Gestion autonome de la qualité de service et de la sécurité dans un environnement Internet des objets

Ahmad Khalil

THESE DE DOCTORAT DE L'ETABLISSEMENT UNIVERSITE BOURGOGNE FRANCHE-COMTE

PREPAREE A L'UNIVERSITE DE BOURGOGNE

Ecole doctorale n°37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat en Informatique

Par

Ahmad KHALIL

Gestion autonome de la qualité de service et de la sécurité dans un environnement Internet des objets

Thèse présentée et soutenue à *Dijon*, le *04/12/2019*

Composition du Jury :

Mme. Francine KRIEF	Professeur, ENSEIRB-MATMECA – Bordeaux INP	Présidente (Rapporteur)
M. Hacène FOUCHAL	Professeur, Université de Reims Champagne-Ardenne	Rapporteur
M. Benoît DARTIES	Maître de conférences, Université de Montpellier	Examinateur
M. Olivier TOGNI	Professeur, Université de Bourgogne	Co-Directeur de thèse
M. Nader MBAREK	Maître de conférences HDR, Université de Bourgogne	Co-Directeur de thèse

UNION EUROPEENNE
RÉGION BOURGOGNE FRANCHE COMTÉ
avec le Fonds européen de développement régional (FEDER)

Remerciements

Cette thèse n'aurait pas pu aboutir sans le soutien et les encouragements continus de tous ceux qui m'ont accompagné pendant ces trois années de recherche. Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à terme.

D'abord, je tiens à remercier le conseil régional de Bourgogne-Franche-Comté d'avoir financé ce travail à travers le dispositif Jeunes Chercheurs Entrepreneurs (JCE) et spécialement la convention FEDER **#2016-9205AAO033S03069**.

Ensuite, Je tiens à remercier mon directeur de thèse M. ***Olivier Togni***, Professeur à l'université de Bourgogne, pour m'avoir assuré toutes les conditions nécessaires et suffisantes et tous les encouragements permettant de bien réussir mes missions. Je remercie grandement mon co-directeur de thèse M**.** ***Nader Mbarek***, Maître de conférences HDR à l'ESIREM – Université de Bourgogne, pour toute son aide. Je suis ravi d'avoir travaillé en sa compagnie car outre son appui scientifique, les échanges très enrichissants que j'ai pu avoir avec lui et ses conseils, il a toujours été là pour me soutenir. Je tiens à le remercier aussi pour sa disponibilité permanente, pour les nombreux encouragements qu'il m'a prodigués et pour son sens de critique qui m'a permis d'élargir ma vision et d'améliorer ma recherche.

Egalement, j'exprime ma gratitude à Mme. ***Francine Krief***, Professeur à l'ENSEIRB-MATMECA – Bordeaux INP, et M. ***Hacène Fouchal***, Professeur à l'Université de Reims Champagne-Ardenne, d'avoir accepté la lourde charge d'être rapporteurs de cette thèse et d'évaluer ce travail. Je remercie aussi M. ***Benoît Darties*** pour avoir accepté d'être examinateur de cette thèse.

Je tiens aussi à remercier tous les amis que j'ai connu à Dijon (***Alissar***, ***Georges***, ***Karam,*** ***Maria***) et tous les collègues du laboratoire LIB (***Carine***, ***Hiba***, ***Anabelle***, ***Rose***, ***Frantz***, ***Armen***, ***Hajer***) avec lesquels j'ai partagé des moments sympathiques et amicaux. J'exprime ma grande gratitude et ma reconnaissance à mes parents (***Ali*** et ***Amné***), mes sœurs (***Ghenwa***, ***Elissa*** et ***Assil***), mon frère (***Raji***) et ma meilleur amie d'enfance (***Sarelle***) pour leurs encouragements, patience, sacrifices, soutien et pour avoir cru en moi.

Résumé

De nos jours, les avancées technologiques font que l'Internet des Objets (IoT) est en train de s'imposer dans notre vie quotidienne afin d'en améliorer la qualité grâce à entre autres l'automatisation de certaines tâches. Un défi majeur dans le déploiement massif des applications et services IoT ainsi que leur utilisation dans différents domaines est l'amélioration du niveau de service correspondant. Ce niveau de service peut être caractérisé selon deux axes importants : la Qualité de Service (*QoS*) et la sécurité. De plus, ce niveau de service doit être géré d'une manière autonome au sein de l'IoT vu l'hétérogénéité et la taille des réseaux qui forment cet environnement rendant difficile, même impossible, leurs gestions d'une façon manuelle par les administrateurs. Dans le cadre de cette thèse, nous proposons un mécanisme de *QoS*, appelé *QBAIoT* (QoS Based Access for IoT environments) permettant d'assurer un traitement différencié des trafics existants dans l'environnement IoT afin de respecter les exigences de chacun des trafics selon différents paramètres de *QoS* (i.e., délai, gigue, taux de livraison de paquets, etc.). *QBAIoT* est ensuite amélioré afin d'être géré d'une manière autonome à travers deux fonctions importantes : l'auto-configuration et l'auto-optimisation. De plus, afin d'assurer la *QoS* au sein de l'environnement IoT, il faut optimiser la consommation énergétique des composants contraints en termes de ressources. Ainsi, nous proposons une adaptation de *QBAIoT* permettant de réduire sa consommation énergétique d'une manière autonome tout en respectant la précision des données remontées par les objets IoT. Notre contribution concernant le deuxième axe du niveau de service dans un environnement IoT, à savoir la sécurité, se traduit par un mécanisme permettant de contrôler l'accès des objets aux passerelles IoT. Nous appelons cette méthode de contrôle d'accès *IoT-MAAC* (IoT Multiple Attribute Access Control) car elle prend en compte différents paramètres spécifiques à l'environnement IoT (i.e., confiance des objets, identificateur de l'objet, empreinte digitale de l'objet, etc.). La prise de décision concernant le contrôle d'accès des objets IoT est gérée d'une façon autonome par les passerelles IoT et vise à respecter les exigences de cet environnement en termes de confiance.

Mots-clés : Internet des objets, IoT, QoS, QBAIoT, Sécurité, Vie privée, Confiance, IoT-MAAC, Gestion autonome, Auto-configuration, Auto-optimisation, Consommation énergétique

Abstract

Nowadays, the Internet of Things (IoT) is becoming important in our daily lives thanks to technological advances. This paradigm aims to improve the quality of human life through automating several tasks. In this context, service level guarantee within IoT environments is a major challenge while considering a massive deployment of IoT applications and services as well as extending their usage to different domains. The IoT service level can be characterized in two parts: Quality of Service (*QoS*) and security. Moreover, this service level must be managed in an autonomic manner within the IoT environment given the heterogeneity and the size of its infrastructure making it difficult, even impossible, their management in a manual manner by the administrators. In this thesis, we propose a *QoS* based channel access control mechanism, called *QBAIoT* (QoS Based Access for IoT environments), to ensure a differentiated processing of existing traffics in the IoT environment. The differentiated processing allows satisfying the requirements of each traffic according to different *QoS* parameters (i.e., delay, jitter, packet delivery ratio, etc.). Then, *QBAIoT* is improved and upgraded to integrate self-management capabilities thanks to two important functions of the closed control loop: self-configuration and self-optimization. In addition, to offer a better *QoS* within the IoT environment, it is necessary to optimize the energy consumption of resources' constrained components. Thus, we propose an adaptation of *QBAIoT* allowing to reduce its energy consumption in an autonomic manner while respecting the data accuracy. Our contribution concerning the second part of service level guarantee within an IoT environment, which is security, consists is a mechanism enabling IoT objects access control to IoT gateways, called *IoT-MAAC* (IoT Multiple Attribute Access Control). This mechanism takes into account different parameters that are specific to IoT environments (i.e., IoT object trust, IoT object identifier, IoT object fingerprint, etc.). Finally, the decision making process regarding IoT object access control is autonomously managed by IoT gateways and aims to meet the requirements of IoT environment in terms of trust.

Keywords: Internet of Things, IoT, QoS, QBAIoT, Security, Privacy, Trust, IoT-MAAC, Autonomic Computing, Self-configuration, Self-optimization, Energy consumption

Table des matières

Liste des Abréviations

ABAC	Attribute Based Access Control
ABSD	Adaptive Beacon Order, Superframe Order and Duty cycle
AC	Autonomic Computing
ACPT	Access Control Policy Testing
ACTS	Automated Combinatorial Testing for Software
AE	Authorization Engine
AIOTI	Alliance for IoT Innovation
ALOHA	Abramson's Logic of Hiring Access
AM	Autonomic Manager
API	Application Programing Interface
AWS	Amazon Web Services
BCC	Block Check Character
BE	Back off Exponent
BI	Beacon interval
BLE	Bluetooth Low Energy
BLE	Battery Life Extension
BO	Beacon Order
BP	Backoff Periods
CAP	Contention Access Period
CapBAC	Capability Based Access Control
CCA	Clear Channel Assessment
CFP	Contention Free Period
CH	Cluster head
COA	Center Of Area
COG	Center Of Gravity
cSLA	cloud Service Level Agreement
CSMA/CA	Carrier Sense Multiple Access / Contention Avoidance
CSP	Cloud Service Provider
CW	Contention Window
DDoS	Distributed Denial of Service
DGI	Data Generation Interval
DoS	Denial of Service
DSME	Deterministic and Synchronous Multi-Channel Extension
ECC	Elliptic Curve Cryptosystem
EDR	Effective Data Rate
FSM	Finite State Machine

GE	Gateway Equivalent
GEP	Goal Enforcement Point
gSLA	gateway Service Level Agreement
GTS	Guaranteed Time Slot
GUI	Graphical User Interface
GWD	Gateway Device
HL-Gw	High Level Gateway
HMAC-SHA	Hashing for Message Authentication - Secure Hash Algorithm
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IdP	Identity Provider
IDS	Intrusion Detection System
iSLA	IoT Service Level Agreement
IMA	Integrity Measurement Architecture
IMASC	Integrity Measurement Architecture using a Smart Card
IoT	Internet of Things
IoT-C	IoT Client
IoT-SP	IoT Service Provider
IPSEC	Internet Protocol Security
IT	Intervalle de Temps
IoT-MAAC	IoT Multiple Attribute Access Control
JSON	JavaScript Object Notation
L2TP	Layer 2 Tunneling Protocol
LDAP	Lightweight Directory Access Protocol
LECIM	Low-Energy Critical Infrastructure Monitoring
LIFS	Long Inter-Frame Spacing
LL-Gw	Low Level Gateway
LR-WPAN	Low Rare Wireless Personal Area Network
M2M	Machine to Machine
MAC	Medium Access Control
MAPE-K	Monitor Analyze Plan Execute – Knowledge
MDT	Mean Down Time
ME	Managed Element
MIMO	Multiple Input Multiple Output
MISO	Multiple Input Single Output
MND	Mobile Node Device
MOM	Mean Of Maxima
MSC	Message Sequence Chart
MTTR	Mean Time To Repair
NaaS	Network as a Service

NB	Number of Backoffs
NRT	Non Real Time
nSLA	network Service Level Agreement
NSP	Network Service Provider
OASIS	Organization for Advancement of Structured Information Standards
OPEX	Operational Expenditure
orBAC	Organization Based Access Control
PaaS	Platform as a Service
PAN	Personal Area Network
PAP	Policy Administration Point
PATH	Program for Appropriate Technology in Healt
PCA	Priority Channel Access
PDP	Policy Decision Point
PDR	Packet Delivery Ratio
PEP	Policy Enforcement Point
PIP	Policy Information Point
PKI	Public Key Infrastructure
PPR	Punctual Packets Ratio
PPTP	Point to Point Tunneling Protocol
QBAIoT	QoS Based Access Control for IoT environments
QHP	Quantitative Hierarchy Process
QoE	Quality of Experience
QoS	Quality of Service
QPT	Quantitative Policy Trees
RBAC	Role Based Access Control
RGPD	Règlement Général sur la Protection des Données
RoT	Root of Trust
RTMC	Real Tome Mission Critical
RTNMC	Real Time Non Mission Critical
SaaS	Software as a Service
SAML	Security Assertion Markup Language
SDN	Software Defined Networking
SECcSLA	Security cloud Service Level Agreement
SECgSLA	Security gateway Service Level Agreement
SECiSLA	Security IoT Service Level Agreement
SecSLA	Security Service Level Agreement
SD	Superframe Duration
SED	Self-Encrypting Drive
SIFS	Short Inter-Frame Spacing
SKKE	Symmetric-Key Key Establishment

SHA	Secure Hash Algorithm
SLA	Service Level Agreement
SLO	Service Level Objective
SMV	Symbolic Model Verification
SO	Superframe Order
SOAP	Simple Object Access Protocol
SP	Service Provider
SSO	Single Sign On
SSTP	Secure Socket Tunneling Protocol
TAXII	Trusted Automated eXchange of Indicator Information
TCG	Trusted Computing Group
TPM	Trusted Platform Module
TSCH	Time Slotted Channel Hopping
TSK	Takagi Sugeno Kang
V2I	Vehicle to Infrastructure
V2V	Vehicle to Vehicle
VM	Virtual Machine
WHO	World Health Organisation
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

Liste des Figures

Liste des Tableaux

Chapitre 1 – Introduction générale

1.1. Contexte général

De nos jours, les avancées technologiques deviennent de plus en plus importantes dans nos sociétés. Dans ce contexte, différentes villes, cités et pays dans le monde sont en train de mettre en place des services de plus en plus intelligents. Devenir une cité intelligente où l'on prend conscience de la protection de l'environnement et de la qualité de vie humaine, est désormais l'objectif des gestionnaires des grandes métropoles mais aussi les collectivités dans les pays des différents continents. Cette transition vers un monde intelligent assure le développement durable de nos environnements urbains ainsi qu'une amélioration de la qualité de vie et la sécurité des citoyens tout en assurant le respect de leurs vies privées.

La vision d'un monde intelligent prévoit que l'ensemble de l'espace urbain soit interconnecté afin qu'il puisse interagir avec le monde numérique et par conséquent l'adoption du paradigme de l'Internet des Objets (IoT : Internet of Things). En effet, l'IoT permet d'interconnecter différents objets, machines, capteurs qui peuvent être portés par des êtres humains afin d'offrir une nouvelle gamme de services et d'applications. En interconnectant différents types d'objets grâce à de nouvelles technologies adaptées et en utilisant les capacités offertes par le cloud computing, l'IoT permet de faciliter la vie humaine en automatisant certaines tâches quotidiennes. De même, l'IoT permet d'offrir des services critiques à distance comme les services médicaux en utilisant les capacités du réseau Internet. Ainsi, l'impact de l'IoT sur nos sociétés sera important et permettra l'amélioration de la qualité de vie tout en réduisant les charges de mise en place des services. A titre d'exemple, la prise en charge des services de santé à travers l'IoT permet de réduire les charges qui incombent aux collectivités tout en améliorant le suivi des patients.

L'amélioration des services offerts par un environnement IoT est un défi primordial que nous devons relever pour permettre un déploiement massif de ses services dans différents domaines d'application et faciliter leur adoption par des utilisateurs avec des besoins différents. Dans ce contexte, une meilleure expérience utilisateur est attendue pour remédier aux limites d'utilisation des services IoT. L'expérience utilisateur peut être traduite par un niveau de service comprenant non seulement la Qualité de Service (*QoS* : Quality of Service) attendue mais aussi le niveau de sécurité et de respect de la vie privée dans l'environnement IoT. Dans cet environnement, les objets connectés de l'IoT ont des contraintes en termes de capacité mémoire, de capacité de calcul et de consommation énergétique. Néanmoins, les mécanismes de *QoS* ainsi que les mécanismes de sécurité et de protection de la vie privée des systèmes traditionnels existants ne prennent pas en considération les limitations des objets connectés. A ce titre, il est primordial de concevoir et de développer de nouveaux mécanismes de *QoS* et de sécurité ou encore des adaptations et des améliorations de ces mécanismes dans le contexte de l'IoT. Afin de garantir la *QoS* demandée par les utilisateurs, différents mécanismes doivent être prévus au niveau des différentes couches d'une architecture IoT. Ainsi, ces mécanismes doivent aussi être conçus en adéquation avec des protocoles de communication bien spécifiques à cet

environnement et doivent convenir aux capacités de calcul et de mémoire des différentes composantes d'une architecture IoT. Dans ce contexte, différents services IoT coexistent dans une même infrastructure. Ces services possèdent des caractéristiques différentes et nécessitent des garanties adaptées selon différents paramètres de *QoS*. A titre d'exemple, les services temps réel nécessitent des garanties en termes de délai et de gigue alors que les services de diffusion nécessitent une garantie de la gigue uniquement. Ainsi, dans un environnement IoT, un challenge consiste à assurer une *QoS* satisfaisante et adéquate à chaque type de service IoT à travers différents mécanismes adaptés. Dans ce contexte, divers travaux de recherche ont été initiés afin de répondre aux challenges d'intégration de mécanismes de *QoS* adaptés dans l'IoT, mais plusieurs défis restent à relever. De plus, d'autres défis restent aussi à relever pour les environnements IoT dans le cadre de la garantie du deuxième pilier du niveau de service qui est la sécurité. Ainsi, les mécanismes de sécurité doivent prendre en compte les caractéristiques spécifiques à l'IoT. La garantie de la sécurité dans un environnement IoT doit considérer les menaces issues de l'interconnexion d'un grand nombre d'objets ainsi que l'hétérogénéité des équipements. De plus, les mécanismes de sécurité doivent être présents au niveau des différentes couches d'une architecture IoT et doivent être en mesure de respecter la vie privée des utilisateurs. A ce titre, la sécurité et la vie privée dans l'IoT est un autre challenge important à prendre en considération dans ce type d'environnement.

Une nouvelle tendance dans le domaine des nouvelles technologies permet d'apporter une autonomie dans les réseaux et dans les systèmes informatiques grâce au paradigme de l'Autonomic Computing (*AC*) qui permet une gestion autonome des ressources. Dans les environnements IoT, la gestion de l'infrastructure sous-jacente est d'une complexité importante due à la taille des réseaux d'objets connectés et leur hétérogénéité. Ainsi, la gestion manuelle par un administrateur n'est plus efficace étant donné qu'elle implique des coûts élevés, des erreurs fréquentes et une lenteur caractérisant son temps de réaction. Dans ce sens, la gestion autonome est proposée comme une alternative à la gestion traditionnelle des infrastructures. Cette nouvelle tendance de gestion se base sur des systèmes composés par des éléments gérés et des gestionnaires autonomes permettant de s'adapter aux changements grâce à l'analyse de l'environnement et la prise de décision adaptée. Ainsi, l'adoption des principes de l'*AC* pour la gestion de la qualité de service et de la sécurité dans un environnement IoT est un troisième challenge important pour une meilleure gestion du niveau de service global relatif aux services IoT offerts aux utilisateurs.

1.2. Objectifs de la thèse

L'objectif de nos travaux de recherche est de proposer des mécanismes adaptés à l'environnement IoT afin d'assurer la garantie du niveau de service en termes de QoS et de sécurité ainsi que leur gestion d'une manière autonome. L'offre de niveau de service de bout en bout dans ce type d'environnement nécessite la proposition de mécanismes de *QoS* et de sécurité au niveau des différentes couches d'une architecture IoT.

Ainsi, nous proposons dans cette thèse une architecture permettant d'assurer une *QoS* de bout en bout tout en se basant sur un ensemble de contrat de niveau de services établis entre les fournisseurs de services IoT et les fournisseurs d'infrastructures sous-jacentes (i.e., fournisseur de service cloud

et fournisseur de service réseau) d'une part ; d'autre part entre les fournisseurs de services IoT et les clients de ces services. Les besoins spécifiés à travers les contrats de niveau de services établis avec les clients IoT seront traduits par différentes caractéristiques spécifiées dans les contrats de niveau de services établis avec les fournisseurs d'infrastructures. Nous devons par la suite garantir ces contrats grâce à des mécanismes adaptés que nous devons appliquer au niveau de chaque couche de l'architecture IoT. A ce titre, nous devons spécifier un mécanisme permettant d'assurer un traitement différencié des divers trafics émanant des objets connectés et à destination des passerelles au niveau de la couche basse de l'architecture IoT. Ce mécanisme sera ensuite amélioré avec des foncions d'auto-configuration et d'auto-optimisation permettant entre autres d'améliorer la consommation énergétique des objets IoT.

De plus, notre architecture IoT sera étendue afin de prendre en considération la sécurité dans un environnement IoT. Ainsi, les contrats du niveau de service établis par le fournisseur de service IoT avec les fournisseurs d'infrastructures sous-jacentes et avec les clients IoT seront étendus pour assurer les besoins de sécurité relatifs aux différentes couches de l'IoT. A ce titre, nous devons spécifier un mécanisme de sécurité permettant de contrôler l'accès des objets aux passerelles IoT selon différents critères spécifiques à l'environnement IoT dont le niveau de confiance des objets.

1.3. Organisation du manuscrit

La suite de ce manuscrit de thèse est organisée en neuf chapitres dont une conclusion générale comme suit.

Le chapitre 2 présente un état de l'art concernant le contexte général de l'Internet des Objets à travers les définitions et les architectures spécifiées par différents organismes de standardisation et projets internationaux. De plus, nous décrivons les différents domaines d'application de l'IoT dans nos sociétés contemporaines. Enfin, nous réalisons une étude comparative des différents protocoles et technologies de communication utilisés au sein d'un environnement l'IoT.

Le chapitre 3 présente les défis concernant les trois axes de recherche de la thèse à savoir la QoS, la sécurité et la gestion autonome dans l'IoT. Nous étudions dans un premier temps la *QoS* dans l'IoT à travers les motivations et les chalenges ainsi que les besoins de *QoS* au niveau de chaque couche de l'architecture IoT et les contrats de niveau de service. Par la suite, nous nous intéressons à la sécurité dans ce type d'environnement en décrivant les motivations, les challenges ainsi que les différents services de sécurité et les mécanismes correspondants dans le contexte de l'IoT. La protection de la vie privée et la confiance sont deux aspects aussi importants en relation avec la sécurité dans l'IoT. Nous définissons ces concepts ainsi que les réglementations et les travaux de recherches existants. Finalement, le troisième axe concernant la gestion autonome est étudié à travers les motivations et les challenges ainsi que les travaux de recherche existants et les objectifs à atteindre dans le cadre de l'IoT.

Nous présentons à travers le chapitre 4 notre Framework de garantie de *QoS* dans l'IoT comportant une architecture structurée en trois couches (i.e., sensing, réseau et cloud) et des contrats de niveau de service (i.e., *SLA* : Service Level Agreement) correspondants aux besoins de QoS de ces couches. Nous spécifions dans ce chapitre différents types de contrats de niveau de service portant sur la garantie de *QoS* au niveau des différentes couches de notre architecture. Ces contrats sont établis entre le fournisseur de service IoT et les fournisseurs de services cloud (i.e., *cSLA* : cloud SLA) et les fournisseurs réseau (i.e., *nSLA* : network SLA) et en interne (i.e., *gSLA* : gateway SLA). Enfin, nous définissons un contrat de niveau de service global (i.e., *iSLA* : IoT SLA) qui répond aux besoins du client IoT en se basant sur les contrats précédemment mentionnés et nous spécifions les processus d'établissement de ces différents *SLA*.

Le chapitre 5 nous permet de proposer un mécanisme de QoS au niveau de la couche sensing de notre architecture IoT afin de respecter le niveau de service décrit dans le contrat concernant cette couche (i.e., *gSLA*). Pour ce faire, nous spécifions une méthode de contrôle d'accès au canal basée sur la *QoS* appelée *QBAIoT* (QoS Based Access for IoT environnements) qui adapte le standard IEEE 802.15 en définissant des périodes d'accès en fonction de la classe de service à laquelle appartient un objet IoT. Par la suite, nous décrivons l'environnement de simulation et nous évaluons les performances de *QBAIoT* selon différents scénarios. Enfin, nous comparons les performances de *QBAIoT* avec d'autres techniques existantes dans la littérature.

Nous présentons dans le chapitre 6 notre Framework de garantie de sécurité en définissant les besoins au niveau de chaque couche de l'architecture IoT. Nous enrichissons dans ce chapitre les *SLA* de *QoS* définis dans le chapitre 4 en prenant en considération les services de sécurité à assurer dans un environnement IoT. Ainsi, nous définissons les contrats de type *SECiSLA*, *SECcSLA*, et *SECgSLA*. Ces extensions permettent d'ajouter des paramètres qualitatifs et quantitatifs de sécurité, de vie privée et de confiance spécifiques à ce type d'environnement. Enfin, nous décrivons les processus d'établissement de ces différents contrats.

Le chapitre 7 nous permet de spécifier notre méthode de contrôle d'accès des objets IoT à une passerelle appelée *IoT-MAAC* (IoT Multiple Attribute Access Control). Ce mécanisme de sécurité est basé sur plusieurs attributs dont le niveau de confiance des objets IoT. Nous présentons en premier lieu notre Framework de contrôle d'accès basé sur les deux standards *XACML* (eXtensible Access Control Markup Language) et *SAML* (Security Assertion Markup Language). Par la suite, nous décrivons notre système d'évaluation du niveau de confiance des objets IoT basé sur une approche de logique floue. De plus, nous présentons notre méthode *IoT-MAAC* en définissant l'algorithme de prise de décision, l'échange d'informations et la stratégie de contrôle d'accès à appliquer.

Dans le chapitre 8, nous décrivons la gestion autonome de notre méthode d'accès au canal *QBAIoT* à travers les deux fonctions d'auto-configuration et auto-optimisation de la boucle de contrôle fermée. Ainsi, nous présentons dans un premier temps l'auto-configuration de *QBAIoT* qui permet l'adaptation de la configuration de cette méthode en fonction des classes de trafic existantes

dans l'environnement IoT. Par la suite, nous spécifions la fonction d'auto-optimisation de *QBAIoT* permettant d'améliorer l'utilisation des ressources en temps réel. Enfin, nous évaluons ces deux fonctions de gestion autonome en prenant en compte différents scénarios de simulation dans le but de déterminer leur efficacité dans un environnement IoT.

La gestion autonome de *QBAIoT* ne se limite pas à ces deux fonctions mais prend en compte aussi l'auto-optimisation de la consommation énergétique que nous spécifions dans le chapitre 9. Ainsi, nous proposons dans ce chapitre un processus permettant de minimiser la consommation énergétique du système IoT afin d'étendre sa durée de vie. Pour ce faire, nous définissons un Framework permettant de réduire la consommation énergétique des objets IoT en se basant sur un système de logique floue et des algorithmes de prise de décision. De plus, nous évaluons la durée de vie du système IoT pour déterminer l'efficacité de notre proposition d'auto-optimisation de la consommation énergétique.

Enfin, le dernier chapitre conclut le manuscrit de thèse et présente les perspectives des travaux de recherche réalisés.

Chapitre 2 – Internet des objets : Généralités

2.1. Introduction

Les environnements de type Internet des objets (i.e., IoT : Internet of Things) deviennent de plus en plus présents dans nos sociétés contemporaines. Ce paradigme est né de l'idée d'interconnexion de différents types d'objets entre eux à travers le réseau Internet afin d'améliorer la qualité de vie des humains. Ainsi, plusieurs organismes se sont intéressés à l'IoT afin de définir ses différents composants selon divers architectures en couches mais aussi les concepts et technologies qui permettent de le déployer.

Ce chapitre présente les différentes notions de base concernant l'environnement IoT. Premièrement, nous présentons dans la section 2 l'historique et le contexte de l'Internet des objets. Ensuite, nous décrivons dans section 3 différentes définitions et architectures de cet environnement. La section 4 nous permet de présenter les domaines d'application de l'IoT. Enfin, nous étudions dans la section 5 les diverses technologies de communication utilisées pour assurer la connectivité des objets de l'IoT avant de conclure ce chapitre avec la section 6.

2.2. Historique et contexte de l'Internet des Objets

2.2.1. Historique

Le terme Internet des objets (IoT : Internet of Things) remonte à la fin des années 90 lorsque Kevin Ashton, cofondateur du centre Auto-ID à Massachusetts [1], l'a utilisé pour définir le lien entre la technologie RFID et l'Internet [2]. Auto-ID est un groupe de laboratoires de recherche travaillant sur les technologies de radiofréquence et les technologies de capteurs émergentes. Selon Ashton, l'IoT n'est pas une vision mais un nouvel environnement capable de collecter des informations indépendamment de l'interaction humaine et de comprendre le monde sans être opéré par les humains [2].

En 2000, la notion d'objets connectés a été abordée pour la première fois par les industriels suite aux expérimentations du fabricant coréen LG (Life's Good) permettant d'interconnecter des appareils de la vie quotidienne à travers l'Internet. Ainsi, LG a annoncé le plan « réfrigérateur interconnecté » en 2000 [3]. Ces expérimentations ont été effectuées bien avant que l'alliance IPSO (Internet Protocol for Smart Objects), alliance regroupant les grands industriels technologiques, promeut l'utilisation du protocole *IP* (Internet Protocol) pour l'interconnexion des objets en 2008 [4]. En revanche, Cisco IBSG (Internet Business Solutions Group) a indiqué que l'IoT est né entre 2008

et 2009, simplement au moment où plus d'objets que d'humains étaient connectés à l'Internet tout en citant la croissance massive du nombre d'appareils connectés [5]. Par ailleurs, l'IoT a été ajouté en 2011 au Gartner Hype Cycle, organisme de recherche et de conseil international, qui suit les cycles de vie des technologies du moment du déclenchement à la production. De plus, l'IoT a atteint le pic des attentes exagérées (Peak of Inflated Expectations) du cycle Gartner Hype en 2014 [6]. Depuis son lancement, l'IoT a fait l'objet d'une immense évolution et il a été considéré par plusieurs organismes de standardisation mais aussi par différentes commissions européennes et mondiales. En effet, l'Union Internationale des Télécommunications – secteur de la normalisation des Télécommunications (UIT-T) a proposé en 2005 un rapport portant sur l'IoT nommé "The Internet of Things" [7]. En outre, la première conférence internationale sur le sujet de l'IoT a pris place à Zurich en 2008 sous le nom « Internet of Things » [8]. En même temps, le NIC (National Intelligence Council) a énuméré l'IoT comme l'une des six technologies civiles qui ont des répercussions potentielles sur les intérêts américains jusqu'en 2025 [9] [10]. Par la suite, l'UIT-T a proposé le document Y.2060 présentant une vue d'ensemble de l'IoT en 2012 [11] et des recommandations pour l'environnement IoT à travers le document Y.2066 en 2014 [12]. D'autres organismes de standardisation se sont intéressés à l'environnement IoT ainsi que les technologies associées. Ainsi, l'IETF (Internet Engineering Task Force) a créé le groupe de travail 6Lo [13] en 2014, précédé par le groupe de travail 6LoWPAN [14], pour adapter les technologies radio à l'environnement IoT. De plus, cet organisme propose plusieurs documents de références décrivant les challenges à considérer dans l'IoT [15] [16].

De nos jours, plus de 26 milliards d'objets connectés existent dans le monde pour offrir différents services de l'IoT dans les différents domaines d'application de ce nouveau paradigme. D'ici 2020, les estimations indiquent que plus de 30 milliards d'objets seront connectés [17].

2.2.2. Contexte de l'IoT

L'IoT n'est plus un sujet de science-fiction mais une évidence dans notre vie de tous les jours. Actuellement, un humain possède en moyenne 2 appareils numériques ou électroniques connectés permettant de lui offrir des services intelligents [18]. L'intégration de l'IoT dans la vie quotidienne des humains permet d'offrir de nouvelles applications améliorant la qualité de vie à travers l'interconnexion des objets. L'IoT est à l'origine d'une révolution dans le rapport des humains avec la technologie. Cette révolution concerne divers secteurs de la vie quotidienne induisant un changement dans les habitudes et les comportements. Ainsi, l'IoT a permis la création de réfrigérateurs connectés, de chaussures connectées, de compteurs électriques connectés, de caméras connectées, etc. Dans ce contexte, la gestion de notre résidence principale peut être effectuée à distance et même d'une manière autonome et intelligente. Cette gestion à distance et intelligente peut rendre la vie humaine plus confortable, plus facile, voire même plus sûre. En effet, même les domaines critiques, comme le domaine médical, sont impactés par l'intégration des technologies de l'IoT en connectant tout type d'objets. Dorénavant, les médecins pourront réaliser le diagnostic de

leurs patients à distance, et ce, à travers des capteurs portés ou implantés sur les patients. Cette avancée technologique améliore les services vitaux en optimisant les performances et en améliorant la qualité de ces services.

L'IoT est à l'origine d'une collecte et/ou d'une création d'un volume important de données. Cet énorme volume de données, connu par le terme « Big Data », permet d'une part, d'avoir une richesse incroyable en termes d'informations permettant l'offre de services avancés. D'autre part, ce volume de données crée de nouveaux défis à prendre en considération tels que la sécurisation, le traitement et l'accessibilité en temps-réel de ces données.

2.3. Définitions et architectures de l'Internet des objets

2.3.1. Définitions

2.3.1.1. Organismes de standardisation

Différents organismes de standardisation ont spécifié plusieurs définitions portant sur le nouveau paradigme de l'Internet des objets. Nous présentons dans cette section les définitions les plus importantes proposées par des organismes de standardisation internationaux.

L'UIT-T étudie l'environnement IoT et ses différents domaines d'application à travers le groupe de travail SG20, qui a pris la relève suite à la clôture des travaux du groupe SG13 en 2016. Dans ce contexte, le document Y.2060 de l'UIT-T définit l'IoT comme étant un réseau ubiquitaire disponible n'importe où, à n'importe quel moment et à n'importe qui [11]. Ainsi, l'Internet des objets est une infrastructure globale de la société d'information, permettant d'offrir des services avancés en interconnectant des objets à travers des technologies de communication variées [19]. De plus, l'UIT-T a défini les objets dans l'environnement IoT en tant qu'objets physiques et objets virtuels. Un objet physique peut être représenté par un ou plusieurs objets virtuels, et des objets virtuels peuvent exister sans la nécessité d'une correspondance physique. Ces objets peuvent communiquer entre eux par le biais de passerelles, à travers un réseau de communication, ou directement. De même, l'UIT-T spécifie dans [11] [12] plusieurs exigences pour l'environnement IoT telles que l'inter-connectivité, l'hétérogénéité, la protection de la vie privée, la sécurité, l'identification, etc.

En outre, le groupe de travail SWG10, précédé par SWG5, de l'ISO/IEC (International Organization for Standardization / International Electrotechnical Commission) a présenté dans [19] une définition de l'IoT ainsi qu'une spécification du vocabulaire utilisé au sein de ce type d'environnement. Selon cet organisme, l'Internet des objets est un réseau d'objets physiques qui collectent et transmettent des données. Il s'agit d'une infrastructure d'objets interconnectés, d'humains et de ressources d'informations qui permettent de traiter les informations remontées par les objets et de réagir en conséquence [20][21]. L'ISO/IEC a défini différentes exigences pour l'IoT comme l'auto-configuration, l'identification unique, la standardisation des interfaces, la connectivité, la fiabilité, la mobilité, etc.

Selon l'IETF (Internet Engineering Task Force), l'idée générale de l'IoT est de connecter des objets pour assurer des services contextuels, et cela à travers différentes technologies pour aboutir à un service accessible depuis n'importe où et à n'importe quels moments [19]. L'IETF considère l'IoT comme étant un réseau d'objets interconnectés, adressables d'une manière unique et utilisant des protocoles standardisés pour la communication entre ces objets [22]. De plus, l'IETF insiste sur les exigences spécifiées par l'ISO/IEC pour l'environnement IoT.

Pareillement, d'après le groupe de travail T2TRG (Thing-to-Thing Research Group) de l'IRTF (Internet Research Task Force) créé en 2015, l'IoT assure la communication entre les objets en utilisant Internet pour offrir de meilleurs services. Ce groupe de travail a défini l'IoT comme un réseau où les nœuds à faibles ressources communiquent entre eux et avec le réseau public Internet pour assurer des innovations [23].

L'IEEE (Institute of Electrical and Electronics Engineers), à travers le projet et le groupe de travail P2413, a étudié l'environnement IoT [24]. Le projet P2413 comporte des membres industriels comme Cisco Systems et des organismes comme l'IIC (Industrial Internet Consortium). L'IIC est un consortium formé de plusieurs industriels connus dans le monde des technologies de l'information comme IBM, HUAWEI, Intel, etc. Ainsi, l'IoT est décrit dans [19] comme étant l'interconnexion de différents systèmes possédant des standards bien adaptées comme par exemple différentes technologies de communications permettant de respecter les besoins de chacun de ses systèmes.

Le NIST (National Institute of Standards and Technology) considère l'IoT comme étant un système cyber-physique [25]. Ce système connecte des objets intelligents à travers de nouvelles techniques pour améliorer la qualité de vie des utilisateurs [26]. Finalement, OASIS (Organization for the Advancement of Structured Information Standards) s'est focalisé sur l'adaptation des technologies existantes pour l'IoT [27]. Il définit l'IoT comme étant un système à travers lequel l'Internet est connecté au monde physique via des capteurs [28].

2.3.1.2. Autres définitions de l'IoT

L'émergence de l'IoT dans la société contemporaine, a permis la création de différents projets et consortium européens et internationaux traitant cette thématique. Dans cette partie, nous présentons d'autres définitions de l'IoT émanant de plusieurs projets de recherche internationaux.

IoT-I (Internet of Things Initiative) est un projet européen financé par le programme FP7-ICT (de 2010 à 2012) [29]. Il a comme but la création d'un environnement, économiquement durable et socialement acceptable en Europe, pour les technologies IoT. IoT-I a défini l'IoT comme étant une part intégrée de l'Internet du futur formée par une infrastructure réseau globale et dynamique possédant des capacités d'auto-configurations basées sur des standards et des protocoles de communication interopérables. De même, IoT-I a spécifié que les objets IoT doivent posséder des identités et des attributs et doivent utiliser des interfaces intelligentes afin d'être intégrés d'une

manière transparente. Cette définition de l'IoT a été utilisé aussi par le consortium de projets européens sur l'IoT à savoir le CERP-IoT (Cluster of European Research Projects on the Internet of Things) [30].

CASAGRAS (Coordination and Support Action for Global RFID) est un projet européen financé par le programme FP7-ICT (2008 - 2009) [31]. Ce projet s'est concentré sur des études fondamentales pour les technologies sous-jacentes de l'IoT et leurs adaptations. Selon ce projet, l'IoT a été définie comme étant une infrastructure de réseau globale, reliant les objets physiques et virtuels pour la capture de données à travers des capacités de communication. Cette infrastructure intègre les développements existants de l'Internet et du réseau. De plus, elle offre une identification unique aux objets et une capacité de communication qui sont la base pour le développement de services et d'applications coopératives indépendants. Ceux-ci seront caractérisés par la capture de données d'une manière autonome, le transfert des données, la connectivité réseau et l'interopérabilité [32].

2.3.2. Architectures de l'IoT

2.3.2.1. Architectures des organismes de standardisation

Différentes architectures pour l'Internet des objets ont été présentées par des organismes de normalisation. Dans ce qui suit, nous décrivons plusieurs exemples d'architectures proposées pour l'IoT. Ainsi, l'UIT-T a proposé un modèle de référence pour l'Internet des objets basé sur plusieurs couches [12]. Ce modèle de référence (voir Figure 2.1) spécifie 4 couches horizontales (Application, Support, Réseau et Dispositifs) et 2 couches verticales (Gestion et Sécurité).

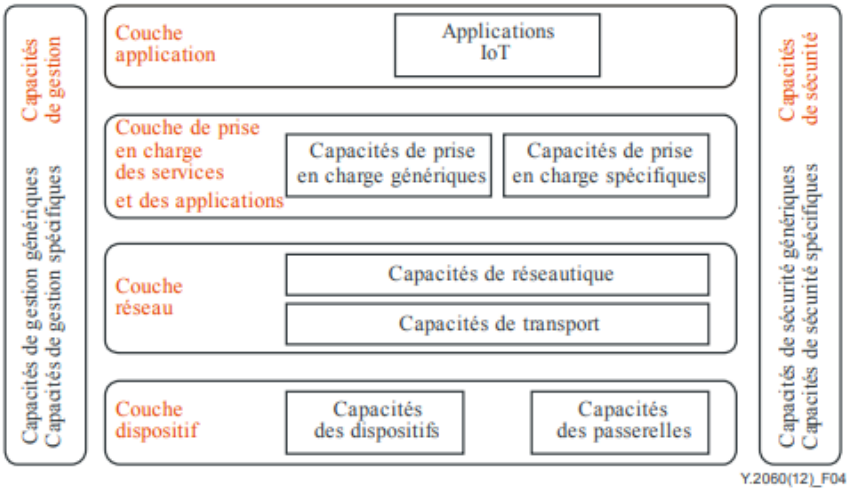

Figure 2.1 : Architecture de l'Internet des objets de l'UIT-T [12]

La couche Application (Application Layer) comprend les applications et les services IoT. Ensuite, la couche Support (Service Support et Application Support Layer) définit les capacités de support génériques communes à toutes les applications comme le traitement et le stockage des données. Elle comprend aussi les capacités de support spécifiques qui répondent aux besoins d'une

application particulière. La couche Réseau (Network Layer) offre deux services. D'une part, les capacités réseaux qui assurent le contrôle de la connectivité, la gestion de la mobilité et la comptabilité au niveau de cette couche. D'autre part, les capacités de transport qui permettent l'acheminement des données provenant des applications ou encore des informations de contrôle et de gestion de l'environnement. La couche Dispositifs (Device Layer) spécifie les capacités propres aux objets et aux dispositifs connectés ainsi que les capacités des passerelles. Quant aux deux couches verticales, elles définissent les capacités génériques de gestion (gestion des objets, du réseau, du trafic et de la congestion) et de sécurité génériques à toutes les couches (autorisation, authentification, intégrité et protection de la vie privée). De plus, ces deux couches verticales présentent des capacités spécifiques qui dépendent du type d'application IoT considérée telles que les exigences de sécurité de paiement mobile.

De plus, le consortium IIC propose à travers le rapport « Industrial Internet Reference Architecture » [33] une architecture système applicable pour l'Internet des objets. Cette architecture trois tiers est basée sur 3 couches verticales ou encore trois niveaux (voir Figure 2.2.). La couche de Bord (Edge Tier) correspond à l'ensemble des nœuds qui collectent les données à travers des réseaux de proximité. Elle permet d'implémenter toutes les fonctions de contrôle. Ensuite, la couche Plateforme (Platform Tier) reçoit, traite et transmet les commandes de contrôle vers la couche de Bord. Elle permet aussi le traitement, l'analyse et l'exécution des opérations sur les données récoltées des objets avant de les transmettre dans l'autre sens vers la couche Entreprise (Entreprise Tier). Ainsi, la couche Entreprise prend les décisions et assure le rôle d'interface pour les utilisateurs finaux. Cette dernière comprend alors les applications qui permettent de générer les commandes de contrôle à envoyer à la couche Plateforme. Les différentes couches de cette architecture sont interconnectées via des réseaux d'accès et des réseaux de services.

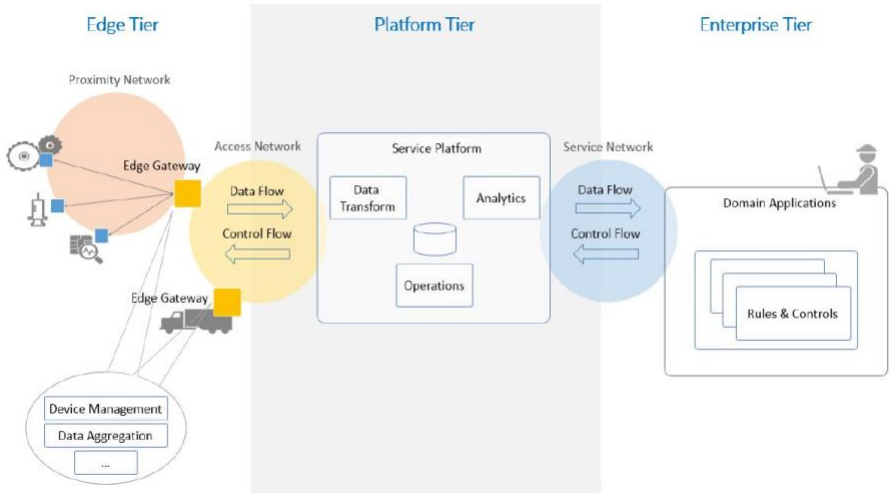

Figure 2.2 : Architecture de l'Internet des Objets de l'IIC [33]

L'IEEE, à travers le Webinaire « Integrating the IoT and Cultural Heritage in the Smart City», a proposé une architecture (voir Figure 2.3) pour l'IoT basée sur 4 couches (couche Interface, couche

Logique de service, couche Données et couche Ressources) [34]. Chaque couche comprend des entités et des éléments spécifiques permettant de fournir diverses fonctionnalités. Ainsi, la couche Interface (Interface Layer) comprend tous types d'interfaces permettant l'interaction entre les utilisateurs et les applications IoT. La couche Logique de service (Service Logic Layer) permet de définir le contexte de l'application à travers l'entité de contexte et l'entité logique de l'application. Elle permet aussi de fournir des descriptifs concernant les ressources et les services. La couche de Données (Data Layer) comprend les données brutes récoltées de la couche directement inférieure (i.e., la couche Ressource) et des entités permettant de restructurer ces données et de les agréger pour les remonter aux couches supérieures. Finalement, la couche Ressources (Ressource Layer) comprend tous les objets connectés ainsi que différentes capacités (i.e., calcul et stockage au niveau de cette couche). Les objets permettent de capter les informations, d'appliquer les requêtes envoyées par l'application, de faire du stockage et du traitement près de la source pour les données récoltées. La communication inter-couches dans cette architecture se fait à travers des *API* (Application Program Interface) bien spécifiques.

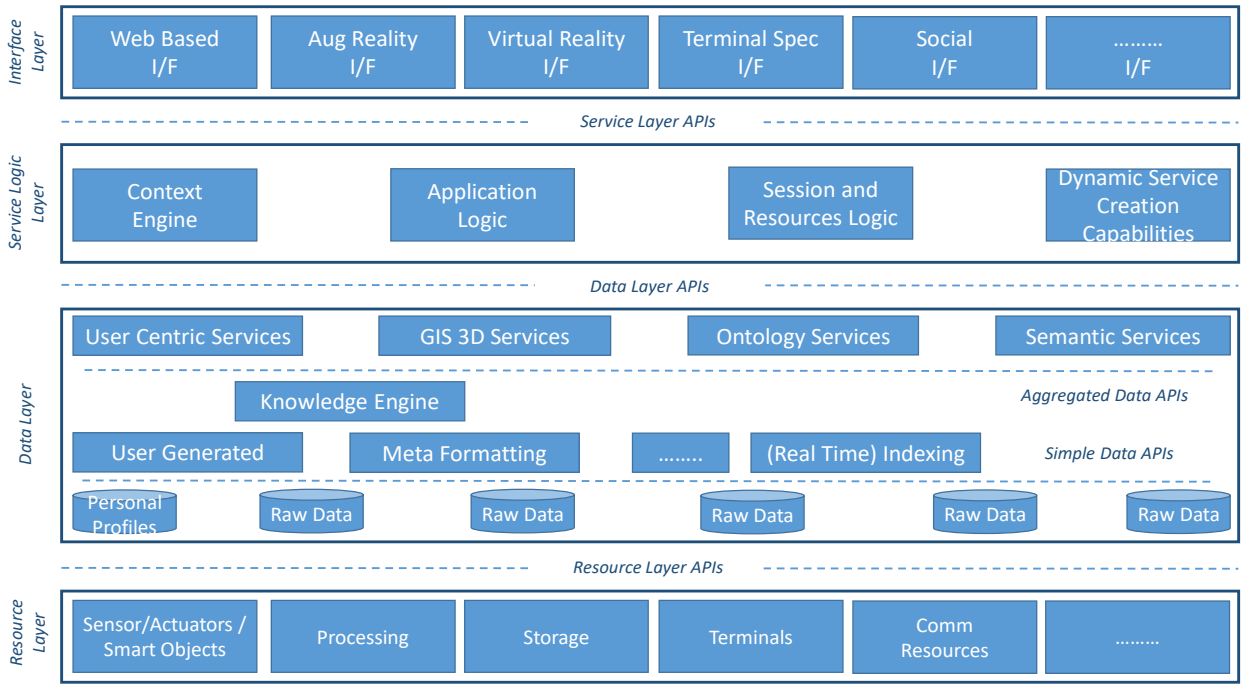

Figure 2.3 : Architecture IEEE [30]

2.3.2.2. Autres architectures de l'IoT

Différents travaux de recherches internationaux et européens mais aussi des industriels du domaine des technologies de l'information ont proposé d'autres architectures pour l'environnement IoT. Dans ce qui suit, nous présentons certaines de ces architectures avec des caractéristiques différentes.

Le projet européen IoT-A (IoT-Architecture), financé par FP7-ICT de septembre 2010 à novembre 2013, a proposé un modèle de référence pour l'architecture IoT nommé « Architectural Reference Model – ARM » [35]. Ce modèle (voir Figure 2.4) est basé sur 5 couches horizontales (i.e., Application, Exécution de Processus et Orchestration de Service, etc.) et 2 couches transversales (i.e., Gestion et Sécurité) pour répondre aux besoins de l'environnement IoT. Chacune des couches représente un groupe de fonctionnalités formé de composants fonctionnels afin d'offrir les ressources, de maintenir et d'organiser les informations reçus et offrir les services.

