Wikipedia. Raspberry Pi. [Online]; https://en.wikipedia.org/wiki/Raspberry_Pi.

[71] Digi International Inc. Digi products. [Online]; http://www.digi.com/products.

[72] Digi International Inc. XBee/XBee-PRO DigiMesh 2.4 OEM RF modules. Technical report, Digi International, 2008.

[73] Jérôme Abel. XBee et Arduino. [Online]; http://faitmain.org/volume-2/xbee-arduino.html.

[74] Digi International Inc. XCTU. [Online]; http://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu.

[75] Elec Freaks. Ultrasonic ranging module HC-SR04. Technical report, Elec Freaks.

[76] Jacob Marsh (developpez.com). Utiliser des capteurs avec le port GPIO-partie 3 : Télémètre à ultrasons. [Online]; http://raspberry-pi.developpez.com/cours-tutoriels/capteur/mag-pi-utiliser-port-gpio/partie-3-telemetre-ultrason/.

[77] Waveshare. MQ-135 gas sensor. [Online]; http://www.waveshare.com/wiki/MQ-135_Gas_Sensor.

Table des figures

Table des matières