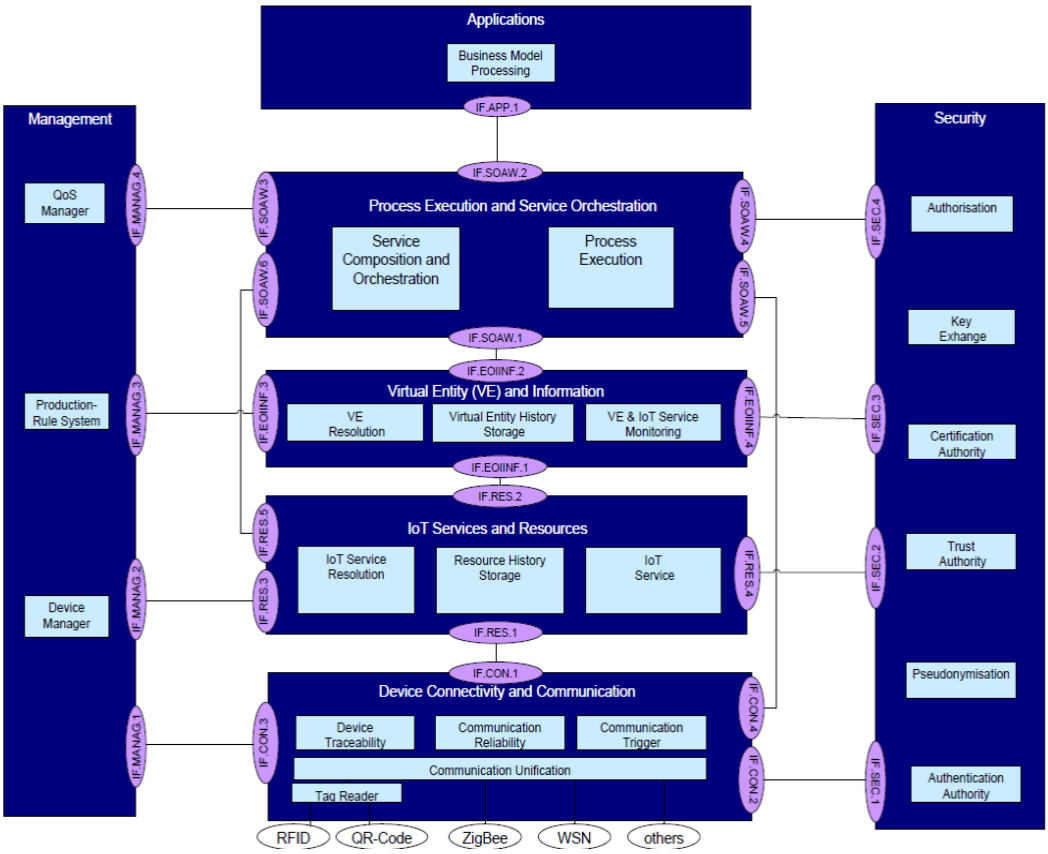

Figure 2.4 : Modèle de référence architectural d'IoT-A [35]

La couche Application (Applications layer) de ce modèle décrit les fonctionnalités assurées par les applications déployées sur une instance de cette architecture. Ensuite, la couche Exécution de Processus et Orchestration de Service (Process Execution and Service Orchestration layer) permet d'instancier les services en déployant les modèles de processus. Les services seront décomposés en processus et les processus seront déployés au niveau de cette couche. L'exécution des processus demandés par l'application assure une amélioration de la qualité de l'information. La couche Entités Virtuelles et Informations (Virtual Entity and Information layer) maintient et organise les liaisons entre les entités virtuelles, créées à partir des ressources disponibles, les ressources et les services. La couche Services et Ressources IoT (IoT Services and Ressources) décrit les services et fournit un lien aux ressources exposées et aux fonctionnalités requises par les services IoT pour le traitement des

données. Cette couche permet aussi de récupérer les descriptions des ressources et des services. Enfin la dernière couche horizontale, à savoir la couche Connectivité et Communication des Dispositifs (Device Connectivity and Communications) assure un ensemble de méthodes et de primitives pour la connectivité des objets. Au niveau de cette couche, différentes technologies de communication inter-objets sont disponibles comme RFID, QR-Code, ZigBee, et les technologies *WSN* (Wireless Sensor Network) d'une façon générale.

Les 2 couches transversales, qui sont utilisées dans l'architecture du projet IoT-A, communiquent avec les 5 couches horizontales. Ainsi, la première couche transversale, à savoir la couche Gestion, permet de gérer les ressources, les objets et la qualité de service. Quant à la couche Sécurité, elle permet d'empêcher les applications non autorisées d'obtenir un accès aux ressources et elle permet aussi d'assurer la protection de la vie privée des utilisateurs en ce qui concerne les données échangées. La communication inter-couches (verticales et transversales) dans cette architecture se fait à travers des interfaces bien spécifiques.

En plus des projets de recherche, les industriels du domaine ont aussi proposé des architectures pour l'environnement IoT. A ce titre, CISCO, industriel connu dans le monde des réseaux et des systèmes informatiques, adopte la nomenclature « Internet of Everything (IoE) » à la place de « Internet of Things » et propose une architecture pour l'IoE représentée par la Figure 2.5 [36].

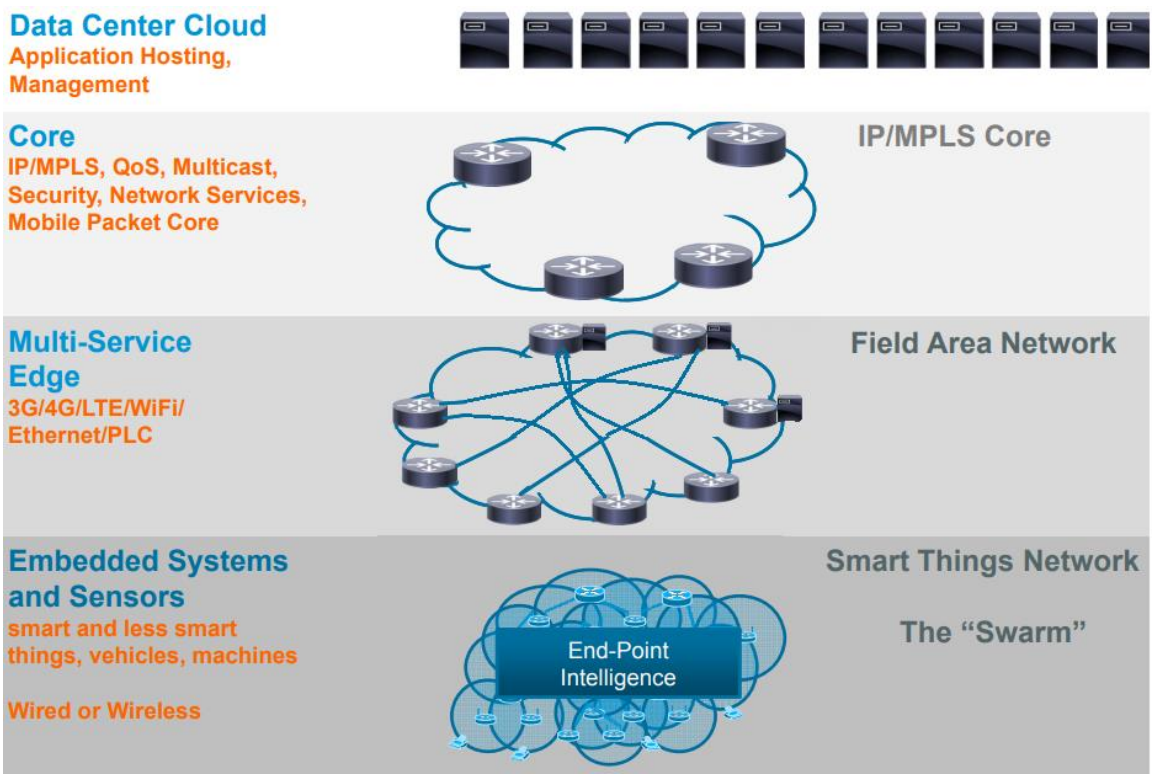

Figure 2.5 : Architecture CISCO pour l'IoE [36]

Cette architecture est basée sur 4 couches horizontales. Premièrement, la couche Systèmes embarqués et Capteurs (Embedded Systems and Sensors layer) permet de récolter les données et de capter les événements qui se produisent dans l'environnement IoE. Ensuite, la couche Multi-Services (Multi-Service Edge layer) permet de gérer le réseau interconnectant les objets aux centres de calcul. Cette couche permet aussi d'assurer des capacités de type « Fog computing » pour répondre exigences des applications sensibles au délai en effectuant le traitement des données près des objets. Les termes Fog Computing ou Edge Computing désignent l'extension du cloud au bord du réseau afin de faciliter le fonctionnement des services de calcul, de stockage et de mise en réseau entre les périphériques et les centres de calcul cloud. La troisième couche de l'architecture proposée par CISCO, à savoir la couche Cœur (Core layer) spécifie différentes fonctionnalités comme l'analyse des performances des réseaux, la détection d'intrusion, le monitoring de l'environnement et la surveillance. Finalement, la couche Data Center Cloud (Data Center Cloud layer) permet de centraliser l'intelligence et d'assurer les capacités de calcul requises pour les applications IoE. A travers cette architecture, CISCO présente aussi les différentes technologies qui peuvent être utilisées au niveau de chaque couche.

Intel, un autre acteur dans le domaine des technologies de l'information, a proposé une architecture pour l'IoT représentée par Figure 2.6 [37]. Cette architecture repose sur 3 couches : Objets (Things), Réseau (Network) et Cloud. La couche Objets consiste en une collection d'objets connectés directement à Internet ou non. Cette couche inclut aussi des capacités de sécurité, de gestion et des API. La couche Réseau comprend des passerelles qui collectent les données des objets et assurent une agrégation et un filtrage des données. Les passerelles gèrent aussi les différents objets en assurant leurs configurations et leur surveillance. La connexion des passerelles vers le cloud est assurée par des fournisseurs de services réseaux pour acheminer le trafic jusqu'à l'infrastructure cloud du fournisseur de service IoT ou bien vers une infrastructure cloud d'une tierce partie. La couche Cloud permet le stockage de données, l'application de la sécurité au niveau des serveurs de stockage et de traitement, la gestion et les capacités de calcul, etc.

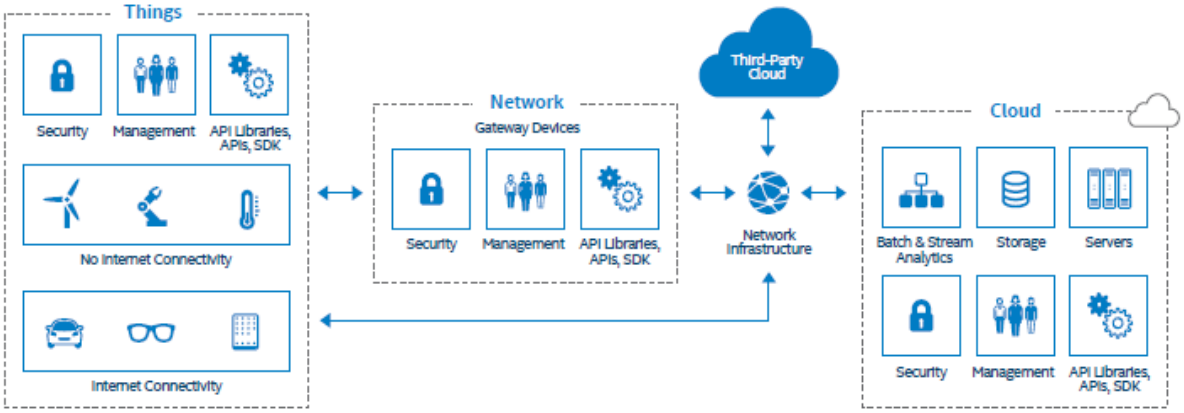

Figure 2.6 : Architecture IoT proposée par Intel [37]

2.4. Domaines d'applications de l'Internet des objets

L'Internet des objets améliore la qualité de vie des humains dans différents domaines de la vie courante. Nous citons à ce titre, le domaine de la santé, de la ville intelligente, des réseaux véhiculaires, etc. Dans ce contexte, l'ISO/IEC se focalise sur la standardisation des technologies sous-jacentes utiles dans les différents domaines d'applications de l'IoT. Ainsi, le groupe de travail 9 du comité technique 1 (JTC 1/WG9) de l'ISO/IEC s'intéresse à la normalisation des technologies Big Data dans les domaines de l'IoT [21]. En outre, divers fournisseurs proposent des solutions pour l'implémentation et l'offre de services IoT dans différents domaines d'applications. A titre d'exemple, la plateforme Kaa [38] fournit une gamme de fonctionnalités permettant aux développeurs de créer des applications avancées pour leurs dispositifs intelligents, de gérer de manière flexible les écosystèmes de leurs périphériques, d'orchestrer le traitement de données de bout en bout, etc. Dans ce qui suit, nous présentons les caractéristiques et les spécificités de divers domaines d'applications de l'IoT.

2.4.1. E-Santé

Le vieillissement de la population entraine un besoin de surveillance et de contrôle des personnes âgées. Ce suivi est basé sur un ensemble de capteurs connectés permettant de créer un système de soins décentralisé. Chaque patient est associé à un système de surveillance lui permettant d'être monitoré et surveillé sans besoin de se déplacer au centre médical. Les données médicales récoltées améliorent les soins de santé en personnalisant les traitements et en facilitant la vie des patients. Ainsi, une grande partie du travail des médecins (tests, diagnostic, prescriptions, modifications de comportement) peut être effectuée par des systèmes automatisés, qui collectent et analysent de manière passive et active des données concernant les patients. Par conséquent, une base de données riche en informations sera disponible et permettra d'alerter les médecins en cas de besoin tout en mettant à leur disposition une vue plus globale pour chaque patient.

Différents services sont inclus dans le domaine de l'e-santé. L'aide à la vie autonome est un service essentiel. Il implique l'utilisation de maisons intelligentes pour permettre la surveillance et les soins des patients dans des environnements indépendants. Les applications déjà développées comprennent la surveillance de la pression artérielle et du diabète, la surveillance de la température corporelle et la surveillance de la saturation en oxygène [39] [40]. Ce domaine d'application a attiré l'attention d'un grand nombre d'industriels qui essaient de proposer différents produits utiles pour l'e-santé. En effet, Ericsson et ses partenaires proposent des prototypes portables permettant de surveiller les patients atteints de diabètes avec des durées de vie de batterie importantes [41].

De même, ce domaine d'application attire l'attention de plusieurs organisations internationales qui essaient de normaliser les technologies utilisées pour bien répondre à ses exigences [21]. Les organisations internationales et mondiales de la santé visent à promouvoir l'usage des technologies de l'e-santé dans le monde. Ainsi, l'Organisation Mondiale de la Santé (*WHO* : World

Health Organisation) et le Programme des Technologies Appropriées en Santé (*PATH* : Program for Appropriate Technology in Health) ont signé un partenariat visant à accélérer l'évolution de la santé numérique [42].

2.4.2. Villes intelligentes

Dans le contexte des bâtiments intelligents, l'Internet des objets offre des systèmes de gestion qui permettent d'automatiser les fonctions de contrôle à travers des dispositifs intelligents fournissant des données et des analyses en temps réel. Grâce aux informations remontées par les objets de l'IoT, les besoins et préférences en termes d'éclairage, de chauffage ou de ventilation seront anticipés. De même, ces informations peuvent concerner les systèmes de sécurité, les compteurs d'électricité ou l'élimination des eaux et des déchets. La réactivité vis-à-vis de ces données récoltées grâce aux objets de l'IoT permet aux bâtiments de s'adapter aux changements et d'effectuer des modifications en conséquence en temps réel pour avoir une efficacité accrue et réduire les coûts d'exploitation.

La mise en place des bâtiments intelligents permet la création des villes intelligentes et cela en intégrant d'autres dispositifs IoT lors de l'offre de différents services dans la ville tels que le transport, l'eau et la qualité de l'air. L'intelligence des bâtiments est assurée par l'utilisation des appareils intelligents au niveau des domiciles et des bureaux. Selon IHS Markit [43], un des leaders mondiaux dans le monde de l'informatique proposant des solutions d'analyse et de conseil, une croissance significative de la demande en appareils intelligents pour les foyers a été remarquée, avec plus de 161 millions d'unités vendus entre 2010 et 2016. Plus de la moitié de ces appareils ont été vendus en 2016, soit une augmentation de 64% par rapport à 2015. Cette augmentation comprenait l'achat de systèmes de gestion intelligente de l'énergie tels que les thermostats, les solutions de sécurité telles que les serrures intelligentes et les assistants personnels tels que Google Home, Bosch Mykie et Alexa d'Amazon [40] [43].

Différents organismes de standardisation se focalisent sur ce domaine d'application de l'IoT à travers des groupes de travail qui s'intéressent à la normalisation des technologies utilisées. A titre d'exemple, l'ISO et l'IEC, à travers le sous-comité technique JTC1/SC25, normalisent les systèmes de microprocesseurs et les supports d'interconnexion associés aux équipements pour les environnements commerciaux et résidentiels [44]. De plus, le papier blanc de l'IEC [45] décrit les bâtiments intelligents et les villes intelligentes comme étant des domaines d'application de l'IoT et spécifie comment orchestrer l'infrastructure nécessaire aux villes intelligentes et durables. De plus, des industriels comme Nokia ont proposé sur le marché plusieurs services et technologies pour gérer la surveillance vidéo, le stationnement et l'environnement d'une ville intelligente d'une façon générale en se basant sur des réseaux de capteurs. Ainsi, Nokia a proposé une infrastructure avec des applications permettant de transformer la cité contemporaine en une ville intelligente [46].

2.4.3. Systèmes de transport intelligents

Tous les types de systèmes de transport peuvent bénéficier des avantages qu'offre l'IoT. En effet, les solutions IoT promettent de rendre les systèmes de transport plus intelligents et plus performants en améliorant la sécurité, l'efficacité des déplacements, la maintenance des véhicules et en proposant une gestion plus stratégique du trafic [47]. Ainsi, l'utilisation de l'IoT dans les systèmes de communication entre véhicules et infrastructures (*V2I* : Vehicle to Infrastructure) et les systèmes de communication entre véhicules (*V2V* : Vehicle to Vehicle) améliore la sécurité, l'efficacité et les performances des transports publics et privés. De plus, ces systèmes contribuent à la réduction de la congestion et à l'amélioration de la gestion de l'espace. Dans ce contexte, les conducteurs des voitures connectées bénéficient d'un grand nombre de services comme la navigation assistée, le trafic en temps réel et les informations de stationnement ainsi que l'intégration de téléphones intelligents avec des tableaux de bord et des appareils portables (i.e., montres connectées, etc.) [21].

La révolution du monde de transport à travers le déploiement de l'Internet des objets dans le domaine des réseaux véhiculaires s'effectue grâce à l'utilisation des réseaux de capteurs et différents types d'applications IoT. En effet, il existe un nombre important de capteurs dans différents types de véhicules. Certains capteurs ou encore les objets IoT d'une façon générale sont utilisés dans le système embarqué dans la voiture pour surveiller le fonctionnement du moteur, le contrôle des émissions ou encore les dispositifs de freinage [40]. D'autres capteurs sont également intégrés pour faire partie de l'infrastructure de transport. Par exemple, les routes intelligentes déterminent le nombre de voitures sur chaque ligne à travers des capteurs et ensuite opèrent les feux de circulation suivant cette information pour minimiser la congestion des routes.

L'efficacité de ce domaine d'application de l'IoT est démontrée à travers la mise en place de cette révolution technologique dans différents projets. Ainsi, le projet ParkDC [48] du département de transport de Washington D.C. utilise un système de surveillance basé sur l'IoT pour informer les conducteurs des places de stationnement disponibles et appliquer des tarifs appropriés en fonction de la demande en temps réel [49]. Des investissements importants ont également été réalisés au Royaume-Uni avec l'introduction du programme Smart Motorways de Highways England [50] qui permet de réduire les embouteillages en surveillant en temps réel la charge de trafic véhiculaire, et ce en changeant la vitesse limite en temps réel selon les caractéristiques des routes. UKCITE, un autre projet britannique financé par le Centre des véhicules connectés et autonomes a pour objectif de fournir un environnement de test pour les véhicules de conduite connectés et autonomes [51].

2.5. Protocoles de communication dans l'Internet des objets

Pour assurer la communication entre les objets de l'IoT, différents technologies de communication peuvent être utilisées en prenant en considération les besoins et les capacités des environnements IoT. Dans ce contexte, une technologie de communication spécifique est utilisée conformément aux caractéristiques des objets IoT et aux exigences des applications IoT

correspondantes. Les objets sont généralement dotés de peu de ressources mémoires et de capacités de calcul réduites. Par conséquent, les technologies de communication utilisées dans un environnement IoT doivent être conformes aux caractéristiques de ces objets.

Les technologies de communication au sein de l'IoT peuvent correspondre à une adaptation d'une technologie existante ou encore à une nouvelle technologie spécifiquement mise en place pour ce type d'environnement. Dans ce qui suit, nous proposons une classification des technologies et protocoles de communication de l'IoT selon deux groupes (voir Figure 2.7) : le groupe des technologies de communication sans fil cellulaires et celui des technologies de communication sans fil non cellulaires.

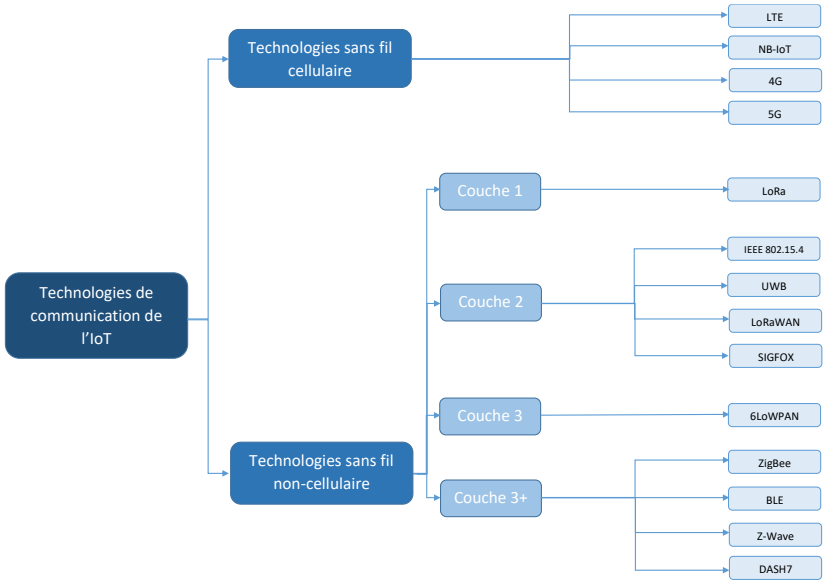

Figure 2.7 : Classification des technologies de communication de l'IoT

2.5.1. Technologies sans fil cellulaires

Différentes technologies et variantes de technologies sans fil cellulaires ont été proposées pour une utilisation dans l'Internet des objets. Dans ce qui suit, nous présentons une liste non exhaustive de ces technologies utilisées dans l'IoT. Ainsi, LTE (Long Term Evolution) est une évolution des normes de téléphonie mobile qui peut être adaptée et utilisée dans l'IoT en assurant une bonne couverture étendue arrivant à 5 Km pour un grand nombre d'objets (20000 objets par cellule) et avec un débit théorique entre 50 et 100 Mbit/s [52] [53]. A ce titre, différentes variantes de LTE existent telle que LTE-M qui permet d'économiser de l'énergie, d'optimiser les interférences et de se conformer ainsi aux caractéristiques des objets IoT [52]. D'autre part, LTE-Advanced, connu par l'appellation 4G, permet d'offrir des réseaux plus rapides et plus efficaces avec de meilleures bandes passantes et une meilleure intensité du signal dans un environnement urbain dense. Par conséquent, la 4G assure une fiabilité accrue en termes de perte de paquets et une meilleure sécurité pour une

utilisation dans un environnement IoT [52]. Enfin, NB-IoT (Narrowband IoT) est une technologie radio offrant une consommation énergétique minimale pour assurer une longue durée de vie des batteries et s'adapter ainsi aux contraintes des objets IoT. Elle supporte un grand nombre d'objets par cellule (55000 objets par cellule) avec un faible débit, de l'ordre de 200 Kbit/s [52] [54]. La 5G est la 5ème génération de réseaux mobiles, une évolution significative des réseaux 4G LTE Advanced actuels. La 5G est conçue pour répondre à la forte croissance des flux de données échangés sur les réseaux. La 5G fonctionnera initialement avec les réseaux 4G existants avant de devenir des réseaux entièrement autonomes dans les versions ultérieures. Il existe trois grandes catégories de cas d'utilisation pour la 5G. La première concerne les communications machine à machine (*M2M*) qui consistent à connecter des milliards de périphériques sans intervention humaine. La deuxième concerne des communications ultra-fiables à faible temps de latence. La troisième catégorie concerne l'offre de débits de transmission de données considérablement plus importants. Avec la 5G le débit de liaison descendante peut atteindre 20 Gbit/s et celui de la liaison montante 10 Gbit/s avec une latence entre 10 et 20 ms [55].

2.5.2. Technologies sans fil non cellulaires

Des technologies sans fil non cellulaires peuvent être utilisées pour la communication entre les objets dans l'IoT. Ces technologies opèrent à différents niveaux du modèle OSI (Open Systems Interconnection) comme le montre la Figure 2.7. Nous présentons dans cette section une liste non exhaustive des différentes technologies sans fil non cellulaires utilisées dans l'environnement IoT. A ce titre, LoRa (Long Range) [56] est une technologie sans fil offrant une longue durée de vie des batteries et une bonne couverture pouvant atteindre les 15 Km. Cette technologie est utilisée dans l'IoT pour des applications à débit réduit entre 0.3 Kbit/s et 50 Kbit/s. La technologie LoRa se limite à la spécification de la couche 1 du modèle OSI tandis que LoRaWAN étend la spécification à la couche 2 du modèle OSI. Ainsi, LoRaWAN définit le protocole de communication et l'architecture réseau correspondante [56]. D'autre part, IEEE 802.15.4 est un standard de communication adapté aux réseaux à faible débit et à faible portée. Ce standard définit les caractéristiques des couches 1 et 2 du modèle OSI et peut être utilisé comme base par d'autres protocoles et technologies de l'IoT [57]. A titre d'exemple, ZigBee [58] et 6LowPAN [59] sont basées sur les spécifications des couches 1 et 2 du standard IEEE 802.15.4. Ce dernier spécifie les fréquences, puissances, modulation et technique de contrôle d'accès au medium partagée avec un débit pouvant atteindre au maximum 250 Kbit/s. Dans ce groupe de technologies sans fil utilisées dans l'IoT, nous pouvons citer Z-Wave [60] qui est un standard de communication permettant une faible consommation énergétique. Cette technologie assure des débits entre 9.6 Kbit/s et 40 Kbit/s pour une couverture de l'ordre de 30 mètres. De plus, BLE (Bluetooth Low Energy) [61] est une variante du standard Bluetooth adaptée pour les capteurs ayant peu de ressources. Cette technologie assure une couverture allant jusqu'à 60 m avec un débit pouvant atteindre les 305 Kbit/s.

2.5.3. Etude comparative des technologies de communication de l'IoT

Nous proposons à travers le Tableau 2.1 une étude comparative concernant les différents protocoles et technologies de communication sans fil cellulaire et non cellulaire qui peuvent être utilisés dans l'Internet des objets. Nous avons pris en considération différents critères pour l'établissement de cette étude comparative : la consommation énergétique, la topologie réseau, la modulation, le débit, la portée, la bande de fréquence, le chiffrement utilisé et l'utilisation dans des projets de recherche portant sur l'IoT. Nous remarquons à travers notre étude comparative que les standards IEEE 802.15.4 et ZigBee sont les plus utilisés dans les projets de recherche européens. De plus, ZigBee se base sur IEEE 802.15.4 pour la spécification de ses couches basses. Le standard IEEE 802.15.4 assure des communications à faible débit adaptées à une large gamme de service IoT (services médicaux, services véhiculaires, etc.).

Tableau 2.1 : Tableau comparatif des différentes technologies de communication dans l'IoT

	Consommation énergétique	Topologie réseau	Modulation	Débit	Portée	Bande de fréquence	Chiffrement	Utilisation dans les projets
LoRaWAN	*Faible*	*Etoile*	*CHIRP*	*0.3 Kbps, 50 Kbps*	*2 Km – 5 Km*	*868 Mhz*	*AES - 128*	*FIESTA IoT*
DASH7	*Faible*	*Etoile, P2P, Cluster*	*GFSK*	*9,6 Kbps, 55.55 Kbps, 166.7 Kbps*	*5 Km*	*433 MHz, 868 MHz, 915 MHz*	*AES -128*	---
IEEE 802.15.4	*Faible*	*Etoile, P2P*	*BPSK, O-QPSK*	*20 Kbps, 40 Kbps, 250 Kbps*	*Faible portée (Dizaines, centaines de mètres)*	*868 MHz, 915 MHz, 2,4 GHz*	*AES - 128*	*SymbIoTe, FiT IoT Lab FIESTA IoT*
UWB	*Faible*	*Etoile*	*PPM, PAM, OOK*	*200 Mbps*	*10 m*	*3.1 GHz -10.6 GHz*	---	---
ZigBee	*Faible (~ 40 mA) **	*Etoile, P2P, Cluster*	*BPSK, ASK, O-QPSK*	*250 Kbps*	*10 – 100 m*	*868 MHz, 915 MHz, 2,4 GHz*	*AES - 128*	*AGILE, SymbIoTe, VICINITY*
BLE	*Faible (~ 12.5 mA) **	*Etoile*	*GFSK*	*260Kbps, 305 Kbps **	*60 m*	*2,4 GHz, 2,5 GHz*	*AES-CCM*	*AGILE, BigIoT, IIC*
Z-Wave	*Faible (~ 2.5 mA) **	*Mesh*	*GFSK*	*9,6 Kbps, 40 Kbps, 100 Kbps*	*30 m*	*868 MHz*	*AES -128*	*AGILE*
LTE	---	*Etoile*	*QPSK*	*50 Mbps, 100 Mbps, (Théorie)*	*2.5 Km – 5 Km*	*450 MHz à 3,8 GHz*	---	---
LTE Advanced	---	*Etoile*	*QPSK*	*75 Mbps, 1Gbps*	*2,5 Km -5Km*	*800 MHz, 1800 MHz, 2,6 GHz*	----	*FIESTA IoT*
NB-IoT	*Faible (10 ans)*	*Etoile*	*QPSK*	*300 bps, 200 Kbps*	*10 Km -15 Km*	*800 MHz, 1800 MHz, 2,6 GHz*	---	---

2.6. Conclusion

Nous avons présenté dans ce chapitre des généralités concernant l'environnement IoT. Ainsi, nous avons décrit l'évolution de l'IoT depuis sa proposition au début des années 2000. De plus, nous avons étudié différentes définitions et architectures de ce nouveau paradigme. Ces architectures en

couches ont été proposées par différents organismes de standardisation (i.e., NIST, IEEE, etc.) et projets de recherche (i.e., IoT-I, CASAGRAS, etc.). Nous avons présenté dans ce chapitre différents domaines d'application de l'IoT dans nos sociétés contemporaines ainsi que leur utilité pour l'amélioration de la qualité de vie humaine. Nous citons à ce titre, le domaine de la e-santé ou encore les villes intelligentes et les systèmes de transport intelligents. Finalement, nous avons comparé les différents protocoles de communications utilisés au sein de l'IoT pour assurer la connectivité entre les objets. Ces différents protocoles (i.e., Zigbee, LoRa, NB-IoT, IEEE 802.15.4, etc.) sont utilisés dans différentes scénarios selon les besoins des applications IoT. L'étude que nous avons menée dans ce chapitre nous a permis de maitriser les concepts et caractéristiques de ce nouveau type d'environnement afin de mieux cerner dans le chapitre suivant les challenges et les défis de recherche à relever dans l'Internet des Objets en termes de qualité de service, sécurité et gestion autonome.

Chapitre 3 – Qualité de Service, Sécurité et Gestion autonome dans l'Internet des objets

3.1. Introduction

Afin d'assurer un niveau de service adéquat dans un environnement Internet des objets, des mécanismes de Qualité de Service (*QoS*) et de sécurité doivent être implémentés au niveau des différentes couches de l'architecture IoT. Ces différents mécanismes permettront ensuite de fournir une *QoS* et une sécurité de bout en bout pour l'utilisateur des services IoT. Par conséquent, il est primordial d'étudier ces deux aspects dans un contexte Internet des objets afin de trouver les mécanismes permettant de les garantir et de les gérer d'une manière autonome dans cet environnement IoT. Dans ce sens, ce chapitre présente l'état de l'art relatif aux trois axes de recherche de la thèse: la qualité de service, la sécurité et la gestion autonome.

Ainsi, nous étudions dans la section 2 les motivations et les challenges pour la *QoS* dans l'IoT à travers la description des besoins de *QoS* au niveau de chaque couche de l'architecture IoT ainsi que les mécanismes permettant de les garantir. De plus, nous présentons à travers cette même section les travaux de recherche concernant les contrats de niveau de service à utiliser dans l'IoT. Ensuite, nous nous focalisons dans la section 3 sur les motivations et challenges ainsi que les besoins et mécanismes existants concernant la sécurité dans l'IoT en suivant une classification selon les services de sécurité (i.e., confidentialité, intégrité, authentification, etc.). D'autre part, nous analysons dans cette même section la protection de la vie privée et la notion de confiance dans l'IoT tout en mettant en avant les réglementations en cours. Enfin, nous décrivons dans la section 4 le concept de gestion autonome en présentant les motivations, les challenges, les objectif et travaux existants dans le contexte de l'Internet des objets.

3.2. Qualité de service dans un environnement Internet des objets

3.2.1. Motivations et challenges

L'Internet des objets consiste en une combinaison de technologies différentes et réseaux hétérogènes fournissant divers types de services. Par conséquent, plusieurs types de données et de flux avec des contraintes différentes voire contradictoires peuvent coexister dans un même environnement IoT. De ce constat est née la nécessité de l'intégration de la *QoS* pour satisfaire les exigences de chaque type de données [62]. La *QoS* devient un élément critique dans l'Internet des objets, étant donné que le grand nombre d'objets connectés entraine une augmentation considérable du volume de données dans ce type d'environnement. Cette augmentation peut créer une congestion affectant les performances du système IoT. Dans ce contexte, les mécanismes de *QoS* permettent

d'optimiser les performances en identifiant les trafics et en leur appliquant un traitement différencié. La gestion de la performance permet également de réduire les coûts et d'assurer une meilleure évolutivité dans l'environnement IoT [63].

Les modèles de *QoS* utilisés dans les réseaux informatiques traditionnels tels que *DiffServ* [64] et *IntServ* [65] sont difficiles à mettre en place dans l'Internet des objets. Ceci est dû non seulement aux contraintes en termes de mémoire et de capacité de calcul des objets de l'IoT, mais aussi à la grande envergure de l'IoT et le déploiement aléatoire de ces objets. Ainsi, de nouveaux modèles de *QoS* bien adaptés doivent être proposés pour offrir des services IoT avec des garanties de connectivité, de performance et de *QoS* d'une façon générale [66]. Les approches et modèles de *QoS* adaptés doivent être présents au niveau de chaque couche de l'architecture IoT. En effet, le non-respect d'un paramètre de *QoS* au niveau de n'importe quelle couche peut être ressenti par l'utilisateur final de l'application IoT et freiner l'utilisation de cette application et par conséquent, freiner l'utilisation de l'IoT. Ainsi, pour gérer les paramètres de *QoS* d'une manière optimale dans un environnement IoT, des approches et des architectures doivent être mises en place pour tenter de prévenir, signaler et agir dans le but de respecter le niveau de service requis par les utilisateurs. De plus, une communication entre les couches de l'IoT est nécessaire pour la garantie de *QoS* dans cet environnement. Par exemple, un délai important au niveau de la couche inférieure de l'architecture IoT (couche dispositifs) doit être signalé aux couches supérieures afin de prendre des mesures adéquates pour compenser cette dégradation. Ainsi, les flux subissant cette dégradation doivent être priorisés dans les autres couches afin d'offrir le niveau de services requis.

L'importance de la garantie de *QoS* dans l'Internet des objets a été mise en avant par différents organismes internationaux. A ce titre, l'UIT-T a étudié l'importance de l'intégration de la *QoS* dans l'environnement IoT à travers la recommandation Y.2066 [12] en mentionnant le fait que prioriser les services IoT est une exigence primordiale. En outre, Y.2066 indique que la fonctionnalité de priorisation permet de satisfaire les différentes exigences de service de divers groupes d'utilisateurs IoT. En d'autres termes, les services différenciés devraient être pris en charge afin que l'environnement IoT puisse fournir différents accords de niveau de service (*SLA*). D'autre part, LinkLabs, une société américaine développant des technologies pour les réseaux informatiques, indique que l'intégration de la *QoS* dans l'IoT permet de mieux gérer les capacités et les ressources dans ce type d'environnement, et fournir ainsi une infrastructure fiable et optimisée pour la connectivité des objets de l'IoT. Selon LinkLabs, les mécanismes de *QoS* permettront de gérer les délais, la gigue, la bande passante et la perte de paquets en classifiant le trafic et en enregistrant les limites des canaux de communication [67]. Ainsi, il serait possible d'offrir des services IoT sûrs et prévisibles.

Nous décrivons dans la section suivante les besoins en termes de garantie de qualité de service ainsi que les réponses apportées dans les différentes couches de l'architecture de l'Internet des objets.

3.2.2. Garantie de QoS dans l'IoT

3.2.2.1. Couche dispositifs de l'architecture IoT

3.2.2.1.1. Besoins de QoS

La couche dispositifs de l'architecture en couches de l'IoT comprend les objets connectés et les passerelles qui gèrent ces derniers. Ainsi, la *QoS* à ce niveau doit répondre à des besoins correspondant aux objets et aux passerelles.

Un besoin essentiel pour assurer une qualité de service au niveau de la couche dispositifs est la différenciation et la priorisation des flux. L'Internet des objets regroupe un ensemble varié d'applications différentes et appartenant à divers domaines. Par conséquent, les données générées par ces différentes applications n'ont pas la même priorité. Ceci est à l'origine du besoin de classifier les différents flux suivant leur criticité et d'appliquer une certaine priorisation à travers différents mécanismes de *QoS* adaptés. Ainsi, lors de l'accès au canal partagé, les objets remontant des informations critiques auront une priorité plus importante que les objets remontant des informations non critiques. En se basant sur la même notion d'hétérogénéité des applications, les mécanismes de *QoS* appliqués dans le cadre de l'IoT doivent être en adéquation avec les caractéristiques de ces différentes applications. Dans ce contexte, il est important de classifier les applications de l'IoT suivant des critères spécifiques afin de proposer des mécanismes de *QoS* adaptés à un ensemble homogène d'applications. Par conséquent, chaque ensemble d'applications IoT disposera des mécanismes de *QoS* bien adaptés à leurs exigences.

La gestion de la performance d'un système IoT, à travers l'optimisation de l'utilisation des ressources pour offrir les meilleures performances, est un mécanisme important à proposer pour assurer une *QoS* au niveau de la couche dispositifs. Afin de répondre aux besoins des clients des applications IoT et offrir un service respectant les exigences correspondantes, une surveillance de la performance de l'environnement IoT doit être effectuée. Cette surveillance consiste en une collecte d'informations précises sur le système. Il est donc important de bien choisir les paramètres permettant de mesurer les performances relatives à l'environnement IoT. Ce choix permet de spécifier des paramètres propres à l'Internet des objets qui doivent être quantifiés par des métriques adaptées. De plus, au niveau de cette couche dispositifs, la minimisation de la consommation énergétique permettant une extension de la durée de vie des systèmes IoT est un paramètre de *QoS* à prendre en considération dans ce type d'environnement. Afin d'assurer un meilleur niveau de service, la durée de vie du système IoT doit être étendue pour pouvoir répondre aux attentes des utilisateurs et ne pas augmenter les coûts opérationnels des infrastructures dûs aux changements des sources d'alimentation électriques des composants du système.

Enfin, comme décrit dans le chapitre 2, différents protocoles et technologies de communication sont utilisés pour l'échange d'informations entre objets et passerelles au sein de la

couche dispositifs de l'architecture IoT. Par conséquent, les mécanismes de *QoS* relatifs à la couche dispositifs dépendent aussi de ces technologies. Certaines de ces technologies ont des bases communes ce qui permet de généraliser certains mécanismes de *QoS* utilisés au niveau de cette couche de l'IoT.

3.2.2.1.2. Projets et travaux de recherche

Différents projets et travaux de recherche ont proposé l'intégration de la *QoS* au niveau de la couche dispositifs de l'architecture IoT grâce à des mécanismes adaptés à ce type d'environnement. Nous étudions dans ce qui suit certains de ces projets et travaux de recherche.

OpenIoT, un projet européen financé par FP7 (Décembre 2011 – Décembre 2014), a spécifié différents paramètres de *QoS* à garantir dans l'Internet des objets [68]. Ces paramètres sont variés et peuvent être classés en plusieurs catégories. D'une part, le projet identifie les paramètres généraux comme la qualité des capteurs pour déterminer la précision et la sensibilité des mesures mais aussi la consommation énergétique. De plus, OpenIoT prend en considération le volume de données produit par un capteur physique, la couverture du réseau des objets et la confiance corrélée à la qualité d'un capteur. D'autre part, différents paramètres de *QoS* concernant le réseau d'objets de la couche dispositifs, comme la durée de vie, le délai et la gigue, sont étudiés dans le cadre de ce projet. Ainsi, pour chaque paramètre de *QoS* spécifié dans le cadre du projet OpenIoT, différentes métriques sont prises en compte pour le quantifier. Par exemple, le paramètre relatif à la consommation énergétique est quantifié grâce à la métrique niveau de batterie. Enfin, ce projet permet de gérer la qualité de service en utilisant un gestionnaire appelé « QoS Manager » qui permet la collecte des informations relatives aux différents paramètres de *QoS*. Les informations collectées vont permettre à ce gestionnaire de prendre des décisions pour garantir et améliorer la qualité de service au sein de la couche dispositifs de l'architecture IoT [68].

QUASIMODO [69], un projet de recherche national français (Mars 2011 - Décembre 2014), a proposé différents algorithmes et méthodes permettant de fournir une prise en charge de bout en bout et auto-adaptative de la qualité de service pour les applications temps réel. De plus, ce projet considère l'efficacité énergétique dans les réseaux de capteurs sans fil (*WSN* : Wireless Sensor Networks) et les systèmes IoT. Dans le cadre du projet QUASIMODO, un protocole de routage multi-sauts et prenant en compte multiples contraintes (i.e., énergie, fiabilité de la liaison, délai) a été spécifié pour une utilisation dans un environnement IoT. Ce protocole (Operator Calculus - based routing protocol) a été implémenté avec succès en utilisant la plate-forme Contiki / TelosB.

Plusieurs projets de recherche universitaires ont porté sur la garantie du niveau de service dans un environnement IoT. Ainsi, les auteurs dans [70] proposent de classifier les différentes applications de l'IoT selon 3 modèles de services (i.e., Open Service Model, Supple Service Model, Complete Service Model). Par la suite, ils font correspondre à chaque modèle une topologie physique pour l'implémentation des capteurs. Dans ce cadre, « Open Service Model » correspond aux applications

interactives, non temps réel et non critiques, « Supple Service Model » correspond aux applications interactives, temps réel souples (Soft Real Time) et critiques alors que « Complete Service Model » correspond aux applications interactives, temps réel dures (Hard Real Time) et critiques. Ce travail de recherche a fait correspondre les modèles de service proposés à des topologies physiques au niveau de la couche dispositifs de l'IoT afin de répondre aux besoins de chaque modèle. En effet, les applications appartenant au modèle « Complete » doivent être fournies à travers un réseau de capteurs ayant une topologie physique en étoile pour obtenir de meilleurs délais. D'autre part, les applications appartenant au modèle « Open » doivent être fournies à travers un réseau de capteurs ayant une topologie physique aléatoire pour une meilleure consommation énergétique.

D'autres travaux de recherche se sont intéressés à la priorisation de l'accès au canal de communication partagé entre les objets IoT pour garantir une différenciation de trafic lors de l'envoi des données aux passerelles. Le travail de recherche mené dans [71] spécifie différentes files d'attente et un ordonnanceur dans un objet afin de privilégier le traitement des flux prioritaires. De plus, certains travaux de recherche ont tenté d'adapter les algorithmes d'accès au canal *Slotted CSMA/CA* (Slotted Carrier Sense Multiple Access / Collision Avoidance) pour assurer une garantie de qualité de service au niveau de la couche dispositifs. A ce titre, le travail de recherche mené dans [62] a adapté le standard IEEE 802.15.4 et en particulier l'algorithme *Slotted CSMA/CA* en intégrant un mécanisme de qualité de service. La nouvelle méthode d'accès qui en résulte, appelée *SDA* (Service Differentiated and Adaptive)-*CSMA/CA*, repose sur la spécification de nouvelles valeurs pour les variables de l'algorithme *CSMA/CA*. Ainsi, les nouvelles valeurs de Contention Window (*CW*) et Backoff Exponent (*BE*) (minimal, maximal) permettent de créer différentes priorités en fonction des types de trafic à acheminer dans la couche dispositifs. Ainsi, trois niveaux de priorité sont pris en compte avec *SDA-CSMA/CA*. Dans ce contexte, des fenêtres CW moins importantes et des intervalles *BE* (*BE* minimal et *BE* maximal) moins larges sont affectés aux trafics de priorité élevée. Par conséquent, les messages importants ont plus de chance d'accéder au canal avec un temps d'attente réduit.

D'autre part, différents travaux de recherche se sont focalisés sur la minimisation de la consommation énergétique dans l'environnement IoT. A ce titre, le travail présenté dans [72] propose une méthode permettant d'étudier l'optimisation de la consommation d'énergie des composants du réseau de capteurs (*WSN*) et par conséquent l'optimisation de la durée de vie système IoT. Les auteurs proposent une nouvelle approche en combinant l'approche logique floue et l'algorithme *A-star*. Cette approche permet de sélectionner une route optimale entre la source et la destination en tenant compte de différents critères (i.e., énergie restante dans les objets, nombre de sauts et charge du trafic) et en les équilibrant pour allonger la durée de vie d'un système IoT utilisant un réseau de capteurs. Dans le cadre de ce travail, l'algorithme *A-Star* permet de trouver un chemin à faible coût et la logique floue permet de modéliser la logique humaine pour la prise de décision suivant différents critères. En utilisant cette approche, les auteurs ont pu avoir une meilleure durée de vie du système (i.e., 1.89 fois plus importante qu'avec la technique logique floue seul et 2.79 fois plus importante qu'avec la

technique *A-star* seul). Un autre travail de recherche [73] suggère une modification de la méthode *EASR* (Energy Aware Sink Relocation) visant à augmenter son efficacité en diminuant la consommation d'énergie des objets et en assurant une communication hautement sécurisée entre les objets à l'aide de la cryptographie à courbe elliptique (*ECC* : Elliptical Curve Cryptography). En se basant sur le principe que la consommation d'énergie est inversement proportionnelle à la distance entre l'objet et le coordinateur, les auteurs proposent une technique de recherche du chemin le plus court qui améliore l'efficacité énergétique ainsi que la durée de vie du système IoT. En effet, ils proposent de déplacer le coordinateur de la couche dispositifs lors d'une dégradation du niveau d'énergie en dessous d'un seuil donné afin de minimiser la consommation énergétique.

3.2.2.2. Couche réseau de l'architecture IoT

3.2.2.2.1. Besoins de QoS

La couche réseau de l'architecture IoT comprend toutes les fonctionnalités réseau tels que le routage, le transfert et la gestion des chemins (sélection et récupération du chemin) via une infrastructure à chemins multiples. Cette couche agit en tant qu'infrastructure réseau interconnectant le réseau d'objets de l'IoT à l'infrastructure cloud.

L'intégration de la *QoS* dans cette couche réseau permettra de traiter le grand nombre de requêtes et informations transitant de la couche dispositifs à la couche support de l'architecture IoT. Ce traitement doit être différencié. Il doit en d'autres termes prioriser les requêtes selon leur importance. De ce fait, les besoins de *QoS* pour la couche réseau de l'architecture IoT correspondent aux besoins de *QoS* d'une infrastructure réseau tout en adaptant ces besoins aux caractéristiques de l'environnement Internet des objets. Pour répondre aux besoins de *QoS* dans la couche réseau de l'architecture IoT, différents mécanismes peuvent être utiles comme la priorisation des requêtes et le choix des chemins sur la base d'un contexte de garantie de qualité de service. Dans ce contexte, nous étudions certains des projets et travaux de recherche qui ont contribué à l'intégration de la *QoS* dans la couche réseau de l'architecture IoT.

3.2.2.2.2. Projets et travaux de recherche

F-Interop Platform, un projet européen financé par H2020 (Novembre 2015- Novembre 2018), a développé des outils de test pour la *QoS* et la qualité d'expérience (*QoE* : Quality of Experience) qui se basent sur les solutions de type *SDN/NFV* (Software Defined Networking / Network Functions Virtualisation) pouvant être utilisés dans l'Internet des objets [74]. Ces technologies peuvent remplacer les technologies classiques pour assurer la *QoS* dans la couche réseau de l'architecture IoT. En effet, la technologie *SDN* consiste à séparer les fonctions de contrôle des fonctions d'acheminement de données. Le contrôleur *SDN* (composant centralisant la prise de décision et la gestion du réseau) peut imposer des politiques spécifiques sur les flux pour fournir différents niveaux de *QoS*. D'autre part, la technologie *NFV* permet de fournir des modules pour

l'application de ces différentes politiques, le déploiement des agents et pour simuler des conditions de réseau tout en collectant des mesures concernant les paramètres de *QoS* [74]. A travers la solution de F-Interop, la collecte des statistiques concernant les paramètres de *QoS* (Délai, Gigue, etc.) peut être assurée en utilisant une méthode passive (i.e., métrologie passive) ou une méthode active (i.e. métrologie active). La méthode passive s'appuie sur la collecte de statistiques de *QoS* à partir des commutateurs *SDN* sur lesquels des compteurs sont implémentés pour mesurer le débit et la perte de paquets par exemple. La méthode active s'appuie sur l'utilisation d'agents *NFV* permettant de mesurer les paramètres de *QoS* en temps réel. La collecte des informations porte sur les données concernant la perte de paquets, le délai et la gigue [74].

3.2.2.3. Couches support et application de l'architecture IoT

3.2.2.3.1. Besoins de QoS

La couche support de l'architecture IoT comprend les services cloud permettant d'avoir des capacités de calcul et de stockage. En outre, la couche application comprend les applications IoT permettant de traiter les données et d'effectuer des calculs spécifiques.

La garantie de *QoS* dans le cloud hébergeant les applications IoT est un enjeu stratégique important et constitue une discipline émergente présentant plusieurs défis de recherche. Ceci s'explique par le manque d'approches standardisées pour l'assurance de la qualité de service dans ce type d'environnement. De plus, il existe une diversité de contraintes et de paramètres de *QoS* spécifiques à chaque service cloud. En effet, les besoins de *QoS* dans le cloud sont variés et dépendent du service fourni par ce dernier. Ces services sont l'*IaaS* (Infrastructure as a Service), le *PaaS* (Platform as a Service) et le *SaaS* (Software as a Service). Le cloud peut aussi fournir des services réseaux appelés *NaaS* (Network as a Service) qui peuvent constituer l'infrastructure connectant, dans un environnement IoT, la couche dispositifs à la couche support offrant des services de type *IaaS*, *PaaS* ou *SaaS*.

Les besoins de *QoS* dans un environnement cloud et par conséquent dans la couche support et la couche application de l'architecture IoT peuvent concerner la disponibilité du service, la nécessité de fournir des mécanismes permettant de traiter de manière différenciée les différents types de flux, l'allocation des ressources en fonction des besoins des flux et la standardisation des *SLA* [75]. Nous décrivons dans ce qui suit certains projets et travaux de recherche qui se sont intéressés à ces défis de recherche.

3.2.2.3.2. Projets et travaux de recherche

Différents projets et travaux de recherche étudient l'intégration de la technologie cloud dans l'Internet des objets ainsi que la garantie de *QoS* dans l'environnement qui en résulte.

Le projet européen BigClouT, financé par H2020 (Juillet 2016 – Juin 2019), vise à utiliser les trois principaux outils technologiques actuels, à savoir l'IoT, le cloud et le Big Data, dans le but d'accroître l'efficacité des infrastructures urbaines, économiques et naturelles partagées par une population croissante [76]. Un autre travail de recherche mené dans [77], a considéré que l'intégration du cloud dans l'environnement IoT permet de garantir un niveau de *QoS* aux applications grâce aux ressources importantes du cloud. En effet, le cloud est capable de gérer des données diversifiées et de provisionner les applications IoT avec des ressources d'une manière dynamique pour optimiser la performance tout en respectant un moindre coût. Dans ce contexte, les auteurs ont proposé différentes fonctions permettant de mesurer plusieurs attributs de *QoS* (i.e., temps de réponse, disponibilité, débit, etc.). D'autre part, l'étude menée dans [78] concerne l'environnement cloud mais aussi IoT et permet de mettre en évidence différents concepts concernant le maintien d'une bonne *QoS*. A ce titre, ils ont classifié les différents paramètres dont dépend la *QoS* selon les couches de l'IoT. De plus, ils ont étudié les différents problèmes rencontrés pour maintenir ces paramètres et les solutions possibles pour surmonter ces problèmes dans le cadre de l'intégration du cloud dans l'IoT. Ce travail spécifie différents paramètres à prendre en considération au niveau des couches support et application de l'architecture IoT comme la charge de trafic, le délai de service, le temps de service, la précision du service et la priorité de service. Selon ce travail, les paramètres de *QoS* dans l'environnement IoT doivent prendre en considération les caractéristiques liées aux applications IoT hébergées dans le cloud.

3.2.2.4. Approche transversale dans l'IoT

3.2.2.4.1. Besoins de QoS

Après avoir définit les différents besoins de *QoS* propres à chaque couche de l'architecture IoT ainsi que les mécanismes pouvant apporter cette qualité de service pour chacune de ces couches, il est nécessaire de spécifier les besoins et les mécanismes assurant une *QoS* de bout en bout à travers les différentes couches de l'architecture IoT d'une façon globale et transversale (cross-layer). Cette *QoS* de bout en bout permet au client final de l'environnement IoT d'apprécier le niveau de service garanti. En effet, le client (utilisateur de l'application IoT) perçoit le niveau global de *QoS* de bout en bout et ne distingue pas la déclinaison de cette *QoS* en plusieurs niveaux selon les différentes couches de l'architecture IoT.

Différents projets et travaux de recherche prennent en charge l'étude de la *QoS* de bout en bout dans l'Internet des objets et spécifient les différent besoins relatifs à ce type d'environnement. Premièrement, l'environnement IoT doit être en mesure de prioriser certains flux et certains services en fonction de leur importance. Ainsi, les utilisateurs de l'IoT ayant payé pour des services privilégiés doivent en bénéficier. De plus, le déploiement de l'Internet des objets, en partant de la couche dispositifs jusqu'à l'hébergement de l'application au niveau de la couche support, doit être effectué d'une manière optimale pour prendre en considération les capacités demandés par les différents composants de l'architecture. A ce titre, les passerelles doivent être capables de traiter le volume de

données remontées par les objets alors que le service cloud doit être en mesure de fournir les capacités de calcul et de stockage nécessaires. De même, l'ensemble des composants de l'environnement IoT doit être capable d'effectuer des auto-adaptations en temps réel pour respecter les besoins des différents flux des utilisateurs. Ensuite, les besoins de *QoS* au niveau des différentes couches de l'architecture IoT doivent être traduits d'une manière compréhensible au niveau des autres couches. Cette traduction permet aux différentes couches de l'IoT de communiquer entre elles afin de respecter les exigences globales en termes de *QoS*. Par conséquent, la garantie d'une qualité de service de bout en bout dans l'Internet des objets nécessite, non seulement la décomposition en premier lieu des besoins de *QoS* suivant les différentes couches de l'architecture IoT, mais aussi la spécification des mécanismes à appliquer d'une façon complémentaire entre ces couches.

3.2.2.4.2. Projets et travaux de recherche

Différents projets et travaux de recherche ont pour objectif d'étudier la garantie de *QoS* dans l'environnement IoT en se basant sur une approche transversale impliquant les différentes couches de l'architecture IoT. Nous présentons dans ce qui suit une liste non exhaustive des propositions émanant de ces études.

SymbIoTe, un projet européen financé par H2020 (Janvier 2016 - Janvier 2019) [79], a comme but de présenter un système offrant des services IoT à condition que leur disponibilité et leur qualité de service puissent être garanties par l'infrastructure sous-jacente. De plus, le système IoT correspondant doit offrir aux utilisateurs privilégiés une priorité plus importante que celle des utilisateurs de base lors d'une tentative d'accès simultané à certains services IoT. Cette priorité doit être assurée au niveau de toutes les couches de l'architecture IoT pour garantir une *QoS* de bout en bout. Par ailleurs, le système proposé doit permettre le contrôle de l'accès aux services IoT en accord avec la législation locale et mondiale (la sécurité de l'accès aux données critiques est une condition nécessaire demandée par plusieurs gouvernements et organisations internationales), la charge actuelle, les exigences de sécurité, etc. Ce contrôle d'accès doit être assuré au niveau de toutes les couches de l'architecture IoT [79].

Plusieurs travaux de recherche s'intéressent à l'approche transversale pour la garantie de *QoS* dans l'environnement IoT. Ainsi, le travail de recherche présenté dans [80] se base sur le principe que deux couches adjacentes de l'architecture IoT doivent communiquer entre elles pour assurer la qualité de service. En effet, la couche supérieure effectue une requête de *QoS* vers la couche directement inférieure et cette dernière traduit la demande en un nombre de paramètres *QoS* à garantir au niveau local. Cette communication inter-couche se fait à l'aide d'interfaces dédiées avec des paramètres bien spécifiques. D'autre part, la *QoS* au niveau de la couche la plus haute de l'architecture IoT (i.e. support et application) est perçue directement par les clients. Elle se base sur des paramètres comme l'heure de service, la précision, la charge et la priorité, etc. De plus, la *QoS* au niveau de la couche réseau de l'architecture IoT dépend du type de réseau utilisé. Elle se base sur des paramètres comme la bande passante, le délai, la gigue et le taux de perte de paquets, etc. Enfin, la *QoS* au niveau

de la couche dispositifs se base sur les paramètres d'échantillonnage, de couverture, de synchronisation et de mobilité.

Nous spécifions dans cette thèse un mécanisme permettant de fournir la qualité de service au niveau de la couche basse de l'architecture IoT grâce à une différenciation des flux (cf. chapitre 5).

3.2.3. Contrats du niveau de service

Après avoir présenté différents travaux de recherche concernant la garantie de *QoS* au niveau des différentes couches de l'architecture IoT, nous présentons dans cette section les travaux de recherche concernant les contrats de niveau de service permettant de définir les besoins de *QoS* suivant des critères et des paramètres bien définis.

Le contrat du niveau de service *SLA* est un document définissant le niveau de service attendu par un client concernant un service fourni par un fournisseur. Autrement dit, il s'agit de clauses définissant les objectifs précis attendus et le niveau de service que souhaite obtenir un client de la part du prestataire. Ce contrat fixe les responsabilités et les conséquences des violations. De plus, le niveau de service garanti par ce contrat comprend généralement des objectifs (*SLO* : Service Level Objective) concernant non seulement la qualité de service mais aussi la sécurité. Les *SLA* ont été utilisé pour définir les relations entre les fournisseurs de services et les clients pour différents services informatiques dans les réseaux *IP*, les environnements cloud, etc. Nous pouvons aussi utiliser ces contrats dans le cadre de l'Internet des objets lors de l'offre de services IoT. Ces contrats permettent de traduire les attentes du client en des paramètres mesurables facilitant ainsi la surveillance et la gestion du niveau de service attendue [81] [82]. Les contrats de niveau de service comportent différentes parties essentielles. A ce titre, nous citons l'identification des parties signataires du contrat, les paramètres de performances, les pénalités appliquées lors des violations ainsi que la tarification. Les contrats de niveau de service doivent prendre en considération les besoins et les exigences de toutes les couches de l'architecture IoT. Nous étudions dans ce qui suit les *SLA* garantissant un niveau de service basé sur la seule composante relative à la *QoS*.

Différents projets et travaux de recherche ont spécifié divers contrats de niveau de service adaptés à différents environnements informatiques en se focalisant sur la partie qualité de service. Ainsi, le travail effectué dans [83] propose une architecture permettant de conclure des *SLA* dans le domaine des réseaux de capteurs (*WSN*) (voir Figure 3.1). Cette architecture comporte trois niveaux (i.e., niveau de négociation – Human decisions, niveau d'admission – Offline decisions, niveau d'exécution – Online process). Le premier niveau permet au client de décrire les besoins et les caractéristiques des applications (i.e., type de trafic, criticité, etc.) en les négociant avec le fournisseur de service. Ces caractéristiques et besoins sont décrits à travers différents paramètres de performance dans les objectives du niveau de service (*SLO*). Le deuxième niveau permet d'évaluer les besoins et les exigences du client en termes de ressources nécessaires en comparant les ressources libres de l'infrastructure et les ressources exigées par le niveau de service demandé. En effet, le fournisseur

doit être sûr de pouvoir offrir les ressources nécessaires pour le bon fonctionnement de l'application. A ce titre, le composant « SLA Admitter » de l'architecture proposée (voir Figure 3.1) prend une décision relative à l'acceptation ou le refus du *SLA* suite à l'évaluation des ressources existantes. Si le *SLA* est refusé, le processus recommence à partir du premier niveau, sinon le troisième niveau (i.e., niveau d'exécution) est sollicité pour mettre à jour les ressources utilisées par la nouvelle application. De plus, ce troisième niveau permet d'appliquer les mécanismes nécessaires pour l'adoption et la garantie du *SLA* dans l'infrastructure correspondante. Ce travail de recherche a aussi spécifié les différentes mesures à effectuer pour surveiller le respect des *SLA* en utilisant des paramètres comme la charge de trafic, l'énergie résiduelle, le ratio de livraison de paquet, etc.

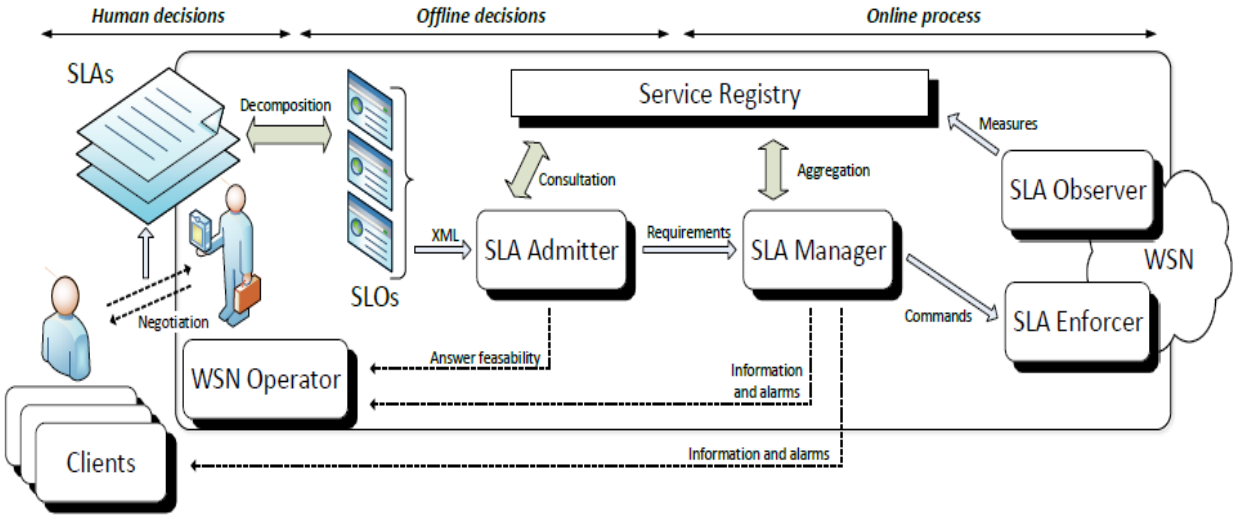

Figure 3.1 : Architecture pour l'établissement des SLA dans un environnement WSN [83]

D'autre part, les auteurs dans [84] proposent un modèle conceptuel incluant les entités clés d'un contrat de niveau de service de bout en bout ainsi que leurs interactions dans le cadre des environnements IoT. Des *SLO* sont également pris en compte dans le cadre de ce travail pour exprimer les contraintes de *QoS*. De plus, les auteurs proposent un outil de spécification et de composition de SLA qui peut être utilisé comme modèle pour générer des *SLA* avec un format lisible par les machines. L'outil permet de spécifier les *SLO* au niveau de l'application, tels que le niveau de disponibilité requis, le temps de réponse de l'application, etc. Ensuite, l'utilisateur de l'outil sélectionne les activités requises par l'application (i.e., capture de données, analyse de données, stockage de données, etc.) et mappe chaque activité aux ressources correspondantes de l'infrastructure. Cet outil permet alors de créer le *SLA* au format *JSON* (JavaScript Object Notation) afin d'être interprétable par les machines. Le travail de recherche décrit dans [85] présente une boite à outils qui permet la spécification des *SLO* d'une application IoT, mais aussi le mappage de chacune des tâches des applications aux ressources logicielles et matérielles requises ainsi que la création des *SLA* au format *JSON*. L'architecture de cette boite à outils repose sur une première couche qui comporte une interface de type *GUI* (Graphical User Interface) permettant à l'utilisateur de spécifier les exigences et les caractéristiques de l'application (i.e., type de l'application, temps de réponse du

service, les tâches de l'application et leur flux d'exécution, etc.). Une deuxième couche, à savoir la couche de programmation, permet de retrouver le fichier comprenant les *SLO* correspondant au type de l'application IoT et mappe les tâches des applications aux modules correspondants pour la configuration et le déploiement du service. Ainsi, la boite à outils permet la génération du ficher *SLA* au format *JSON* et fournit les moyens pour décrire les exigences de chaque couche de l'architecture IoT concernant l'offre d'un service IoT avec garantie de *QoS*.

Nous spécifions dans cette thèse un *SLA* de *QoS* adapté à un environnement IoT permettant de définir les besoins de *QoS* des différentes couches de l'architecture IoT en se basant sur plusieurs sous *SLA* concernant un service IoT (cf. chapitre 4). Le niveau de service comprend un autre volet que celui de la *QoS*. Il s'agit de la sécurité qui doit être prise en compte lors de l'offre des services IoT. Ainsi, nous présentons dans la section suivante les services de sécurité à prendre en considération dans un environnement IoT.

3.3. Sécurité et vie privée dans un environnement Internet des objets

3.3.1. Motivations et challenges

La sécurité des systèmes d'information constitue l'ensemble des moyens techniques, organisationnels, juridiques et humains nécessaires pour empêcher l'utilisation non-autorisée, le mauvais usage, la modification ou le détournement du système d'information. Actuellement, la sécurité est un enjeu majeur dans le monde de l'informatique et a comme but de maintenir la confiance des utilisateurs et la cohérence de l'ensemble du système d'information. Plusieurs normes concernent des notions relatives à la sécurité comme par exemple la recommandation X.800 de l'UIT-T [86] qui insiste sur le rôle des différents services de sécurité et leur applicabilité.

L'Internet des objets se caractérise par un environnement comportant des contraintes à plusieurs niveaux ce qui rend difficile l'adoption des mécanismes de sécurité conçus pour les systèmes traditionnels. En effet, un environnement de type IoT comprend des objets avec peu de ressources mémoires et des capacités de calcul limitées. Or, les techniques habituellement utilisées dans les réseaux traditionnels ont été conçues pour des systèmes disposant de microprocesseurs puissants et des capacités de stockage importantes [87]. Par conséquent, il est nécessaire d'adapter les techniques de sécurité existantes. En outre, le grand nombre d'objets dans un environnement IoT rend la tâche d'adaptation de certains algorithmes de sécurité existants difficile et lourde. A titre d'exemple, les méthodes et algorithmes d'identification et de contrôle d'accès des objets deviennent de plus en plus complexes avec le nombre d'objets de plus en plus important.

Avant qu'un périphérique ou un utilisateur puisse accéder aux services de l'IoT, l'authentification mutuelle et l'autorisation entre le périphérique/utilisateur et le système IoT doivent être exécutées conformément à des politiques de sécurité prédéfinies. Ces politiques de sécurité doivent être spécifiées avec précision afin d'intégrer d'une façon exhaustive tous les cas possibles

d'utilisation et doivent suivre des modèles standardisés pour répondre aux exigences de l'Internet des objets. Ainsi, il est important de standardiser des politiques de sécurité pour l'environnement Internet des objets. De plus, l'accès aux données ou aux services de l'IoT doit être entièrement transparent, traçable et reproductible. Par conséquent, un énorme volume de fichiers de traces est créé dans un environnement IoT étant donné le nombre important d'objets connectés. Ainsi, des mécanismes pour l'optimisation de la traçabilité doivent être conçus dans le contexte de l'Internet des objets. Dans un tel environnement IoT, une diversité de systèmes d'exploitation avec des architectures différentes sont disponibles pour les objets de l'IoT. Nous citons à ce titre Android Things de Google (anciennement connu sous le nom de Brillo) [88], LiteOS de Huawei [89] et Windows 10 IoT Core [90], etc. Cette diversité peut rendre encore plus difficile la normalisation des mécanismes et des mesures de sécurité.

En ce qui concerne la vie privée des utilisateurs, les données dans les systèmes IoT peuvent être collectées sans l'implication de ces derniers. Dans ce contexte, il est nécessaire de sécuriser cette remontée d'informations et de garantir le respect de la vie privée pendant la collecte, la transmission, l'agrégation, le stockage, l'extraction et le traitement des données. Afin de répondre à ces besoins, des mécanismes adaptés pour la confidentialité, l'authenticité et l'intégrité des données doivent être fournis au sein de l'Internet des objets tout en respectant les exigences de ce type d'environnement [11].

Différents organismes internationaux s'intéressent aux notions relatives à la sécurité et la vie privée dans l'Internet des objets, soit en proposant des mécanismes de sécurité adaptés, soit en proposant des méthodologies à appliquer à travers les différentes couches de l'architecture IoT. Ainsi, la recommandation Y.2060 de l'UIT-T [11] vise à sécuriser l'environnement IoT en commençant par une analyse des menaces propres à une application IoT. Ensuite, des mécanismes et des services de sécurité spécifiques seront pris en charge au niveau de chaque couche de l'architecture IoT pour assurer une sécurité globale au sein de cet environnement. Ainsi, au niveau de la couche application du modèle de référence IoT de l'UIT-T, différents services de sécurité sont pris en compte tels que l'autorisation, l'authentification, la confidentialité et l'intégrité des données applicatives, mais aussi la protection de la vie privée. Au niveau de la couche réseau de ce même modèle d'architecture IoT, les services de sécurité incluent l'autorisation, l'authentification, la confidentialité des données applicatives et des données de signalisation (configurations et commandes) et la protection de l'intégrité de la signalisation du réseau. Enfin, au niveau de la couche la plus basse de l'architecture IoT de l'UIT-T, à savoir la couche dispositifs, les principaux services et mécanismes à offrir pour garantir la sécurité sont l'authentification, l'autorisation, la validation de l'intégrité de l'appareil, le contrôle d'accès, la confidentialité des données et la protection de l'intégrité. Suivant la recommandation Y.2066 [12], plusieurs capacités de sécurité spécifiques sont à considérer dans l'environnement IoT : une capacité de communication sécurisée pour garantir la confidentialité et

l'intégrité des données durant la transmission ; une capacité de gestion des données sécurisée pour garantir la confidentialité et l'intégrité lors du stockage. De plus, cette même recommandationspécifie une capacité de provision de service sécurisé garantissant l'interdiction des services frauduleux et une capacité d'authentification et d'autorisation mutuelle entre les objets et les utilisateurs suivant des politiques prédéfinies pour garantir la sécurisation de l'accès à l'information. Ces capacités sont étroitement liées aux exigences spécifiques des applications IoT et dépendent du domaine d'application. La recommandation Y.2060 [11] insiste aussi sur la nécessité de la prise en charge des fonctions et mécanismes de sécurité par les passerelles IoT interconnectant les différents composants des différentes couches de l'architecture IoT spécifiée par l'UIT-T. Nous décrivons dans la section suivante les différents services de sécurité qui doivent être pris en considération dans un environnement Internet des objets.

3.3.2. Services de sécurité dans l'IoT

3.3.2.1. Identification et Authentification dans l'IoT

3.3.2.1.1. Définitions

L'identification consiste à établir l'identité de l'utilisateur d'un service IoT. Son principe repose sur l'utilisation d'un identifiant attribué individuellement à un utilisateur. L'authentification succède à l'identification et permet à l'utilisateur d'apporter la preuve de son identité. L'utilisateur utilise un authentifiant ou un code secret que lui seul connait. Néanmoins, l'authentification ne donne pas un droit d'accès, c'est le contrôle d'accès qui assure ce privilège si l'authentification a été réussie [86]. Les mécanismes d'identification et d'authentification peuvent fournir des avantages à l'environnement IoT. Ainsi, cet environnement, à travers ces mécanismes d'identification et d'authentification, intègre des dispositifs robustes capables de réduire les risques d'intrusion et d'éviter les violations [91].

En outre, les mécanismes d'identification et d'authentification traditionnels doivent être adaptés pour respecter les exigences de l'environnement IoT en termes d'évolutivité, du grand nombre d'entités, etc. Plusieurs organisations utilisent des certificats numériques basés sur l'infrastructure à clé publique (*PKI* : Public Key Infrastructure) pour les opérations d'identification et d'authentification des périphériques [92]. Toutefois, des adaptations doivent être effectuées pour considérer ce type de solutions dans l'IoT. Premièrement, les infrastructures *PKI* devraient être en mesure de soutenir efficacement le processus d'émission de certificats numériques en grande quantité et à grande vitesse. A ce titre, une infrastructure *PKI* basée sur le cloud permet d'offrir un moyen plus économique et plus réaliste pour atteindre l'échelle IoT. Ensuite, les certificats numériques ont une durée de vie limitée, ce qui signifie qu'ils ont une date d'expiration. Dans le cadre de l'environnement IoT, certains cas d'utilisation peuvent nécessiter des certificats de courte durée, tandis que de nombreux autres demandent des certificats de plus longue durée. Ainsi, un certificat de durée de vie plus longue est requis lorsqu'un périphérique nécessite une authentification à long terme. Par

conséquent, les responsables de projets IoT doivent déterminer avec soin la durée de vie requise des certificats numériques et en déterminer les avantages et les inconvénients. En revanche, les certificats, étant considérés comme des éléments critiques ayant un cycle de vie, doivent être gérés d'une manière efficace. En effet, la gestion manuelle de suivi des certificats n'est pas faisable dans un environnement IoT. Ainsi, l'infrastructure *PKI* devrait être associée à des fournisseurs de gestion de certificats ayant une plateforme évolutive. Ces plateformes devraient alors être capables de gérer les cas d'utilisation de l'IoT [92]. D'autres solutions et technologies peuvent aussi être utilisées pour l'authentification dans l'IoT tel que *SAML* (Security Assertion Markup Language). Ce dernier est un standard, développé par *OASIS*, définissant un protocole pour échanger des informations liées à la sécurité. *SAML*, basé sur le langage *XML*, propose une authentification unique sur le web. De plus, le standard *SAML* est modulaire grâce à des messages d'authentification qui peuvent être acheminés en utilisant plusieurs protocoles de communication (*HTTP* : Hypertext Transfer Protocol, *SOAP* : Simple Object Access Protocol, etc.) [93]. Enfin, ce standard est compatible avec d'autres standards pour le contrôle d'accès tel que *XACML* (eXtensible Access Control Markup Language) [94].

3.3.2.1.2. Projets et travaux de recherche

Les services de sécurité de type identification et authentification ont été traités dans différents projets et travaux de recherche. A ce titre, BUTLER (uBiquitous, secUre inTernet-of-things with Location and contExt-awaReness) [95], un projet de recherche européen financé par FP7 (Octobre 2011 – Octobre 2014), a étudié les mécanismes d'identification et d'authentification dans l'environnement IoT. Ce projet a proposé un mécanisme pour gérer la possession des objets par les différents utilisateurs. Ces derniers possèdent dans ce cas des objets connectés. Ainsi, un utilisateur (propriétaire d'un objet) dispose d'un compte auprès d'un gestionnaire de confiance (Trust Manager) implémenté sur un serveur d'autorisation. L'utilisateur se connecte au serveur d'autorisation et enregistre une nouvelle ressource (un nouvel objet connecté). La ressource doit comporter un identifiant unique (généralement une adresse *URL*) et des informations d'identification (resource security credentials). L'utilisateur doit alors configurer la ressource avec les informations d'identification et ainsi l'identité de l'utilisateur possédant l'objet pourra être vérifiée. De même, BUTLER propose un mécanisme permettant l'identification des objets auprès des passerelles à travers des certificats numériques gérés par le serveur d'autorisation [96].

Des travaux de recherche académiques ont aussi traité le sujet de l'identification et l'authentification dans l'IoT. Ils ont utilisé le cadre *SAML* pour authentifier les utilisateurs dans un environnement IoT. A titre d'exemple, le travail de recherche réalisé dans [97] décrit un cadre utilisant le standard *SAML* afin de permettre un contrôle d'accès fin et flexible aux objets IoT connectés avec une puissance de traitement et une mémoire très limitées. Ce travail se base sur trois entités: des ressources ou périphériques, un utilisateur souhaitant accéder à une ressource et un moteur d'autorisation (*AE* : Authorization Engine) qui évalue les règles et délivre les assertions d'autorisations à l'utilisateur. L'utilisateur envoie ces assertions à la ressource avec la demande

d'accès. Le moteur d'autorisation agit pour le compte d'un propriétaire de ressources qui a configuré les stratégies d'accès alors que l'utilisateur envoie les assertions à la ressource avec la demande d'accès.

3.3.2.2. Contrôle d'accès dans l'IoT

3.3.2.2.1. Définitions

Le contrôle d'accès permet de lutter contre l'utilisation non autorisée d'une ressource IoT. Afin de réaliser ce contrôle, une liste des entités autorisées à accéder à une ressource, avec leurs niveaux d'accès est définie conformément à une politique de sécurité. Ce service est offert pour mettre en place différents types d'accès aux ressources (lecture, écriture, modification, suppression d'information et exécution d'une tâche). Le contrôle d'accès est basé sur un ou plusieurs éléments (une liste de contrôle d'accès, une matrice de structure hiérarchique ou répartie, etc.) à travers des bases d'informations conservées par des centres d'autorisation ou par la ressource. Ces bases d'informations comprennent des informations d'authentification (mots de passe, étiquettes de sécurité, etc.) [86]. Dans le cadre du contrôle d'accès dans l'IoT, deux types d'entités sont importants : les détenteurs de données (les utilisateurs des services IoT) et les collecteurs de données (les objets IoT) qui envoient les données ou reçoivent les commandes. Ces deux types d'entités doivent être authentifiés mutuellement [98].

3.3.2.2.2. Projets et travaux de recherche

Plusieurs modèles existent dans la littérature pour assurer le service de contrôle d'accès. Les solutions de contrôle d'accès basées sur le modèle *RBAC* (Role Based Access Control) utilisent trois éléments pour prendre la décision de contrôle d'accès : l'utilisateur, le rôle et les permissions. L'autorisation est affectée selon le rôle de l'utilisateur. Plusieurs travaux de recherche qui étendent le modèle *RBAC* ont été proposés pour une utilisation dans l'environnement IoT et dans les réseaux de capteurs. Nous citons à ce titre le modèle *CA-RBAC* (Context-Aware – *RBAC*) [99] et le modèle *BTG-RBAC* (Break The Glass – RBAC) [100]. Le premier modèle, à savoir *CA-RBAC*, permet de fournir une connaissance du contexte et d'adapter les propriétés de sécurité en conséquence. Dans ce modèle, le processus de prise de décision est divisé en trois situations de contexte modulaires: situation critique, situation d'urgence et situation normale. Sur la base de ces situations, les privilèges d'accès aux données seront différents [99]. Le deuxième modèle, à savoir *BTG-RBAC*, permet de rassembler les informations nécessaires auprès des utilisateurs finaux afin de définir une stratégie de contrôle d'accès utilisable dans des situations d'urgence. En effet, la règle *BTG* spécifiée par *BTG-RBAC* permet aux utilisateurs d'avoir un accès urgent au système lorsqu'une authentification normale ne fonctionne pas ou ne fonctionne pas correctement [100]. D'autre part, *ABAC* (Attribute Based Access Control) est un autre modèle qui permet de prendre la décision de contrôle d'accès selon différents attributs de l'entité demandant un accès à une ressource. Les attributs peuvent appartenir à l'entité (i.e., identité) ou peuvent dépendre de l'environnement (i.e., temps de la demande) [101].

D'autres modèles de contrôle d'accès existent tels que le *capBAC* (capability Based Access Control) offrant des tickets ou des jetons aux entités autorisées, l'*orBAC* (organizational Based Access Control) utilisant des données de contexte (historiques, spatiales, temporelles, déclarées par l'utilisateur, etc.) [101].

Plusieurs projets de recherche européens ont étudié l'adaptation du service de sécurité de type contrôle d'accès pour l'environnement Internet des objets. Ainsi, ARMOUR [102], projet européen financé par H2020 (Février 2016 - Février 2018), a traité certains défis de sécurité et de confiance dans l'Internet des objets. Les travaux effectués dans le cadre de ce projet permettent de définir un ensemble de composants qui interagissent pour autoriser ou bloquer des requêtes de données sécurisées dans un environnement IoT. Pour cela, ARMOUR définit dans cet environnement plusieurs entités. En premier lieu, le point de décision stratégique ou *PDP* (Policy Decision Point) est un composant comportant les politiques d'accès qui permet d'autoriser ou non un dispositif IoT (capteur) à effectuer une action sur une ressource (serveur d'enregistrement de données) en évaluant les politiques de contrôle d'accès. Par exemple, une décision « PERM*IT* » provenant du *PDP* permet au gestionnaire de fonctionnalité ou Capability Manager (serveur communicant avec le *PDP*) de générer et d'envoyer un jeton au capteur pour la publication des données sur la plateforme IoT. Le serveur de publication de données (Publish/Subscribe Server) enregistre les données et permet alors la mise à jour et l'exécution de la requête de données si le jeton du capteur reçu par le Capability Manager permet cette action [103].

SMARTIE (Secure and SMArter ciTIes data management) [104], un autre projet européen financé par FP7 (Septembre 2013 - Décembre 2016), s'est focalisé sur le contrôle d'accès dans l'Internet des objets. Il a comme objectif de développer de nouveaux mécanismes qui établissent la confiance et la sécurité dans les différentes couches de l'IoT. Les résultats du projet indiquent que le contrôle d'accès basé sur les attributs (i.e., *ABAC*) est une solution adéquate pour une spécification de politiques fines de contrôle d'accès dans un environnement IoT. L'identité des utilisateurs de service IoT n'est plus limitée à un seul attribut mais repose sur plusieurs attributs (i.e., id utilisateur, rôle, etc.) qui composent cette identité. Pour ces raisons, le contrôle de type *ABAC* permet des améliorations substantielles pour l'autorisation et le contrôle d'accès dans l'IoT. Les solutions basées sur *ABAC* permettent de surmonter les inconvénients des solutions de contrôle d'accès centralisées. Chaque requête nécessite deux étapes : une vérification d'autorisation (contrôle d'identité et authentification) et par la suite une décision de contrôle d'accès (autorisation ou prohibition). Pour chaque demande d'accès, une authentification de l'utilisateur du service IoT auprès de son domaine est effectuée et la requête d'autorisation d'accès de l'utilisateur est obtenue. La requête d'autorisation d'accès de l'utilisateur est signée par une autorité de confiance du domaine. Ainsi, l'utilisateur peut envoyer la requête aux dispositifs IoT qui vérifient la signature. Si la vérification de la signature est réalisée avec succès, les informations demandées sont renvoyées à l'utilisateur [104] [105] [106].

3.3.2.3. Confidentialité dans l'IoT

3.3.2.3.1. Définitions

Le service de confidentialité fournit une protection contre l'analyse du trafic par des entités non autorisées et contre la divulgation des flux de données. Pour assurer ce service de sécurité, le chiffrement des données est le mécanisme de sécurité le plus adapté. Le chiffrement peut être effectué avec un système symétrique (à clé secrète) ou asymétrique (à clé publique). Le système de chiffrement symétrique implique la connaissance de la clé secrète qui permet le chiffrement et le déchiffrement. Alors que pour le système de chiffrement asymétrique, la connaissance de la clé publique de chiffrement par toutes les entités n'implique pas la connaissance de la clé privée correspondante pour le déchiffrement. En plus des mécanismes de chiffrement, l'existence d'un mécanisme de gestion de clés est nécessaire pour l'échange de ces clés entre les entités communicantes [86].

Dans l'environnement IoT, plusieurs points sont à prendre en considération lors de la mise en place du service de confidentialité, notamment lors du processus d'échange de clés utilisé pour le chiffrement. A ce titre, l'extensibilité est une caractéristique importante étant donné le nombre élevé d'objets connectés. En effet, le nombre d'entités pouvant être impliquées dans le processus d'échange de clés peut être limité en utilisant les systèmes traditionnels alors que de nouvelles entités peuvent être impliquées après le premier échange de clés et de nouveaux objets peuvent s'intégrer à l'environnement IoT après l'initiation des services. D'autre part, l'évolutivité est une caractéristique importante à prendre en considération. En effet, suite à l'implication de nouvelles entités dans le processus d'échange de clé dans l'Internet des objets, le volume des données cryptographiques à stocker sur les objets devient de plus en plus important alors que les objets IoT ont des restrictions en termes de stockage et de capacité de traitement [107].

3.3.2.3.2. Projets et travaux de recherche

Un des challenges importants lors de la mise en œuvre d'un système de chiffrement sur un objet connecté dans un environnement IoT est la disponibilité de librairies logicielles adéquates et respectant les contraintes de ces objets en termes de capacité de mémoire, de capacité de calcul et de consommation énergétique. Dans ce contexte, certains travaux de recherche ont été effectués pour remédier à ce problème qui reste toujours un défi de recherche et demande des études plus avancées et plus adaptées aux besoins de l'Internet des objets afin de garantir ce service de sécurité d'une façon optimale. Nous citons comme exemple de librairies existantes qui peuvent être utilisées dans un environnement IoT, la librairie «AVR-Crypto-Lib» [108], fournissant des implémentations spéciales respectant les ressources limitées de microcontrôleurs. Cette librairie propose des fonctions de chiffrement à clés symétriques comme *AES* (Advanced Encryption Standard), *RC5* (Rivest Cipher 5), *RC6, DES* (Data Encryption Standard), etc. D'autre part, la librairie «Relic-Toolkit» [109] propose un éventail assez large d'algorithmes de chiffrement asymétrique comme *RSA* (Rivest Shamir–Adleman), Rabin crypto system, etc. « Relic-Toolkit » est utilisé dans le projet TinyPBC implémenté

sur le simulateur TOSSIM [110] utilisant le système d'exploitation TinyOS. Les librairies que nous venons de décrire assurent entre autres un service de confidentialité dans un environnement IoT afin de permettre des communications sécurisées, de sorte que l'accès non autorisé au contenu des données soit interdit et que ce contenu soit protégé pendant son transfert entre deux entités de l'environnement IoT. Divers algorithmes de chiffrement (Curupira [111], Katan [112], Katantan [112], RECTANGLE [113], TWINE [114], mCrypton [115]) ont été aussi proposés pour les environnements contraints comme celui de l'IoT. Certains de ces algorithmes ont été standardisés ou proposés par des organismes internationaux tels que Present [116], Simon [117], Speck [117], HIGHT [118], CLEFIA [116] et CAMELLIA [118]. Nous présentons dans le tableau 3.1 une étude comparative de ces différents algorithmes de chiffrement pour les environnements contraints suivant plusieurs critères comme la longueur de la clé, la taille du bloc, la métrique *GE* (Gateway Equivalent) et le nombre de cycle de l'algorithme. La métrique *GE* permet de quantifier la complexité des circuits électroniques numériques, indépendamment de la technologie de fabrication, nécessaire à l'implémentation physique des algorithmes.

Tableau 3.1 : Etude comparative des algorithmes de chiffrement pour les environnements contraints

	Curupira	Katan	Ktantan	Present	Simon	Speck	RECTANGLE	HIGHT	CELFIA	CAMELLIA	TWINE	XTEA
Longueur de la clé	96	80	80	80	128	128	80	128	128	128	80	128
	144				192	192			192		128	
	192			128	256	256	128		256			
Taille du bloc	96	32	32	64	128	128	64	64	128	128	64	64
		48	48									
		64	64									
GE	9450	802 à 1054	462 à 688	1075	1317	1396	1599	3048	4950	6511	1799	3490
				1391			2063				2235	
Cycles	10 à 33	254	254	32	68	32	25	32	18	18	36	64
					69	33			22	24		
					72	34			26			
Recommandations et spécifications				ISO 29192-2:2012	Proposé par NSA	Proposé par NSA		ISO 18033-3:2010	ISO 29192-2:2012 Recommandé par cryptec	ISO 18033-3:2010 Recommandé par cryptec		

Des projets européens de recherche se sont aussi focalisés sur la confidentialité des données dans l'Internet des objets. Ainsi, le projet SMARTIE [104] utilise *CP-ABE* (Ciphertext Policy Attribute Based Encryption), une technique permettant à l'utilisateur du service IoT de déchiffrer le message provenant des objets avec une clé secrète si les attributs de la politique correspondent aux attributs de la clé. *CP-ABE* permet de ne pas chiffrer les données individuellement mais en revanche de les chiffrer pour un groupe d'utilisateurs selon des politiques d'accès. Cette technique relie le contrôle d'accès et la confidentialité. Elle est utile lorsque les données d'un objet doivent être reçues par plusieurs utilisateurs du service IoT. Ainsi, le chiffrement sera effectué une seul fois [104] [105]. D'autre part, le projet européen BUTLER [95] s'est intéressé à la protection du canal de

communication dans l'IoT. Ce canal est vulnérable en raison de sa caractéristique sans fil et de la diffusion de l'information. BUTLER propose des améliorations aux normes de sécurité utilisées dans les technologies de communication de l'IoT. A ce titre, le projet propose un système de sécurité basé sur l'utilisation de clés symétriques afin de compléter et renforcer les fonctionnalités de sécurité fournies par la norme ZigBee. Cette dernière utilise deux clés obligatoires et une facultative. La clé maître (Master Key) et la clé réseau (Network Key) sont obligatoires alors que la clé de lien (Link Key) est facultative. La clé maître est utilisée lors de la phase d'initialisation et elle est implémentée dans les nœuds à travers un canal hors bande. La clé réseau assure la sécurité au niveau de la couche réseau. Elle est partagée par tous les nœuds et dérivée de la clé maître. La clé facultative de lien est déduite de la clé maître. Elle assure la sécurité du lien entre une paire d'objets au niveau de l'application. Dans ce contexte, le projet BUTLER a implémenté des mécanismes pour gérer le déploiement, la maintenance et la révocation de la clé maître. De plus, il propose d'implémenter dans le nœud, lors de sa fabrication, une clé symétrique supplémentaire appelée clé globale. Cette clé est utilisée par la couche *MAC* (Medium Access Control) et elle est partagée par tous les nœuds. La clé globale assure une sécurité au niveau des couches basses et elle est gérée par le standard IEEE 802.15.4 et non par ZigBee. Alors que la clé de réseau fournie par la norme ZigBee sera utilisée comme une clé de groupe (Group Key). La clé de groupe sera partagée entre les nœuds et gérée par la couche réseau de ZigBee. Elle permet d'adresser en toute sécurité un groupe de nœuds partageant une fonctionnalité commune. Par conséquent, une sécurité plus accrue grâce à l'utilisation d'une clé globale pour tous les nœuds et une clé de réseau par groupe de nœuds sera assurée dans le réseau d'objets communicant à travers ZigBee ainsi qu'une sécurité supplémentaire lors de l'utilisation de la clé facultative de lien au niveau de la couche application pour sécuriser l'échange entre la passerelle et un objet [96].

3.3.2.4. Intégrité dans l'IoT

3.3.2.4.1. Définitions

L'intégrité est un service de sécurité qui englobe deux grandes parties dans le contexte de l'Internet des objets : l'intégrité des données et l'intégrité des objets. L'intégrité des données vise à assurer que les données échangés dans un environnement IoT n'ont pas été modifiées ou détruites d'une manière non autorisée lors de leur acheminement. Elle est nécessaire pour fournir un service fiable et s'assurer que les informations collectées et les commandes reçues par les objets sont légitimes. La vérification de l'intégrité des données implique deux processus, l'un au niveau de l'émetteur et l'autre au niveau du récepteur. L'entité émettrice ajoute des informations de contrôle (code de contrôle par bloc comme le *BCC* (Block Check Character) ou valeur de contrôle cryptographique tel que le hachage) en fonction des données envoyées. L'entité réceptrice génère les mêmes informations de contrôle en se basant sur les données reçues et les compare à celles reçues pour déterminer si les données ont été modifiées pendant l'acheminement dans l'environnement IoT [86].

Le deuxième type d'intégrité dans le contexte de l'IoT est celui concernant les objets. Ainsi, l'intégrité des objets est nécessaire car les objets dans l'IoT peuvent être déployés dans des environnements non fiables et peuvent être attaqués physiquement afin de modifier par exemple le code en cours d'exécution sur ces objets. Ce deuxième service d'intégrité à assurer dans l'Internet des objets permet de détecter et d'empêcher les modifications apportées au système d'exploitation et à la configuration des objets. L'intégrité des objets permet aussi de verrouiller et d'éliminer le périphérique non conforme. Pour mettre en place ce type d'intégrité, une empreinte numérique (digital fingerprint) de l'objet en question est utilisée pour comparer les données existantes sur l'objet et les données qui doivent exister sur ce dernier.

3.3.2.4.2. Projets et travaux de recherche

Le service de sécurité de type intégrité avec ces deux volets portant sur les objets et les données dans un environnement IoT a été étudié dans différents projets de recherche. A ce titre, le projet SMARTIE prend en considération plusieurs architectures pour la mise en place du service d'intégrité dans un environnement IoT. En effet, il utilise l'architecture de mesure d'intégrité présente dans le noyau Linux [104] pour vérifier l'intégrité des objets. De plus, ce projet s'appuie sur les mécanismes de contrôle d'intégrité présents dans les cartes à puce tout en s'inspirant de l'architecture *IMA* (Integrity Measurment Architecture). Ainsi, SMARTIE propose un composant d'attestation de nœud permettant de vérifier l'intégrité d'un objet IoT en testant le hachage de la liste des logiciels et de fichiers qui ont été exécutés sur cet objet. Le composant d'attestation de nœud est un mécanisme d'attestation à distance (Remote Attestation) entre les objets IoT et l'unité distante centrale chargée de la mesure d'intégrité des objets. L'attestation à distance permet à une partie distante, la passerelle ou le serveur chargé de la vérification de l'intégrité des objets, d'inspecter l'état d'un périphérique ou d'un objet IoT à tout moment. La partie distante peut demander le hachage de la liste de logiciels ou de fichiers et elle est en mesure de vérifier si les enregistrements fournis par le dispositif ont été falsifiés ou non en comparant le hachage reçu à celui calculé. Le composant d'attestation de nœud développé dans SMARTIE permet de fournir une solution pratique qui est un compromis entre la solution matérielle et les approches basées sur le logiciel en utilisant le module *IMA* et l'architecture de mesure d'intégrité présente dans le noyau Linux [104] [105]. Le module *IMA* mesure l'intégrité du code binaire avant que le noyau ne procède au chargement de ce code dans la mémoire pour exécution. Le résultat de la mesure est enregistré et envoyé au service *IMASC* (Integrity Management Architecture using a SmartCard). Le service du système *IMASC* transmet le résultat à la carte à puce où il est horodaté et signé, de sorte qu'aucune manipulation de l'entrée n'est possible par la suite. Cette carte à puce conserve un registre avec une valeur de hachage. En ce qui concerne l'attestation à distance, la partie vérificatrice peut inspecter l'état d'un périphérique à distance à tout moment en demandant le hachage et en vérifiant les signatures. Lors d'une demande d'attestation à distance, le service *IMASC* interagit avec la carte à puce et la partie distante, pour fournir la preuve d'attestation. Dans ce contexte, des librairies ont été conçues pour les objets de l'IoT afin de réaliser les fonctions de hachage. A ce titre, la librairie « Cryptosuite » [119] est une bibliothèque pour Arduino qui

supporte différents algorithmes de hachage comme *SHA-1* (Secure Hash Algorithm 1), *SHA-256*, *HMAC-SHA-1 (Hash-based Message Authentication Code*- Secure Hash Algorithm) et *HMAC-SHA-256*.

Différents algorithmes de hachage ont été conçus pour les environnements contraints en ressources de stockage et de puissance de calcul. Ces algorithmes permettent d'avoir des digests de tailles différentes. Nous présentons dans le tableau 3.2 une étude comparative de trois algorithmes de hachage utilisables dans l'environnement IoT : Quark [120], PHOTON [121] et SPONGENT [121]. Cette comparaison se base sur différents critères tels que la taille du digest, le débit de hachage, GE et la possibilité de collisions. Parmi ces critères, la métrique débit de hachage utilisé dans le tableau 3.2 indique la vitesse à laquelle le hachage fonctionne. Le taux de hachage est calculé en digest par seconde (d / s).

Tableau 3.2 : Etude comparative des algorithmes de hachage pour les environnements contraints

	PHOTON	SPONGENT	Quark
Taille du digest	80	80	136
	128	128	
	160	160	176
	224	224	
	256	256	256
Débit de hachage	20/16	8	8
	16	8	
	36	16	16
	32	16	
	32	16	32
GE	865 / 1168	738/1127	1379/2392
	1122/1708	1060/1687	
	1396/2117	1329/2190	1702/2819
	1736/2786	1728/2903	
	2177/4362	1950/3281	2296/4640
Possibilité de collisions	2^{40}	2^{40}	2^{80}
	2^{64}	2^{64}	
	2^{80}	2^{80}	2^{80}
	2^{112}	2^{112}	
	2^{128}	2^{128}	2^{112}
Recommandations et spécifications	ISO/IEC 29192-5:2016	ISO/IEC 29192-5:2016	

3.3.2.5. Non répudiation dans l'IoT

3.3.2.5.1. Définitions

Le service de sécurité de type non répudiation permet de ne pas nier la participation aux échanges par une extrémité de communication. Ce service peut se présenter sous plusieurs formes.

La première forme est la non répudiation avec preuve de l'origine, où le destinataire reçoit la preuve de l'origine des données. Cette preuve peut être constituée par une signature numérique en utilisant le chiffrement asymétrique appliqué au résultat de hachage des données échangées. La deuxième forme est la non répudiation avec preuve de la remise des données, où l'émetteur reçoit cette preuve sous la forme d'un accusé de réception par exemple [86]. Ce servie de sécurité de type non répudiation est nécessaire dans l'Internet des objets afin de fournir une preuve d'envoi des données par les objets mais aussi une preuve d'envoi d'ordre d'exécution par les utilisateurs des services IoT. Ceci permet de suivre et de sauvegarder tous les évènements qui ont eu lieu dans un environnement IoT dans des fichiers de traces.

3.3.2.5.2. Projets et travaux de recherche

La non répudiation dans sa première forme (i.e., avec preuve de l'origine) repose sur des mécanismes utilisés pour garantir l'intégrité telle que la signature des données. Par conséquent, l'adaptation des mécanismes de non répudiation à un environnement Internet des objets peut hériter de l'adaptation des services d'intégrité dans l'IoT. La non répudiation a été garantie dans le projet européen SMARTIE grâce à la mise en place d'une signature de la liste des logiciels et du système d'exploitation des objets pour vérifier l'identité de l'émetteur du hachage (cf. section 3.3.2.4.2).

3.3.2.6. Disponibilité dans l'IoT

3.3.2.6.1. Définitions

La disponibilité concerne la possibilité d'accéder et d'utiliser sur demande des ressources par une entité autorisée suite à une authentification et un contrôle d'accès. La disponibilité est un service de sécurité puisqu'un service non disponible suite à une attaque de type *DoS* (Denial of Service) par exemple est un service non sécurisé et peut être compromis à n'importe quel moment [122]. La disponibilité dans l'Internet des objets est essentielle pour fournir un environnement connecté à Internet et pleinement opérationnel. Dans le contexte de l'IoT, ce service intègre d'une part la disponibilité des dispositifs (i.e., objets, passerelles) permettant de collecter des données sans interruptions. D'autre part, il assure aussi la disponibilité du service IoT offert aux utilisateurs. Ce deuxième type de disponibilité est déterminé par la configuration de l'environnement IoT, d'où la nécessité de faire des choix appropriés de protocoles de gestion et d'administration et de se protéger contre des attaques de type déni de service (*DoS* ; *DDOS* : Distributed DoS). Les services offerts dans l'IoT doivent être disponibles d'une façon continue tout en prenant en compte la criticité de quelques-uns de ces services [122]. Dans ce contexte, une disponibilité permanente de 24 heures sur 24 pendant les 365 jours de l'année est requise pour des services critiques dans l'IoT, telles que certaines applications dans le domaine de l'e-santé.

3.3.2.6.2. Projets et travaux de recherche

Suivant les recommandations de GSMA (Global System for Mobile Communications Association), un organisme commercial qui représente les intérêts de 800 opérateurs mobiles dans le monde entier, la disponibilité a été mise en relief en indiquant que les nœuds doivent être capables de communiquer en permanence entre eux, avec les utilisateurs et avec les services « back-end » [123]. Le projet européen iCore (Internet Connected Objects for Reconfigurable Ecosystem) [124], financé par FP7 (Octobre 2011 – Octobre 2014), définit les exigences de sécurité à prendre en considération dans un framework concernant l'Internet des objets. Le framework, appelé « Open Cognitive framework », prend en considération trois niveaux : Objets Virtuels (*VO* : Virtual Objects) qui fournit une représentation virtuelles des objets, Objets Virtuels Composites (*CVO* : Composite Virtual Objects) qui représente des fusions de plusieurs et le niveau utilisateur. Ce framework permet d'assurer une réutilisation des objets dans de nouveaux services et fournir ainsi une certaine redondance afin d'améliorer la disponibilité. De même, la fusion de certains objets virtuels à travers les assure une meilleure disponibilité. De plus, une description sémantique des capacités des objets permet d'assurer cette réutilisation. D'autre part, le projet iCore présente des recommandations à respecter dans différents cas d'utilisation pratiques de l'IoT. Ainsi, ce projet de recherche propose de fournir des mécanismes pour protéger l'infrastructure informatique contre les menaces de déni de services et de mettre en place les mécanismes nécessaires pour prendre en charge la récupération de service après une défaillance [124].

Le travail de recherche décrit dans [125] spécifie un outil portable de test de *DoS* basé sur un logiciel pour les objets IoT. Ce test d'attaque de type *DoS* doit être effectué au stade de la conception et du développement du produit. L'outil comprend une entité attaquante et un moniteur. L'attaquant effectue une attaque *DoS* sur le périphérique cible (i.e., objet IoT) et transmet les informations de trafic au moniteur. Dans ce contexte, l'attaque « Mirai » a été utilisée. Elle cible les périphériques sous Linux et les transforme en « bots » contrôlés à distance et pouvant être utilisés dans le cadre d'une attaque de réseau à grande échelle. Ainsi, l'outil de test *DoS*, présenté dans ce travail de recherche, vérifie efficacement si les appareils IoT sont résistants ou vulnérables aux attaques de type déni de service [125].

Nous spécifions dans cette thèse un mécanisme permettant d'assurer le service de contrôle d'accès au niveau de la couche basse de l'architecture IoT. Le contrôle d'accès des objets IoT est effectué au niveau des passerelles (cf. chapitre 7).

3.3.3. Contrats de niveau de service de sécurité

Afin de spécifier les attentes des clients des services IoT concernant les mesures de sécurité prises en charge, les contrats de niveau de service (i.e., *SLA*) doivent inclure une partie sécurité permettant de spécifier les mécanismes et les algorithmes de sécurité proposés par le fournisseur de

service IoT. Globalement, le *SLA* de sécurité doit indiquer ce que le client s'attend à recevoir, ce que le fournisseur promet de fournir, les outils de mesure des paramètres de sécurité et les pénalités.

Différents travaux de recherche ont été menés pour spécifier des *SLA* de sécurité concernant différents environnements comme les réseaux IP, le cloud et les grilles. SLA Ready [126] est un consortium d'industriels et d'universités ayant un conseil consultatif formé par de grandes organisations de standardisation comme le NIST et de grandes entreprises comme CISCO. Les travaux de SLA Ready se focalisent sur la partie sécurité des contrats de niveau de service dans le Cloud. Cet organisme essaie d'inclure les politiques de sécurité dans les *SLA* de sécurité ou encore *SecSLA*. D'autre part, le travail de recherche présenté dans [127] a été mené pour quantifier la sécurité dans les environnements cloud. Cette quantification permet de comparer les différentes offres des fournisseurs de cloud suivant les besoins des clients et de définir les paramètres à inclure dans les contrats de type *SecSLA*. La quantification est effectuée grâce à des arbres de quantification (*QPT* : Quantitative Policy Trees) et des processus de hiérarchisation (*QHP* : Quantitative Hierarchy Process). *QPT* est une arborescence ET/OU (AND/OR) dans laquelle les exigences du client sont représentées. La technique d'évaluation *QHP* permet de classer les fournisseurs de service cloud en fonction de leur conformité aux exigences du client.

Un contrat de niveau de service de type *SecSLA* est utilisé pour énoncer explicitement les obligations du fournisseur, les mécanismes et outils de sécurité mis en œuvre, leur efficacité et les conséquences d'une éventuelle mauvaise gestion [128]. Il permet de mesurer la qualité de protection (*QoP* : Quality of Protection), qui comprend la capacité d'un fournisseur de services à fournir un service conformément à un ensemble d'exigences de sécurité spécifiques [129]. *SecSLA* a un cycle de vie bien spécifique qui comprend la publication, la négociation, l'engagement, le provisionnement, la surveillance et la résiliation [129]. Par conséquent, le *SecSLA* doit être adapté à l'environnement IoT pour inclure des mécanismes adaptés et répondre ainsi aux exigences et caractéristiques de cet environnement émergent. Nous spécifions dans cette thèse un *SLA* de sécurité adapté à un environnement IoT permettant de présenter les garanties de sécurité en termes de mécanismes et mesures à considérer au niveau de chaque couche de l'architecture IoT (cf. chapitre 6).

3.3.4. Protection de la vie privée et confiance dans l'IoT

3.3.4.1. Vie privée dans l'IoT

La protection de la vie privée nécessite une considération particulière dans l'Internet des objets pour protéger les informations concernant la vie privée des utilisateurs de services IoT contre l'exposition dans ce type d'environnement. Même si les données récoltées par un seul objet peuvent ne pas générer de problèmes portant atteinte à la vie privée des individus si elles ne sont pas rattachées à d'autres données provenant d'autres objets. Cependant, lorsque des données fragmentées provenant de plusieurs objets sont rassemblées, compilées et analysées, elles peuvent générer des informations sensibles nécessitant une protection adaptée.

Comme déjà décrit dans le chapitre 2, l'Internet des objets opère dans différents domaines d'applications à travers lesquels des données personnelles des utilisateurs sont collectées. Ainsi, les consommateurs des services IoT risquent de divulguer leur vie privée, peu à peu, sans s'en rendre compte, car ils peuvent ignorer la nature des données collectées et leurs utilisations dans ce type d'environnement. Les approches actuelles de protection des données personnelles dans l'IoT reposent principalement sur le chiffrement ou encore le contrôle d'accès aux informations collectées. Cependant, les menaces concernant la vie privée dans l'Internet des objets peuvent ne pas être couvertes par les mécanismes proposés par ces approches. En effet, les données peuvent être externalisées pour le traitement (data outsourcing) créant un risque de vente de ces données pour des fins de marketing ou autre à des parties tiers [130].

3.3.4.2. Confiance dans l'IoT

La confiance est un concept complexe influencé par de nombreuses propriétés mesurables et non mesurables. La confiance est étroitement liée à la sécurité des systèmes et des utilisateurs. Cependant, la confiance concerne non seulement la sécurité, mais également de nombreux autres facteurs, tels que la qualité du service rendu par le système, sa fiabilité, sa disponibilité et donc les services offerts par ce dernier, etc. Un autre concept important lié à la confiance est la protection de la vie privée. Un système de confiance devrait préserver la vie privée de ses utilisateurs, afin de gagner la confiance de ces derniers. La confiance, la sécurité et la protection de la vie privée sont des enjeux cruciaux dans les domaines émergents des technologies de l'information, tel que l'IoT [131].

La gestion de la confiance se fait grâce à plusieurs processus intervenant dès la collecte des données jusqu'à l'offre du service au client. Ainsi, la gestion de la confiance dans l'Internet des objets fournit un moyen efficace pour évaluer les relations de confiance entre les entités de l'IoT et pour les aider à prendre des décisions afin de communiquer et de collaborer entre elles. Pour assurer cette confiance, la détection et la collecte des données doivent être fiables dans l'Internet des objets. Ainsi, il faut accorder une attention particulière aux propriétés de confiance dans ce type d'environnement. Ces propriétés comprennent la sensibilité, la précision, la sécurité, la fiabilité et la persistance de l'objet, ainsi que l'efficacité de la collecte de données. La collecte génère un énorme volume de données qui doit être traité et analysé de manière digne de confiance en matière de fiabilité, de préservation de la vie privée et de précision. De plus, les données doivent être transmises et communiquées en toute sécurité dans un environnement IoT. La gestion des clés de chiffrement dans l'Internet des objets est un défi important à relever pour atteindre cet objectif puisque la confidentialité des données est commune aux services de sécurité, de protection de la vie privé et de la confiance. En outre, il faut prendre en considération les mesures nécessaires pour remédier aux attaques qui peuvent porter atteinte à tous les niveaux de l'environnement IoT (objets, passerelles, cloud, etc.). Il faut aussi être sûr que le système est robuste face à tous types d'attaques afin d'obtenir une confiance suffisante des utilisateurs en leur environnement IoT. Finalement, une gestion d'identité évolutive et efficace est attendue. Cette gestion d'identité concerne toutes les couches de

l'architecture IoT en partant de l'objet jusqu'à l'utilisateur du service hébergé sur le cloud. La gestion d'identité doit respecter la confidentialité de l'identité de l'utilisateur du service afin de respecter la vie privée correspondante. Le contexte du service IoT est susceptible d'influencer les stratégies de gestion des identités.

A ce titre, un service IoT critique dans le domaine e-santé requiert une gestion d'identité plus fine et plus spécifique pour s'assurer que des personnes bien identifiées peuvent accéder aux données personnelles et critiques ce qui permet de gagner la confiance des utilisateurs du service [131].

3.3.4.3. Réglementations

Plusieurs réglementations pour la protection de la vie privée ont été présentées par les états unis et l'union européenne. Une des plus importantes est la directive 95/46/EC du Parlement européen qui insiste sur la protection des personnes à l'égard du traitement des données à caractère personnel et à la libre circulation de ces données [132]. Ainsi, le Règlement Général sur la Protection des Données (*RGPD*) est un règlement de l'Union européenne constituant le texte de référence en matière de protection des données privées. Il est entré en vigueur depuis le 25 mai 2018 et engendre un changement important dans le traitement des données utilisateurs. Cette réglementation doit être respectée lors de l'offre des services dans l'Internet des objets et comprend différentes exigences [133] :

- Traiter les données personnelles équitablement, licitement et de manière transparente ;
- Collecter et conserver seulement les données personnelles réellement nécessaires ;
- Détruire les données personnelles après utilisation ;
- S'assurer que les données détenues sont exactes et à jour ;
- S'assurer d'être prêt à gérer les droits accrus des personnes ;
- Nommer un délégué à la protection des données ;
- Élaborer et tenir à jour un registre des activités de traitement ;
- Partager les données personnelles de manière responsable

3.3.4.4. Projets et travaux de recherche

Plusieurs travaux de recherche ont porté sur la confiance et la protection de la vie privée dans l'IoT. Ainsi, l'Alliance pour l'innovation dans l'IoT (*AIOTI* : Alliance for IoT Innovation) indique dans son rapport [134] les principales exigences de sécurité et de protection de la vie privée pour différents domaines d'application de l'Internet des objets. Ces exigences portent sur le contrôle des données par l'utilisateur, la transparence et le contrôle de l'interface utilisateur, le chiffrement par

défaut, l'isolation des données, le contrôle continu, etc. De plus, le rapport de l'alliance insiste sur l'importance de l'application de mécanismes supplémentaires comme la minimisation de la collecte des données et la nécessité de la responsabilisation pour éviter une utilisation abusive des données personnelles collectées [134].

D'autre part, *TCG* (Trusted Computing Group) [135], un groupe formé par AMD, Hewlett-Packard, IBM, Intel et Microsoft, a pour objectif d'implémenter les concepts du « Trusted Computing » (i.e., de confiance) dans les ordinateurs personnels. Dans ce contexte, le sous-groupe Internet des objets de *TCG* a défini un système de confiance comme étant un système conçu pour être prévisible, même sous le stress [136]. En outre, il a été spécifié que pour construire un système IoT de confiance, il faut mettre en place une racine matérielle de confiance (*RoT* : Root of Trust), utiliser le chiffrement lors du stockage, ajouter une automatisation de sécurité et protéger les systèmes hérités. Le *RoT* permet de générer des nombres aléatoires, de stocker et d'utiliser des clés à long terme et de vérifier l'intégrité du système dans le but de réduire les risques et d'assurer une protection forte au système.

De plus, *TPM* (Trusted Platform Module) est un standard ouvert et interopérable de l'ISO/IEC [137] qui permet de spécifier un *RoT* matériel. La spécification technique de ce standard a été rédigée par le groupe *TCG*. *TPM* assure des fonctionnalités de sécurité comme l'authentification, le chiffrement et l'attestation (garantissant la sécurité d'un logiciel ou d'un matériel à un tiers). Actuellement, *TPM* est intégré dans des milliards d'objets connectés. Le chiffrement de stockage matériel (Hardware Storage Encryption) est un composant de *TPM* permettant d'assurer le service de chiffrement. Le chiffrement de stockage matériel utilise le *SED* (Self-Encrypting Drive) afin d'assurer un chiffrement continu et sans impact sur la performance du système. Ce composant permet de se protéger contre les attaques physiques, la perte ou le vol en assurant un effacement instantané des données et des suites cryptographique. L'effacement des données stockées suite à une attaque assure une protection de la confidentialité des données et ainsi la protection de la vie privée des utilisateurs concernées par ces données.

D'autre part, l'automatisation de la sécurité (Security Automation) permet le traitement automatique des tâches liées aux opérations de sécurité. Cette automatisation est effectuée lors de toutes les phases de sécurisation du système (détection d'incident, analyse de l'incident, réponse à l'incident) [136]. Cette automatisation de sécurité est normalisée à travers différents standards comme IEEE802.1AR [138] et *TAXII* (Trusted Automated eXchange of Indicator Information) [139].

Enfin, le travail de recherche décrit dans [140] présente un modèle de gestion de la confiance dans l'Internet des objets nommé *TRM-IoT*. Ce modèle permet de se défendre contre les attaques des nœuds malicieux en les identifiant et en prenant les décisions adéquates. Pour effectuer cette tâche, la mesure de la confiance des objets est nécessaire. Elle est effectuée en prenant en compte des aspects complexes tels que l'évaluation de la crédibilité des nœuds à titre d'exemple. La théorie de la logique floue est ensuite utilisée pour déterminer la confiance en prenant en compte différents paramètres

d'entrée : ratio de transmission des paquets de bout en bout, la consommation énergétique et le ratio de livraison des paquets.

3.4. Gestion autonome dans l'IoT

3.4.1. Motivations de la gestion autonome dans l'IoT

L'existence des environnements informatiques complexes (i.e., IoT, Cloud, Grilles, etc.) augmentent les coûts humains et opérationnels liés à la gestion des infrastructures et à la résolution de problèmes [141]. En effet, l'augmentation du nombre d'objets connectés hétérogènes utilisant des technologies différentes rend difficile, voire même impossible, la configuration des périphériques par des administrateurs humains d'où le besoin d'automatisation de certains processus de gestion [141]. Dans ce contexte, le paradigme de gestion autonome (*AC*) [142] permet de faire face à la complexité croissante et au coût opérationnel important de la gestion de l'infrastructure IoT. L'importance de la gestion autonome dans l'environnement IoT a été soulignée par différentes organisations telles que l'UIT-T dans son document Y.2066 [12]. Ainsi, l'UIT-T décrit l'importance de l'intégration de la gestion autonome dans l'IoT à travers différentes fonctions comme l'auto-configuration, l'auto-optimisation et l'auto-protection. En outre, l'automatisation des processus s'avère importante puisqu'elle permet de minimiser le risque d'erreurs humaines ou encore la non-cohérence pouvant avoir lieu lors de la gestion des infrastructures IoT.

La gestion autonome est nécessaire pour les différentes couches de l'IoT. Premièrement, au niveau de la couche inférieure comportant les objets connectés, les communications entre objets doivent être minimisées pour assurer la conservation énergétique. Ainsi, le réseau d'objets connectés doit être configuré avec soin et sans erreurs pour éviter les communications inutiles ainsi que la perte des paquets. Ensuite, au niveau de la couche réseau, il est important de reconfigurer le réseau en période de congestion afin de minimiser les pertes de paquets et de maximiser les performances des infrastructures et ce en acheminant par exemple les messages différemment dans un réseau à sauts multiples. Au niveau de la couche supérieure, où se situe le cloud hébergeant les applications IoT et comportant les ressources de calcul et de stockage, une gestion autonome est nécessaire pour assurer la disponibilité des ressources grâce à une distribution de la charge du trafic sur la totalité des ressources en temps réel selon les sollicitations des couches inférieures. De même, les applications IoT peuvent nécessiter des capacités de gestion autonome qui modifient les exigences pendant l'exécution. Ainsi, la gestion autonome permettra de reconfigurer les objets IoT afin de collecter des données spécifiques pour améliorer l'efficacité de l'application et la précision du service rendu [143].

Sans une gestion autonome au niveau des différents couches de l'IoT, les problèmes seront résolus soit par un approvisionnement surdimensionné des ressources augmentant énormément le coût des services ; soit à travers une utilisation excessive des ressources conduisant à des inefficacités et à une durée de vie réduite des systèmes IoT. La gestion autonome permet d'éviter une intervention humaine très coûteuse, mais également sujette aux erreurs. Par conséquent, pour répondre aux

demandes des applications IoT et faire face aux changements continus de l'environnement, l'adoption du concept de gestion autonome est primordiale dans l'IoT [143].

3.4.2. Définitions et historique de la gestion autonome

L'Autonomic Computing (*AC*) permet de minimiser la complexité de la gestion des systèmes en utilisant différentes technologies pour gérer ces derniers d'une manière automatisée. Le terme autonome est inspiré de la biologie humaine. En effet, le système nerveux autonome surveille et adapte le fonctionnement des organes vitaux sans aucun effort conscient de la part de l'être humain. De la même façon, la gestion autonome anticipe les exigences du système informatique et permet de résoudre les problèmes tout en minimisant l'intervention humaine [142]. Dans la vision de l'*AC*, les administrateurs spécifient simplement les objectifs de haut niveau (High Level Business Goals) correspondant à leurs attentes du système informatique. Ces objectifs permettent ensuite de guider les processus autonomes sous-jacents. Ainsi, les administrateurs se concentrent sur la définition de stratégies de haut niveau et délèguent aux entités de la gestion autonome, à savoir le gestionnaire autonome (*AM* : Autonomic Manager) et les éléments gérés (*ME* : Managed Elements), l'exécution des tâches techniques de niveau inférieur nécessaires pour atteindre ces objectifs [144]. Les éléments gérés comportent des interfaces de type capteur permettant de récolter des informations nécessaires aux gestionnaires autonomes pour la prise de décision. De plus, ces éléments gérés comportent aussi des interfaces de type effecteur permettant d'exécuter les décisions prises par les gestionnaires autonomes.

Le concept de gestion autonome a été introduit pour la première fois par Paul Horn à l'Académie nationale de génie de l'Université de Harvard en 2001 [144]. Dans ce contexte, un système de gestion autonome peut être caractérisé par différents éléments : l'auto-configuration, l'auto-restauration, l'auto-optimisation, l'auto-protection, mais aussi la connaissance du système et de son environnement ainsi que l'ouverture et l'anticipation [145]. D'autres définitions de la gestion autonome ajoutent d'autres caractéristiques telles que l'auto-adaptation et la prise en compte de l'environnement et du contexte. La gestion autonome s'appuie sur un certain nombre de disciplines existantes comme la conception de protocoles, la gestion de réseaux, l'intelligence artificielle, l'informatique omniprésente etc.

3.4.3. Objectifs de la gestion autonome

Les caractéristiques fondamentales pour les systèmes autonomes concernent les quatre fonctions de gestion qu'ils peuvent assurer : l'auto-configuration (Self-Configuration), l'auto-restauration (Self-Healing), l'auto-optimisation (Self-Optimisation) et l'auto-protection (Self-Protection). Ces fonctions forment le *self-CHOP* qui a été étendu par plusieurs travaux de recherche avec d'autres fonctions telles que l'auto-organisation, l'auto-installation, l'auto-diagnostic, etc. Nous détaillons dans ce qui suit les caractéristiques des quatre fonctions de gestion autonome formant le *self-CHOP* [144].

L'auto-configuration permet à un système autonome de se configurer et se reconfigurer pour s'adapter à diverses conditions, éventuellement imprévisibles. De la même manière que la nouvelle cellule est intégrée dans le corps humain, un nouvel élément autonome doit s'intégrer dans le système autonome existant et ce dernier doit s'adapter à ce nouvel élément. De plus, un système autonome se déploie et se configure en fonction des objectifs prédéfinis et des ressources actuelles du système.

L'auto-restauration permet au système autonome de détecter, de diagnostiquer et de résoudre les problèmes en temps réel tout en essayant de minimiser les interruptions de service. De plus, le système doit prévoir les problèmes potentiels et prendre des mesures préventives. L'auto-restauration implique premièrement la détection des symptômes signalant un problème existant ou potentiel, par exemple un goulot d'étranglement. Deuxièmement, cette fonction de gestion autonome permet la détermination d'une solution viable pour éviter ou encore résoudre le problème. Les méthodes de récupération ou de réparation comportent la recherche d'une autre utilisation des ressources, le téléchargement de mises à jour logicielles, le remplacement de composants matériels défaillants, le redémarrage d'éléments défaillants ou simplement le lancement d'une exception pour avertir un administrateur humain.

L'auto-optimisation permet au système autonome de chercher perpétuellement des moyens pour améliorer son fonctionnement. L'auto-optimisation a comme but d'améliorer continuellement les performances du système et par conséquent la *QoS* rendu aux utilisateurs. De plus, un système autonome optimise ses performances en ajustant l'utilisation des ressources selon l'évolution de la charge. D'autre part, certains critères d'optimisation peuvent être conflictuels, par exemple performance et sécurité ou encore consommation énergétique et confort de l'utilisateur. Cela nécessite des compromis à faire lors de l'établissement des configurations optimales du système.

L'auto-protection assure une anticipation des dangers et des attaques de sécurité pouvant se produire dans le système. Elle permet une détection, une identification et une protection contre les menaces internes et externes, afin de préserver l'intégrité du système et d'assurer différents services de sécurité tels que la confidentialité et la protection des données. L'auto-protection s'adapte à divers types de menaces, notamment les attaques malveillantes ou encore accidentelles. Les actions d'auto-protection peuvent inclure la suppression de ressources lors de la détection d'intrusions, le renforcement des contrôles d'accès suite à des soupçons de menaces potentielles et enfin l'alerte générale des administrateurs système [144].

3.4.4. Boucle de contrôle (MAPE-K)

Le passage d'un système manuel faisant appel à l'intervention humaine à un système totalement autonome se fait progressivement en suivant plusieurs étapes définies par le groupe Tivoli d'IBM [146]. Selon ce même groupe, les éléments autonomes (i.e., *AE*) interagissent pour gérer d'une manière autonome le système correspondant. Chaque *AE* comporte deux composants : les gestionnaires autonomes (i.e., *AM*) qui exécutent des règles définis par des administrateurs et mettent

en œuvre les fonctions de la gestion autonome (i.e., *self-CHOP*) pour surveiller et contrôler le comportement des ressources gérés (i.e., *ME*). Ce dernier peut être un périphérique, un logiciel ou encore un système entier. La ressource gérée (*ME*) comprend des capteurs et des effecteurs. Le gestionnaire autonome (*AM*) est l'entité qui assure le contrôle central et l'analyse de ces ressources à travers une boucle de contrôle appelé *MAPE-K* [146].

La boucle de contrôle *MAPE-K* spécifiée par IBM et représentée par la Figure 3.2, correspond à plusieurs fonctionnalités telles que Monitor (Surveiller), Analyze (Analyser), Plan (Planifier), Execute (Exécuter) mais aussi un composant appelé Knowledge (Base de Connaissance). Chaque fonctionnalité a un rôle spécifique dans le processus de la boucle fermée de contrôle. La fonctionnalité « Surveiller » collecte des informations à partir des ressources gérées, via les interfaces de type capteur. Les données récoltées sont prises en compte par la fonctionnalité « Analyser » pour prédire les états futurs du système. Par la suite, la fonctionnalité « Planifier » permet de programmer les actions en fonction des informations de stratégie et des règles fournies par la « Base de Connaissances ». Enfin, la fonctionnalité « Exécuter » contrôle l'exécution du plan fourni et distribue les actions recommandées sur les effecteurs des ressources gérées. Dans ce contexte, la base de connaissances stocke les informations de politique et les règles correspondant à différents scénarios pouvant se produire dans l'environnement du système autonome [147]. Les interfaces de type capteur sont utilisés pour communiquer des informations détectées par les ressources gérées aux gestionnaires autonomes, alors que les interfaces de type effecteur sont utilisés par les gestionnaires autonomes pour exécuter des actions.

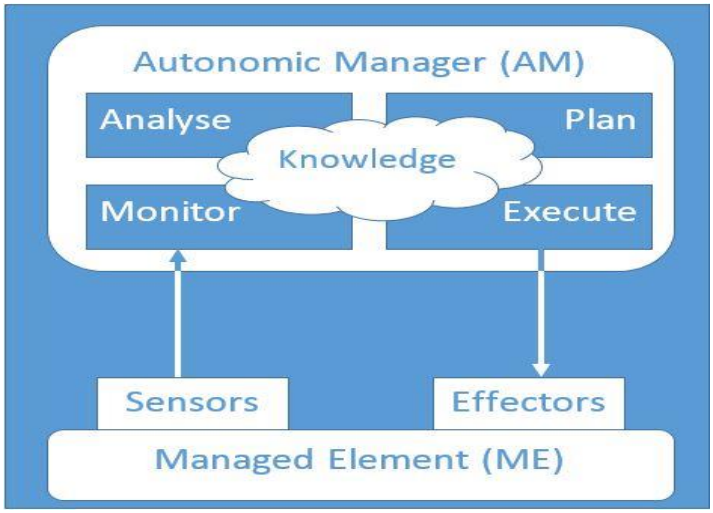

Figure 3.2 : Boucle de contrôle MAPE-K

3.4.5. Projets et travaux de recherche

L'importance de l'adoption de la gestion autonome dans l'environnement IoT a poussé différents projets de recherche à considérer ce défi de recherche. Ainsi, le projet européen OpenIoT

(Décembre 2011 – Décembre 2014) a proposé des adaptations des services afin d'utiliser le concept de gestion autonome dans l'IoT. Les systèmes IoT autonomes doivent adapter de manière dynamique les services qu'ils fournissent et les ressources qu'ils utilisent pour répondre aux besoins changeants des utilisateurs. De même, ces systèmes doivent répondre à des conditions environnementales variables. Dans ce contexte, OpenIoT a proposé un cadre de travail pour la gestion autonome de l'environnement IoT représenté par la Figure 3.3 [148].

Ce cadre de travail de gestion autonome dans l'IoT repose sur plusieurs éléments. Ainsi, l'élément création de service (Creation) permet de traduire les objectifs du service IoT en une description technique. Cette description englobe toutes les fonctionnalités nécessaires pour répondre aux exigences du client. Un service IoT est conçu sous la forme d'instructions pour fournir les mécanismes nécessaires permettant son offre. Le deuxième élément est la personnalisation des services (Customization) qui est nécessaire pour permettre au fournisseur de services IoT d'offrir à ses consommateurs une possibilité de personnalisation des services IoT en fonction de leurs besoins et/ou souhaits personnels. Un troisième élément essentiel est la gestion du service (Management). Les principales tâches de la gestion des services sont la distribution, la maintenance, l'appel, l'exécution et l'assurance. Un aspect fonctionnel important de la mise en œuvre de la gestion des services est le déploiement dynamique. Ainsi, lorsqu'un service IoT est déployé, des décisions doivent être prises afin de déterminer quels capteurs ou périphériques doivent être considérés pour l'offre du service. Cette activité s'effectue grâce à l'utilisation de règles logiques qui permettent de cartographier l'utilisateur avec les sources de données souhaitées et les capacités des objets qui prennent en charge le service. Enfin, le cadre de travail de gestion autonome du projet OpenIoT spécifie l'élément opération (Operation) qui permet d'appliquer des algorithmes adaptatifs d'optimisation en fonction des caractéristiques de chaque scénario de service IoT déployé [149].

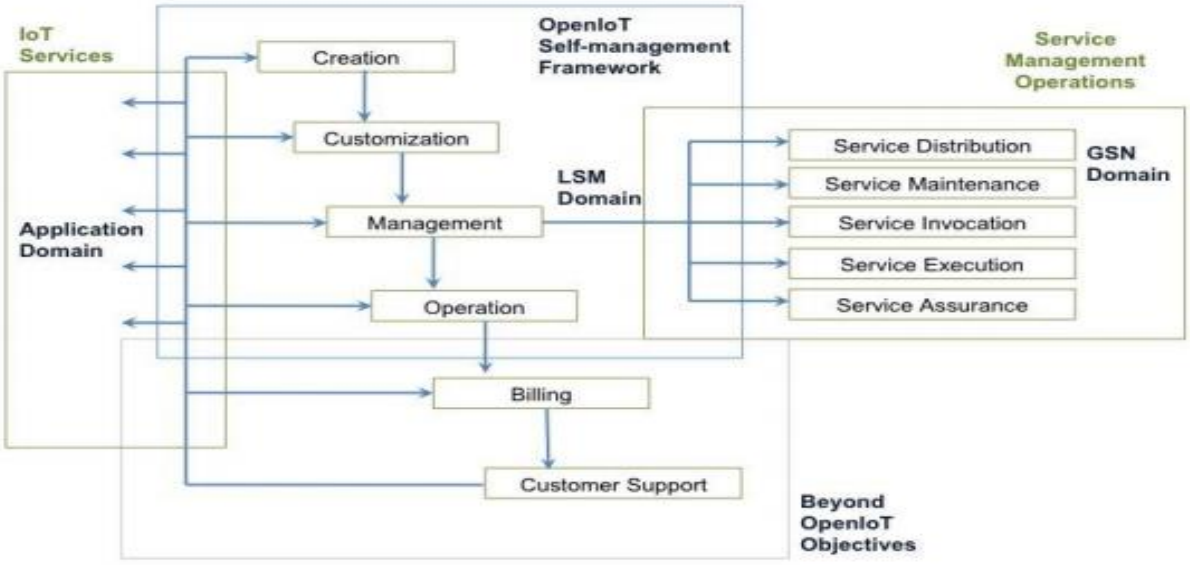

Figure 3.3 : Cadre de travail OpenIoT pour la gestion autonome [148]

Différentes études ont été menées pour étudier les possibilités d'adoption de la gestion autonome dans l'environnement IoT. Dans [146], les auteurs ont présenté une architecture autonome pour l'IoT basée sur la boucle *MAPE-K*. Cette architecture comprend une passerelle principale (*GWD* : Gateway Device) et des dispositifs mobile (*MND* : Mobile Node Devices). Dans cette architecture autonome, la passerelle joue le rôle du gestionnaire autonome (*AM*) et comprend différents composants permettant la surveillance des nœuds, l'analyse des exigences et la recherche de services afin de réaliser les fonctionnalités de la boucle de contrôle *MAPE-K*. D'autre part, les dispositifs mobiles de cette architecture jouent le rôle de ressources gérées et comprennent des capteurs et des effecteurs permettant une interaction avec l'environnement du système considéré. Ainsi, le composant « surveillance de nœuds » dans la passerelle permet de recevoir des informations concernant les dispositifs mobiles. Cette fonction permet de récupérer ces données localement au niveau de la passerelle principale mais aussi de les filtrer et les transmettre via une interface à une base de données externe. De plus, le composant « analyse des exigences » de la passerelle est responsable d'obtenir des exigences concernant le service souhaité en faisant appel à une source externe ou suite à une décision locale. Enfin, le composant « recherche de service » de la passerelle permet de trouver des services appropriés en fonction de la décision du composant « analyse des exigences » et d'envoyer aux dispositifs mobiles les éléments de configuration de services correspondants.

Egalement, les auteurs dans [149] ont présenté une architecture autonome pour l'IoT. Cette dernière a pour objectif d'optimiser les contrats de niveau de service de différents clients. Cette architecture comporte un agent principal qui spécifie des objectifs globaux du système traduits par des règles de gestion. Ensuite, ces règles sont distribuées à différentes périphériques appelées points d'application des objectifs (*GEP* : Goal Enforcement Points), qui exécutent des politiques locales et définissent les mécanismes de contrôle et de configuration pour répondre aux objectifs globaux.

D'autre part, les auteurs dans [150] mettent en avant l'idée que les périphériques loT ne nécessitent aucune configuration manuelle. Ainsi, un appareil IoT devrait être simplement branché, démarré et tout le reste devrait être configuré automatiquement. Les auteurs ont utilisé des données sémantiques pour décrire les capacités des objets et de l'environnement IoT en général. Dans ce travail de recherche, les appareils déduisent leurs configurations en trouvant d'autres appareils avec des états similaires, grâce au traitement des données sémantiques. Enfin, le travail de recherche décrit dans [151] présente un cadre utilisant plusieurs boucles *MAPE-K* pour les systèmes complexes et hétérogènes. Pour l'IoT, une seule boucle *MAPE-K* peut ne pas être suffisante pour gérer la totalité du système. Cette hiérarchisation permet aux boucles *MAPE-K* de niveau supérieur de déterminer les valeurs définies pour les boucles subordonnées et ainsi former le plan de connaissances pour ces dernières.

Nous spécifions dans cette thèse des mécanismes permettant de gérer d'une manière autonome la *QoS* et la sécurité dans l'environnement IoT afin de maximiser l'autonomie du processus de l'offre du niveau de service avec ses deux composantes (cf. chapitres 8 et 9).

3.5. Conclusion

A travers ce chapitre, nous avons étudié les trois axes de recherche de cette thèse, à savoir la qualité de service, la sécurité et la gestion autonome et cela en présentant les définitions, les motivations, les challenges et les travaux de recherche existants pour chacun de ces trois axes. Ainsi, le premier axe de recherche, qui a fait l'objet de notre état de l'art, concerne la qualité de service dans l'environnement IoT. Nous avons étudié cet axe en définissant les besoins et les challenges en termes de *QoS* au niveau de chaque couche de l'architecture IoT. La couche dispositifs, comportant les objets IoT, doit prendre en compte la diversité des technologies de communication pour l'offre de *QoS*. D'autre part, l'offre de *QoS* au niveau de la couche réseau doit considérer le volume du trafic créé par le service IoT et acheminé via cette couche vers la couche supérieure. Enfin, la couche support doit prendre en compte la diversité des offres de services cloud (i.e., *SaaS*, *IaaS*, *PaaS*) et leurs caractéristiques. Le deuxième axe de recherche concerne la sécurité dans l'environnement IoT qui est d'une grande importance de nos jours. Nous avons considéré cet axe en étudiant les besoins et les challenges de chaque service de sécurité à part. Ainsi, nous avons présenté les différents services de sécurité (i.e., identification et authentification, contrôle d'accès, confidentialité, intégrité, non répudiation et disponibilité) et leurs utilités. De plus, comme certaines couches de l'architecture de l'IoT possèdent des exigences bien définies, il est nécessaire d'adapter ces services de sécurité pour respecter ces exigences. Les objets IoT sont majoritairement contraints en termes de ressources et par conséquent les mécanismes de sécurité doivent être légers pour minimiser la consommation énergétique et maximiser la durée de vie des objets. Le troisième axe de recherche concerne la gestion autonome dans l'environnement IoT. Nous avons traité cet axe d'une manière globale grâce à une étude de l'importance du déploiement des différentes fonctions de gestion *self-CHOP* ainsi que la boucle de contrôle correspondante au sein d'une architecture IoT. L'état de l'art présenté dans ce chapitre permet d'introduire le contexte de nos contributions présentées dans les chapitres à venir et de valoriser leur apport par rapport aux travaux existants.

Chapitre 4 – Proposition d'un Framework pour la garantie du niveau de service dans un environnement Internet des objets

4.1. Introduction

L'environnement IoT utilise différents types d'infrastructures pour l'offre de service à l'utilisateur des applications IoT. En effet, une infrastructure cloud permet d'héberger les applications IoT, stocker les informations et fournir des capacités de traitement. De plus, un deuxième type d'infrastructure, celle du réseau, rend possible l'acheminement des trafics de données émanent du troisième type d'infrastructure, celle des objets IoT collectant les informations, jusqu'à l'infrastructure cloud. Par conséquent, la spécification d'une architecture IoT déterminant les relations entre ces différentes infrastructures mais aussi les entités IoT nécessaires à l'offre de service selon les attentes des utilisateurs, devient primordiale. Dans ce contexte, plusieurs types de contrats de niveau de services (*SLA*), tels que celui établi entre le fournisseur de service IoT (*IoT-SP* : IoT Service Provider) et les fournisseurs de service cloud (*CSP* : Cloud Service Provider) mais aussi établi avec le fournisseur de service réseau (*NSP* : Network service Provider), doivent être définis afin de garantir les besoins des clients de services IoT en terme de qualité de service.

Nous proposons dans ce chapitre un Framework permettant de garantir le niveau de service et en particulier la qualité de service (cf. chapitre 6 pour son extension à la sécurité) dans un environnement IoT grâce à une architecture en trois couches et des contrats de niveau service relatifs à ces couches afin d'offrir un service IoT avec *QoS* de bout en bout. Ainsi, nous présentons dans la section suivante l'architecture globale de notre Framework basée sur la *QoS* dans l'IoT et comprenant trois couches et différents composants. Notre architecture, conforme aux recommandations des organismes de standardisation, permet aux différents composants de cette architecture de se mettre d'accord sur les attentes en termes de *QoS* à travers des contrats de niveau de service. En outre, nous spécifions dans la section 3 les différents *SLA* de *QoS* de notre Framework : *cSLA* (cloud Service Level Agreement), *nSLA* (network Service Level Agreement) et *gSLA* (gateway Service Level Agreement). Ainsi, nous détaillons dans cette section la procédure d'établissement d'un *SLA* global de *QoS* appelé *iSLA* (IoT Service Level Agreement) établi entre le client du service IoT (*IoT-C* : IoT Client) et le fournisseur de service IoT (*IoT-SP*). Finalement, nous décrivons les échanges entre les différentes entités de notre architecture lors l'établissement de l'*iSLA* avant de conclure ce chapitre dans la section 4.

4.2. Spécification d'une architecture globale de QoS dans l'IoT

Nous décrivons dans ce qui suit notre contribution portant sur une architecture globale permettant la garantie du niveau de service (en particulier la *QoS*) dans un environnement IoT (voir Figure 4.1). Cette architecture permet de prendre en considération différents domaine d'application de l'IoT (i.e., domaine de l'e-santé, domaine des réseaux véhiculaires et domaines des villes intelligentes). Notre architecture est basée sur trois couches comprenant les composants et les infrastructures requises pour l'offre d'un service IoT. La structuration en trois couches horizontales permet de différencier entre différents types d'infrastructures et permettre ainsi la possibilité de les offrir par différents fournisseurs.

L'architecture proposée est conforme au modèle de référence de l'UIT-T (cf. Chapitre 2). Elle repose sur deux entités principales ayant chacune un rôle spécifique :

- L'*IoT-C* : c'est l'entité qui demande la souscription d'un service auprès d'un *IoT-SP*. Il peut utiliser une infrastructure d'objets fournie par l'*IoT-SP* ou bien sa propre infrastructure pour le service en question. L'*IoT-C* n'est pas nécessairement l'utilisateur final du service IoT. En effet, il peut s'agir d'une entité en charge de plusieurs utilisateurs finaux.
- L'I*oT-SP :* c'est l'entité qui offre des services IoT pour différents *IoT-Cs*. Il peut disposer de sa propre infrastructure cloud ou alors souscrire des accords de niveau de service spécifiques auprès de différents fournisseurs cloud (*CSP)*. De plus, l'*IoT-SP* souscrit des accords de niveau de service auprès de différents fournisseurs réseau (*NSP)*.

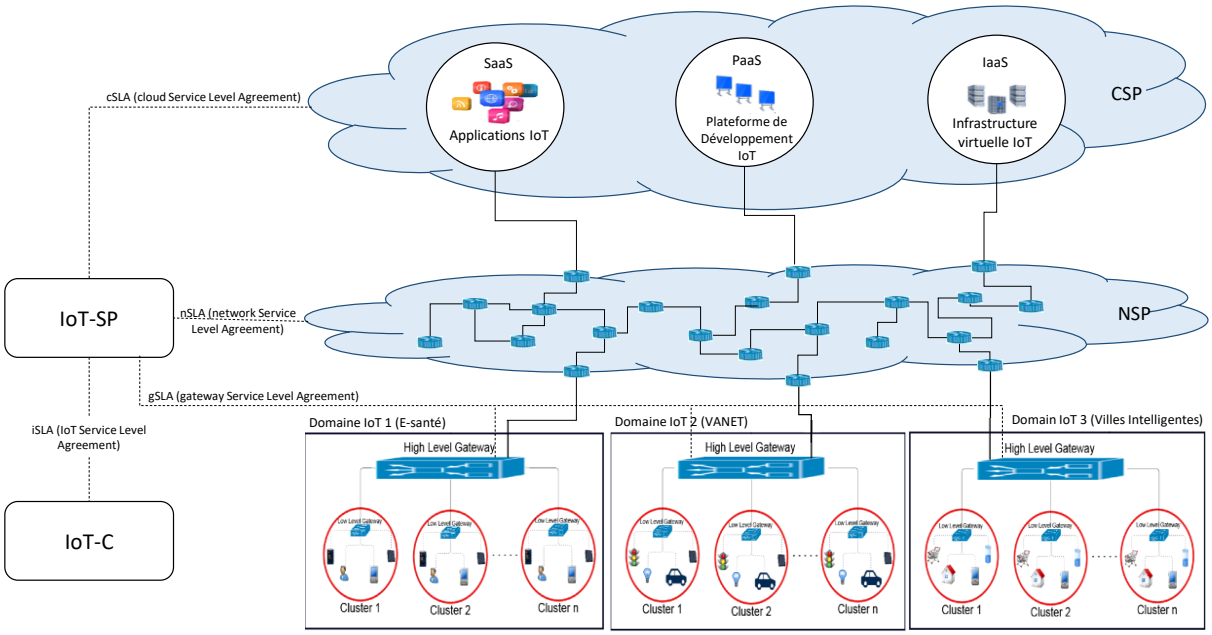

Figure 4.1 : Architecture globale IoT basée sur la QoS

L'architecture que nous proposons (voir Figure 4.1) repose sur trois couches : couche sensing, couche réseau et couche cloud. La couche cloud de l'architecture représente les services cloud indépendamment de leur source. En effet, l'infrastructure cloud peut appartenir à l'*IoT-SP* ou bien à un *CSP*. Si l'infrastructure appartient à un *CSP*, un *cSLA* est établi entre le fournisseur IoT et le fournisseur cloud. Dans ce contexte, différents services cloud peuvent être proposés [152] :

- Le service cloud de type infrastructure (*IaaS* : Infrastructure as a Service) est offert par le fournisseur de service cloud à travers des machines virtuelles et des capacités de stockage tout en permettant à l'utilisateur (i.e., *IoT-SP*) de les configurer suivant ses besoins, de développer et d'héberger les applications IoT. A travers le service *IaaS*, le fournisseur (i.e., *CSP*) gère les serveurs, la virtualisation et le stockage. Alors que le client *IaaS* (i.e., *IoT-SP*) gère les applications, les données, le système d'exploitation et les configurations des machines. Différents fournisseurs cloud offrent le service *IaaS*. Nous citons à ce titre *AWS* (Amazon Web Services) [153], Cisco Metapod [154], Microsoft Azur [155], Google Compute Engine [156], etc.

- Le service cloud de type plateforme (*PaaS* : Platform as a Service) est offert par le fournisseur de service cloud à travers une plateforme en se basant sur une machine virtuelle configurée avec un système d'exploitation et un environnement de développement. Ainsi, un client peut développer sa propre application pour l'héberger sur la plateforme. Dans ce contexte, le fournisseur *PaaS* gère les serveurs, la virtualisation, le stockage, le système d'exploitation et les configurations des machines. Alors que le client *PaaS* gère les applications et les données. Différents fournisseurs cloud offrent le service *PaaS*. Nous citons à ce titre *AWS* Elastic Beanstalk [157], Windows Azure [155], Heroku [158], Google App Engine [159], etc.

- Le service cloud de type logiciel (*SaaS* : Software as a Service) est offert au client à travers une application entièrement fonctionnelle hébergée sur le cloud et prête à être utilisée. Le fournisseur *SaaS* gère les serveurs, la virtualisation, le stockage, le système d'exploitation, les configurations des machines, les applications et les données. Dans ce cas le client n'a rien à gérer, il utilise directement l'application. Différents fournisseurs cloud offrent le service cloud de type *SaaS* tels que Google Apps [160], GoToMeeting [161], SalesForce [162], etc.

Ainsi, la couche la plus haute de notre architecture, à savoir la couche cloud, inclut des fonctionnalités de stockage et de traitement des données. Toutes les données remontées par les couches inférieures de notre architecture (i.e., couche sensing et couche réseau) mais aussi créées au niveau de cette même couche cloud, suite à différents traitements et calculs, doivent être stockées au niveau du cloud. Dans ce contexte, nous spécifions 4 classes d'applications IoT que nous hébergeons au niveau de cette couche cloud. Nous définissons ainsi les applications critiques à temps réel (*RTMC* : Real Time Mission Critical), les applications non critiques à temps réel (*RTNMC* : Real Time Non Mission Critical), les applications de diffusion (*Streaming*) et les applications non temps

réel (*NRT* : Non Real Time). La classification des applications se fait en se basant sur des critères de *QoS*. En effet, les applications *RTMC* nécessitent un délai minimal, une disponibilité accrue et un taux de livraison de paquet important (p. ex., surveillance des patients à distance, chirurgie à distance, etc.). D'autre part, les applications *RTNMC* nécessitent un délai minimal mais une disponibilité et un taux de livraison de paquet moins importants que ceux des applications *RTMC* (p. ex., diagnostic des patients à distance, système de détection d'anomalie dans les services critiques des villes intelligentes, etc.). Enfin, les applications *Streaming* nécessitent une gigue minimale et un taux de livraison de paquet important (p. ex., vidéosurveillance des locaux, etc.) alors que les applications *NRT* sont des applications avec des besoins de *QoS* assez faibles (p. ex., surveillance et gestion des équipements des villes intelligentes comme les lampes et les barrières, etc.).

La couche réseau de l'architecture proposée représente l'infrastructure fournie par un *NSP* afin d'interconnecter l'infrastructure d'objets, y compris les passerelles, à l'infrastructure cloud. Cette couche fournit les fonctionnalités réseaux telles que le routage, le transfert et la gestion des chemins (i.e., sélection et récupération du chemin). Enfin, la couche sensing comprend les passerelles et les objets. Nous spécifions deux type de passerelles IoT dans la couche sensing : des passerelles de haut niveau (*HL-Gw* : High Level Gateway) et des passerelles de bas niveau (*LL-Gw* : Low Level Gateway). Ces passerelles sont fournies obligatoirement par l'*IoT-SP* comme elles implémentent les mécanismes offrant la gestion autonome de la *QoS* et de la sécurité. Ces deux types de passerelles fournissent des fonctionnalités communes mais aussi des fonctionnalités spécifiques à chaque niveau. Les fonctionnalités communes comprennent le filtrage, la mise en file d'attente et la classification des données. En ce qui concerne les fonctionnalités spécifiques, la *HL-Gw* agrège les données pour minimiser la consommation de bande passante au niveau de la couche réseau lors de la transmission de données vers le cloud. Cette passerelle de haut niveau peut offrir des capacités de type « Fog computing » pour traiter en local les données des services IoT non tolérants au délai. Ainsi, les capacités de « Fog computing » permettent d'assurer un traitement des données près de leurs sources afin de minimiser le délai de bout en bout en éliminant le délai d'acheminement vers l'infrastructure de traitement de la couche cloud. D'autre part, les *LL-Gw* assurent d'autres fonctionnalités spécifiques telles que la classification des données récoltés. Ces passerelles de bas niveau sont responsables de la mesure de la qualité des données pour assurer la fiabilité des informations transmises vers la *HL-Gw* et par la suite vers l'application IoT au niveau de la couche cloud. Finalement, la couche sensing comprend aussi différents types d'objets IoT connectés permettant la collecte des données de leurs environnements. De plus, ces objets exécutent les commandes envoyées par les utilisateurs via l'application IoT et peuvent communiquent avec d'autres objets dans le cadre de l'offre d'un service IoT.

4.3. Spécification des SLA de QoS du Framework

La garantie de *QoS* dans un environnement particulier se fait en accord avec un *SLA* conclu entre le fournisseur et le client permettant de définir les caractéristiques attendues du service. Dans le

cadre d'une offre de service IoT dans un environnement Internet des objets le contrat *SLA* doit inclure les attentes du client de service IoT relatives aux différentes couches de l'architecture IoT. Dans ce contexte, nous proposons un *SLA* global adapté à un environnement IoT appelé *iSLA*. Ce dernier est établi entre l'*IoT-SP* et l'*IoT-C* et utilise différents sous-*SLA* portant sur les différentes couches de notre architecture IoT. Ainsi, le premier sous-*SLA*, à savoir le *cSLA*, est établi avec le *CSP* et concerne la couche cloud. Le deuxième sous-*SLA*, à savoir le *nSLA*, est établi avec le *NSP* et concerne la couche réseau. Le troisième sous-*SLA*, à savoir le *gSLA*, est un contrat interne au fournisseur de service IoT qui concerne la couche sensing et en particulier la *HL-Gw* permettant de définir les caractéristiques des passerelles et des objets connectés de cette couche. Dans ce qui suit, nous détaillons chacun de ses contrats de niveau de service (*cSLA, nSLA, gSLA* et *iSLA*).

4.3.1. SLA pour la couche Cloud : cSLA

4.3.1.1. Besoins du cSLA

La couche cloud comporte les fonctionnalités de stockage et de traitement nécessaires à l'offre d'un service IoT. Dans ce contexte, le contrat du niveau de service établi entre l'*IoT-SP* et le *CSP*, appelé *cSLA*, doit prendre en considération des paramètres de *QoS* relatifs au service cloud souscrit (*IaaS*, *PaaS* ou *SaaS*). Le *cSLA* concerne un service IoT bien défini. Il est identifié d'une manière unique par un identifiant, à savoir le *cSLA ID*. De plus, le *cSLA* doit comporter plusieurs parties importantes voire même nécessaires (voir Figure 4.2) telles que la validité, les partenaires, l'identification du service, les paramètres de performance et finalement les paramètres commerciaux.

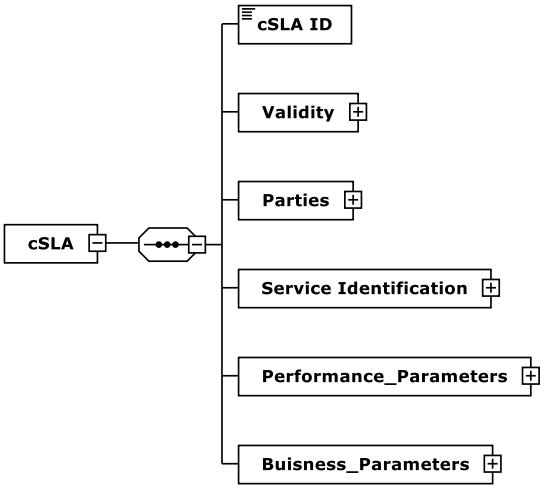

Figure 4.2 : Représentation globale du schéma XML du cSLA

Le *cSLA* doit permettre aux différents partenaires de se mettre d'accord sur les différents paramètres de *QoS* à garantir au niveau de la couche cloud. Ces paramètres permettent de quantifier la *QoS* dans le cloud et ont été choisis pour leurs utilités dans la garantie de la *QoS* de bout en bout au sein de l'IoT. Nous citons à ce titre, la disponibilité pour assurer la continuité du service, le temps

de réponse qui influence le délai de bout en bout, la gigue permettant de mesurer la variation du délai, les techniques de redondance de données et de recouvrement suite à une panne, le ratio de livraison de paquet et la bande passante au niveau de cette couche.

4.3.1.2. Paramètres du cSLA

Le *cSLA* est identifié par un *cSLA ID* et comporte entre autres les paramètres « Validity », « Parties » et « Service Identification ». Nous représentons les détails de ces paramètres dans la Figure 4.3. Premièrement, nous spécifions la validité de ce contrat en utilisant une date de début et de fin par rapport à la mise en application de ce contrat. Par la suite, les deux partenaires du *cSLA* doivent être identifiés de manière unique par un identifiant. Ainsi, nous utilisons un I*oT-SP ID* pour identifier le client du *cSLA* (soit l'*IoT-SP*) et un *CSP ID* pour identifier le fournisseur du *cSLA* (soit le *CSP*).

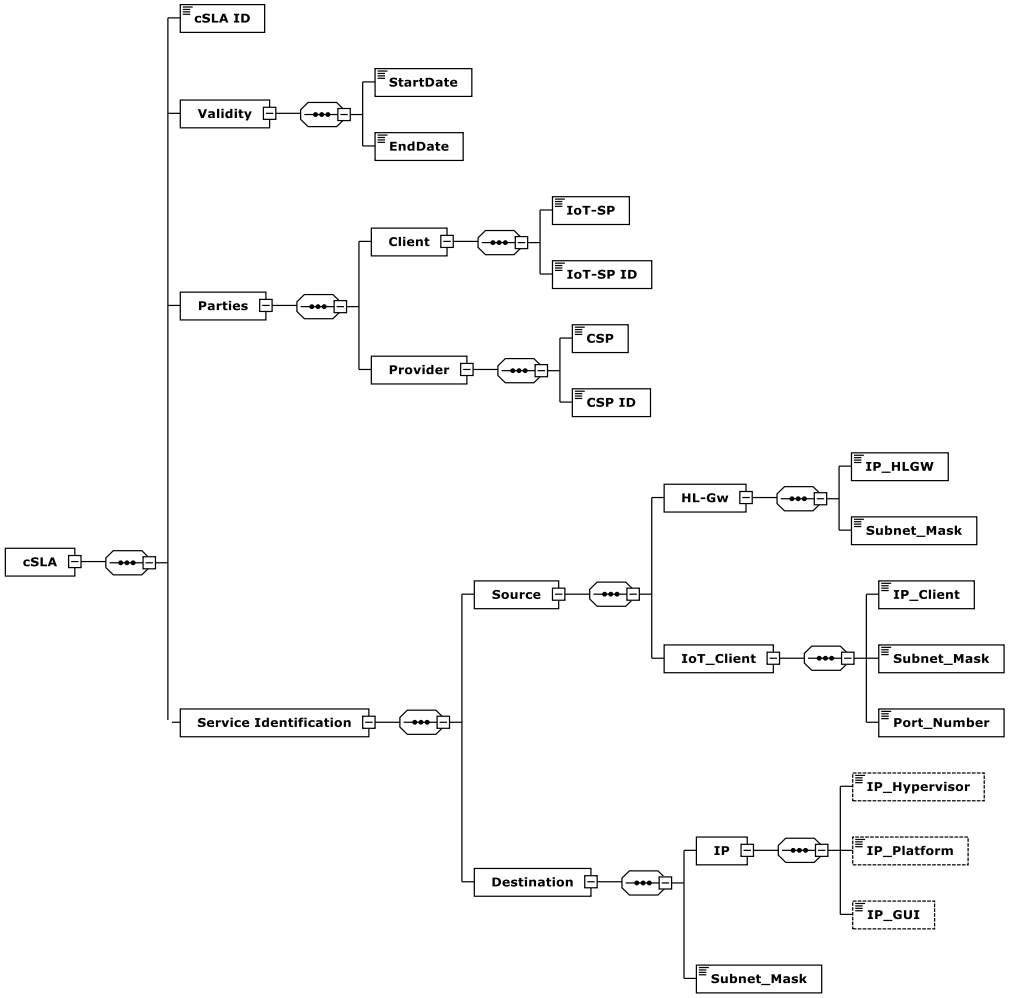

Figure 4.3 : Représentation du schéma XML des paramètres d'identification et de validité du cSLA

En plus de l'identification des partenaires de ce contrat de type *cSLA*, nous spécifions le paramètre « Service Identification » pour identifier le service garanti par le *cSLA*. L'identification du service se base sur les deux attributs « Source » et « Destination ». Concernant la source, le service garanti par le *cSLA* est identifié par deux éléments, la *HL-Gw* et l'*IoT-C* à travers leurs adresses *IP* et le numéro de port de l'application IoT utilisée par le client. En effet, les données reçues au niveau du cloud pour le traitement et le stockage proviennent de la passerelle *HL-Gw* qui est affectée à un *IoT-C* demandant un service IoT de l'*IoT-SP*. Ainsi, la source du service est définie par les caractéristiques de la *HL-Gw* et de l'*IoT-C*. De plus, les données ont été récoltées pour l'offre d'un service fourni via une application. Cette dernière possède un numéro de port permettant de l'identifier et ainsi d'identifier la source du service demandé par l'*IoT-SP* auprès du *CSP*. D'autre part, du côté de la destination, le service garanti par le cSLA est identifié par l'adresse *IP* de l'hyperviseur (dans le cas de souscription d'un service *IaaS*), ou par l'adresse *IP* de la plateforme (dans le cas de la souscription d'un service *PaaS*) ou par l'adresse *IP* de l'interface graphique de l'application (dans le cas de la souscription d'un service *SaaS*).

Une autre partie importante que nous spécifions dans le *cSLA* est « Performance_Parameters » permettant de décrire des paramètres de performance pour les services *IaaS, PaaS* et *SaaS*. Dans notre *cSLA*, nous regroupons les paramètres de performance des services *IaaS* et *PaaS* car le *PaaS* comporte quelques paramètres supplémentaires par rapport à l'IaaS (voir Figure 4.4). D'autre part, un seul type de service cloud (*IaaS, PaaS* ou *SaaS*) peut être considéré dans un contrat *cSLA*. Par conséquent, une occurrence de 1 est utilisée pour le service en question et une occurrence de 0 pour les autres services lors de l'instanciation du schéma *XML* du *cSLA* correspondant. Il est à noter qu'un élément non obligatoire (occurrence minimale de 0) est représenté par des pointillés dans un schéma *XML* et dans ce cas celui du *cSLA*.

Les paramètres de performance que nous spécifions pour les services *IaaS/PaaS* décrivent le type de machine virtuelle en définissant plusieurs caractéristiques : les mémoires volatiles et non volatiles, la vitesse et l'architecture du processeur, l'heure de démarrage de la machine et le nombre de machines virtuelles créées. De plus, nous spécifions les paramètres de stockage ainsi que des paramètres concernant la plateforme mais ces derniers ne sont pas obligatoires (i.e., une occurrence minimale de 0 et maximale de 1 dans le schéma *XML* et donc en pointillés dans la Figure 4.4). En effet, ces paramètres sont utilisés uniquement si le *cSLA* concerne un service de type *PaaS* offert par le *CSP*. Dans ce cas, les paramètres de plateforme comportent la liste des systèmes d'exploitation supportés, la liste des plateformes de développement IoT et le nombre maximal d'accès simultanés à la plateforme. Enfin, nous spécifions les paramètres de *QoS* relatifs aux services de type *IaaS* et *PaaS*. Ces paramètres définissent la disponibilité des machines virtuelles et des capacités de stockage, le temps de réponse au niveau de la couche cloud, la gigue au niveau de la couche cloud, les techniques de recouvrement et de redondance des données, le taux de livraison de paquet et la bande passante au niveau de la couche cloud. Tous ces paramètres sont essentiels pour la garantie de *QoS* des services *IaaS/PaaS* (i.e., une occurrence de 1). Parmi ces paramètres, nous définissons la disponibilité à travers un ou plusieurs des attributs suivants : Temps moyen d'arrêt du service (*MDT* :

Mean Down Time), Temps moyen de reprise du service après arrêt (*MTTR* : Mean Time To Repair) et pourcentage de disponibilité. Quant au temps de réponse au niveau de la couche cloud, nous le décomposons en de deux parties : le délai de traitement des machines virtuelles (*VMs*) et le temps d'acheminement des requêtes dans l'infrastructure cloud vers les machines virtuelles. D'une façon similaire pour la gigue, elle dépend de la variation du délai de traitement des *VMs* ainsi que la variation du délai l'acheminement des requêtes.

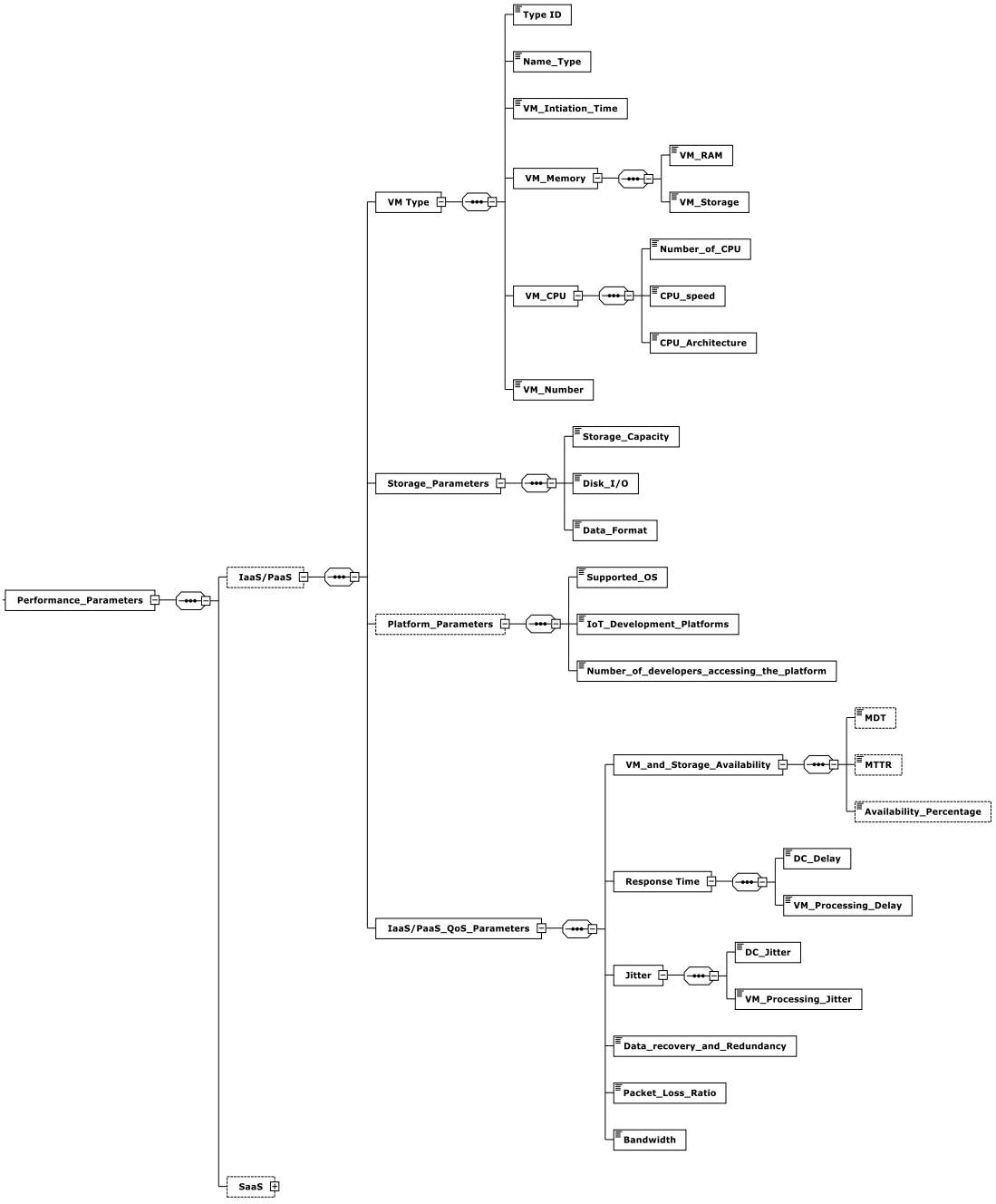

Figure 4.4 : Représentation du schéma XML des paramètres de performance du cSLA (IaaS/PaaS)

Les paramètres de performance que nous spécifions pour un service de type *SaaS* comportent trois attributs : « Application_Characteristics », « Storage » et « SaaS_QoS_Parameters » (voir Figure 4.5).

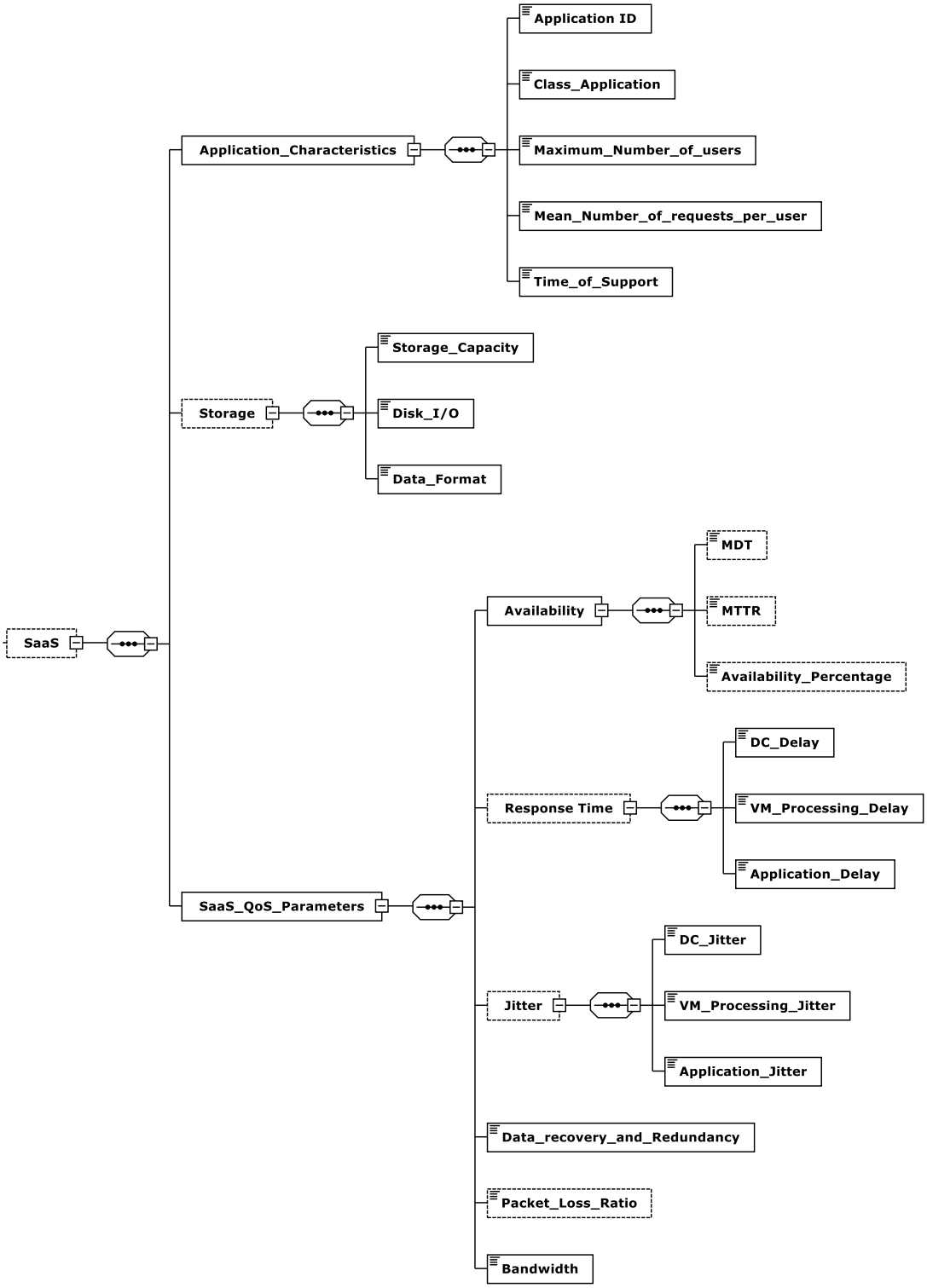

Figure 4.5 : Représentation du schéma XML des paramètres de performance du cSLA (SaaS)

Le premier attribut est obligatoire (i.e., occurrence de 1). Il définit les caractéristiques de l'application en utilisant un identifiant de l'application, la classe de l'application (une liste comportant les différents types d'applications *RTMC, RTNMC, Streaming, NRT*), le nombre d'utilisateurs de l'application, le nombre de requête moyen reçu par l'application et le temps de support (voir Figure 4.5). Le deuxième attribut est optionnel (occurrence minimale de 0 et maximale de 1). Il définit les caractéristiques de stockage en spécifiant la capacité, le nombre autorisé d'entrées/sorties sur le disque et le format des données stockés. Finalement, le troisième attribut concernant les paramètres de *QoS* du service de type SaaS est obligatoire. Il définit la disponibilité de l'application, la liste des techniques de recouvrement et de redondance des données ainsi que la bande passante pour les 4 classes d'application (i.e., *RTMC, RTNMC, Streaming* et *NRT*). En effet, ces paramètres de *QoS* sont communs pour ces 4 classes d'applications alors que nous définissons d'autres paramètres de *QoS* optionnels tels que le temps de réponse, la gigue et le taux de livraison de paquets qui sont requis par certaines de ces classes. Les mêmes techniques de quantification des paramètres de *QoS* sont utilisées pour les services de type *IaaS*/*PaaS* et *SaaS* (par exemple *MDT* et *MTTR* pour le pourcentage de la disponibilité).

Le dernier ensemble de paramètres que nous spécifions dans le *cSLA*, à savoir « Business_Parameters », comporte des attributs relatifs à la tarification et à la gestion des violations et des pénalités (voir Figure 4.6).

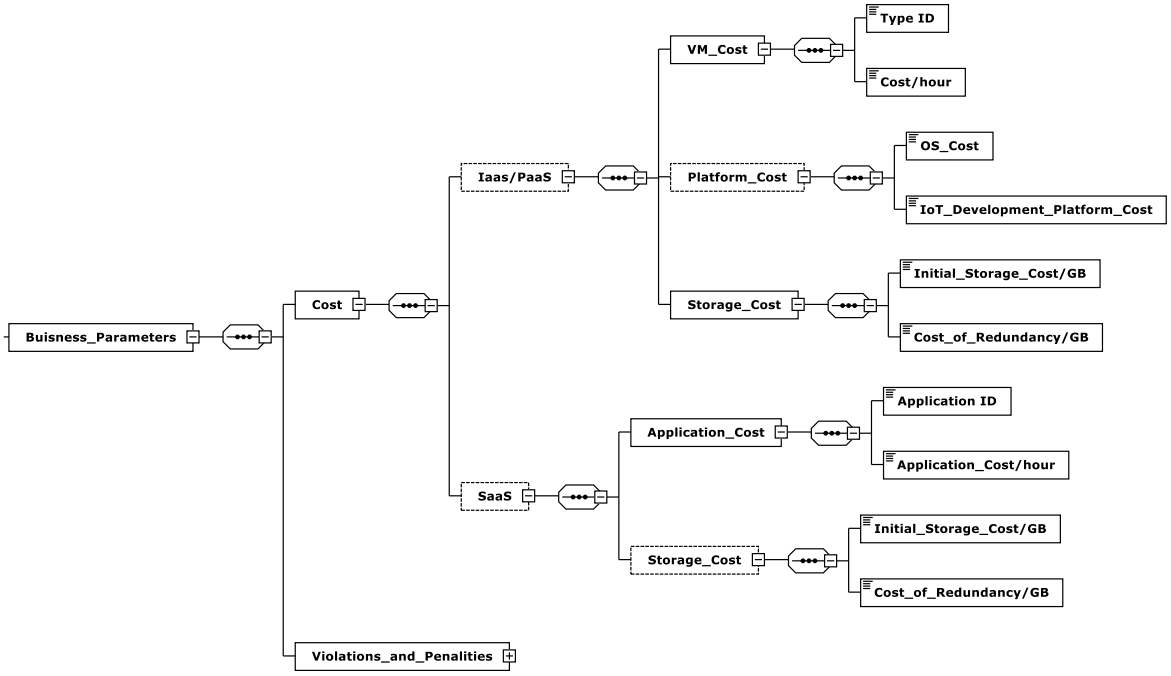

Figure 4.6 : Représentation du schéma XML des paramètres commerciaux du cSLA (Cost)

En ce qui concerne la tarification, le paiement suivant l'utilisation (*Pay As You Go*) est utilisé pour tous les services cloud. Cependant la méthode de tarification dépend du type de service cloud (*IaaS*, *PaaS* ou *SaaS*). Néanmoins, nous regroupons les services *IaaS/PaaS* ensemble grâce à la

similitude des paramètres entre les deux services. Ainsi, pour l'*IaaS/PaaS*, le coût de la machine virtuelle et celui du stockage sont obligatoires (occurrence de 1) pour les deux services. Ces coûts dépendent du type de la machine virtuelle utilisée, de son utilisation par heure mais aussi de l'espace de stockage par Go utilisé et l'espace de stockage redondant par Go utilisé pour améliorer la disponibilité grâce à la redondance. Par contre, le coût de plateforme est optionnel (occurrence minimale de 0 et occurrence maximale de 1) car il est utilisé uniquement dans le cas du *PaaS*. Ce coût dépend du système d'exploitation et de la plateforme de développement installés sur la machine virtuelle. D'autre part, pour le service *SaaS*, le coût dépend de l'utilisation de l'application par heure et du coût de l'espace de stockage par Go utilisé ainsi que celui de l'espace de stockage redondant.

```
<xs:element minOccurs="1" maxOccurs="1" name="Violations_and_Penalities" >
    <xs:complexType >
        <xs:sequence >
            <xs:element  minOccurs="0" maxOccurs="1" name="IaaS/PaaS">
                <xs:complexType >
                    <xs:sequence >
                        <xs:element  minOccurs="1" maxOccurs="1" name="Monitoring_Interval_Time">
                            <xs:complexType >
                                <xs:sequence >
                                    <xs:element minOccurs="1" maxOccurs="1"  name="Availability" type="Time_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Response_Time" type="Time_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Packet_Loss_Ratio" type="Time_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Bandwidth" type="Time_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Jitter" type="Time_Type"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element  minOccurs="1" maxOccurs="1" name="Thresholds">
                            <xs:complexType >
                                <xs:sequence >
                                    <xs:element minOccurs="1" maxOccurs="1" name="Availability"  type="Percentage_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1"  name="Response_Time" type="Percentage_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Packet_Loss_Ratio" type="Percentage_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Bandwidth" type="Percentage_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Jitter" type="Percentage_Type"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element minOccurs="0" maxOccurs="1" name="SaaS" >
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="1" name="Monitoring_Interval_Time" >
                            <xs:complexType >
                                <xs:sequence >
                                    <xs:element minOccurs="1" maxOccurs="1" name="Availability" type="Time_Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Response_Time" type="Time_Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Packet_Loss_Ratio" type="Time_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Bandwidth" type="Time_Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Jitter" type="Time_Type"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                        <xs:element  minOccurs="1" maxOccurs="1" name="Thresholds">
                            <xs:complexType >
                                <xs:sequence >
                                    <xs:element minOccurs="1" maxOccurs="1"  name="Availability" type="Percentage_Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Response_Time"  type="Percentage-Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Packet_Loss_Ratio" type="Percentage_Type"/>
                                    <xs:element minOccurs="1" maxOccurs="1" name="Bandwidth" type="Percentage_Type"/>
                                    <xs:element minOccurs="0" maxOccurs="1" name="Jitter" type="Percentage_Type"/>
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 4.7 : Schéma XML des paramètres commerciaux du cSLA (Violations and Penalities)

Le deuxième attribut des paramètres commerciaux du *cSLA* concerne la gestion des violations et des pénalités (voir Figure 4.7). En effet, un intervalle de surveillance est défini pour chaque paramètre de *QoS* du service souscrit, au cours duquel des valeurs de performance sont collectées pour tester la conformité du service fourni aux attentes de l'utilisateur grâce à des seuils prédéfinis. De même, les services *IaaS/PaaS* sont regroupés ensemble et les paramètres disponibilité, temps de réponse, taux de livraison de paquets, bande passante et gigue sont obligatoirement évalués avec un intervalle de surveillance et un seuil prédéfinis dans le contrat. En ce qui concerne le service *SaaS*, les paramètres disponibilité et bande passante sont aussi obligatoirement surveillés. Par contre, les paramètres temps de réponse, taux de livraison des paquets et gigue sont optionnels et dépendent du type de l'application souscrite (Temps réel, Streaming, etc.).

4.3.1.3. Procédure d'établissement du cSLA

Différents échanges doivent avoir lieu entre le *CSP* et l'*IoT-SP* pour établir le *cSLA*. Ces échanges permettent aux différentes parties de se mettre d'accord sur les valeurs des paramètres de performances demandées par l'*IoT-SP*. Nous spécifions dans la Figure4.8 l'ensemble des états de l'automate fini (*FSM* : Finite State Machine) concernant l'établissement du *cSLA* au niveau du *CSP*.

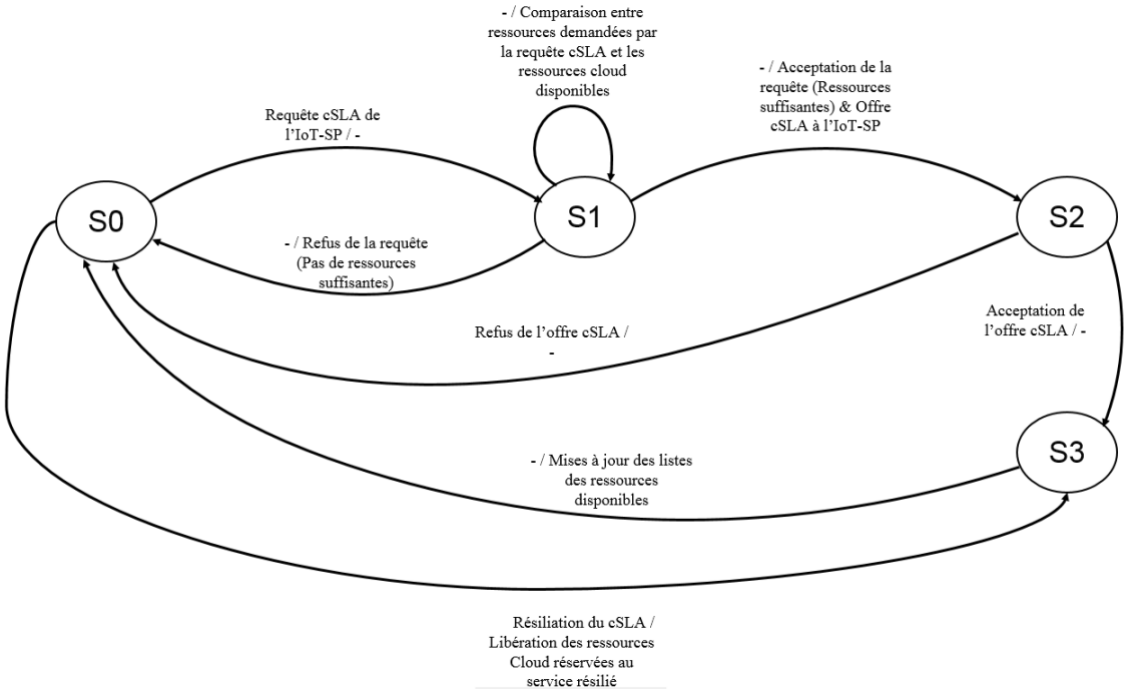

Figure 4.8 : FSM du CSP concernant l'établissement du cSLA

Dans l'état S0, le *CSP* attend la réception d'une nouvelle requête concernant l'établissement d'un *cSLA* avec un client (i.e., *IoT-SP*). Suite à la réception de cette requête, le *CSP* passe à l'état S1 et boucle sur lui-même pour déterminer sa capacité à offrir le service demandé. En effet, le *CSP* compare les ressources demandées par l'*IoT-SP* et ses propres ressources disponibles. Si les ressources du *CSP* ne sont pas suffisantes pour offrir le service demandé, le *CSP* revient à l'état S0

en rejetant la requête d'établissement du *cSLA*. Si les ressources sont suffisantes, le *CSP* envoie une offre *cSLA* à l'*IoT-SP* et passe à l'état S2. Dans cet état, le *CSP* attend la réponse de l'*IoT-SP* concernant son offre. Si l'offre n'est pas acceptée par l'*IoT-SP*, le *CSP* revient à l'état S0. Sinon, le *CSP* passe à l'état S3 et effectue les configurations nécessaires pour le nouveau service. De plus, le *CSP* met à jour la liste des ressources disponibles et revient à l'état S0 en attendant la réception d'une nouvelle requête ou encore la résiliation d'un contrat. A l'état S0, le *CSP* peut recevoir une demande de résiliation du contrat *cSLA*. Par conséquent, les ressources cloud réservées pour le service seront libérées par le *CSP* en passant à l'état S3. Finalement, le *CSP* passe de nouveau à l'état S0 en mettant à jour la liste des ressources disponibles.

4.3.2. SLA pour la couche réseau : nSLA

4.3.2.1. Besoins du nSLA

Les informations remontées par les d'objets connectés doivent être acheminées jusqu'à l'infrastructure cloud pour le traitement et le stockage. Cependant, le traitement peut être exécuté sur les passerelles *HL-Gw* pour les informations sensibles au délai afin d'éliminer celui de l'acheminement des données et par conséquent minimiser le délai global. D'autre part, le cloud héberge des applications IoT qui peuvent envoyer des commandes aux objets connectés pour les gérer. Ainsi, l'acheminement des données entre les d'objets IoT de la couche sensing de notre architecture IoT et l'infrastructure cloud de cette architecture doit se faire dans le respect des caractéristiques de ces flux de données. Cette interconnexion est offerte par les *NSP* ayant des infrastructures réseaux à grande échelle et à chemin multiples. Un service est demandé par l'*IoT-SP* auprès d'un *NSP* pour interconnecter les passerelles *HL-Gw* à l'infrastructure cloud. Nous proposons pour ce service l'établissement d'un contrat de niveau de service, appelé *nSLA*, entre le *NSP* (en tant que fournisseur) et l'*IoT-SP* (en tant que client). Ce contrat est identifié par un *nSLA ID*.

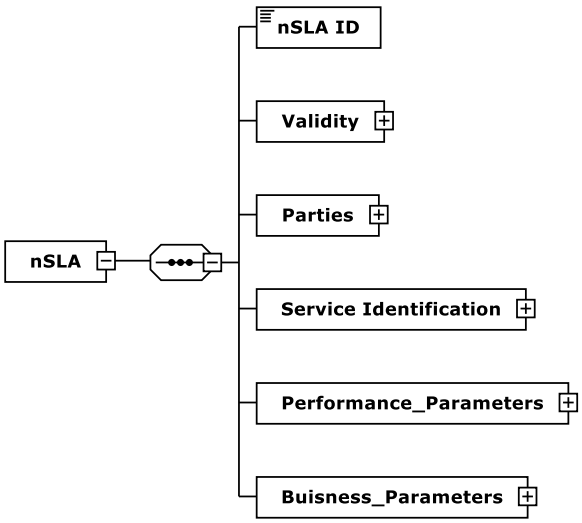

Figure 4.9 : Représentation globale du schéma XML du nSLA

Le *nSLA* comporte plusieurs parties importantes (voir Figure 4.9) telles que : la validité, les partenaires, l'identification du service, les paramètres de performance (basés sur les paramètres de *QoS* d'un service réseau traditionnel comme la bande passante, la latence, la gigue, le taux de perte des paquets et la disponibilité) et finalement les paramètres commerciaux.

4.3.2.2. Paramètres du nSLA

Nous représentons dans la Figure 4.10 certaines parties du schéma *XML* du *nSLA* à savoir « Validity », « Parties » et « Service Identification ».

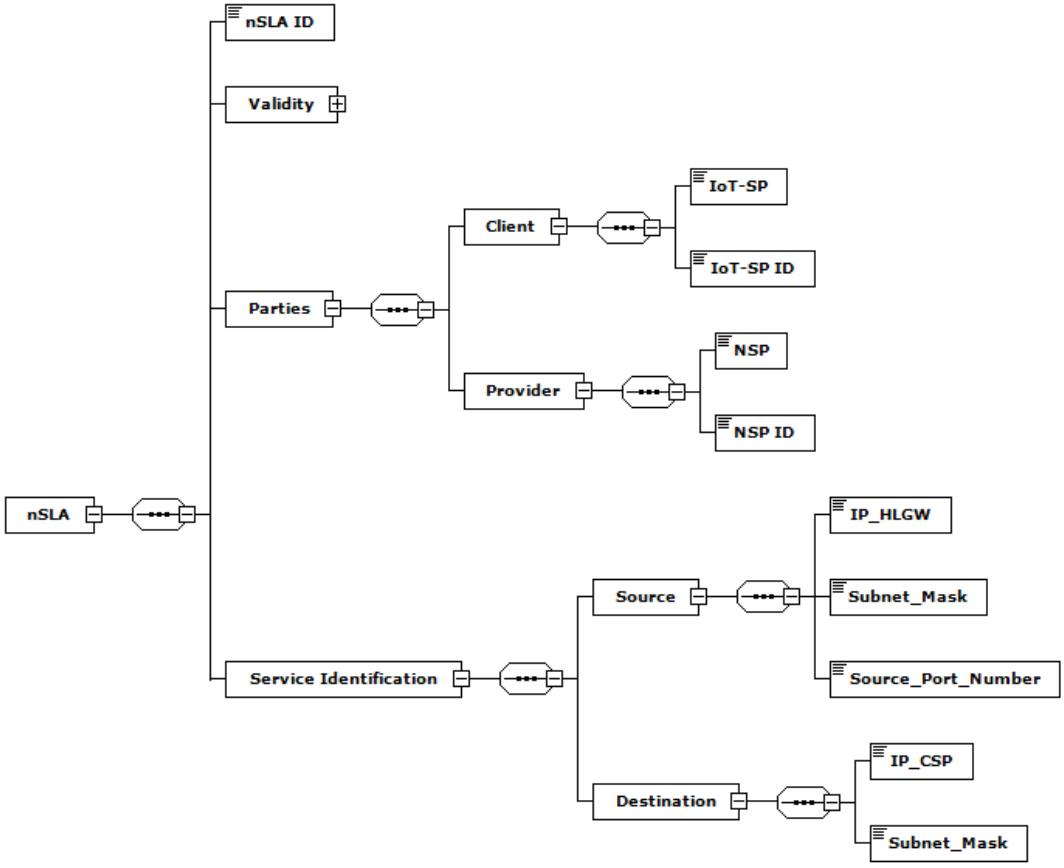

Figure 4.10 : Représentation du schéma XML des paramètres d'identification et de validité du nSLA

Premièrement, nous spécifions la validité de ce contrat en utilisant une date de début et de fin de mise en application de ce dernier. Par la suite, les deux partenaires du contrat doivent être identifiés de manière unique par un identifiant. Ainsi, l'*IoT-SP ID* est utilisé pour identifier le côté client du *nSLA* (soit l'*IoT-SP*) et le *NSP ID* pour identifier le côté fournisseur du *nSLA* (soit le *NSP*). En plus de l'identification des partenaires de ce contrat de type *nSLA*, nous spécifions le paramètre « Service Identification » pour identifier le service garanti par ce contrat. L'identification du service se base sur les deux attributs « Source » et « Destination ». Concernant la source, le service est identifié par l'adresse *IP* de la passerelle de la *HL-Gw* avec sa longueur de préfixe et le numéro de port source

utilisé pour la connexion au cloud. Alors que pour la destination, le service est identifié par l'adresse *IP* relative au fournisseur de cloud avec la longueur de préfixe. Cette adresse *IP* peut être celle de l'hyperviseur (dans le cas de la souscription d'un service *IaaS* avec le *CSP*), de la plateforme (dans le cas de la souscription d'un service *PaaS* avec le *CSP*) ou de l'interface de l'application (dans le cas de la souscription d'un service *SaaS* avec le *CSP*).

Une autre partie importante que nous spécifions dans le *nSLA* est « Performance_Parameters » permettant de décrire les paramètres de performances à garantir grâce à ce contrat (voir Figure 4.11). Dans ce contexte, nous utilisons les paramètres de *QoS* réseaux classiques pour quantifier cette partie du *nSLA*. En effet, le service souscrit avec le *NSP* est un service réseau classique permettant d'interconnecter deux extrémités distantes (la *HL-Gw* et le cloud).

Ainsi, les paramètres de *QoS* bande passante, délai et gigue concernent les caractéristiques de l'acheminement du trafic à travers l'infrastructure du *NSP* depuis la passerelle jusqu'au cloud. De plus, le taux de perte des paquets est un autre paramètre de *QoS* utilisé dans le *nSLA* pour mesurer la fiabilité du système IoT au niveau de l'infrastructure réseau. Enfin, la disponibilité est un paramètre important qui donne des informations concernant la disponibilité de l'infrastructure réseau fourni par le *NSP*. Ce paramètre peut être évalué soit par le *MDT*, soit par le *MTTR*, ou alors avec un pourcentage.

```
<xs:element minOccurs="1" maxOccurs="1" name="Performance_Parameters" >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="QoS_Parameters" >
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="1" name="Network_BW" type="Bandwidth_Type" />
                        <xs:element minOccurs="1" maxOccurs="1" name="Network_Latency" type="Latency-Jitter_Type" />
                        <xs:element minOccurs="1" maxOccurs="1" name="Network_Jitter" type="Latency-Jitter_Type" />
                        <xs:element minOccurs="1" maxOccurs="1" name="Packet_Loss_Ratio" type="Percentage_Type" />
                        <xs:element minOccurs="1" maxOccurs="1" name="Availability" >
                            <xs:complexType >
                                <xs:sequence >
                                    <xs:element minOccurs="0" maxOccurs="1" name="MDT" type="MDT-MTTR_Type" />
                                    <xs:element minOccurs="0" maxOccurs="1" name="MTTR" type="MDT-MTTR_Type" />
                                    <xs:element minOccurs="0" maxOccurs="1" name="Avaliability_Percentage" type="Percentage_Type" />
                                </xs:sequence>
                            </xs:complexType>
                        </xs:element>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 4.11 : Schéma XML des paramètres de performance du nSLA

Le dernier ensemble de paramètres que nous spécifions dans le *nSLA*, à savoir « Business_Parameters », comporte des attributs relatifs à la tarification et à la gestion des violations et des pénalités (voir Figure 4.12).

En ce qui concerne la tarification, l'*IoT-SP* paie le service d'acheminement des données suivant la bande passante souscrite. La gestion des violations et des pénalités est assurée grâce à un intervalle de surveillance et un seuil défini pour chaque paramètre de *QoS* spécifié. Lorsque

l'intervalle arrive à son terme, les paramètres de *QoS* sont mesurés pour tester leur conformité et déterminer les éventuelles pénalités en cas de violations.

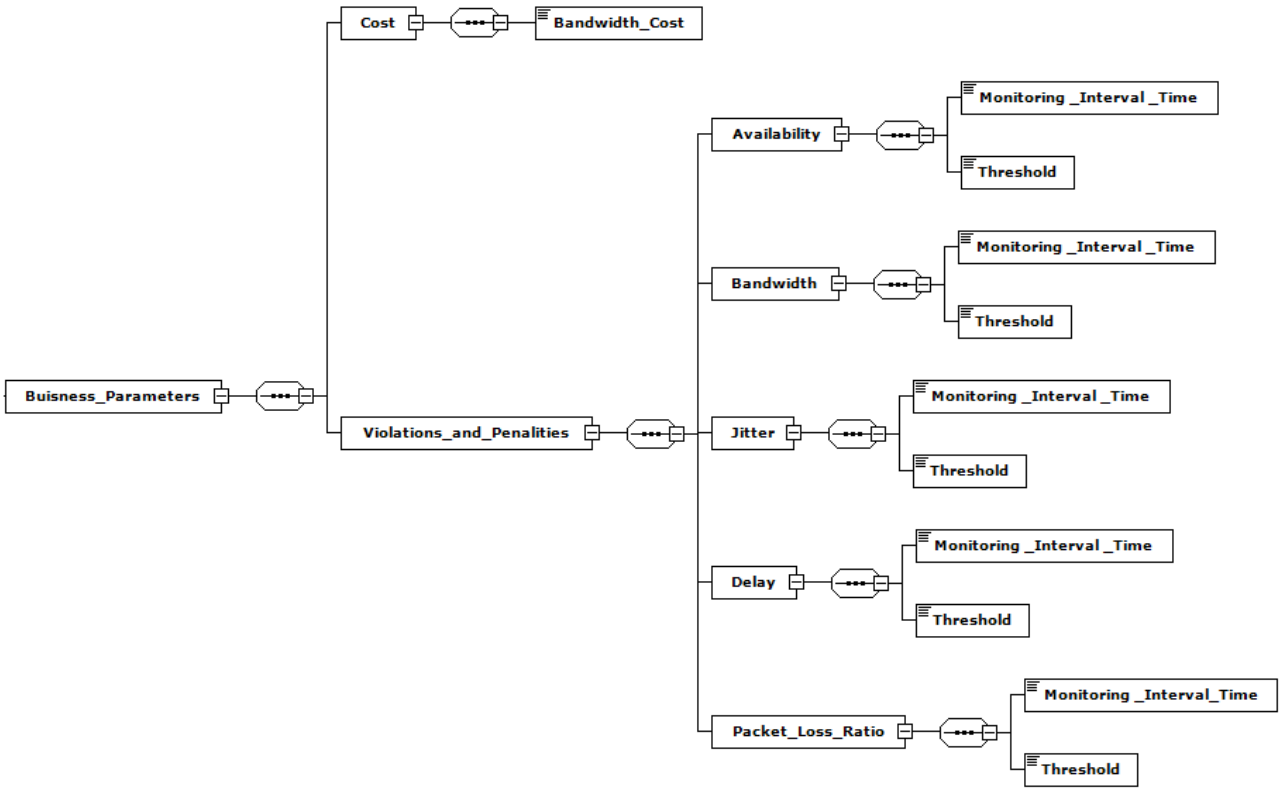

Figure 4.12 : Représentation du schéma XML des paramètres commerciaux du nSLA

4.3.2.3. Procédure d'établissement du nSLA

Différents échanges doivent avoir lieu entre le *NSP* et l'*IoT-SP* pour établir le *nSLA*. Ces échanges permettent aux différentes entités de se mettre d'accord sur les valeurs des paramètres de performances demandées par l'*IoT-SP* et garantis par le *NSP*. Nous spécifions dans la Figure 4.13 l'ensemble des états de l'automate fini (*FSM*) concernant l'établissement du *nSLA* au niveau du *NSP*.

Dans l'état S0, le *NSP* est dans l'attente d'une demande d'établissement d'un contrat de type *nSLA* par un *IoT-SP*. Suite à la réception de cette demande, le *NSP* passe à l'état S1 et boucle sur lui-même pour comparer les ressources demandées par le client avec ses propres ressources. Ainsi, si le *NSP* n'est pas capable de fournir le service en respectant les besoins spécifiques du client, il repasse à l'état S0 en informant le client de son incapacité à satisfaire sa demande. Sinon, le *NSP* passe à l'état S2 en envoyant une offre à l'*IoT-SP*. Si l'offre est rejetée, le *NSP* repasse à l'état S0, sinon il passe à l'état S3 en configurant les besoins du nouveau service sur ses équipements. Dans l'état S3, le *NSP* met à jour la liste de ces ressources disponibles suite à l'acceptation du contrat de type *nSLA* et passe à l'état S0 dans l'attente d'une nouvelle demande. D'autre part, dans l'état S0, le *NSP* peut recevoir une demande de résiliation du *nSLA* de la part de l'*IoT-SP*. Par conséquent, le *NSP* passe à

l'état S3 en libérant les ressources réservées au service résilié et par la suite, il met à jour la liste de ses ressources libres en passant de nouveau à l'état S0.

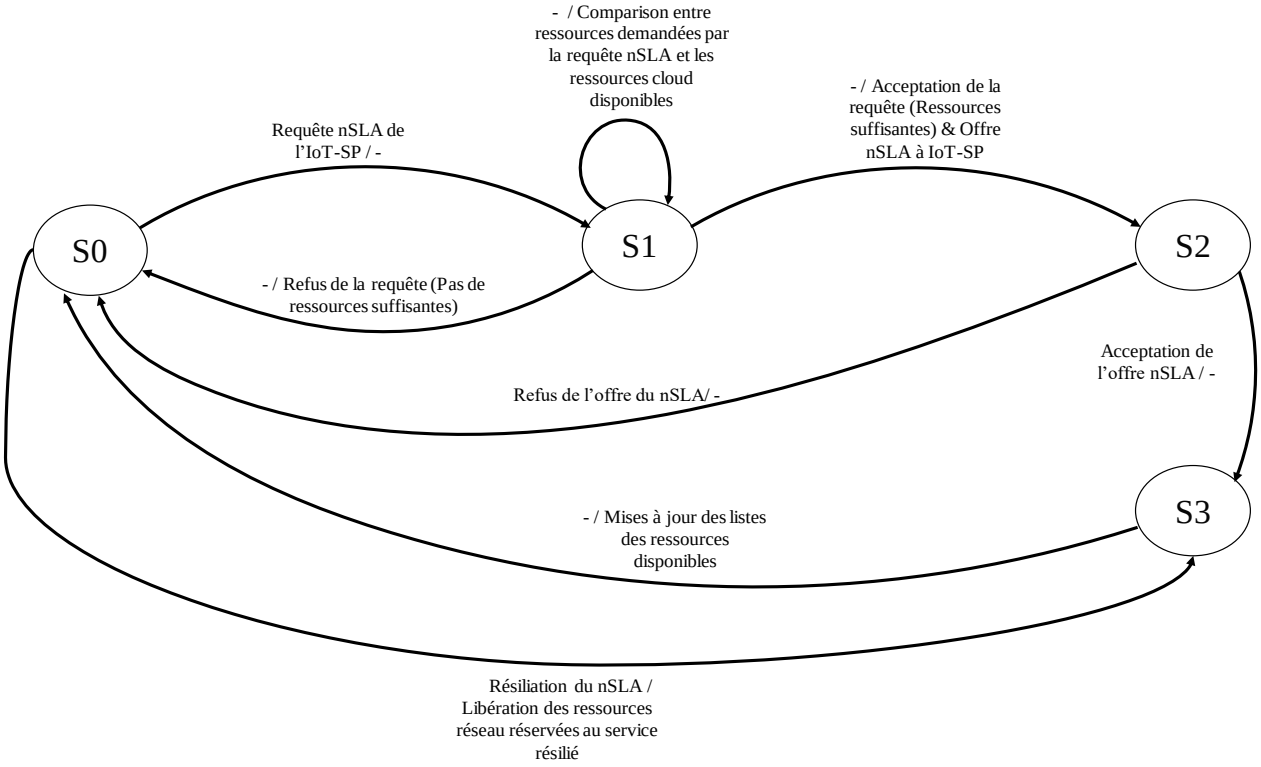

Figure 4.13 : FSM du NSP concernant l'établissement du nSLA

4.3.3. SLA pour la couche sensing : gSLA

4.3.3.1. Besoins du gSLA

Les objets IoT sont directement gérés par les passerelles de bas niveau (*LL-Gw*) en suivant les directives des passerelles de haut niveau (*HL-Gw*). Ces objets jouent un rôle important pour l'offre du service IoT. Ainsi, les informations remontées par les objets IoT sont traitées et stockées au niveau des passerelles *HL-Gw* mais aussi de l'infrastructure cloud. Par conséquent, les paramètres de performance de la couche sensing doivent être pris en compte car ils contribuent au niveau de service global requis par le client du service IoT.

Dans ce contexte, nous proposons le *gSLA*, un contrat interne à l'*IoT-SP*, afin de spécifier les attentes en termes de *QoS* au niveau de la couche sensing (i.e., au niveau de la *HL-Gw*, la *LL-Gw* et les objets) lors de l'offre d'un service IoT. Pour ce faire, nous prenons en considération au niveau de cette couche sensing des paramètres de *QoS* traditionnels tels que le délai, la gigue, la disponibilité, le taux de perte de paquet et la bande passante de la *HL-Gw* vers le cloud spécifiant le débit de sortie des données de la couche sensing. Le débit de sortie des données de la couche sensing ne doit pas dépasser le débit souscrit dans le *nSLA*. Nous prenons aussi en considération d'autres paramètres de

QoS spécifiques à l'environnement IoT de la couche sensing tels que la durée de vie des systèmes d'objets IoT et la qualité des données.

Le contrat de niveau de service de type *gSLA* est identifié par le *gSLA ID* et comporte plusieurs parties (voir Figure 4.14) telles que : la validité, les partenaires, l'identification du service mais aussi les caractéristiques et les paramètres de *QoS* concernant la couche sensing. Le *gSLA* que nous proposons est un contrat interne à l'*IoT-SP* et par conséquent nous ne spécifions pas de paramètres commerciaux relatifs à la tarification et la gestion des violations et des pénalités.

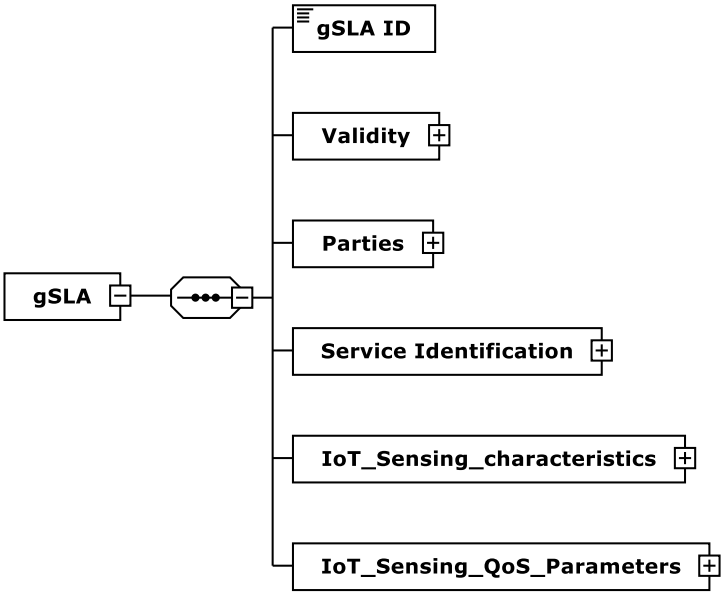

Figure 4.14 *:* Représentation globale du schéma *XML* du gSLA

4.3.3.2. Paramètres du gSLA

Nous représentons dans la Figure 4.15 certaines parties du schéma *XML* du *gSLA* à savoir « Validity », « Parties » et « Service Identification ». Premièrement, nous spécifions la validité du *gSLA* à l'aide d'une date de début obligatoire et une date de fin optionnelle comme le contrat est interne et l'*IoT-SP* joue le rôle du client et du fournisseur, la date de fin n'est pas mandataire comme il sera possible de résilier le contrat sans pénalités.

Par la suite, nous identifions le client du contrat *gSLA* comme étant l'*IoT-SP*, alors que le fournisseur est la passerelle *HL-Gw* détenue par l'*IoT-SP* et identifiée par son adresse *IP*. De plus, nous identifions le service qui fait l'objet de ce contrat de type *gSLA*, non seulement grâce aux *IDs* des *LL-Gw* qui gèrent les objets IoT concernés par ce service (en tant que source), mais aussi grâce à l'*ID* de la *HL-Gw* qui gère ces *LL-Gw* (en tant que destination).

```
<xs:element minOccurs="1" maxOccurs="1" name="gSLA ID" type="xs:integer" />
<xs:element minOccurs="1" maxOccurs="1" name="Validity" >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="StartDate" type="xs:date" />
            <xs:element minOccurs="0" maxOccurs="1" name="EndDate" type="xs:date" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element minOccurs="1" maxOccurs="1" name="Parties"  >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="Client">
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="1" name="IoT-SP" type="xs:string" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element minOccurs="1" maxOccurs="1" name="Provider">
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="1" name="IP_HL-Gw" type="xs:string" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element minOccurs="1" maxOccurs="1" name="Service Identification" >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="Source" >
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="n" name="LL-Gw ID" type="xs:integer"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element minOccurs="1" maxOccurs="1" name="Destination" >
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="1" maxOccurs="1" name="HL-Gw ID " type="xs:integer" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 4.15 : Schéma XML des paramètres d'identification et de validité du gSLA

Nous décrivons dans le *gSLA* les caractéristiques de la couche sensing à travers celles de ses trois composants, à savoir la *HL-Gw*, le groupe de *LL-Gw* et le groupe d'objets IoT (voir Figure 4.16). Nous spécifions dans le *gSLA* les caractéristiques de la passerelle *HL-Gw* en termes de « Fog computing » grâce aux capacités de son processeur (vitesse exprimée en GHz) et de sa mémoire volatile (exprimée en GB). Nous décrivons aussi le nombre de *LL-Gw* gérées par cette *HL-Gw* ainsi que les techniques de recouvrement et de redondance des données utilisées par cette dernière.

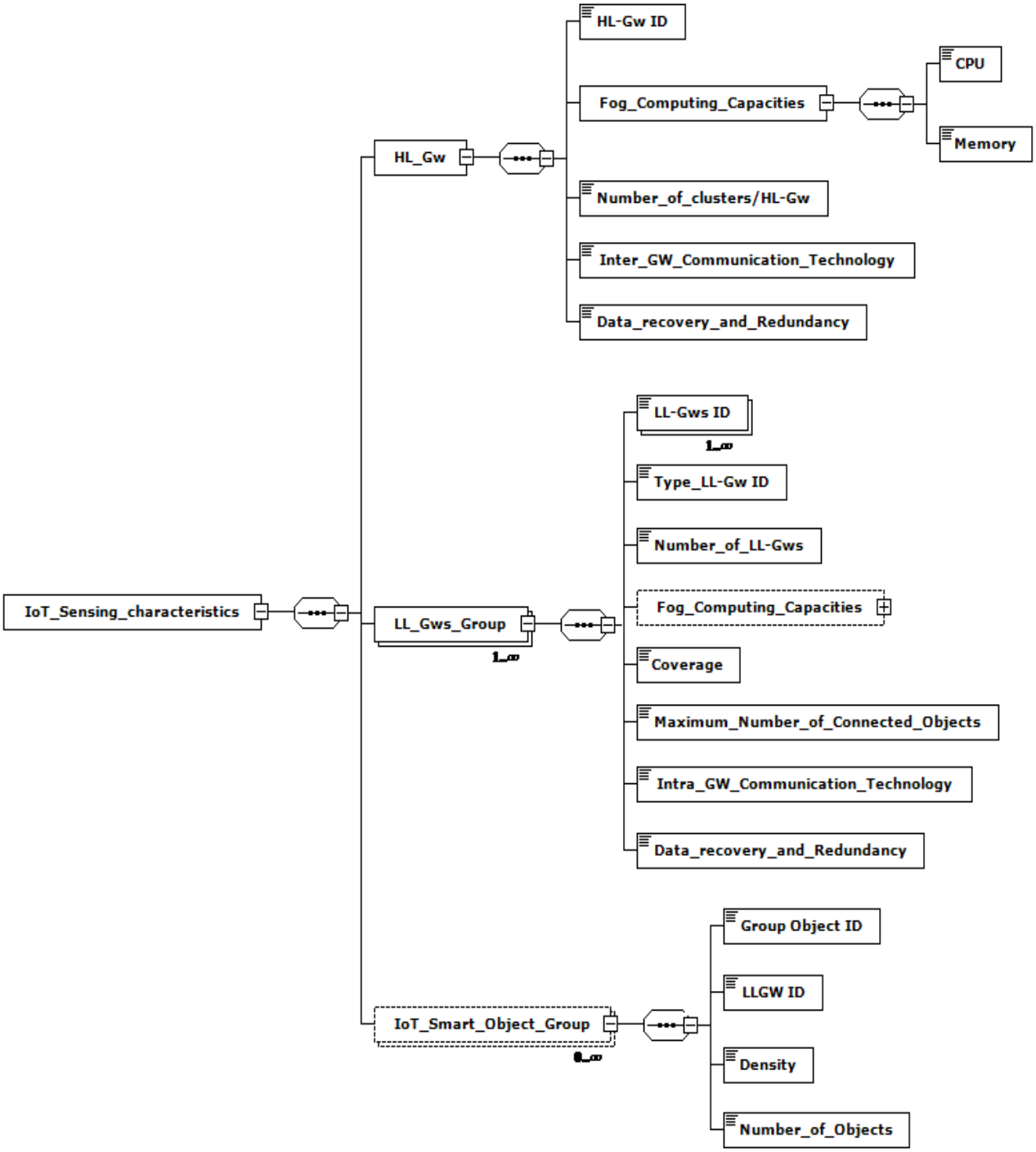

Figure 4.16 : Représentation du schéma XML des caractéristiques de la couche sensing dans le gSLA

Dans ce contexte, un service IoT nécessite obligatoirement une *HL-Gw* fournie par l'*IoT-SP* et par conséquent l'occurrence de l'attribut *HL-Gw* dans le schéma *XML* du *gSLA* est de 1. D'autre part, le deuxième composant des caractéristiques de la couche sensing concerne le groupe de

passerelles de bas niveau. En effet, nous regroupons les *LL-Gw* qui ont les mêmes spécificités (type, nombre maximal d'objets connectés, couverture, capacités de « Fog computing », technologie de communication au sein de la *LL-Gw* et techniques de recouvrement et de redondance) et chaque groupe est identifié d'une façon unique. Enfin, le troisième composant concerne les caractéristiques des groupes d'objets IoT dans le cas où ces objets sont fournis par le fournisseur *IoT-SP*.

Par contre, si les objets IoT de la couche sensing appartiennent au client IoT alors les caractéristiques des groupes d'objets IoT ne sont pas spécifiées dans le contrat de type *gSLA*. Par conséquent, l'occurrence minimale de ce composant dans le schéma *XML* du *gSLA* est de 0 et l'occurrence maximale est infinie comme plusieurs groupes d'objets peuvent coexister lors de l'offre d'un service IoT. Chaque groupe d'objets comprend obligatoirement les informations suivantes : l'identificateur du groupe d'objets, l'identificateur de la passerelle *LL-Gw* gérant le groupe d'objets en question, la densité des objets selon la couverture de la passerelle ainsi que le nombre d'objets connectés (voir Figure 4.16).

Le dernier ensemble de paramètres que nous proposons dans le *gSLA*, à savoir « IoT_Sensing_QoS_Parameters », spécifie les caractéristiques qualitatives et quantitatives des paramètres de QoS de la couche sensing de notre architecture IoT (voir Figure 4.17). Les caractéristiques qualitatives sont obligatoires (occurrence de 1) afin de décrire le type d'application IoT (i.e., *RTMC*, *RTNMC*, Streaming, *NRT*) dont nous garantissons la qualité de service. En effet, le type d'application, spécifié aussi au niveau du *cSLA*, permet de déterminer les caractéristiques quantitatives car chaque type d'application est sensible à un ensemble de paramètres de QoS.

En ce qui concerne les caractéristiques quantitatives, nous considérons un ensemble de paramètres de QoS dont certains sont obligatoires tels que la disponibilité (de la *HL-Gw*, *LL-Gw* et éventuellement les objets IoT), la bande passante de la HL-Gw au cloud et la qualité des données (mesurée par l'écart type (i.e., standard deviation), la marge d'erreur (i.e., standard error) et la fréquence de remontée d'informations (i.e., sensing frequency).

D'autres paramètres de *QoS* sont optionnels et dépendent du type d'application IoT spécifiée dans les caractéristiques qualitatives. Nous citons à ce titre le délai, la gigue, le taux de perte des paquets et la durée de vie des groupes d'objets. Dans ce contexte, le délai ainsi que la gigue au niveau de la couche sensing dépendent de l'acheminement des données depuis les objets à la *LL-Gw* et de la *LL-Gw* à la *HL-Gw* mais aussi du traitement des données au niveau de la *LL-Gw* et de la *HL-Gw*. De plus la disponibilité porte sur les passerelles (i.e., *HL-Gw* et *LL-Gw*) et les objets IoT s'ils sont fournis par l'*IoT-SP*.

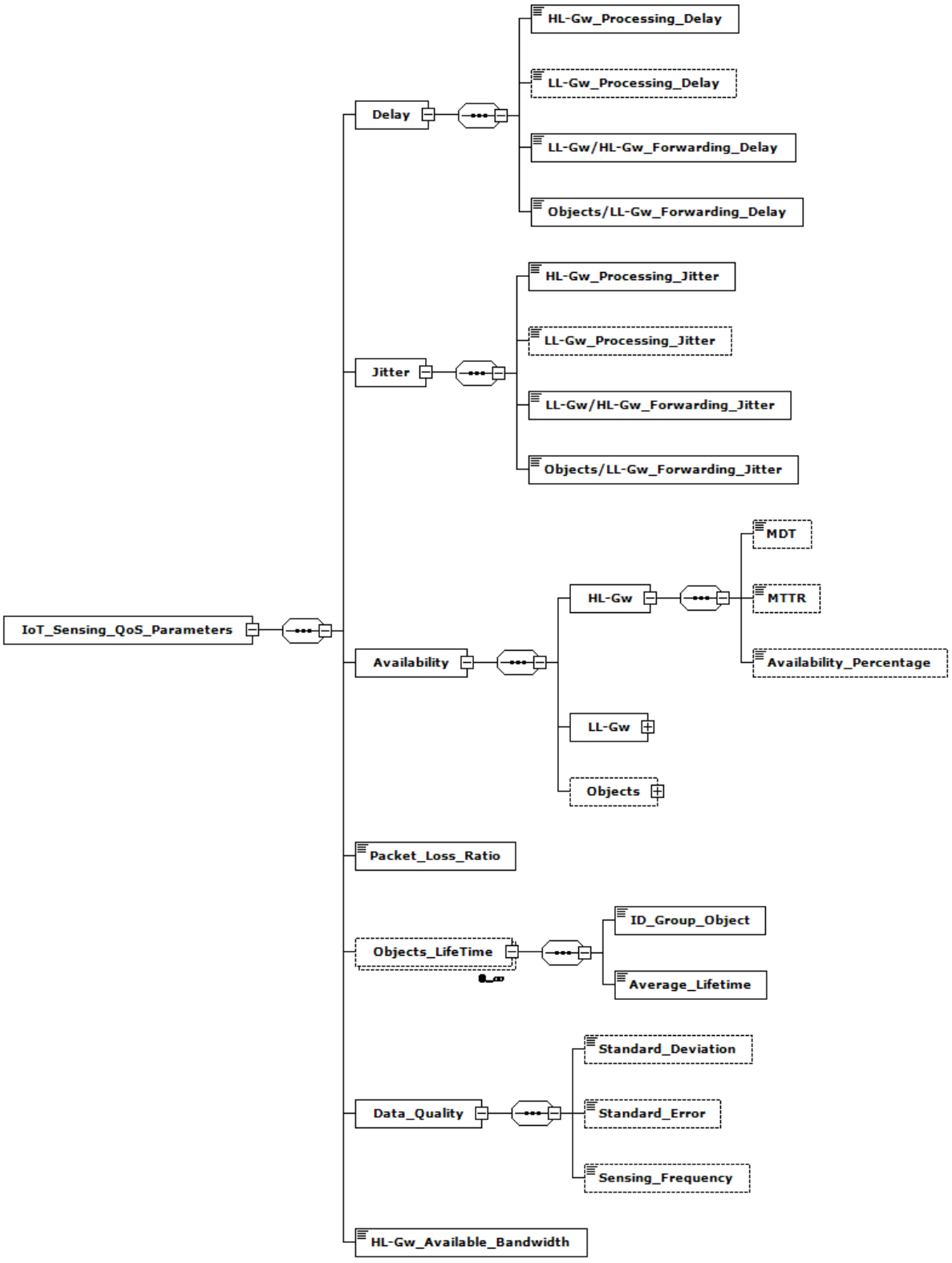

Figure 4.17 : Représentation du schéma XML des paramètres de QoS de la couche sensing dans le gSLA

4.3.3.3. Procédure d'établissement du gSLA

Durant la procédure d'établissement du *gSLA*, le choix entre l'utilisation d'une *HL-Gw* existante ou d'une nouvelle *HL-Gw* doit être effectué afin d'identifier ce type de contrat. Ce choix peut être effectué par l'administrateur du service ou bien d'une manière autonome par la *HL-Gw* elle-même en comparant les ressources de la *HL-Gw* et les ressources requises par le service IoT. Dans le cas où les ressources de la *HL-Gw* sont suffisantes, le contrat est établi et les configurations nécessaires seront exécutées au niveau de la *HL-Gw* et des *LL-Gw* concernées. Nous spécifions dans la Figure 4.18 l'ensemble des états de l'automate fini (*FSM*) concernant l'établissement du *gSLA* au niveau de la *HL-Gw*.

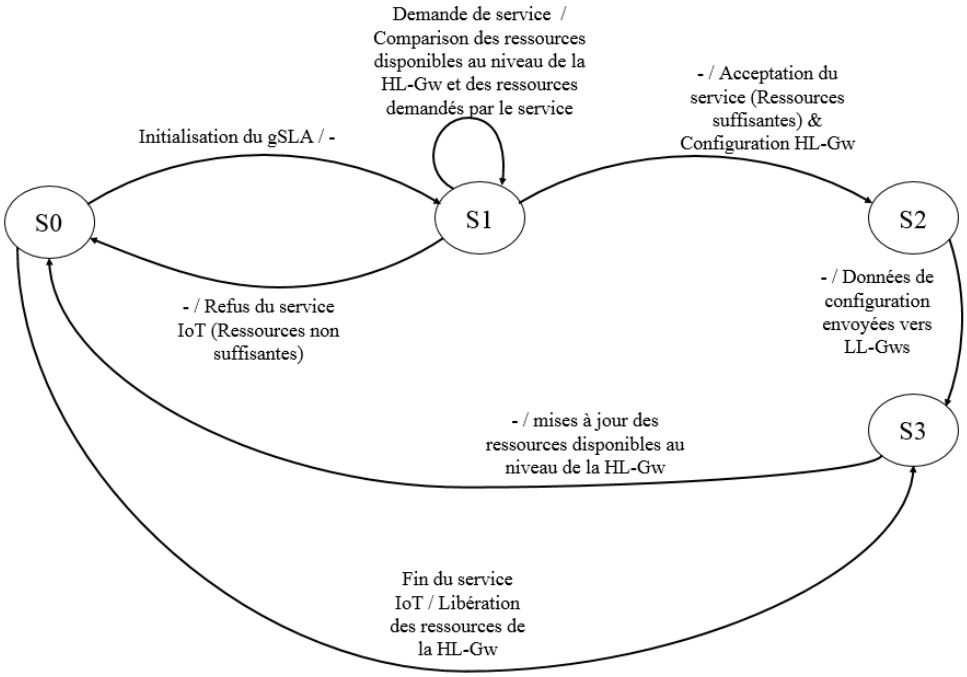

Figure 4.18 : FSM de la HL-Gw concernant l'établissement du gSLA

Dans l'état S0, la *HL-Gw* est dans l'attente de l'établissement d'un contrat de type *gSLA* concernant un nouveau service IoT. Suite à cette demande, la *HL-Gw* passe à l'état S1 où elle compare les ressources demandées par le service IoT et ses ressources disponibles. Si les ressources ne sont pas suffisantes, la *HL-Gw* repasse à l'état S0 en rejetant la requête d'établissement de *gSLA*. Dans ce cas, le *gSLA* en question sera initialisé de nouveau par l'administrateur sur une nouvelle *HL-Gw*. Par contre, si les ressources sont suffisantes alors la *HL-Gw* passe à l'état S2 et procède à son auto-configuration. Par la suite, les données de configuration nécessaires aux *LL-Gw* seront envoyées par la *HL-Gw* en passant à l'état S3. Depuis cet état, la *HL-Gw* met à jour ses ressources disponibles et passe de nouveau à l'état S0. Dans cet état S0, la *HL-Gw* peut recevoir une demande de résiliation d'un *gSLA* déjà établi. Dans ce cas, elle repasse à l'état S3 où elle met à jour ses ressources disponibles avant de revenir à l'état S0.

4.3.4. SLA global pour le service IoT : iSLA

4.3.4.1. Besoins du iSLA

L'*IoT-C* a besoin d'établir avec l'*IoT-SP* un contrat de niveau de service concernant un service IoT afin de demander des garanties pour le niveau de service rendu et se mettre d'accord sur la tarification du service ainsi la gestion des violations et des pénalités. Dans ce contexte, les autres contrats que nous spécifions (i.e., *cSLA*, *nSLA* et *gSLA*) sont transparents pour l'*IoT-C*. En effet, ce dernier s'intéresse au contrat global, appelé *iSLA*, qui doit intégrer des paramètres de *QoS* relatifs aux différentes couches de notre architecture IoT. Ainsi, l'*iSLA* doit prendre en considération les différents *SLA* (i.e., *cSLA*, *nSLA*, *gSLA*) de notre architecture IoT afin de proposer au client (i.e., *IoT-C*) un contrat permettant de caractériser le niveau de service relatif au service IoT d'une façon globale. L'*iSLA* doit comporter plusieurs parties (voir Figure 4.19) telles que : la validité, les partenaires, l'identification du service, les paramètres de performances du service IoT et les paramètres commerciaux du service IoT.

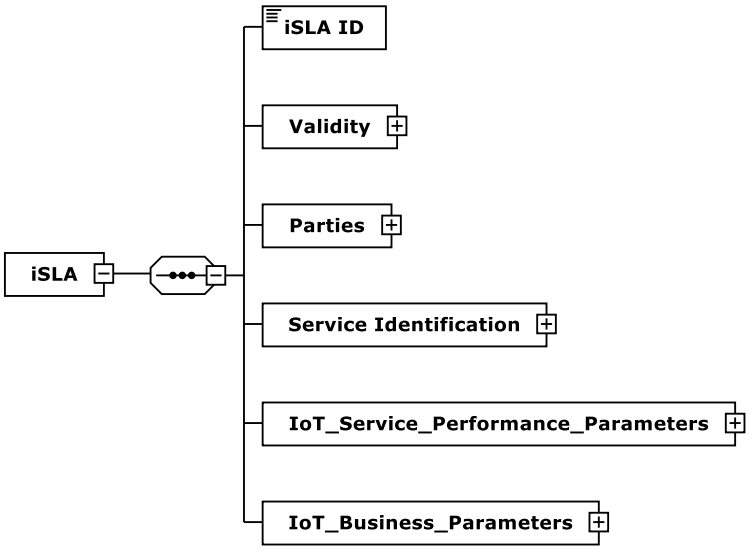

Figure 4.19 : Représentation globale du schéma XML de l'iSLA

4.3.4.2. Paramètres du iSLA

L'iSLA est identifié d'une manière unique par le paramètre *iSLA ID* choisis par l'*IoT-SP*. Nous représentons dans la Figure 4.20 certaines parties du schéma *XML* de l'*iSLA* à savoir « Validity », « Parties » et « Service Identification ».

Premièrement, nous spécifions la validité de ce contrat en utilisant une date de début et de fin de mise en application. Par la suite, nous identifions les deux partenaires du contrat de manière unique par un identifiant, l'ID client est utilisé pour la partie cliente de l'*iSLA* (i.e., l'*IoT-C*) et l'*IoT-SP ID* est utilisé pour la partie fournisseur (i.e., *IoT-SP*). En plus de l'identification des partenaires de ce

contrat de type *iSLA*, nous spécifions le paramètre « Service Identification » pour identifier le service IoT garanti par ce contrat. L'identification du service se base sur les deux attributs « Source » et « Destination ». Ainsi la source du service IoT est identifiée par l'adresse *IP* du client, alors que la destination est identifiée par les attributs de l'application (ID Application, adresse *IP* de l'interface et numéro de port) mais aussi les identifiants de la *HL-Gw* ainsi que les *LL-Gw* concernées par ce service.

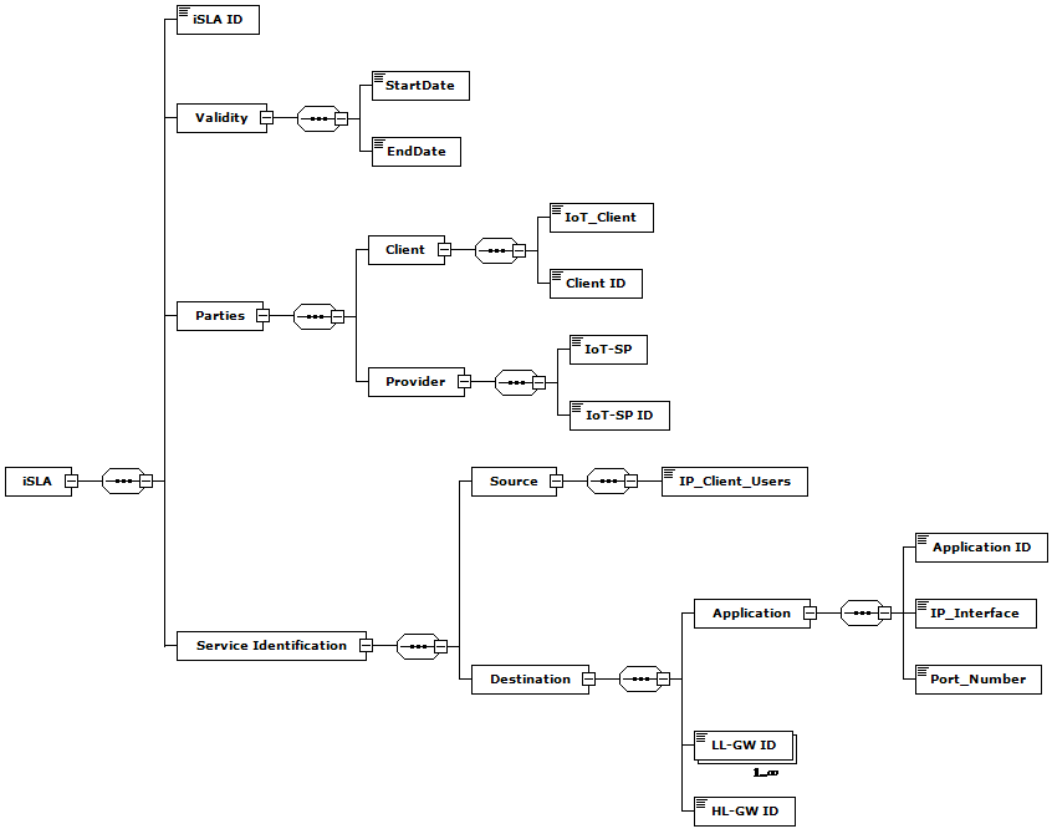

Figure 4.20 : Représentation du schéma XML des paramètres d'identification et de validité de l'iSLA

Une autre partie importante que nous spécifions dans l'i*SLA* est « IoT_Service_Performance_Parameters ». Cette partie comprend plusieurs sous parties : les caractéristiques de la couche sensing, les caractéristiques de l'application et les paramètres de *QoS* (voir Figure 4.21).

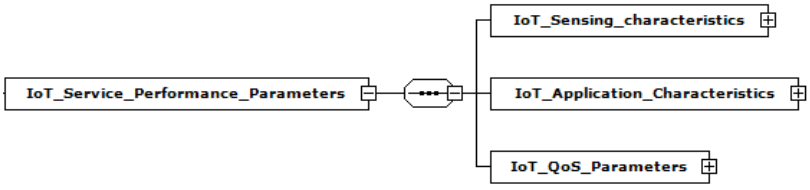

Figure 4.21 : Représentation du schéma XML des paramètres de performance de l'iSLA

En ce qui concerne la première sous partie, à savoir les caractéristiques de la couche sensing, nous reprenons les attributs utilisés dans la même sous partie du *gSLA* (cf. section 4.3.3.2). Ces attributs décrivent des informations relatives à la *HL-Gw*, au groupe de *LL-Gw* et aux objets IoT permettant d'offrir le service IoT au niveau de la couche sensing de notre architecture (voir Figure 4.22).

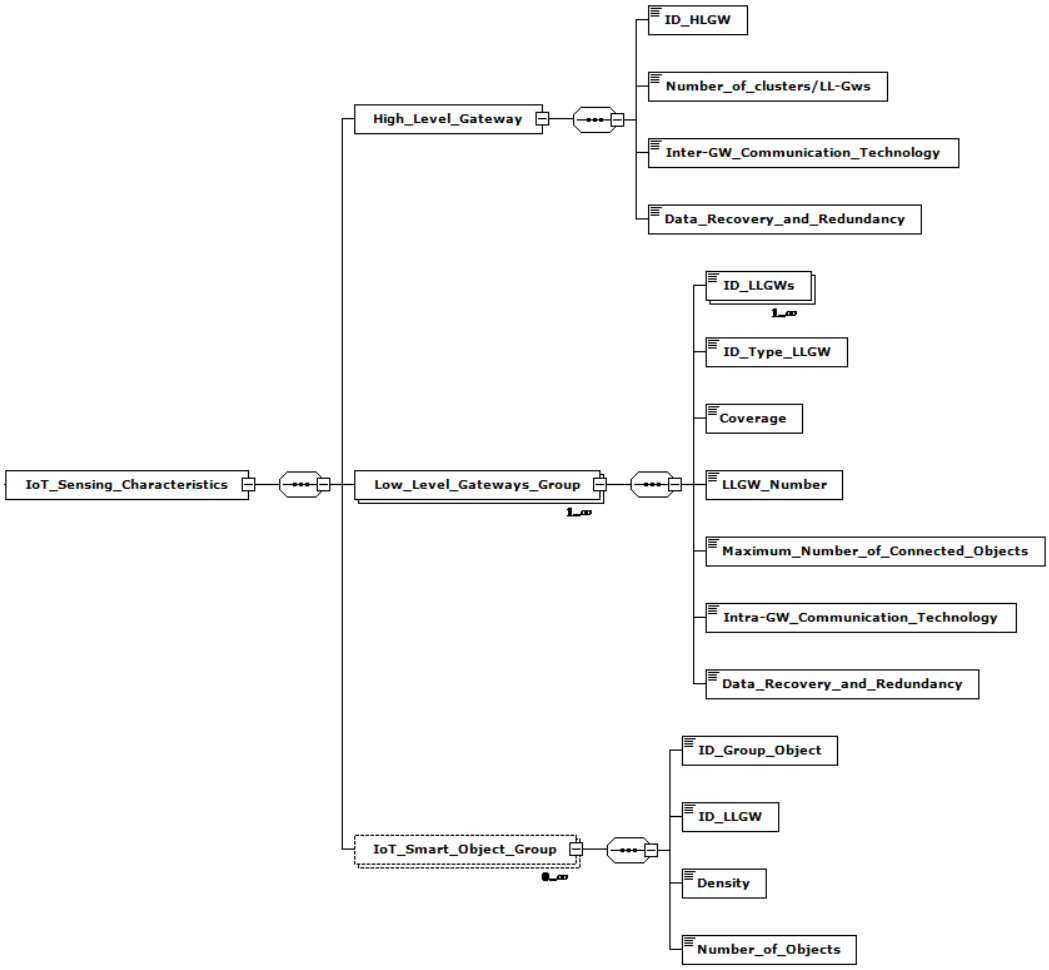

Figure 4.22 : Représentation du schéma XML des paramètres de performance de l'iSLA (couche sensing)

La deuxième sous partie des paramètres de performance du service IoT concerne les caractéristiques de l'application IoT. Cette dernière est soit développée par le *CSP* (à travers la souscription d'un service *SaaS* par l'*IoT-SP* auprès du *CSP*) ou par l'*IoT-SP* (à travers la souscription d'un service *IaaS* ou *PaaS* par l'*IoT-SP* auprès du *CSP*). Ainsi, les garanties correspondantes à cette application sont apportées directement par l'*IoT-SP* ou par l'intermédiaire du *CSP* grâce au *cSLA*. Dans tous les cas, nous spécifions dans la sous partie « IoT_Application_Characteristics » de l'*iSLA* les attributs suivants : l'identifiant *ID Application*, le nombre maximal d'utilisateurs, le nombre moyen de requêtes par seconde et le temps de support qui sont obligatoires. Alors que les caractéristiques de stockage au niveau du cloud qui décrivent les capacités de stockage, les entrées

sorties sur le disque, le format des données et les techniques de redondance de données et de recouvrement sont optionnels (voir Figure 4.23).

```
<xs:element minOccurs="1" maxOccurs="1" name="IoT_Application_Characteristics" >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="Application ID" type="xs:integer" />
            <xs:element minOccurs="1" maxOccurs="1" name="Maximum_Number_of_users" type="xs:integer" />
            <xs:element minOccurs="1" maxOccurs="1" name="Mean_Number_of_requests_per_user" type="xs:integer" />
            <xs:element minOccurs="1" maxOccurs="1" name="Time_of_Support" type="Time_Support_Type" />
            <xs:element minOccurs="1" maxOccurs="1" name="Storage_Parameters">
                <xs:complexType >
                    <xs:sequence>
                        <xs:element minOccurs="0" maxOccurs="1" name="Storage_Capacity" type="Storage_Type" />
                        <xs:element minOccurs="0" maxOccurs="1" name="Disk_I/O" type="I/O_Type" />
                        <xs:element minOccurs="0" maxOccurs="1" name="Data_Format" type="Data_Format_Type" />
                        <xs:element minOccurs="0" maxOccurs="1" name="Data_Redundancy_and_Backup" type="xs:boolean" />
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 4.23 : Schéma XML des paramètres de performance de l'iSLA (application IoT)

La dernière sous partie que nous spécifions dans les paramètres de performance du service IoT concerne les paramètres *QoS* qui peuvent être qualitatifs et obligatoires (i.e., type de l'application) ou quantitatifs et optionnels à l'exception de la disponibilité et de la qualité de données qui sont communs à tous les types de services IoT (voir Figure 4.24).

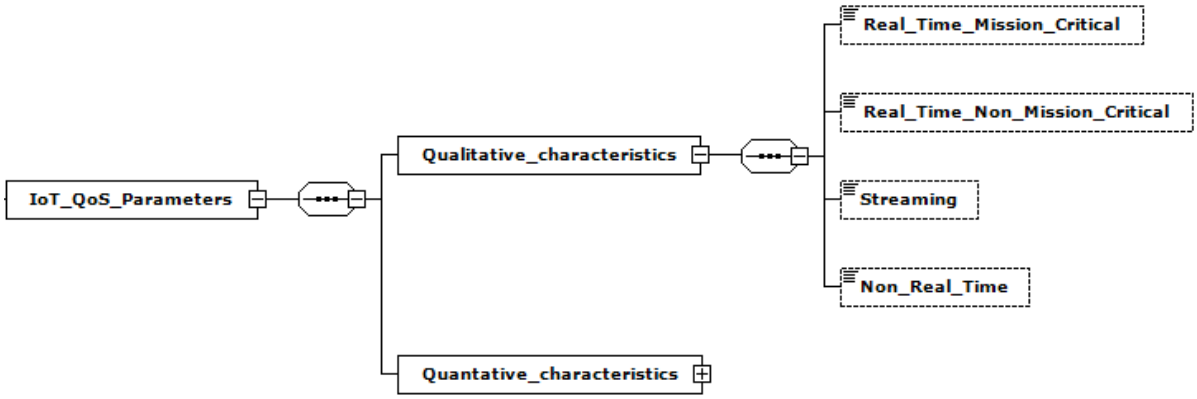

Figure 4.24 : Schéma XML des paramètres de performance de l'iSLA (paramètres de QoS)

De plus, les paramètres de *QoS* que nous spécifions dans le contrat *iSLA* prennent en considération les caractéristiques de toutes les couches de notre architecture (contrairement au *cSLA* avec la couche cloud, le *nSLA* avec la couche réseau et le *gSLA* avec la couche sensing). Nous détaillons dans la Figure 4.25 les paramètres de *QoS* quantitatifs tels que le délai de bout en bout « End_to_End_Delay », la gigue de bout en bout « End_to_End_Jitter », la bande passante « Bandwidth », la disponibilité « Availability », la durée de vie « Lifetime », le taux de perte des paquets « Paket_Loss_Ratio » et la qualité des données « Data_Quality ».

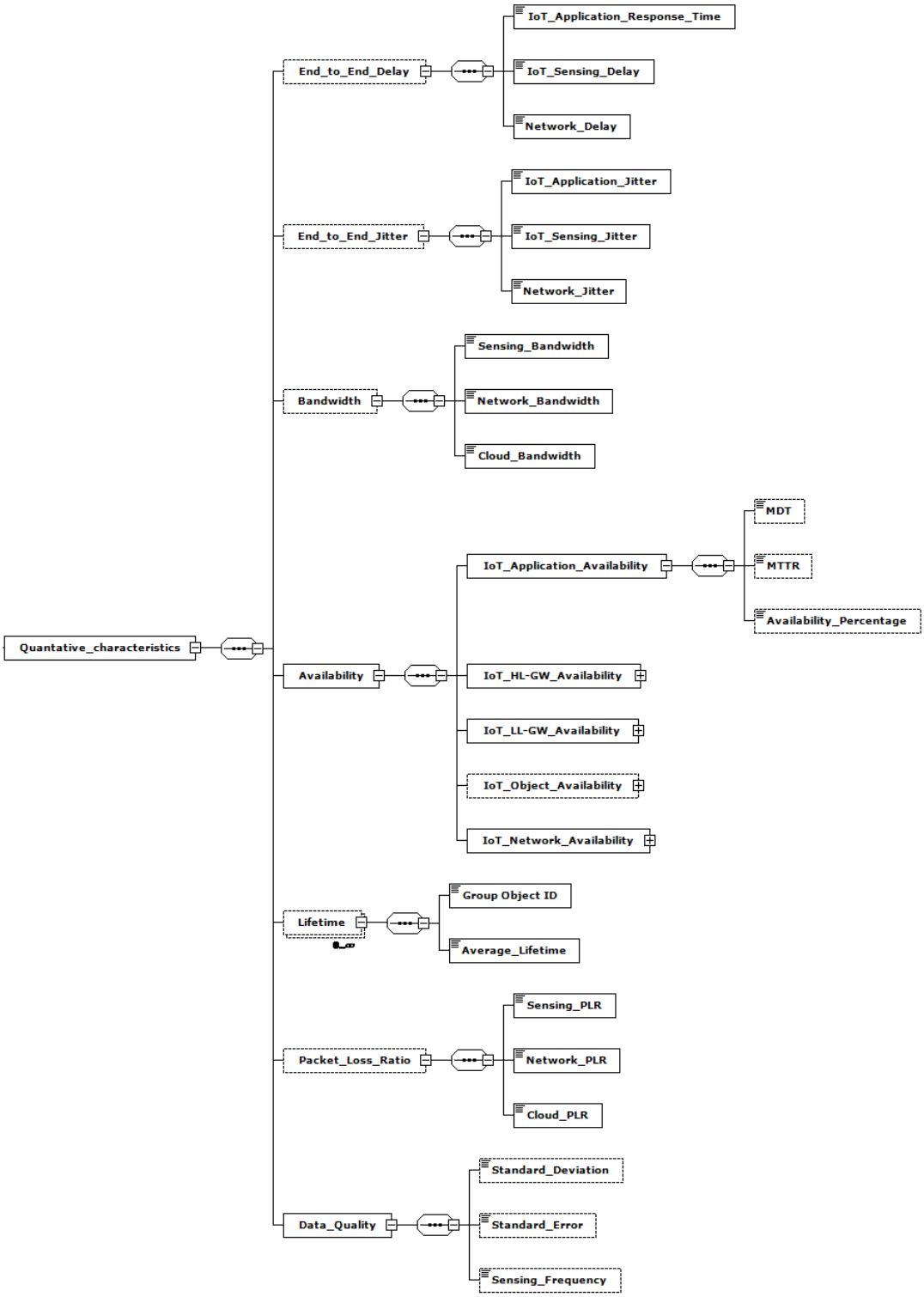

Figure 4.25 : Représentation du schéma XML des paramètres de performance de l'iSLA (paramètres QoS quantitatifs)

Nous définissons dans ce qui suit chacun de ses paramètres de *QoS* spécifiés dans le contexte de l'*iSLA* défini dans la Figure 4.25:

- **Délai de bout en bout (E2E Delay)** : Ce paramètre prend en considération le délai de la couche cloud correspondant au délai de l'application (IoT_Application_RT), le délai de la couche sensing (IoT_Sensing_Delay) correspondant au délai d'acheminement des données et de traitement au niveau des passerelles mais aussi le délai de la couche réseau (Network_Delay) qui correspond au délai d'acheminement des données depuis les passerelles à la couche cloud. Afin de calculer ce délai de bout en bout, nous utilisons l'équation 4.1.

$$E2E\ Delay = \sum_{i=1}^{3} Delay_i = IoT\_Application\_RT + IoT\_Sening\_Delay + Network\_Delay \quad (4.1)$$

- **Gigue de bout en bout (E2E Jitter)** : Ce paramètre prend en considération la gigue au niveau des trois couches de notre architecture. Nous utilisons l'équation 4.2 afin de la déterminer.

$$E2E\ Jitter = \sum_{i=1}^{3} Jitter_i = IoT\_Application\_Jitter + IoT\_Sening\_Jitter + Network\_Jitter \quad (4.2)$$

- **Bande passante (BW)** : Ce paramètre dépend de la bande passante au niveau des trois couches. La bande passante résultante correspond au minimum entre les bandes passantes des couches de notre architecture IoT. Nous calculons ce paramètre de *QoS* en utilisant l'équation 4.3.

$$BW = min(BW_{Sensing}, BW_{Network}, BW_{Application}) \quad (4.3)$$

- **Disponibilité (Availability)** : Ce paramètre prend en considération la disponibilité de toutes les infrastructures de notre architecture IoT. Ainsi, elle intègre la disponibilité de l'infrastructure cloud, la disponibilité de l'infrastructure réseau interconnectant les couches sensing et cloud ainsi que la disponibilité de la couche sensing qui est basée sur la disponibilité des passerelles de haut niveau, de bas niveau et des objets. La disponibilité de chaque infratsructure peut être quantifiée à travers le temps moyen de panne *MDT*, le temps moyen pour réparer la panne *MTTR* ou encore un pourcentage. Dans le cas d'un pourcentage, nous utilisons l'équation 4.4 pour calculer la disponibilité globale alors que nous utilisons les équations 4.5 et 4.6 avec les paramètres *MDT* et *MTTR*.

$$Percentage\ Availability = min(Availability_i) = min\begin{pmatrix} Availability_{Application}, Availability_{HLGw}, \\ Availability_{LLGw}, Availability_{Objects}, Availability_{Network} \end{pmatrix} \quad (4.4)$$

$$MDT_{Global} = max(MDT_i) = max(MDT_{Application}, MDT_{HL-Gw}, MDT_{LL-Gw}, MDT_{Objects}, MDT_{Network}) \quad (4.5)$$

$$MTTR_{Global} = max(MTTR_i) = max\left(MTTR_{Application}, MTTR_{HL-Gw}, MTTR_{LL-Gw}, MTTR_{Objects}, MTTR_{Network}\right) \quad (4.6)$$

- **Durée de vie (Lifetime)** : Ce paramètre est très important pour l'évaluation des groupes d'objets IoT de la couche sensing. Le service IoT spécifié dans contrat *iSLA* peut faire appel à plusieurs groupes d'objets IoT gérés par les passerelles de bas niveau. Chaque groupe d'objets possède une durée de vie moyenne (Lifetime$_i$). Ainsi, nous déterminons la durée de vie du système IoT formé par *n* groupes d'objets concernés par l'offre du service grâce à l'équation 4.7.

$$\text{Lifetime} = \min_{0<i<n}(\text{Lifetime}_i) \quad (4.7)$$

- **Taux de perte des paquets (Packet Loss Ratio)** : ce paramètre correspond au taux de perte global des paquets depuis leurs créations au niveau des objets IoT jusqu'à leur stockage au niveau du cloud. Par conséquent, il faut prendre en considération les pertes des paquets au niveau des différentes couches de notre architecture IoT mais aussi lors de l'acheminement des paquets entre ces couches (i.e., PLR$_{Sensing}$, PLR$_{Network}$ et PLR$_{Cloud}$). Ainsi, nous utilisons l'équation 4.8 pour déterminer le taux de perte global relatif au service IoT qui fait l'objet du contrat i*SLA*.

$$\text{PLR} = 1 - \left[\left(1 - \text{PLR}_{\text{Sensing}}\right) * \left(1 - \text{PLR}_{\text{Network}}\right) * \left(1 - \text{PLR}_{\text{Cloud}}\right) \right] \quad (4.8)$$

- **Qualité des Données (Data Quality)** : ce paramètre ne concerne que la couche sensing qui remonte les données récoltées par les objets IoT. Il peut être mesuré par différentes métriques comme l'écart type (voir Equation 4.9), la marge d'erreur et la fréquence de remontée des informations. Dans l'équation 4.9, N représente la taille de l'échantillon, x_i représente la valeur de l'information remontée et $\overline{x}$ représente la moyenne des valeurs remontée.

$$\text{Ecart Type} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2} \quad (4.9)$$

Le dernier ensemble de paramètres que nous spécifions dans le *nSLA*, à savoir « Business_Parameters », comporte des attributs relatifs à la tarification et à la gestion des violations et des pénalités. Le premier attribut que nous spécifions dans les paramètres commerciaux concerne le coût du service IoT. Grâce à cet attribut, l'*IoT-SP* détermine le coût global du service IoT en prenant en considération ses caractéristiques décrites dans la partie paramètres de performance de l'*iSLA* et par conséquent les ressources utilisées au niveau de chaque couche de notre architecture IoT. Ainsi, le coût global intègre le coût de l'application IoT au niveau de la couche cloud, le coût des ressources au niveau de la couche réseau mais aussi les coûts des passerelles et éventuellement des objets IoT (lorsqu'ils sont fournis par l'*IoT-SP*) au niveau de la couche sensing (voir Figure 4.26).

Figure 4.26 : Représentation du schéma XML des paramètres commerciaux de l'iSLA

Le deuxième attribut des paramètres commerciaux concerne la gestion des violations du niveau de service garanti par l'*iSLA* et des pénalités qui en résultent. Dans ce contexte, nous définissons pour chaque composant d'un paramètre de *QoS*, garanti dans le cadre de l'*iSLA*, un seuil et un intervalle de surveillance afin de vérifier la conformité du niveau de service offert (voir Figure 4.26).

4.3.4.3. Procédure d'établissement de l'iSLA

L'établissement de l'*iSLA* dépend de l'établissement préalable de tous les sous *SLA* (i.e., g*SLA*, *nSLA* et *cSLA*), concernant les différentes couches de notre architecture IoT, afin d'obtenir les informations nécessaires à la mise en place de ce contrat global. Premièrement, le *gSLA* doit être validé en interne au niveau de l'*IoT-SP* pour déterminer l'adresse *IP* de la *HL-Gw* permettant ensuite d'identifier le service du côté source au niveau du *cSLA*. Suite à l'établissement du *gSLA* et du *cSLA* avec le *CSP*, le *nSLA* est établi avec le *NSP* afin de permettre l'interconnexion de la couche sensing et cloud de notre architecture IoT. Nous spécifions dans la Figure 4.27 l'ensemble des états de l'automate fini (*FSM*) de l'*IoT-SP* concernant l'établissement du contrat global de type *iSLA*.

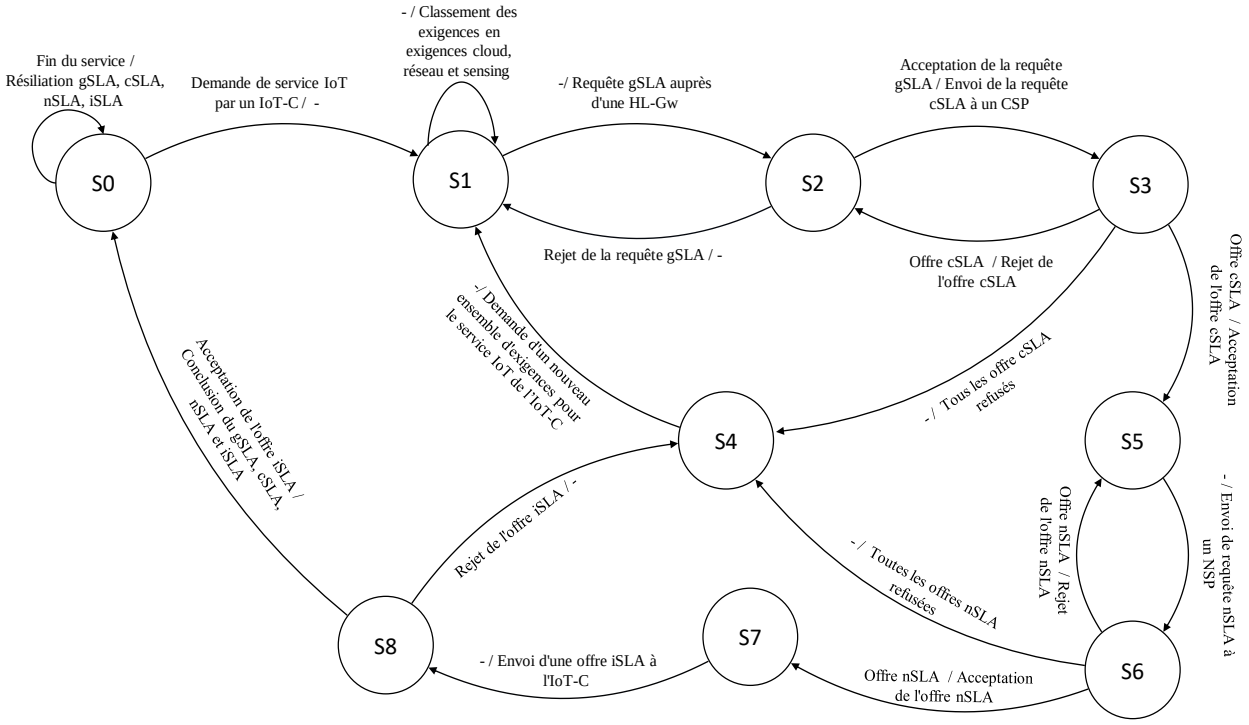

Figure 4.27 : FSM de l'IoT-SP concernant l'établissement de l'iSLA

Comme le montre la Figure 4.27, l'*IoT-SP* passe de l'état S0 à l'état S1 lorsqu'il reçoit de la part d'un *IoT-C* une demande de service IoT avec les exigences correspondante. Dans l'état S1, l'*IoT-SP* classe ces exigences du service IoT selon chaque couche de notre architecture IoT et donc en exigences cloud, réseau et sensing. Par la suite, l'*IoT-SP* initie l'établissement du *gSLA* avec une *HL-Gw* et passe à l'état S2. Dans le cas de refus du *gSLA* de la part de la *HL-Gw* à cause de ressources insuffisantes, l'*IoT-SP* revient à l'état S1 et initie l'établissement du *gSLA* avec une nouvelle *HL-Gw* en passant de nouveau à l'état S2. Dans le cas de l'acceptation du *gSLA* *l'IoT-SP* envoie une requête *cSLA* à un *CSP* et passe à l'état S3. Dans cet état, l'*IoT-SP* attend le retour du *CSP*. Ce dernier envoi une offre *cSLA*. Si cette offre est rejetée par l'*IoT-SP* alors il passe de nouveau à l'état S2 et la requête de *cSLA* est envoyée à un nouveau *CSP*. Dans le cas où toutes les offres des différents *CSP* sont

rejetées par l'*IoT-SP*, ce dernier passe de l'état S3 à l'état S4 afin de demander au client d'adapter ces exigences de service et passer à l'état S1 de nouveau. Sinon, si l'offre *cSLA* est acceptée, l'*IoT-SP* passe de l'état S3 à l'état S5. Depuis cet état, l'*IoT-SP* initie l'établissement du *nSLA* avec un *NSP*. Ainsi, l'*IoT-SP* passe à l'état S6 en envoyant une requête *nSLA* à un *NSP*. Si l'offre envoyée par le *NSP* est rejetée, l'*IoT-SP* repasse à l'état S5 et tente d'envoyer une nouvelle requête vers un nouveau *NSP*. Si toutes les offres de tous les *NSP* sont rejetées, alors l'*IoT-SP* passe à l'état S4 et demande au client d'adapter ces exigences de service IoT en passant à l'état S1 de nouveau. Par contre, si l'*IoT-SP* accepte l'offre d'un *NSP* alors il passe à l'état S7. Depuis cet état, l'*IoT-SP* envoie une offre *iSLA* à l'*IoT-C* et passe à l'état S8 dans l'attente de son retour. Si l'*IoT-C* rejette l'offre *iSLA*, l'*IoT-SP* passe à l'état S4 pour demander à l'*IoT-C* d'adapter ses exigences et revient à l'état S1. Si l'offre *iSLA* est acceptée par l'*IoT-C*, l'*IoT-SP* repasse à l'état S0 dans l'attente d'une une nouvelle demande de service IoT après avoir conclu les contrats *cSLA*, *nSLA* et *iSLA* avec le *CSP*, *NSP* et *IoT-C* respectivement. L'*IoT-SP* attend la fin du service pour résilier les contrats *SLA* correspondant et rester dans l'état S0.

L'établissement de l'*iSLA* nécessite différents échanges entre les composants de notre architecture IoT afin de se mettre d'accord sur les paramètres et les valeurs à considérer dans les sous *SLA*. Nous décrivons dans la Figure 4.28 les échanges effectués entre les composants de l'architecture IoT grâce à un diagramme de séquence de messages (*MSC* : Message Sequence Chart) en considérant le cas où aucune requête de *SLA* n'a été refusée (de la part de la *HL-Gw*, *CSP* et *NSP*).

Le *MSC* de la Figure 4.28 montre que suite à la réception de la demande de service IoT avec les caractéristiques correspondantes (type d'application, délai de bout en bout, disponibilité, nombre d'utilisateurs de l'application, nombre d'objets connectés, qualité des données), l'*IoT-SP* détermine et classe les besoins du service en fonction des couches cloud, réseau et sensing. Par la suite, l'*IoT-SP* demande à une *HL-Gw* l'établissement d'un *gSLA*. Après la vérification de ses ressources, la *HL-Gw* accepte l'établissement du *gSLA* et met à jour ses ressources disponibles. L'étape suivante consiste à choisir le type de service cloud à demander au *CSP*. Suite à ce choix, l'*IoT-SP* envoie une requête *cSLA* au *CSP* en spécifiant les exigences attendues du service choisi. Dans le cas de demande d'un service de type *IaaS*, la requête *cSLA* de l'*IoT-SP* spécifie les caractéristiques de stockage, des mémoires et des processeurs, le nombre de machines virtuelles, la disponibilité et le temps de réponse. Si un service de type *PaaS* est requis, la requête *cSLA* de l'*IoT-SP* spécifie la liste des systèmes d'exploitation et des plateformes de développement, le nombre d'utilisateurs simultanés de la plateforme, les caractéristiques de stockage, la disponibilité et le temps de réponse. Enfin, dans le cas où un service *SaaS* est demandé au *CSP*, la requête *cSLA* de l'*IoT-SP* spécifie le type de l'application, le nombre d'utilisateurs de l'application, les capacités de stockage, la disponibilité et le temps de réponse. Le *CSP* accepte la requête et envoie une offre suivant le service demandé. L'acceptation de cette offre par l'*IoT-SP* lui permet de communiquer avec le *NSP* pour établir un *nSLA* garantissant les ressources réseau entre la *HL-Gw* déjà identifiée et l'infrastructure cloud avec laquelle il vient de conclure le *cSLA*. Ainsi, l'*IoT-SP* envoie une requête *nSLA* en précisant la bande passante, le délai, la gigue et la disponibilité attendus de l'infrastructure réseau. Cette requête sera alors traitée par le

NSP qui envoie une offre *nSLA* à l'*IoT-SP*. Lorsque cette offre est acceptée par l'*IoT-SP*, ce dernier prépare à son tour une offre *iSLA* afin de l'envoyer au client *IoT-C*.

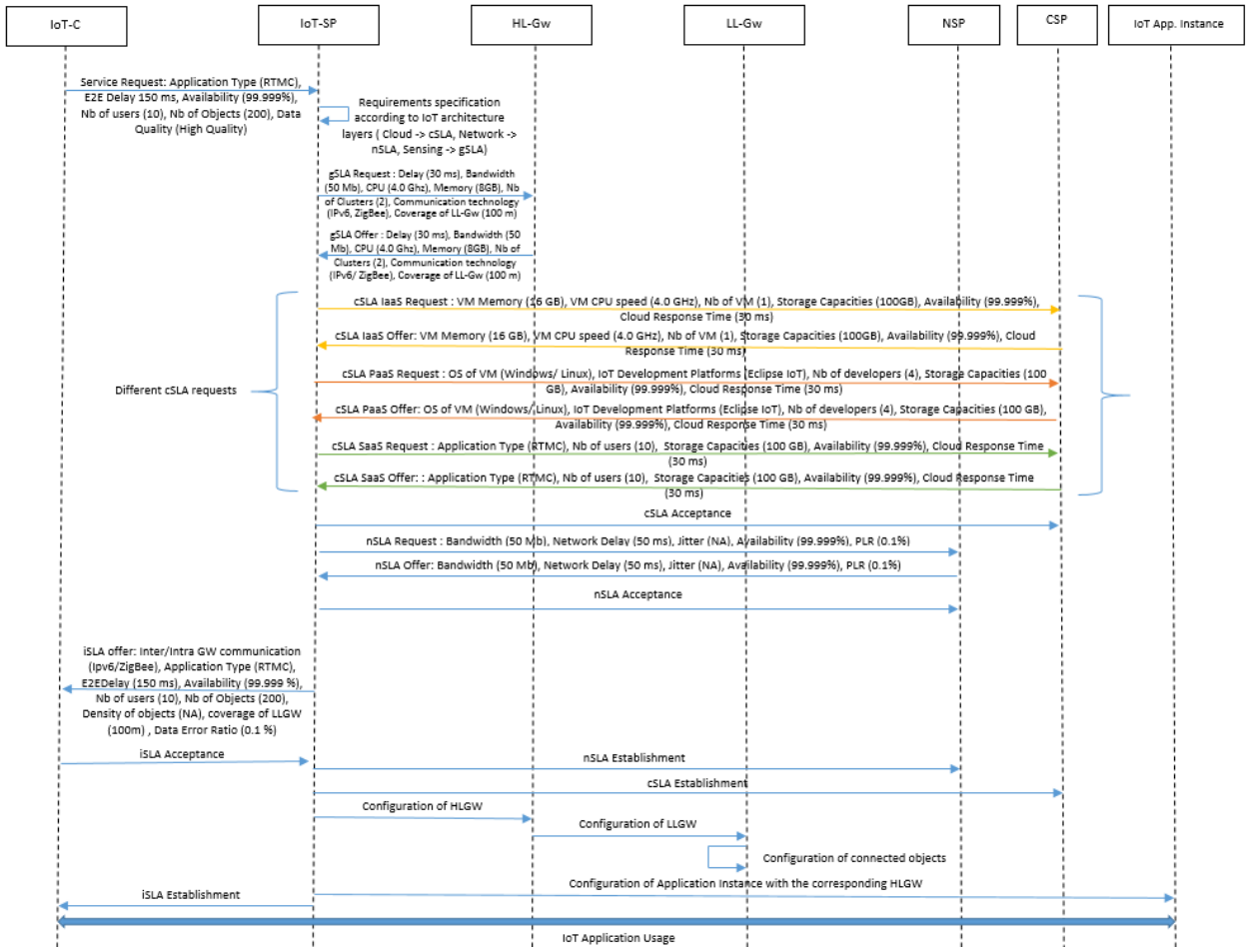

Figure 4.28 : MSC d'établissement de l'iSLA

Cette offre intègre toutes les informations nécessaires à l'*IoT-C* (technologies de communication au sein de la couche sensing, le type d'application, les caractéristiques du service, les valeurs des paramètres *QoS* garantis, etc.). Lorsque cette offre *iSLA* est acceptée par l'*IoT-C* alors les contrats *nSLA* et *cSLA* sont définitivement conclus avec le *NSP* et le *CSP* respectivement. Par la suite, la *HL-Gw* et l'application IoT sont configurées par l'*IoT-SP* et les *LL-Gw* concernées par l'*iSLA* sont configurées par la *HL-Gw*. Ces configurations permettent à l'*IoT-SP* de valider définitivement l'*iSLA* avec l'*IoT-C* qui va pouvoir commencer à utiliser le service IoT avec les garanties spécifiées dans cet *iSLA* qui vient d'être établi.

4.4. Conclusion

A travers ce chapitre, nous avons spécifié dans un premier temps notre architecture IoT basée sur la *QoS* et formée par trois couches différentes, à savoir la couche sensing, la couche réseau et la

couche cloud. Notre architecture IoT repose sur différentes entités et des contrats de niveau de service (i.e., *SLA*) que nous proposons d'établir dans le cadre de cette architecture formant ainsi notre Framework de garantie de niveau de service dans un environnement Internet des objets. Dans ce contexte, nous avons spécifié trois types de contrats (i.e., *gSLA*, *nSLA* et *cSLA*) établis avec les fournisseurs cloud, réseau et en interne. Ces *SLA* concernent la garantie du niveau de service respectivement, dans les couches sensing, réseau et cloud de notre proposition d'architecture IoT. De plus, nous avons utilisé ces contrats afin spécifier un *SLA* global appelé *iSLA* que nous établissons entre l'*IoT-C* et l'*IoT-SP* en prenant en considération toutes les couches de notre architecture IoT lors de l'offre d'un service IoT. Ainsi, l'*iSLA* permet de négocier les différents besoins en termes de *QoS* requis par le l'*IoT-C* auprès d'un *IoT-SP* pour l'offre d'un service IoT Dans ce contexte, nous avons spécifié et présenté les processus d'établissement de ces différents contrats en commençant par le *cSLA* mais aussi le *nSLA* et *gSLA* avant de finir avec le contrat global de type *iSLA*. La spécification d'une architecture IoT basée sur la *QoS* et des *SLA* dans ce chapitre va nous permettre de mieux maîtriser la contribution que nous présentons dans le chapitre suivant et qui porte sur la spécification d'une méthode de contrôle d'accès basée sur la *QoS* dans un environnement IoT.

Chapitre 5 – Spécification de QBAIoT : une méthode de contrôle d'accès au canal basée sur la QoS dans l'IoT

5.1. Introduction

Le contrat de type *iSLA*, établi entre l'I*oT-C* et l'*IoT-SP* dans l'architecture IoT que nous avons spécifié dans le chapitre précédent a pour objectif de satisfaire les besoins en termes de *QoS* des différents trafics et en particulier au niveau de la couche sensing de cette architecture. Pour ce faire, des mécanismes au niveau de chaque couche de l'architecture IoT doivent être mis en place afin d'assurer un traitement différencié des différents trafics. Au niveau de la couche sensing de l'architecture IoT, différents protocoles de communication sont utilisés pour acheminer les informations entre les objets et les passerelles. Dans ce contexte, nous proposons à travers ce chapitre, une méthode de contrôle d'accès basée sur la *QoS* assurant un traitement différencié des trafics des objets utilisant Le standard IEEE 802.15.4 au sein d'un environnement IoT. Cette méthode, appelée *QBAIoT* (QoS based Access for IoT environments), est une adaptation du mécanisme du contrôle d'accès au canal de la norme IEEE 802.15.4.

Ainsi, nous décrivons dans la section suivante le standard IEEE 802.15.4 en mettant l'accent sur les techniques de contrôle d'accès au canal qui seront la base de notre contribution. Cette étude va nous permettre de spécifier dans la section 3 les détails de conception de notre mécanisme *QBAIoT* et en particulier les algorithmes à appliquer au niveau des passerelles *LL-Gw* et des objets permettant la différenciation des trafics. Par la suite, nous évaluons dans la section 4 les performances de *QBAIoT* selon différents scénarios tout en montrant son efficacité par rapport à d'autre travaux de recherche dont le standard IEEE 802.15.4 et ce en considérant différents paramètres de *QoS* (délai moyen, taux de livraison des paquets, etc.). Enfin, la section 5 nous permet de conclure ce chapitre.

5.2. Le standard IEEE 802.15.4

5.2.1. Utilisation du standard IEEE 802.15.4 dans l'IoT

Nous avons présenté dans le chapitre 2 plusieurs protocoles de communications qui peuvent être utilisés pour assurer la communication entre les objets IoT et les passerelles de bas niveau de type *LL-Gw*. Dans ce contexte, le standard IEEE 802.15.4 constitue la base de plusieurs de ces protocoles. En effet, ce standard définit les couches basses (couche physique et couche liaison de données) utilisées par des protocoles tels que ZigBee et 6LowPAN qui spécifient les couches hautes (couche réseau, couche transport, couche application). Ces protocoles basés sur le standard IEEE

802.15.4 assurent les communications dans des environnements sans fil à faible débit et à faible portée (*LR-WPAN* : Low Rate Wireless Personal Area Network) et peuvent servir diverses applications IoT dans différents domaines tels que l'e-santé. En effet, plusieurs projets et travaux de recherche proposent l'utilisation de ZigBee pour interconnecter les objets IoT dans le domaine de l'e-santé. Ainsi, les auteurs proposent dans [163] un système de surveillance des soins de santé en utilisant la technologie sans fil ZigBee afin de fournir des informations en temps réel sur l'état de santé des patients. De plus, le travail de recherche développé dans [164] spécifie un système de surveillance continu des patients basé sur le protocole ZigBee pour la communication entre les objets. La plateforme Zolertia [165], fournissant des solutions logicielles et matérielles de 6LoWPAN pour les applications IoT, propose des solutions utilisées pour les maisons intelligentes afin de permettre la connexion aux objets IoT à distance et assurer l'automatisation de certaines tâches. De plus, selon Zolertia, 6LowPAN est utilisé dans l'agriculture intelligente pour assurer une connexion entre des capteurs éloignés en les connectant les uns aux autres dans les zones isolées.

5.2.2. Description technique du standard IEEE 802.15.4

La couche physique (*PHY)* et la sous couche *MAC (Media Access Control)* du standard IEEE 802.15.4 spécifient des paramètres essentiels tels que le débit de données, les fonctions de contrôle, le format des trames de gestion des données et l'utilisation d'une méthode de contrôle d'accès au canal parmi six disponibles [166]. Ce standard fournit également d'autres fonctionnalités de gestion telles que l'accès au canal, la synchronisation, etc. Le standard IEEE 802.15.4 spécifie six méthodes d'accès au canal différentes dans sa dernière version de 2015 [166] :

- ***Unslotted CSMA/CA*** (Carrier Sense Multiple Access / Collision Avoidance) ***in beacon enabled PAN*** (Personal Area Network)
- ***Slotted CSMA/CA in beacon enabled PAN***
- ***TSCH CCA*** (Time Slotted Channel Hopping Clear Channel Assessment) ***in non-shared slots***
- ***TSCH* CSMA/CA in shared slots**
- ***CSMA/CA with PCA*** (Priority Channel Access)
- ***LECIM (***Low-Energy Critical Infrastructure Monitoring) ***ALOHA (***Abramson's Logic of Hiring Access) ***with PCA***

Nous décrivons dans ce qui suit brièvement ces différentes méthodes d'accès, à l'exception de la méthode ***slotted CSMA /CA*** qui sera détaillé puisqu'elle constitue la base de notre contribution. Ainsi, la première méthode, à savoir ***Unslotted CSMA/CA***, correspond à une méthode d'accès traditionnelle permettant de tester la disponibilité du canal avant d'essayer d'envoyer les données. En ce qui concerne la deuxième méthode, à savoir ***Slotted CSMA/CA***, une structure de trame virtuelle spécifique, appelée supertrame (Superframe), est utilisée. Cette supertrame (voir Figure 5.1) comporte une partie active appelée durée de supertrame (*SD* : Superframe Duration) divisée en 16 périodes de temps appelées créneaux (*Slots*) ou Intervalle de Temps (*IT*) et une période inactive

facultative pendant laquelle tous les nœuds sont inactifs (sleep). La période active est formée par une période d'accès en contention (*CAP* : Contention Access Period) et une période optionnelle sans contention (*CFP* : Contention Free Period). Pendant la *CAP*, les nœuds entrent en concurrence pour accéder au canal partagé, alors que pendant la *CFP*, des créneaux garantis (*GTS* : Guaranteed Time Slots) sont attribués aux objets. À chaque intervalle de balise (*BI* : Beacon Interval) correspondant à la fin de la durée de la supertrame, le coordinateur envoie une balise à tous les nœuds dans son réseau sans fil personnel (*WPAN* : Wireless Personal Area Network). Cette balise permet au coordinateur d'identifier ses nœuds, de les synchroniser et de communiquer différentes valeurs de configuration, telles que l'ordre des balises (*BO* : Beacon Order) et l'ordre des supertrames (*SO* : Superframe Order). Les deux paramètres *BO* et *SO* sont utilisés pour calculer l'intervalle de balise (*BI*) et la durée de supertrame (*SD*) selon les équations 5.1 et 5.2 respectivement. La durée de base d'une supertrame (*BSFD* : Base SuperFrame Duration) est la durée minimale d'une supertrame (*SD*), correspondant à une valeur *SO* de 0 (voir Equation 5.2). Elle est fixée à 960 symboles de 4 bits chacun. De plus, *BO* et *SO* doivent respecter l'inégalité $0 \leq SO \leq BO \leq 14$.

$$BI = BSFD \times 2^{BO} \quad (5.1)$$

$$SD = BSFD \times 2^{SO} \quad (5.2)$$

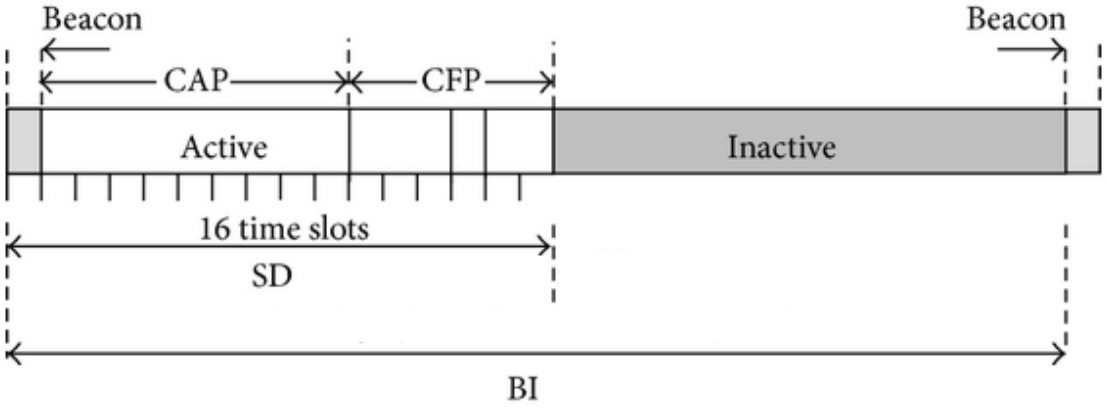

Figure 5.1 : Supertrame IEEE 802.15.4

La troisième méthode, à savoir ***TSCH CCA***, utilisée dans le cas de lien non partagé, se base sur une balise avancée comportant des informations complémentaires concernant la synchronisation des informations *TSCH*, le saut de canal, les plages horaires *TSCH*, etc. Les appareils fonctionnant avec ***TSCH CCA*** utilisent le saut de canal pour éviter les collisions, ce qui élimine le recours aux périodes d'interruption (*BP* : Backoff Period). Lorsqu'un dispositif *TSCH* a un paquet à transmettre, il doit attendre une liaison avec le dispositif de destination. Si *TSCH CCA* est défini sur « ON », la sous couche *MAC* demande à la couche *PHY* d'exécuter un test *CCA* pour pouvoir envoyer les données. Quant à la quatrième méthode ***TSCH CSMA/CA***, elle est utilisée dans le cas de liens partagés (i.e., shared slots). Les liens partagés sont attribuées à plusieurs périphériques, ce qui peut entraîner des collisions et des retransmissions. Pour réduire la probabilité de retransmissions multiples du même paquet, l'algorithme assurant la minimisation des retransmissions est utilisé dans la méthode ***TSCH CSMA/CA***. D'autre part, la méthode ***CSMA/CA PCA*** est utilisée avant la transmission des

messages critiques. Dans cette méthode, la sous couche *MAC* garantit que les opérations de contrôle de collisions *CSMA/CA* restantes et les opérations de transmission peuvent être effectuées avant la fin de la *CAP*. Le mécanisme *PCA* correspond à l'algorithme du choix des périodes d'interruptions (i.e., Backoff Algorithm) à utiliser. Enfin, la dernière méthode appelée ***LECIM ALOHA PCA*** est utilisée lorsque le mode *CCA* est défini à 4. Différents modes pour le *CCA* sont disponibles (1 à 4) afin de déterminer la disponibilité du canal. Le mode *CCA* 4 signale toujours un canal libre. Il est utilisé dans le cas des applications à faible cycle de travail (Low Duty Cycle). ***LECIM ALOHA PCA*** utilise une version modifiée de l'algorithme du choix des périodes d'interruptions ***PCA*** [166] propre à cette méthode et permettant d'étendre la durée des Backoff Periods.

La version 2015 du standard IEEE 802.15.4 utilise une extension de la structure de la supertrame appelée extension multicanal déterministe et synchrone (*DSME* : Deterministic and Synchronous Multi-Channel Extension). Cette dernière étend l'utilisation de *GTS* à tous les périphériques réseau en utilisant plusieurs canaux pour l'attribution des *IT* dans des réseaux denses. Cette extension accroît la complexité de la coordination des créneaux horaires et complique le processus permettant d'éviter les incohérences. Ainsi, la complexité de calcul permet alors une consommation énergétique plus importante [166].

Après avoir décrit les six méthodes d'accès au canal du standard IEEE 802.15.4, nous détaillons dans ce qui suit l'algorithme *CSMA/CA* à créneaux ou à *IT* (i.e., Slotted *CSMA/CA*) utilisé dans la deuxième méthode du standard et qui est la base de notre contribution de contrôle d'accès avec *QoS*. Cet algorithme, représenté par la Figure 5.2, est utilisé par chaque nœud lors de la transmission d'un paquet de contrôle ou de données.

L'algorithme *slotted CSMA/CA* est exécuté uniquement pendant la *CAP*. Dans ce contexte, trois variables sont spécifiées [166] :

- Exposent d'interruptions (*BE* : Backoff Exponent) : Cette variable permet de déterminer le délai d'attente observé par les nœuds avant d'effectuer le test *CCA* sur le canal partagé. La valeur du délai d'attente est choisie au hasard entre 0 et 2^{BE} -1.
- Fenêtre de contention (*CW* : Contention Window) : cette variable indique le nombre de *BP (Backoff Period)* pendant lesquelles le canal doit être libre avant qu'un nœud puisse accéder au canal pour envoyer ses données. La valeur par défaut de *CW* est 2.
- Nombre de périodes d'interruptions (*NB* : Number of Backoffs) : Cette variable correspond au nombre de *BP* observées avant l'accès au canal. La variable *NB* est initialisée à 0 et elle est comparée à une valeur maximale, à savoir « *macMaxCSMABackoffs* », qui est par défaut égale à 5. Si la valeur de *NB* est supérieure à la valeur maximale, un échec se produit.

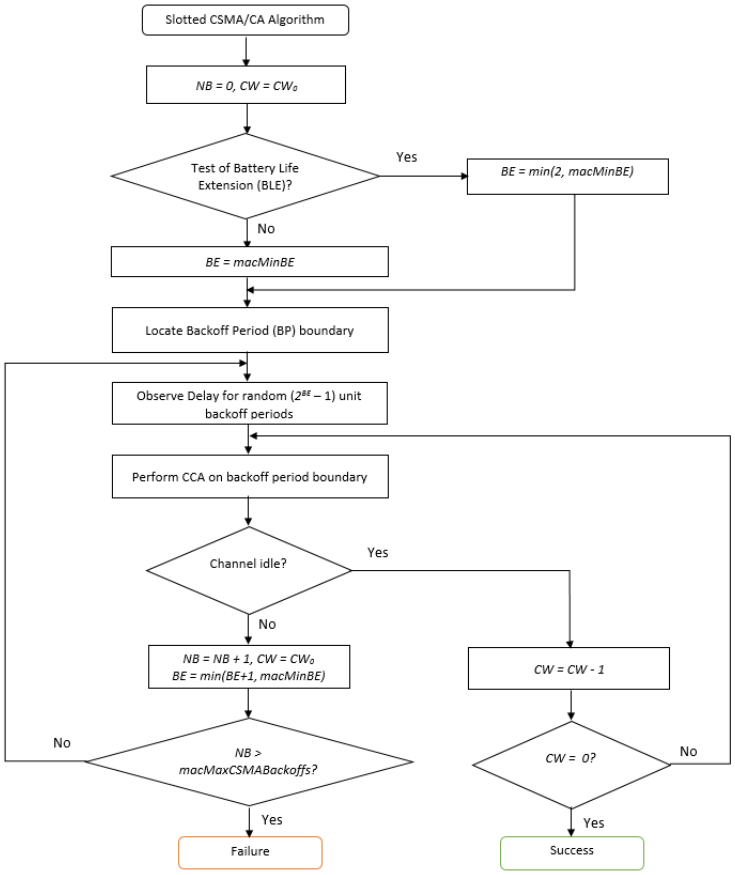

Figure 5.2 : Diagramme de l'algorithme slotted CSMA/CA

L'algorithme *slotted CSMA/CA* est exécuté uniquement pendant la *CAP*. Dans ce contexte, trois variables sont spécifiées [166] :

- Exposent d'interruptions (*BE* : Backoff Exponent) : Cette variable permet de déterminer le délai d'attente observé par les nœuds avant d'effectuer le test *CCA* sur le canal partagé. La valeur du délai d'attente est choisie au hasard entre 0 et 2_{BE} -1.
- Fenêtre de contention (*CW* : Contention Window) : cette variable indique le nombre de *BP (Backoff Period)* pendant lesquelles le canal doit être libre avant qu'un nœud puisse accéder au canal pour envoyer ses données. La valeur par défaut de *CW* est 2.
- Nombre de périodes d'interruptions (*NB* : Number of Backoffs) : Cette variable correspond au nombre de *BP* observées avant l'accès au canal. La variable *NB* est initialisée à 0 et elle est comparée à une valeur maximale, à savoir « *macMaxCSMABackoffs* », qui est par défaut égale à 5. Si la valeur de *NB* est supérieure à la valeur maximale, un échec se produit.

Ainsi, comme le montre la Figure5.2, la valeur de *NB* est initialisée à 0 et celle de *CW* à la valeur « CW_0 » correspondant à 2 par défaut. Après l'initialisation, l'algorithme slotted *CSMA/CA* teste la valeur de la variable booléenne d'extension de la durée de vie de la batterie (*BLE* : Battery

Life Extension). Si la valeur de *BLE* est fixée à vraie, alors la valeur de *BE* est définie au minimum entre 2 et « *macMinBE* » dont la valeur est fixée par défaut à 3. Dans le cas où la valeur de *BLE* est fixée à faux, alors *BE* est initialisé à 2. Après avoir fixé les valeurs des différentes variables, un nœud implémentant l'algorithme slotted *CSMA/CA* observe le délai d'attente et effectue le premier test *CCA*. Deux états de canal possibles existent :

- Le canal est occupé, *CW* est par conséquent réinitialisé à « CW_0 » s'il a été modifié. De plus, un canal occupé entraîne l'incrémentation de *NB* et de *BE*. *BE* ne doit pas dépasser la valeur « *aMaxBE* » fixée par défaut à 5. Lorsque la valeur NB dans le nœud atteint la valeur « *macMaxCSMABackoffs* », l'algorithme *CSMA/CA* à créneaux signale une défaillance à la couche supérieure. Sinon, si *NB* n'a pas atteint la valeur « *macMaxCSMABackoffs* », le délai d'attente avant le test est observé par le nœud et ensuite le test *CCA* est à nouveau exécuté.
- Le canal est libre et *CW* est supérieur à 0. Dans ce cas le *CCA* est répété et la valeur de *CW* est décrémentée. Par contre, si le canal est libre et la valeur de *CW* est égale à 0 alors le nœud tente de transmettre si le temps restant dans la *CAP* actuelle est suffisant pour transmettre la trame et recevoir l'accusé de réception. Sinon, le processus est reporté à la prochaine *CAP* de la prochaine supertrame.

5.3. Spécification de notre méthode de contrôle d'accès : QBAIoT

Après avoir décrit dans la section précédente les caractéristiques techniques du standard IEEE 802.15.4 et présenté dans le chapitre 3 l'importance de la garantie de la *QoS* dans un environnement IoT, nous détaillons dans cette section la spécification de notre contribution appelée *QBAIoT* qui vient enrichir le Framework de garantie de niveau de service dans un environnement IoT que nous avons proposé dans le chapitre 4. En effet, *QBAIoT* est une méthode de contrôle d'accès basée sur la *QoS* pour les nœuds IoT utilisant les protocoles de communication basés sur ce standard IEEE 802.15.4.

5.3.1. Différenciation au niveau de la couche Sensing

Comme présenté dans le Framework du chapitre 4, les applications IoT peuvent appartenir à 4 classes différentes: *RTMC*, *RTNMC*, *Streaming* et *NRT*. Chaque classe correspond à un ensemble de caractéristiques communes que nous traduisons par des besoins en termes de *QoS*. A titre d'exemple, les classes temps réels (*RTMC* et *RTNMC*) sont sensibles au délai et à la gigue alors que la classe Streaming est sensible à la variation du délai et au débit. Par conséquent, pour assurer la différenciation entre les classes de trafics QoS, le protocole de communication assurant l'échange de tous types d'informations à partir de la couche basse de l'architecture IoT (i.e., couche sensing) doit être conçu d'une manière à privilégier certaines classes en assurant un traitement différencié. Afin d'offrir cette différenciation de traitement de trafic et de garantir les valeurs des paramètres de *QoS* négociées dans le contrat de niveau de service de type *iSLA* et en particulier sa composante relative à la couche sensing (i.e., *gSLA*), nous proposons une nouvelle méthode de contrôle d'accès au canal

partagé basée sur la QoS. Cette méthode, appelée *QBAIoT*, est une adaptation de la méthode ***Slotted CSMA/CA*** du standard IEEE 802.15.4.

5.3.2. Spécification de la Supertrame QBAIoT

La méthode *QBAIoT* prend en considération jusqu'à 4 classes de *QoS* tel que défini dans les paramètres qualitatifs de l'*iSLA* (i.e., *RTMC*, *RTNMC*, *Streaming* et *NRT*). Pour adapter la structure de la supertrame IEEE 802.15.4 à cette notion de classes de QoS, *QBAIoT* spécifie une nouvelle supertrame qui remplace la *CAP*, commune à tous les objets, ainsi que la *CFP* par des *QoS CAP* (voir Figure 5.3).

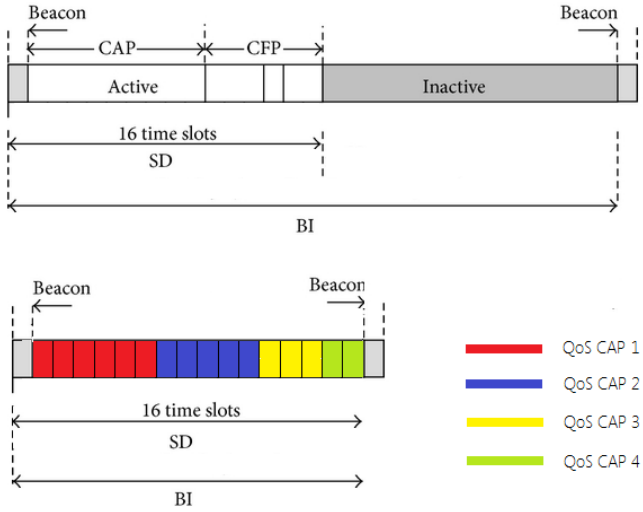

Figure 5.3 : Structure de la supertrame IEEE 802.15.4 et celle de QBAIoT

Chaque *QoS CAP* est spécifique à une classe de *QoS* et sera donc uniquement utilisée par les objets IoT générant un trafic appartenant à cette classe de *QoS*. Ainsi, la nouvelle supertrame *QBAIoT* adaptée peut contenir jusqu'à 4 *QoS CAP* à la place de la *CAP* et la *CFP* de la supertrame IEEE 802.15.4. Ces *QoS CAP* sont configurées au niveau de la passerelle de bas niveau (*LL-Gw*) de notre Framework décrit dans le chapitre 4. Le nombre de *QoS CAP* configurées dans une passerelle *LL-Gw* dépend du nombre de classes *QoS* définies dans les *iSLA* concernant cette passerelle. De plus, la partie inactive de la supertame classique IEEE 802.15.4 est éliminée dans le contexte de *QBAIoT*. Cette suppression entraîne la réduction du délai d'acheminement au niveau de la couche sensing et améliore par conséquent les performances des trafics correspondant aux classes temps réel. D'autre part, le nombre de créneaux (*IT*) disponibles dans la supertrame est fixé à 16 et la durée de chaque *IT* dépend de la durée de la supertrame. La durée de la supertrame (*SD*) est calculée en fonction de la valeur de l'ordre de la supertrame (*SO*). L'ordre de balise (*BO*) est utilisé pour le calcul de l'intervalle d'envoi des balises (*BI*). Dans le cadre de *QBAIoT*, *BO* et *SO* sont toujours équivalents comme la période inactive est supprimée. La Figure 5.3 représente une comparaison entre la structure de la supertrame IEEE 802.15.4 standard et notre adaptation de supertrame *QBAIoT*.

Pendant chaque *QoS CAP* de *QBAIoT*, seuls les objets qui génèrent un trafic appartenant à la classe *QoS* correspondante peuvent entrer en concurrence pour accéder au support partagé. Chaque *QoS CAP* se voit attribuer un nombre d'*IT* parmi les 16 disponibles dans la supertrame. La configuration des *IT* et des valeurs de *BO* et *SO* dépendent du nombre de classes *QoS* existantes dans l'environnement IoT considéré et en particulier du nombre de classes *QoS* temps réel. Ainsi, si une seule classe de *QoS* existe dans l'environnement IoT, *QBAIoT* définit une seule *QoS CAP* et réutilise la structure de la supertrame IEEE 802.15.4 normale mais sans période *CFP* et sans période inactive. Dans ce cas les valeurs de *BO* et *SO* sont fixées à la valeur 14 pour minimiser le nombre de balises envoyées dans l'environnement IoT. La durée de chaque *IT* est de 15.72 s ce qui fait une durée *SD* de 251.52 s. Par conséquent, une balise est envoyée chaque 4 minutes lorsqu'une seule classe existe dans un environnement IoT.

Dans le cas où plusieurs classes existent dans un environnement IoT, *BO* et *SO* sont initialisées à une valeur égale à 2 s'il y a au moins une classe de *QoS* temps réel (*RTMC* ou *RTNMC*), et à une valeur égale à 3 s'il n'y a pas de classes de *QoS* temps réel. En effet, la valeur 2 pour *BO* et *SO* permet de minimiser les délais d'attente pour que les objets IoT soient dans leurs *QoS CAP* respectives à nouveau ce qui permet de minimiser le délai moyen d'attente des paquets des classes temps réel avant d'être servis. Cette configuration permet d'avoir une durée *IT* de 3.84 ms ce qui fait une durée SD de 61.44 ms. D'autre part, la valeur 3 pour *BO* et *SO*, est utilisée dans le cas de non présence de classes temps réel. Cette valeur entraine des délais plus importants pour des classes non sensibles à ce paramètre de *QoS* mais aussi l'envoi d'un nombre moins important de balises libérant ainsi le canal pour l'envoi des données utiles. Cette configuration permet d'avoir un *IT* d'une durée de 7.68 ms ce qui fait une durée *SD* de 122.88 ms.

Afin de configurer la supertrame *QBAIoT* en fonction des caractéristiques de l'environnement IoT, nous réutilisons les champs de la balise IEEE 802.15.4 (voir Figure 5.4). En effet, cette balise comporte un champ « Superframe Specification » qui inclut les informations concernant le *BO*, le *SO*, adresse du coordinateur, etc. En ce qui concerne la configuration des *QoS CAP*, nous utilisons le champ « beacon payload » de la balise pour la communiquer aux objets IoT implémentant la méthode *QBAIoT*.

Octets: 2	**1**	**4/10**	**variable**	**2**	**variable**	**variable**	**variable**	**2/4**
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS Info	Pending address	Beacon Payload	FCS
MHR				MAC Payload				MFR

Figure 5.4 : Structure de la balise IEEE 802.15.4 [166]

La méthode *QBAIoT* est implémentée au niveau de la passerelle *LL-Gw* (agissant en tant que coordinateur) ainsi qu'au niveau des objets IoT de la couche sensing de notre architecture globale IoT. Nous décrivons ci-après les détails de conception de la méthode *QBAIoT* basée sur la *QoS* au

niveau de ces deux types d'entités de notre Framework de garantie de niveau de service dans un environnement IoT.

5.3.3. Spécification du processus *QBAIoT* au niveau de la passerelle LL-Gw

La passerelle *LL-Gw* de notre architecture IoT prend en considération le mécanisme *QBAIoT* proposé en tant que méthode d'accès pour les objets IoT permettant une garantie de *QoS* conformément à un contrat de type *iSLA* conclu entre l'*IoT-SP* et l'*IoT-C*. A travers la Figure5.5, nous illustrons l'ensemble des états de l'automate fini (*FSM*) concernant le processus *QBAIoT* au niveau de la *LL-Gw*.

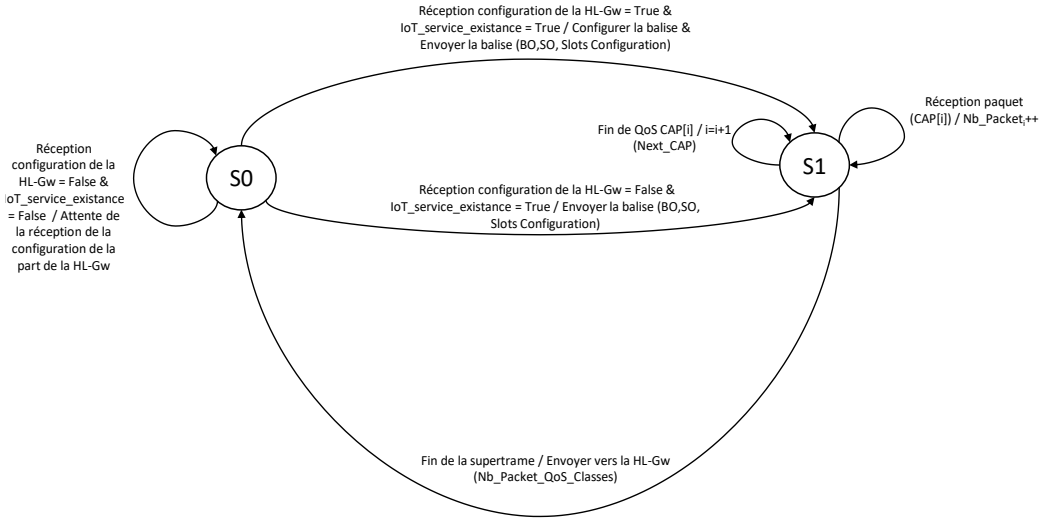

Figure 5.5 : FSM de la LL-Gw concernant QBAIoT

Ainsi, la *LL-Gw* dans l'état S0 est dans l'attente de données de configuration *QBAIoT* émanant de la *HL-Gw*. Dans cet état S0, si la *LL-Gw* reçoit une configuration émanant de la *HL-Gw* concernant *QBAIoT* pour un service IoT décrit dans un *gSLA*, alors elle configure les informations correspondantes de la balise et envoie cette dernière aux objets IoT avant de passer à l'état S1. Dans l'état S1, la *LL-Gw* reçoit des données des différents objets en fonction de leurs *QoS CAP* respectives. La *LL-Gw* évalue le nombre de paquets reçus au cours de chaque *QoS CAP*. Cette évaluation sera utilisée par une fonction d'auto-optimisation que nous proposons dans le chapitre 8. À la fin de chaque supertrame, le nombre de paquets reçus au cours des différentes *QoS CAP* est communiqué à la *HL-Gw* ce qui permet à la *LL-Gw* de revenir à l'état S0. Dans cet état S0, la *LL-Gw* va envoyer de nouveau la balise avec la même configuration si elle ne reçoit pas une nouvelle configuration *QBAIoT* de la *HL-Gw*.

L'algorithme *QBAIoT* au niveau de la *LL-Gw* est présenté à travers la Figure 5.6. Dans ce contexte, la *LL-Gw* implémentant notre méthode d'accès spécifie la configuration de la supertrame

QBAIoT dans la balise en fonction des valeurs reçues de la *HL-Gw*. Ensuite, cette balise est envoyée aux objets IoT s'il existe au moins un trafic dans l'environnement de la *LL-Gw*. L'existence de trafic peut être déduite à partir du nombre de *gSLA* configurés au niveau de la *HL-Gw* et qui concernent la *LL-Gw* correspondante. S'il n'y a pas de trafic dans l'environnement IoT, il n'est pas nécessaire d'envoyer les balises aux objets IoT afin de préserver leurs énergies. Après l'envoi de la balise, la *LL-Gw* reçoit les données des différents objets IoT en fonction des *QoS CAP* correspondantes et compte le nombre de paquets reçus dans chaque *QoS CAP*. En cas de réception d'une nouvelle configuration de la part de la passerelle de haut niveau, la *LL-Gw* met à jour ses valeurs de configuration de supertrame *QBAIoT* et envoie une nouvelle balise aux objets IoT.

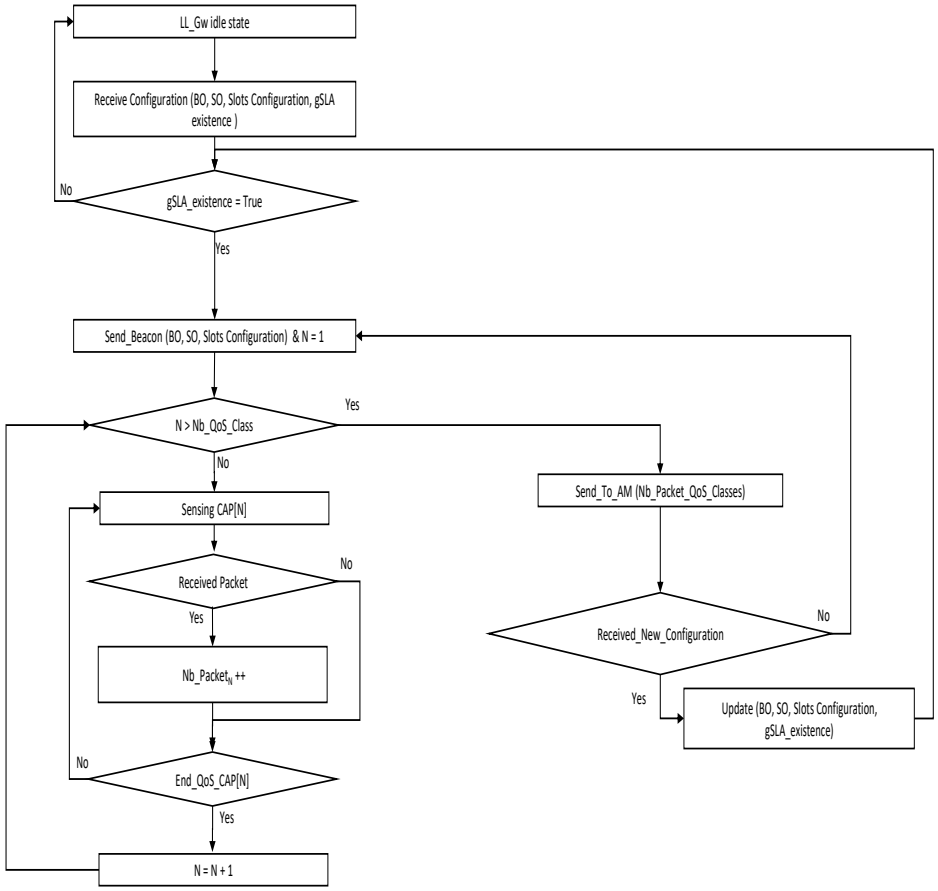

Figure 5.6 : Algorithme QBAIoT au niveau de la passerelle LL-Gw

5.3.4. Spécification du processus *QBAIoT* au niveau des objets IoT

Les objets IoT de notre Framework de garantie de niveau de service utilisent la méthode *QBAIoT* au sein de la couche sensing de l'architecture IoT afin de profiter d'un accès différencié à la passerelle de bas niveau *LL-Gw*. Nous illustrons à travers la Figure 5.7 l'ensemble des états de l'automate finie concernant le processus *QBAIoT* au niveau d'un objet IoT.

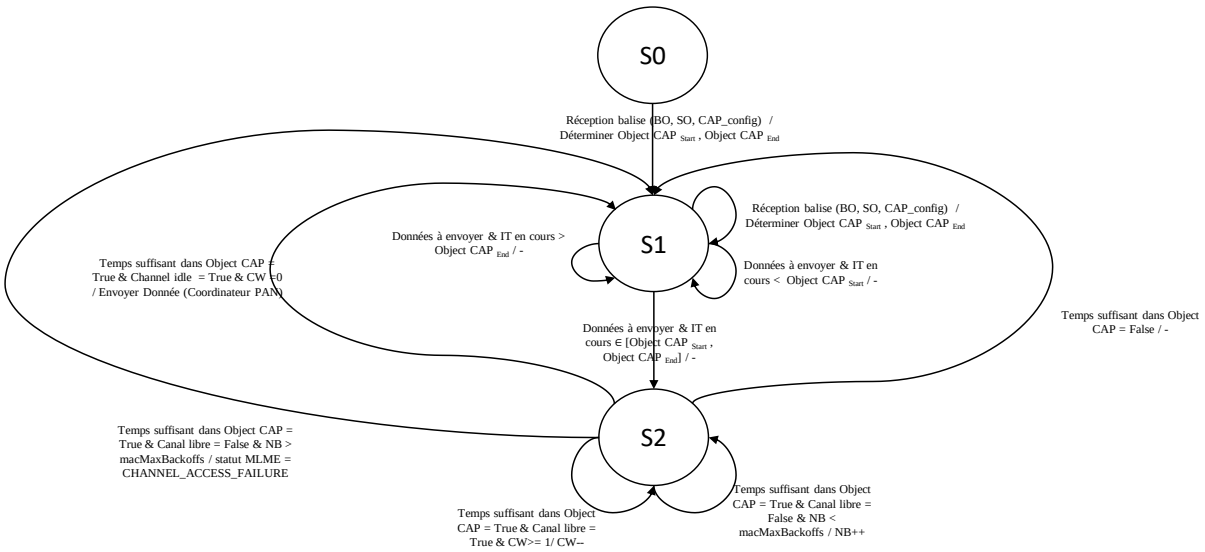

Figure 5.7 : FSM des objets IoT concernant QBAIoT

Ainsi, à l'état S0, un objet IoT reçoit la balise envoyée par la passerelle *LL-Gw* spécifiant la configuration des *QoS CAP, BO* et *SO*. En se basant sur sa classe de *QoS*, l'objet IoT utilise les valeurs des paramètres «*Object_CAPStart*» et «*Object_CAPEnd*» reçu dans le champ «*beacon payload*» de la balise afin de se configurer et passer à l'état S1. Dans cet état S1, si l'objet IoT détermine que sa *QoS CAP* n'a pas encore débuté (i.e., l'*IT* en cours de la supertrame actuelle est inférieur à la valeur spécifié dans «*Object_CAPStart*»), alors il restera dans l'état S1 et attendra sa *QoS CAP* dans la supertrame actuelle. Par ailleurs, si le nœud détermine que sa *QoS CAP* est dépassée (i.e., l'*IT* en cours de la supertrame actuelle est supérieur à la valeur spécifié dans «*Object_CAPEnd*»), alors il restera dans l'état S1 et attendra sa *QoS CAP* dans la supertrame suivante. Enfin, si le nœud détermine que sa *QoS CAP* est en cours (i.e., «*Object_CAPStart*» ≤ *IT* en cours ≤ «*Object_CAPEnd*»), alors il passe à l'état S2. Dans cet état S2, l'objet IoT évalue le temps restant dans sa *QoS CAP*. S'il y a suffisamment de temps pour envoyer un paquet alors l'objet IoT exécute *CCA*. A l'issu de ce test *CCA*, un canal libre réduit la valeur de *CW* tandis qu'un canal occupé n'a aucune conséquence sur cette valeur. Dans le cas d'un canal libre, si la valeur de *CW* est supérieure à 0 alors *CCA* doit être exécuté à nouveau. D'autre part, une valeur de *CW* égale à 0 permet à l'objet IoT d'envoyer ses données à la *LL-Gw* et de revenir à l'état S1. Par la suite, l'objet IoT à l'état S1, attend une nouvelle balise ou essaie d'envoyer un autre paquet de données. En revanche, dans le cas d'un canal occupé pour un objet IoT à l'état S2, *NB* est comparé à «*macMaxBackoffs*». Une valeur de *NB* supérieure à « *macMaxBackoffs*» conduit à un échec d'accès au canal et l'objet IoT revient à l'état S1. Par contre, si *NB* est inférieur à « *macMaxBackoffs* », la valeur de *NB* sera incrémentée et le test *CCA* sera exécuté à nouveau tout en laissant le nœud à l'état S2.

En plus du *FSM* qui décrit les différents états d'un objet IoT implémentant la méthode d'accès *QBAIoT*, nous décrivons à travers la Figure 5.8 notre algorithme d'adaptation du processus slotted *CSMA/CA* pour ces objets IoT.

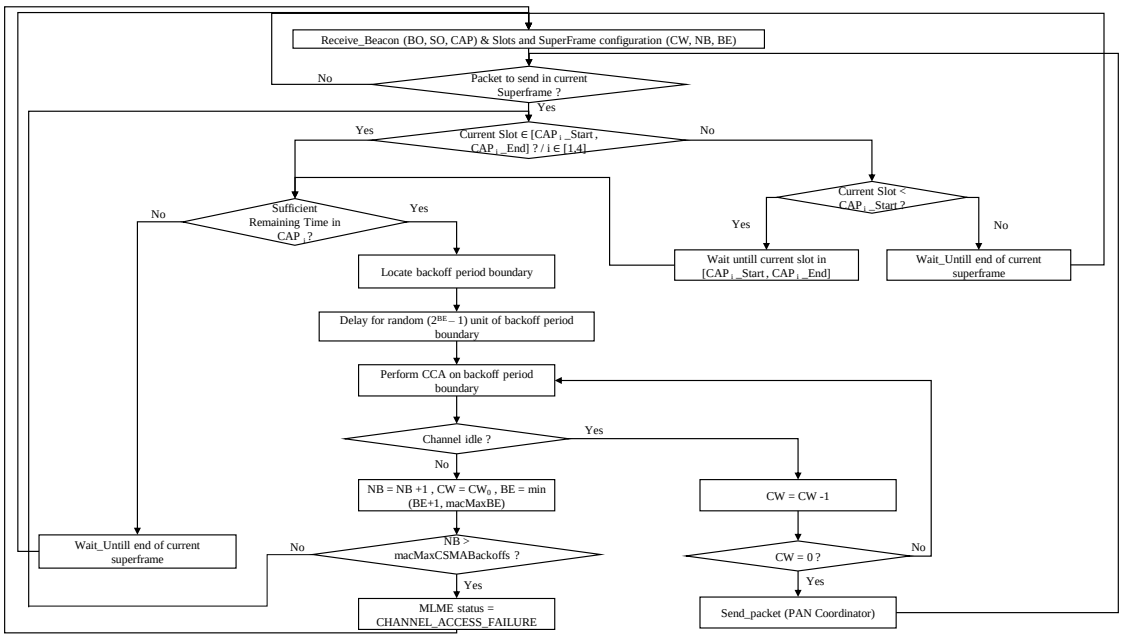

Figure 5.8 : Algorithme QBAIoT au niveau des objets IoT

Ainsi, après la réception de la balise, un objet IoT synchronise la durée de la supertrame *QBAIoT* avec le coordinateur (i.e., *LL-Gw*) en définissant les valeurs de *BO*, *SO*, la durée de sa *QoS CAP* et les différentes valeurs du processus (i.e., *CW, NB, BE*). Par la suite, lors de la création d'un paquet de données à envoyer au coordinateur, l'objet IoT doit vérifier si l'*IT* en cours de la supertrame *QBAIoT* actuelle est dans sa durée de *QoS CAP*. Ainsi, si le numéro de l'*IT* en cours de la supertrame est supérieur au numéro du dernier *IT* de sa *QoS CAP*, l'objet IoT doit attendre sa *QoS CAP* dans la prochaine supertrame QBAIoT. De même, si le numéro de l'*IT* en cours de la supertrame est inférieur au numéro du premier *IT* de sa *QoS CAP*, l'objet IoT doit attendre le début de sa *QoS CAP* dans la supertrame actuelle. Par contre, si l'*IT* en cours est dans l'intervalle des *IT* de sa *QoS CAP*, l'objet IoT exécute le processus *CSMA/CA*. Ainsi, lorsque l'*IT* en cours est dans la *QoS CAP* de l'objet IoT, ce dernier vérifie s'il y a assez de temps restant dans sa *QoS CAP* pour envoyer les données (i.e., au moins un paquet) et recevoir un accusé de réception. Si ce n'est pas le cas alors l'objet IoT doit attendre jusqu'à sa prochaine *QoS CAP* dans la supertrame suivante. Par contre, s'il y a suffisamment de temps, l'objet IoT détermine le début de la *BP (Backoff Period)* suivante (chaque *IT* est divisé en plusieurs *BPs*), car toutes les opérations, entre autres l'envoi des données, doivent commencer avec le début d'une *BP* et non durant la *BP*. Ensuite, le nœud observe un délai d'attente (valeur aléatoire comprise entre 0 et 2^{BE} -1) avant d'effectuer le premier test *CCA*. D'une part, si le canal n'est pas libre, la valeur *NB* est incrémentée, la valeur *CW* est réinitialisée à la valeur initiale et la valeur *BE* permettant le calcul du délai d'attente est changée (voir Figure 5.8). Si la valeur de *NB* est supérieure à la valeur maximale, une défaillance est détectée et le processus reprend depuis le début. Sinon, l'objet IoT vérifie une nouvelle fois que l'*IT* en cours de la supertrame est toujours dans sa *QoS CAP* et reprend à nouveau le processus normalement depuis ce test. D'autre part, si le premier *CCA* a indiqué un canal libre, la valeur de *CW* est décrémentée et un autre *CCA* est exécuté si *CW* n'a pas

encore atteint la valeur 0. Lorsque la valeur de *CW* atteint 0, l'objet IoT peut envoyer les données au coordinateur et reprendre le processus pour voir s'il a d'autres paquets à envoyer.

5.4. Validation et évaluation des performances de QBAIoT

Dans le cadre de la validation et l'évaluation des performances de *QBAIoT*, nous utilisons le simulateur réseau OMNeT ++ en adaptant le modèle du standard IEEE 802.15.4 disponible pour ce simulateur [167]. Notre adaptation consiste non seulement à supprimer la période *CFP* et la partie inactive de la supertrame standard, mais également à créer différentes périodes *QoS CAP* dans une même supertrame tout en spécifiant la configuration de ces *QoS CAP* dans la balise.

5.4.1. Environnement de simulation et scénarios d'utilisation

Plusieurs scénarios de simulation sont regroupés dans différents ensembles afin de valider le bon fonctionnement de *QBAIoT* et étudier l'efficacité de notre méthode d'accès dans un environnement IoT à travers l'évaluation de ses performances. Dans ce contexte, les paramètres communs concernant l'environnement de ces simulations sont spécifiés dans le Tableau 5.1.

Tableau 5.1 : Paramètres communs aux simulations

Paramètres	Valeurs
Temps de simulation	100 s
Fréquence	2.4 Ghz
Topologie	Etoile
Nombre de coordinateur	1
Nombre maximal de retransmission	3
Débit du canal	250 Kb/s

Le premier ensemble de simulations prend en considération différentes classes de *QoS*. Le nombre de ces classes varie entre 1 et 4. De plus, toutes les classes possèdent des caractéristiques de trafic communes : le même nombre de paquets générés pendant la simulation, la même fréquence de génération de paquets et la même taille des paquets. Dans ce premier ensemble, nous définissons 4scénarios (voir Tableau 5.2). Ces scénarios nous permettent d'étudier le comportement de *QBAIoT* en fonction du nombre de classes de *QoS* qui existent dans l'environnement IoT. Dans ces scénarios, les paquets générés par tous les objets ont une charge utile de la sous couche *MAC* de 50 octets. Chaque objet génère 4 paquets par seconde, soit un intervalle de génération de paquet (*DGI* : Data Generation Interval) de 0.25s et tous les objets génèrent les paquets en même temps. La configuration des *QoS CAP* correspond au nombre d'*IT* affectés à chaque *QoS CAP* parmi les 16 *IT disponibles dans la supertrame QBAIoT*. La configuration 1 dans le Tableau 5.2 indique que 16 *IT* sont affectés à la *QoS CAP 1* (*RTMC*) pour le scénario 1 alors que la configuration 6/5/3/2 indique que 6 *IT* sont

affectés à la *QoS CAP 1* (*RTMC*), 5 *IT* à la *QoS CAP 2* (*RTNMC*), 3 *IT* à la *QoS CAP 3* (*Streaming*) et 2 *IT* à la *QoS CAP 4* (*NRT*) pour le scénario 4.

Tableau 5.2 : Paramètres du premier ensemble des simulations

Scénario	Objets RTMC	Objets RTNMC	Objets Streaming	Objets NRT	Configuration des QoS CAP
1	3	0	0	0	16
2	3	3	0	0	9/7
3	3	3	3	0	7/6/3
4	3	3	3	3	6/5/3/2

Le deuxième ensemble de simulations (voir Tableau 5.3) prend en considération 4 classes de *QoS* dans l'environnement de la *LL-Gw* et nous permet d'étudier l'impact de l'augmentation du nombre de paquets générés par classe en faisant varier le paramètre *DGI*. Cet ensemble de simulations spécifie 4 scénarios (scénarios 5 à 8) avec 4 valeurs différentes de *DGI*. Ces scénarios prennent en considération 4 classes de *QoS* avec 3 objets IoT par classe et des paquets ayant une charge utile de la sous couche *MAC* de 50 octets. De plus, ces scénarios utilisent la même configuration des *QoS CAP* (i.e., 6/5/3/2) : 6 *IT* sont affectés à la *QoS CAP 1* (*RTMC*), 5 *IT* à la *QoS CAP 2* (*RTNMC*), 3 *IT* à la *QoS CAP 3* (Streaming) et 2 *IT* à la *QoS CAP 4* (*NRT*). Enfin, le scénario 5 nous permet de générer 2 paquets par objet et par seconde et par conséquent un total de 6 paquets par seconde pour tous les objets d'une même classe, alors que le scénario 8 permet de générer seulement 24 paquets par secondes pour tous les objets d'une même classe. Tous les paquets de tous les objets de toutes les classes sont générés en même temps.

Tableau 5.3 : Paramètres du deuxième ensemble des simulations

Scénario	DGI	Configuration des QoS CAP
5	0.5 s	6/5/3/2
6	0.375 s	6/5/3/2
7	0.25 s	6/5/3/2
8	0.125 s	6/5/3/2

Le troisième ensemble de simulations (voir Tableau 5.4) prend en considération 4 classes de *QoS* dans l'environnement de la *LL-Gw* mais en changeant le nombre d'objets par classe. En effet, cet ensemble de simulations spécifie 3 scénarios (scénarios 9 à 11) avec respectivement 1, 2 et 3 objets par classe de *QoS* afin d'étudier l'impact du nombre d'objets par classe sur le comportement de *QBAIoT*. Dans ces scénarios, les paquets générés par tous les objets ont une charge utile de la sous couche *MAC* de 50 octets et chaque objet génère 4 paquets par seconde, soit une valeur de *DGI* de 0.25s sachant que tous les objets génèrent les paquets en même temps. La configuration des *IT* pour les 3 scénarios est la même à savoir 6/5/3/2.

Tableau 5.4 : Paramètres du troisième ensemble de simulations

Scénario	Objets RTMC	Objets RTNMC	Objets Streaming	Objets NRT	Configuration des QoS CAP
9	1	1	1	1	6/5/3/2
10	2	2	2	2	6/5/3/2
11	3	3	3	3	6/5/3/2

Le quatrième ensemble de simulations (voir Tableau 5.5) prend en considération 4 classes de *QoS* dans l'environnement de la *LL-Gw* et un plus grand nombre d'objets par classe comparé au troisième ensemble de simulations. De plus, ce quatrième ensemble de simulations comprend 3 scénarios (scénarios 12 à 14) avec respectivement 4, 5 et 6 objets par classe de *QoS*. Tout comme l'ensemble de simulations précédent, les paquets générés par tous les objets ont une charge utile de la sous couche *MAC* de 50 octets et chaque objet génère 4 paquets par seconde. Par contre, tous les objets ne génèrent pas les paquets en même temps. En effet, nous avons configuré les objets IoT de manière à ce que les paquets soient générés avec un léger décalage temporel pour ne pas avoir un très grand nombre de collisions avec cet ensemble de simulations caractérisé par un plus grand nombre d'objets par classe. Ainsi nous pouvons étudier les limites de notre méthode d'accès *QBAIoT*.

Tableau 5.5 : Paramètres du quatrième ensemble de simulations

Scénario	Objets RTMC	Objets RTNMC	Objets Streaming	Objets NRT	Configuration des QoS CAP
12	4	4	4	4	6/5/3/2
13	5	5	5	5	6/5/3/2
14	6	6	6	6	6/5/3/2

5.4.2. Résultats des performances de QBAIoT

Dans le cadre de l'évaluation de notre méthode d'accès basée sur la *QoS*, à savoir *QBAIoT*, en fonction des scénarios spécifiés dans la section précédente, nous nous intéressons à différents paramètres de performance et de *QoS* (i.e., Délai moyen, Ratio de livraison de paquets, Débit effectif des données et le Ratio de paquets ponctuels). Nous définissons dans ce qui suit ces différents paramètres avant de présenter les résultats de performance obtenus.

- **Délai moyen (Mean Delay)** : Ce paramètre évalue la durée moyenne d'acheminement des paquets générés par les objets IoT et reçus par la passerelle de bas niveau *LL-Gw*. Il est calculé en divisant la somme des délais de chaque paquet par le nombre total de paquets (voir Equation 5.3).

$$Délai\ moyen\ = \frac{\Sigma\ Délai}{Nombre\ de\ paquets\ reçus} \tag{5.3}$$

- **Ratio de livraison de paquets (*PDR* : Packet Delivery Ratio)** : Ce paramètre évalue la fiabilité en terme de transmission réussie. Il est calculé en divisant le nombre de paquets reçus par le nombre de paquets générés (voir Equation 5.4). Les paquets non reçus sont soit perdus à cause d'une collision, soit toujours dans la mémoire tampon de l'expéditeur (i.e., un objet IoT) en attente d'un accès au canal.

$$PDR = \frac{Nombre\ de\ paquets\ reçus}{Nombre\ de\ paquets\ générés} \quad (5.4)$$

- **Débit effectif des données (*EDR* : Effective Data Rate)** : Ce paramètre évalue l'efficacité de l'utilisation de la bande passante du lien. Il est calculé en multipliant le nombre de paquets reçus par leur taille (charge utile sans surcharge), puis en divisant le résultat par le temps de simulation (voir Equation 5.5). Nous considérons dans notre étude que tous les paquets ont la même taille.

$$EDR = \frac{Nombre\ de\ paquets\ reçus * Taille\ d'un\ paquet}{Temps\ de\ simulation} \quad (5.5)$$

- **Ratio de paquets ponctuels (*PPR* : Punctual Packet Ratio)** : Ce paramètre évalue la proportion de paquets reçus dans les limites temporelles d'une valeur seuil pour le trafic temps réel. Il est calculé en divisant le nombre de paquets reçus conformément au seuil par le nombre total de paquets reçus (voir Equation 5.6). En général, le trafic temps réel devrait avoir un délai de bout en bout de 150 ms (cf. chapitre 4) [168]. Nous avons spécifié le seuil à 80 ms pour le trafic utilisant *QBAIoT* comme méthode d'accès dans la couche sensing de l'architecture IoT afin de laisser un délai maximal de 70 ms pour les couches réseau et cloud de cette architecture et respecter ainsi le délai global de 150 ms.

$$PPR = \frac{Nombre\ de\ paquets\ conforme\ au\ seuil}{Nombre\ de\ paquets\ \text{reçus}} \quad (5.6)$$

Après la définition des paramètres de performance utilisés dans notre étude, nous commençons par évaluer et analyser les résultats correspondant au premier ensemble de simulations. Ainsi, la Figure 5.9 présente les résultats de l'évaluation du délai et du *PDR* correspondant au scénario 1 avec un seul trafic *RTMC*. Nous observons que, dans le cas où une seule classe de *QoS* existe dans l'environnement IoT de la *LL-Gw*, *QBAIoT* se comporte comme la méthode slotted *CSMA/CA* traditionnelle du standard IEEE 802.15.4 tout en allouant la totalité des *IT* à une même et unique classe de *QoS*. En effet, les résultats obtenus en termes de délai moyen et de *PDR* sont très proches pour les deux méthodes. Nous remarquons qu'avec *QBAIoT*, 7 ms de délai moyen supplémentaires sont observés par les objets IoT comparé au standard IEEE 802.15.4. Cette légère différence résulte des tests effectués par les objets *QBAIoT* pour vérifier que l'*IT* en cours est dans leur *QoS CAP* avant de tenter d'accéder au canal alors que ce n'est pas le cas dans l'approche traditionnelle du standard.

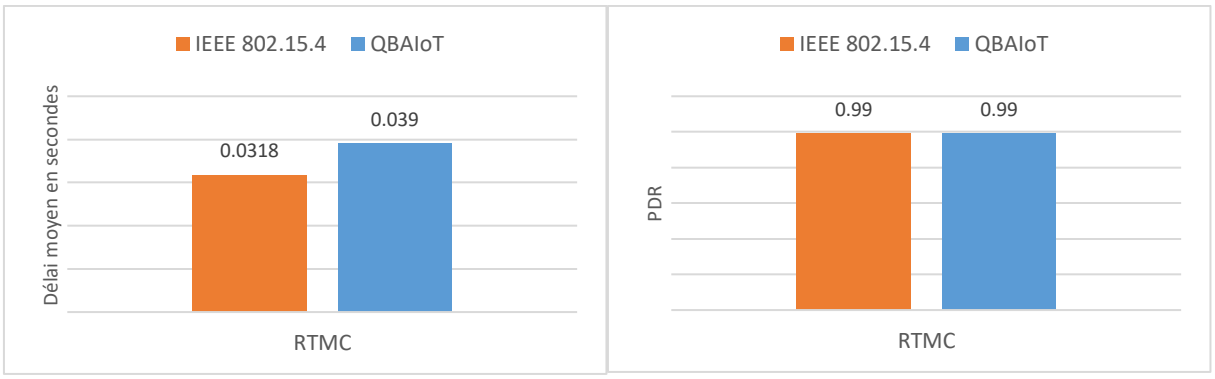

Figure 5.9 : Evaluation du délai et du PDR (scénario 1)

La Figure 5.10 présente les résultats de l'évaluation du délai et du *PDR* correspondant au scénario 2 en présence de deux classes de trafic à savoir RTMC et RTNMC. Nous observons que *QBAIoT* assure un meilleur délai pour *RTMC* et un délai identique pour *RTNMC* comparé à la méthode d'accès standard IEEE 802.15.4.

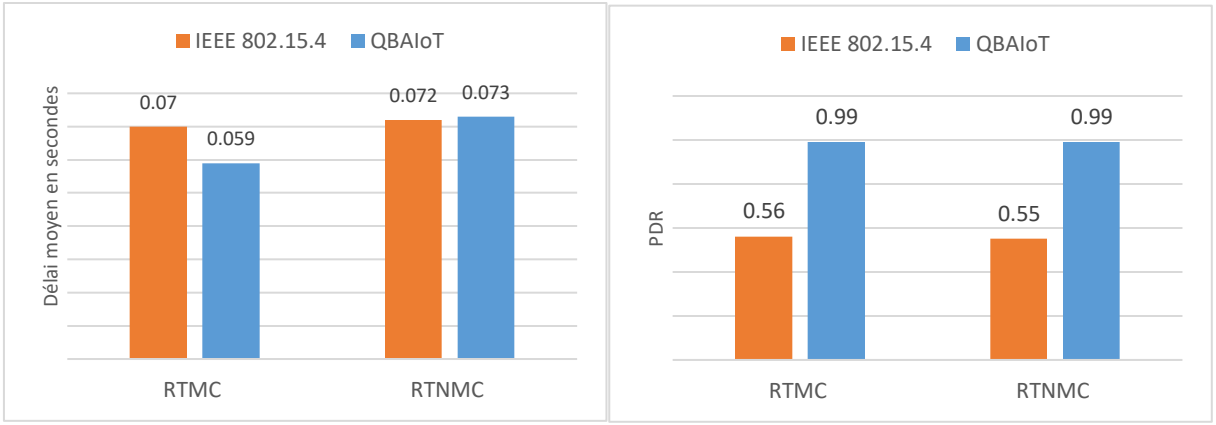

Figure 5.10 : Evaluation du délai et du PDR (scénario 2)

En effet, l'allocation de 2 *IT* supplémentaires pour *RTMC* (i.e., 9 IT) par rapport à *RTNMC* (7 IT) dans le cas d'utilisation de *QBAIoT* résulte en un délai moyen observé par les paquets *RTMC* inférieur de 10 ms comparé à la méthode standard. En ce qui concerne l'évaluation du *PDR* pour le scénario 2, *QBAIoT* garantit de meilleures valeurs pour *RTMC* (99%) et *RTNMC* (98%) comparé à l'approche standard (56% et 55% respectivement). Ainsi, *QBAIoT* permet d'obtenir de meilleures performances en termes de *PDR*, car seuls les objets d'une même classe de *QoS* sont en concurrence pour accéder au canal pendant les *IT* d'une *QoS CAP* particulière. Avec *QBAIoT*, l'accès basé sur les classes permet d'éviter les collisions entre différents trafics générés par des objets appartenant à différentes classes de *QoS*.

La Figure 5.11 présente les résultats de l'évaluation du délai et du *PDR* correspondant au scénario 3 en présence de trois classes de trafic à savoir *RTMC*, *RTNMC* et *Streaming*. Nous observons un meilleur délai, en utilisant *QBAIoT*, pour les trafics *RTMC* (i.e., un gain de 33 ms) et

RTNMC (i.e., un gain de 16 ms) comparé à la méthode du standard IEEE 802.15.4. De même, nous obtenons de meilleurs *PDR* avec *QBAIoT* pour les trois types de trafic de ce scénario grâce à la minimisation du nombre de collisions. A titre d'exemple, le *PDR* pour le trafic *RTMC* avec *QBAIoT* est égal à 98,5% alors qu'il est égal à 26% avec le standard IEEE 802.15.4.

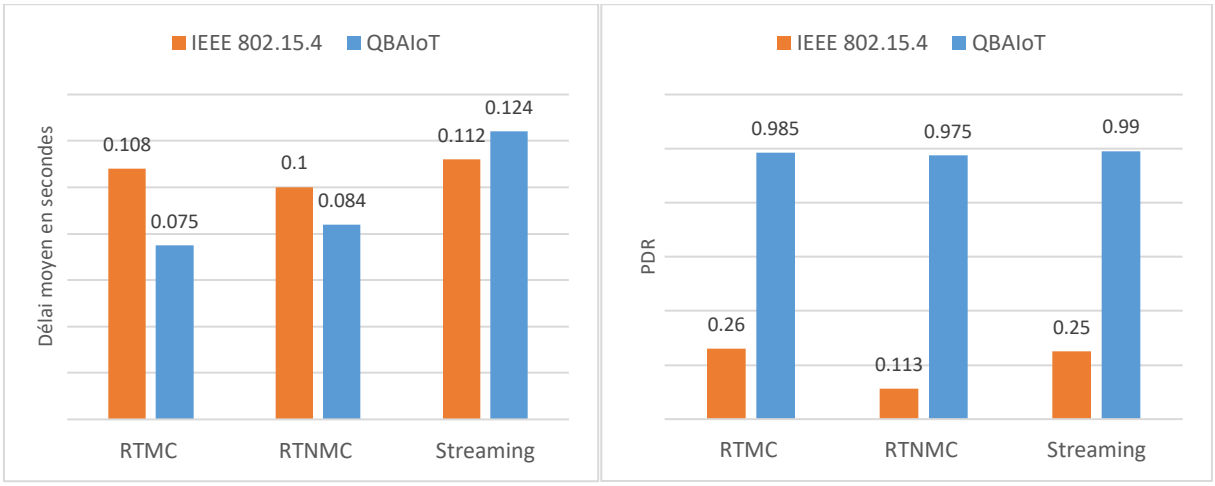

Figure 5.11 : Evaluation du délai et du PDR (scénario 3)

La Figure 5.12 présente les résultats de l'évaluation du délai et du *PDR* correspondant au scénario 4 en présence de quatre classes de trafic à savoir *RTMC*, *RTNMC*, *Streaming* et *NRT*.

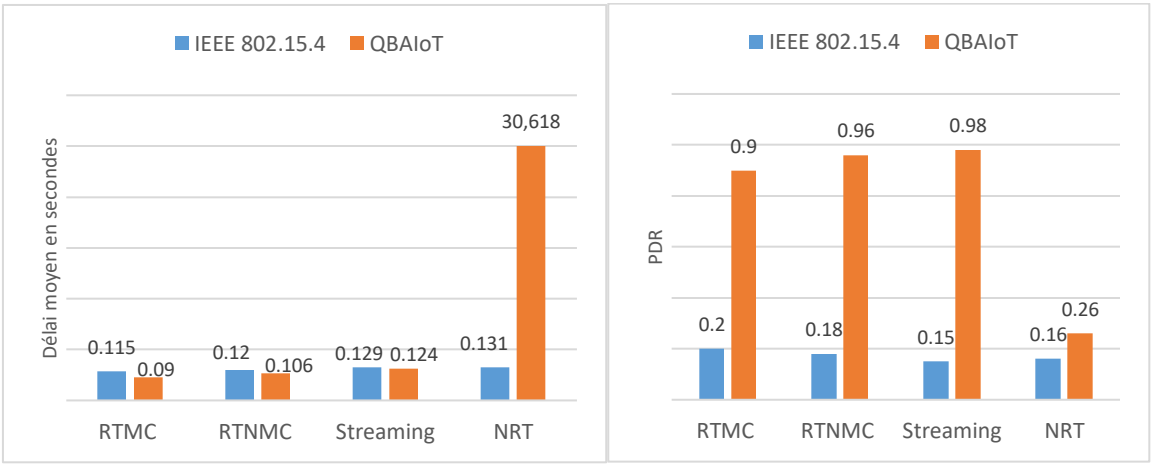

Figure 5.12 : Evaluation du délai et du PDR (scénario 4)

Nous observons que *QBAIoT* assure de meilleurs délais pour les deux classes sensibles à ce paramètre (90 ms pour *RTMC* et 106 ms pour *RTNMC*) comparé à la méthode d'accès standard (115 ms pour *RTMC* et 123 ms pour *RTNMC*). De plus, nous observons un meilleur *PDR* pour toutes les classes de *QoS* avec *QBAIoT* avec plus que 96% pour les 3 classes plus prioritaires. En effet, avec l'utilisation de la méthode d'accès du standard IEEE 802.15.4, toutes les classes de *QoS* sont servies de la même manière et en même temps, entrainant plus de collisions. En revanche, *QBAIoT* permet à

travers la définition de plusieurs *QoS CAP* de minimiser les collisions et de configurer plus d'*IT* pour les classes les plus prioritaires. Enfin, avec *QBAIoT* un délai plus important est observé pour le trafic non temps réel (trafic *NRT* non sensible à ce paramètre) comparé au délai obtenu en utilisant le standard IEEE 802.15.4. Ceci est dû à la configuration d'un nombre limité d'*IT* (i.e., 2 IT) pour cette classe moins prioritaire. Ceci dit, *QBAIoT* offre pour le trafic *NRT* un meilleur *PDR* (26%) que celui obtenu avec la méthode IEEE 802.15.4 (16%).

Dans le cadre du deuxième ensemble de simulations, nous prenons en considération différents intervalles de génération de paquets de données appartenant à 4 classes de *QoS* afin de comparer les performances de la méthode *QBAIoT* à celle du standard IEEE 802.15.4 en termes de délai, *PDR* et *PPR*. Ainsi, la Figure 5.13 montre que pour tous les intervalles de génération (i.e., *DGI*) de paquets considérés dans les différents scénarios (i.e., scénario 5 à 8), *QBAIoT* offre un meilleur délai au trafic *RTMC* comparé à la méthode de contrôle d'accès du standard IEEE 802.15.4. A titre d'exemple, *QBAIoT* offre un délai moyen pour le trafic *RTMC* de 83 ms (cas où *DGI* est égale à 0,375) alors que la méthode standard offre un délai moyen de 109 ms, soit une dégradation. En ce qui concerne le trafic *RTNMC*, *QBAIoT* offre des meilleurs délais que le standard pour les valeurs de *DGI* de 0.5s, 0.375s et 0.25s mais ces délais deviennent moins intéressants pour la valeur de *DGI* de 0.125s suite à la saturation du canal avec cet intervalle de génération des paquets.

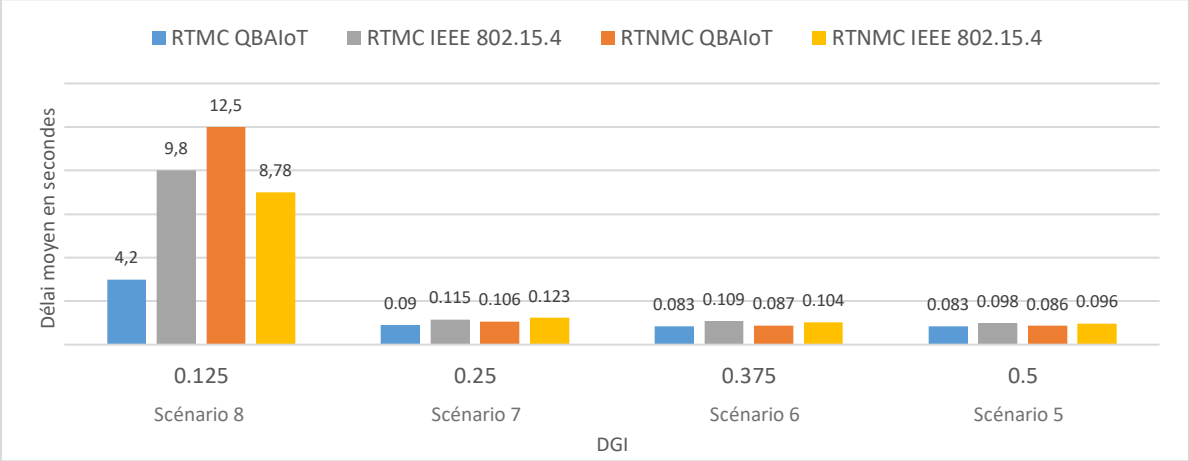

Figure 5.13 : Evaluation du délai de RTMC et RTNMC (scénarios 5 à 8)

La Figure 5.14 présente les résultats de l'évaluation du *PDR* des 4 classes de trafic correspondant aux scénarios allant de 5 à 8. Nous observons que la méthode *QBAIoT* offre un meilleur *PDR* pour toutes les classes de *QoS* comparé à la méthode IEEE 802.15.45. En effet, en créant différentes *QoS CAP*, *QBAIoT* élimine la possibilité d'occurrence de collisions entre paquets de différentes classes de *QoS*, ce qui permet d'optimiser la livraison de ces paquets. D'autre part, le nombre de collisions dépend également de la durée de la *QoS CAP*.

Lorsque la durée de cette dernière est moins importante et que le trafic correspondant est de plus en plus important, la fréquence d'accès au canal devient plus élevée, ce qui entraîne davantage

de collisions. Dans ce contexte, nous pouvons observer (voir Figure 5.14) qu'avec *QBAIoT* et un intervalle *DGI* plus faible (trafic plus important), les valeurs de *PDR* des différentes classes de *QoS* diminuent à cause de l'augmentation du nombre de collisions dans chaque *QoS CAP*. Enfin, les valeurs de *PDR* lors de l'utilisation de *QBAIoT* dépendent du type de classe de *QoS* : le *PDR* des trafics temps réel est globalement meilleur que celui des trafics *Streaming* et *NRT*, alors qu'il est approximativement le même avec la méthode de contrôle d'accès standard qui ne fait pas de différenciation entre les trafics.

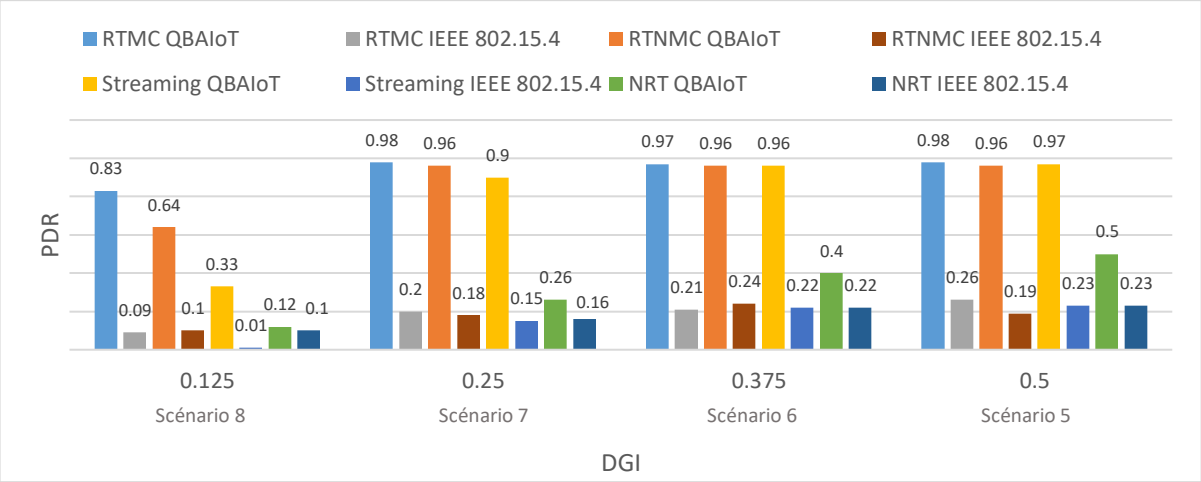

Figure 5.14 : Evaluation du PDR (scénarios 5 à 8)

La Figure 5.15 présente les résultats de l'évaluation du ratio des paquets ponctuels (i.e., *PPR*) des trafics *RTMC* et *RTNMC* qui respectent les délais précisés dans le contrat de niveau de service *iSLA* et qui correspondent aux scénarios 6, 7 et 8 du deuxième ensemble de simulations. Nous observons que la méthode *QBAIoT* fournit un meilleur *PPR* pour le trafic *RTMC* et *RTNMC* comparé à la méthode de contrôle d'accès du standard. Ainsi, la quantité de paquets temps réel conformes au seuil du délai maximum de 80 ms, spécifié dans l'*iSLA*, devient plus importante grâce à l'utilisation de *QBAIoT* et par conséquent ceci permet de diminuer la variation du délai comparé à la méthode slotted *CSMA/CA* standard.

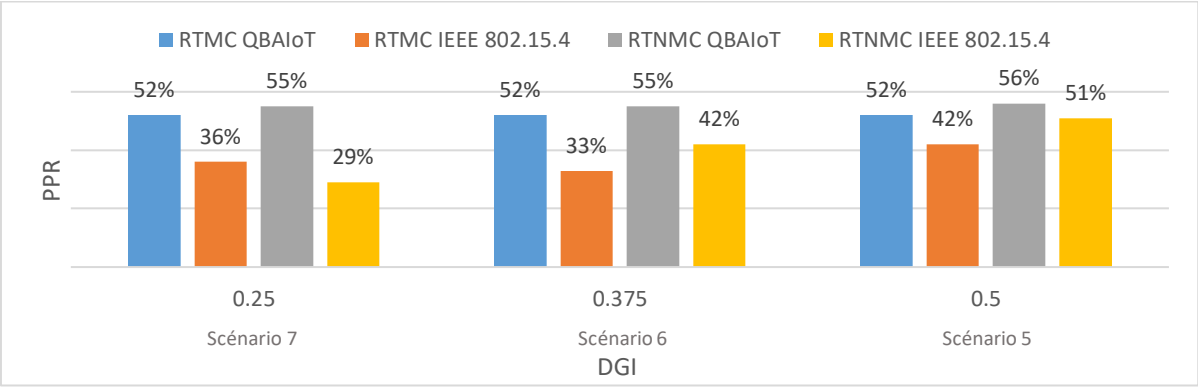

Figure 5.15 : Evaluation du PPR des trafics RTMC et RTNMC (scénarios 6 à 8)

Dans le cadre du troisième ensemble de simulations (i.e., scénarios 9, 10 et 11), nous faisons varier le nombre d'objets par classe (de 1 objet à 3 objets) en présence de 4 classes de *QoS* dans l'environnement IoT de la *LL-Gw* afin de comparer les performances de la méthode *QBAIoT* à celle du standard IEEE 802.15.4 en termes de délai, *PDR* et *EDR*. Ainsi, la Figure 5.16 montre l'évolution des délais subis par les paquets appartenant aux classes *QoS* sensible au délai soit *RTMC* et *RTNMC* en fonction du nombre d'objets par classe *QoS* correspondant aux scénarios 9, 10 et 11. Les résultats obtenus montrent que pour 1 objet par *QoS CAP*, notre méthode *QBAIoT* offre un meilleur délai pour le trafic *RTMC* (10 ms de moins que le standard) et pour le trafic *RTNMC* (7 ms de moins que le standard). Cette différence devient plus importante en augmentant le nombre d'objets par classe.

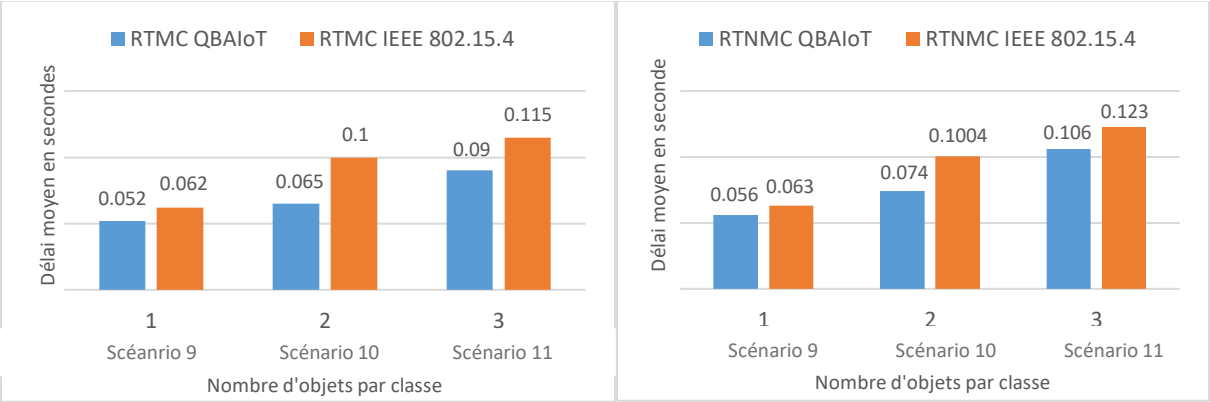

Figure 5.16 : Evaluation du délai des trafics RTMC et RTNMC (scénarios 9 à 11)

Ainsi, dans le cas de 2 objets par *QoS CAP*, les paquets *RTMC* et *RTNMC* observent un meilleur délai avec *QBAIoT* (35 ms de moins de délai pour le trafic *RTMC* et 26 ms de moins pour le trafic *RTNMC*). Ces délais observés par les trafics temps réel avec *QBAIoT* sont meilleurs grâce à l'attribution aux classes de *QoS* temps réel d'un nombre plus important d'*IT* dans lesquels les objets correspondant à ces classes peuvent envoyer leurs données sans aucune collision avec d'autres objets appartenant à d'autres classes de *QoS* non temps réels. Par conséquent, ces paquets temps réel observent un délai d'attente limité dans la mémoire tampon et ils sont servis plus rapidement que les autres types de trafic.

La Figure 5.17 présente les résultats d'évaluation du ratio de livraison des paquets de toutes les classes de *QoS* correspondant aux scénarios 9, 10 et 11. Nous observons que la méthode *QBAIoT*, comparée au standard IEEE 802.15.4, offre pour toutes les classes un *PDR* trois fois supérieur en présence d'un objet par classe, un *PDR* quatre fois supérieur en présence de deux objets par classe et un *PDR* 6 fois supérieur (sauf pour la classe *NRT*) en présence de trois objets par classe. Nous obtenons un meilleur *PDR* avec notre approche grâce à un accès au canal par classe optimisé en évitant les collisions entre les objets de différentes classes de QoS.

En effet, pour chaque *QoS CAP*, seuls les objets de la classe de *QoS* correspondante peuvent entrer en concurrence pour accéder au canal. Par exemple, avec 1 objet par classe de *QoS*, il n'y a pas

de concurrence, en utilisant *QBAIoT*, entre les objets pour accéder au canal, alors que dans ce même scénario l'utilisation du standard ne permet pas la différenciation entre les 4 objets qui sont en concurrence pour l'accès au canal. Ainsi, avec *QBAIoT*, un nombre inférieur d'objets sont en concurrence pour accéder au canal pour un *IT* donné.

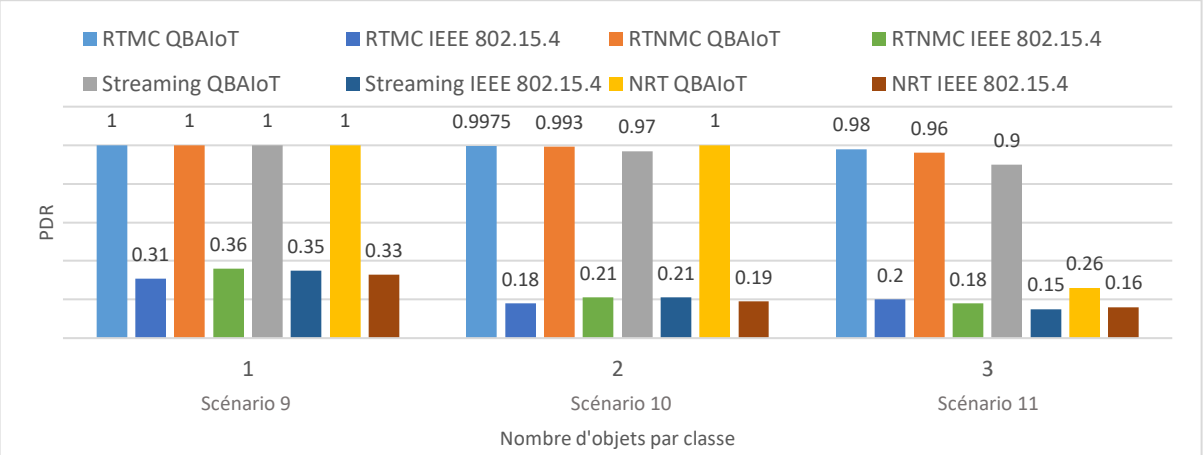

Figure 5.17 : Evaluation du PDR (scénarios 9 à 11)

En ce qui concerne l'évaluation du débit effectif des données (*EDR*), la Figure 5.18 compare les résultats obtenus en utilisant la méthode *QBAIoT* à ceux de la méthode de contrôle d'accès du standard IEEE 802.15.4. Nous observons que *QBAIoT* permet d'avoir, pour tous types de trafics, un meilleur débit de données et une utilisation de la bande passante plus efficace que l'approche traditionnelle. En effet, *QBAIoT* permet un nombre inférieur de collisions et offre un nombre plus élevé de paquets reçus. Par conséquent, le nombre de bits transmis est plus élevé pendant la durée de la simulation, ce qui explique un meilleur *EDR* avec *QBAIoT*. Nous pouvons noter un *EDR* moyen avec *QBAIoT* 4 fois plus important comparé à IEEE 802.15.4 pour toutes les classes de *QoS* dans l'environnement IoT à l'exception du trafic *NRT* avec 3 objets par classe de *QoS* où l'*EDR* avec *QBAIoT* n'est que 1,7 fois plus important que la méthode traditionnelle.

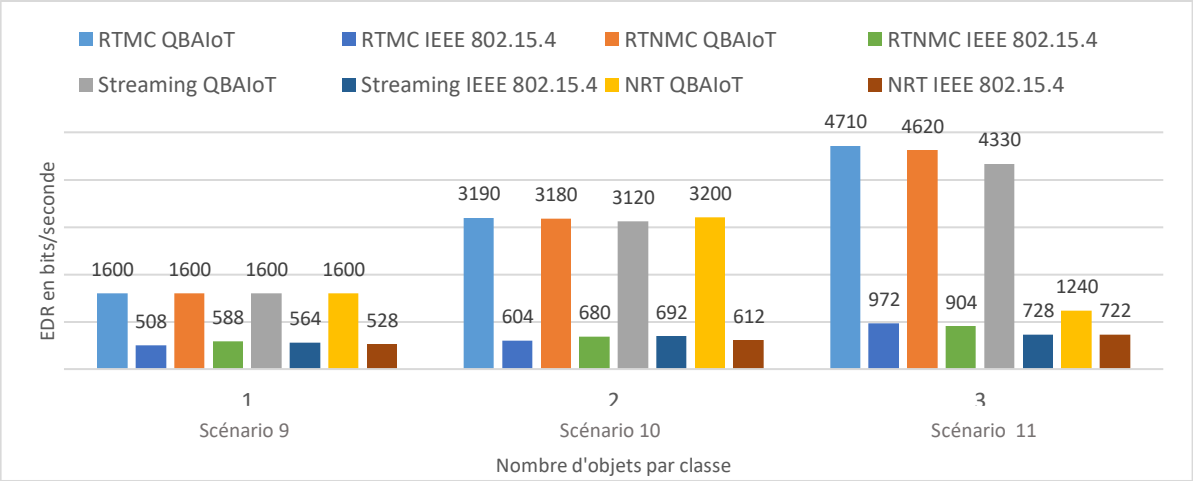

Figure 5.18 : Evaluation de l'EDR (scénarios 9 à 11)

Le quatrième ensemble de simulations nous permet d'étudier l'influence de l'augmentation du nombre d'objets dans chaque *QoS CAP* en utilisant la méthode d'accès *QBAIoT*. En effet, à travers les scénarios 12, 13 et 14, le nombre d'objets par *QoS CAP* augmentent de 4 à 6. Ces objets envoient des données avec une valeur de *DGI* égale à 0.25s sachant que les objets d'une même classe n'envoient pas les données en même temps. Ce décalage entre les temps de génération des paquets permet de minimiser le nombre de collisions possibles dans chaque *QoS CAP* et donc de servir les paquets plus rapidement. La Figure 5.19 montre l'évolution du délai moyen des paquets *RTMC* et *RTNMC* utilisant *QBAIoT* dans les scénarios 12 à 14.

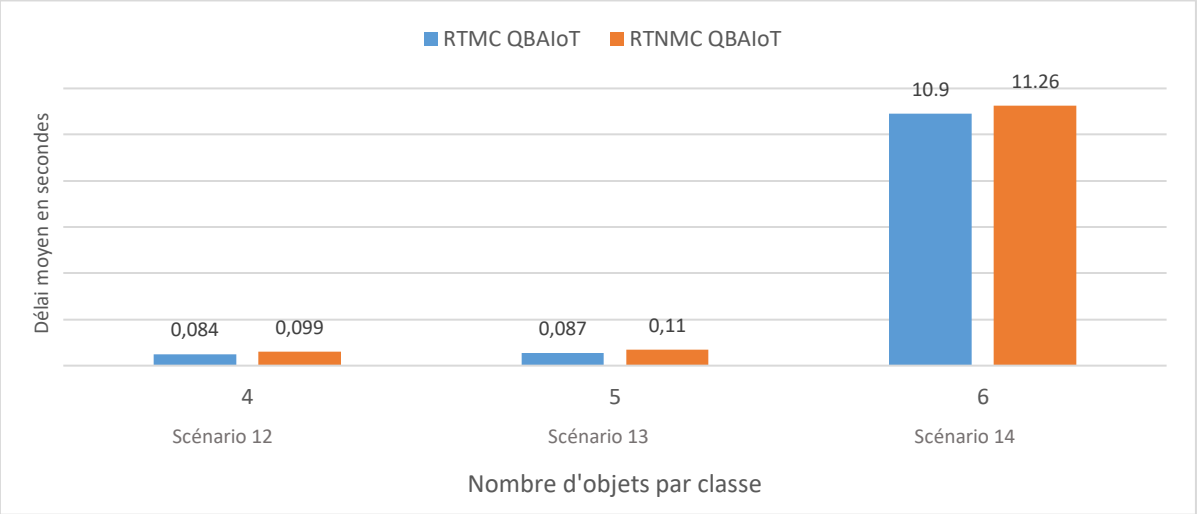

Figure 5.19 : Evaluation du délai moyen de RTMC et RTNMC (scénarios 12 à 14)

Nous observons qu'avec 4 et 5 objets par *QoS CAP*, *QBAIoT* est capable de respecter les exigences des paquets temps réel *RTMC* et *RTNMC* en termes de délais. Par contre, dans le scénario 14 simulant six objets par *QoS CAP*, *QBAIoT* n'est pas en mesure de respecter les exigences des trafics temps réel. En effet, avec six objets ou plus par *QoS CAP* dans le cadre de ces simulations, le grand nombre d'objets qui entrent en compétition au cours de chaque *QoS CAP* est à l'origine d'un nombre important de collisions et par conséquent un délai supplémentaire pour servir les paquets de données. Grâce à l'architecture IoT proposée, nous sommes en mesure de trouver une solution en implémentant une nouvelle passerelle de bas niveau *LL-Gw* (coordinateur) implémentant *QBAIoT* afin d'étendre les capacités de l'environnement IoT et respecter ainsi les exigences spécifiées dans un contrat de type *iSLA* relatif à un nombre croissant d'objets par classe de *QoS*.

5.4.3. Comparaison de QBAIoT avec d'autres méthodes de contrôle d'accès

Après avoir comparé les performances de *QBAIoT* avec celles de la méthode slotted *CSMA/CA* du standard IEEE 802.15.4 selon plusieurs scénarios, il est aussi important de comparer les performances de notre méthode avec d'autres méthodes d'accès basées sur la qualité de service, telles que *SDA-CSMA/CA* [62] décrite dans le chapitre 2. La méthode *SDA-CSMA/CA* prend en compte jusqu'à trois classes de QoS. Chaque niveau de priorité ou classe de *QoS* est caractérisée par un

ensemble de 3 variables (*CW*, « macMinBE » et « macMaxBE ») utilisées dans le processus slotted *CSMA/CA*. Chaque instanciation de cet ensemble de variables donne au niveau de priorité correspondant une certaine probabilité d'accéder au canal pendant le période de contention. Dans ce contexte, nous comparons les performances de *QBAIoT* avec *SDA-CSMA/CA* selon trois scénarios (voir Tableau 5.6) utilisant 3 classes de *QoS*.

Tableau 5.6 : Paramètres des scénarios de simulation (15 à 17)

Scénario	Objets RTMC	Objets RTNMC	Objets Streaming	Configuration des QoS CAP
15	2	2	2	7/6/3
16	3	3	3	7/6/3
17	3	3	3	7/6/3

La configuration initiale de *QBAIoT* pour les trois scénarios est la suivante : 7 *IT* sont attribués à la première classe de *QoS* (*RTMC*), 6 *IT* sont attribués à la deuxième classe de *QoS* (*RTNMC*) et 3 *IT* à la troisième classe de *QoS* (Streaming). De plus, la valeur de *DGI* est fixée à 0,25s pour tous les objets. Enfin, *QBAIoT* est configuré avec une valeur de 2 pour *BO* et *SO*, tandis que *SDA-CSMA/CA* utilise la valeur 7 pour *BO* et 8 pour *SO*. Nous configurons, dans les scénarios 15 et 16, les objets pour générer les paquets au même moment pour prendre en considération le cas le plus critique. Dans ce contexte, la Figure 5.20 présente les résultats obtenus pour les scénarios 15 et 16 et montre que *QBAIoT* offre de meilleurs délais pour les classes de *QoS* sensibles au délai comparé à *SDA-CSMA/CA* qui ne permet pas alors de respecter les contraintes des trafics temps réel.

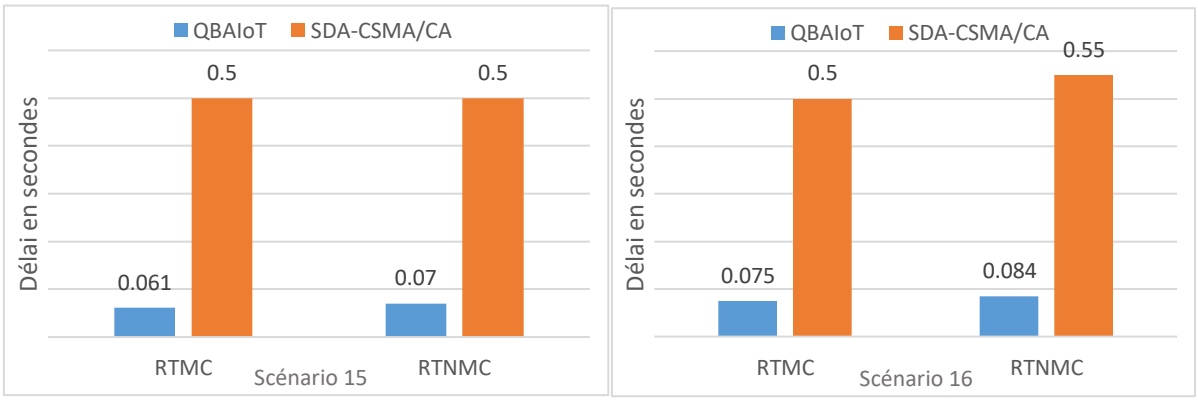

Figure 5.20 : Comparaison du délai moyen avec QBAIoT et SDA-CSMA/CA (scénarios 15 et 16)

De plus, la Figure 5.21 montre le débit effectif des données en utilisant *QBAIoT* et *SDA-CSMA/CA* dans les deux scénarios 15 et 16. Nous remarquons que *SDA-CSMA/CA* offre des meilleurs *EDR* que ceux de *QBAIoT pour* le scénario 15 et cela est dû au fait qu'avec *QBAIoT* un grand nombre de balises sont envoyés sur le canal partagé comme le *BI* est inférieur à celui de *SDA-CSMA/CA*. Cependant dans le scénario 16, *QBAIoT* offre un *EDR* meilleur que celui de *SDA-CSMA/CA* pour les classes prioritaires (*RTMC* et *RTNMC*). Cette amélioration dans les résultats de *QBAIoT* s'explique

par le fait que le rapport entre le nombre des paquets de données et les balises est plus important dans le scénario 16 comparé à celui dans le scénario 15.

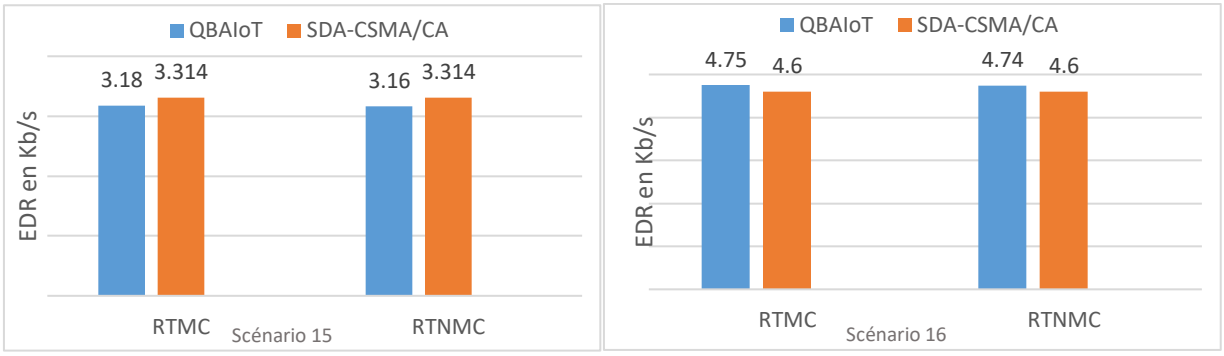

Figure 5.21 : Comparaison de l'EDR moyen avec QBAIoT vs SDA-CSMA/CA (scénarios 15 et 16)

Enfin, le scénario 17 nous permet de comparer *SDA-CSMA/CA* et *QBAIoT* en considérant des objets qui génèrent des paquets avec la même fréquence (*DGI* de 0.25 s) mais à des moments différents pour *QBAIoT* pour minimiser le nombre de collisions entre les paquets. En effet, *SDA-CSMA/CA* permet de différer le temps d'envoi des paquets sur le canal et donc le nombre de collisions en utilisant des valeurs différentes pour les variables *CW*, « macMinBE » et « macMaxBE ». La Figure 5.22 montre les résultats du scénario 17 en termes de délai moyen et débit effectif. Nous observons que les délais moyens des trafics temps réel (i.e., *RTMC* et *RTNMC*) et l'utilisation effective du canal sont meilleurs avec *QBAIoT* qu'avec *SDA-CSMA/CA*. De plus, nous remarquons que les délais moyens et les *EDR* de *QBAIoT* sont meilleurs que ceux du scénario 16 grâce à la diminution du nombre de collisions dans le scénario 17.

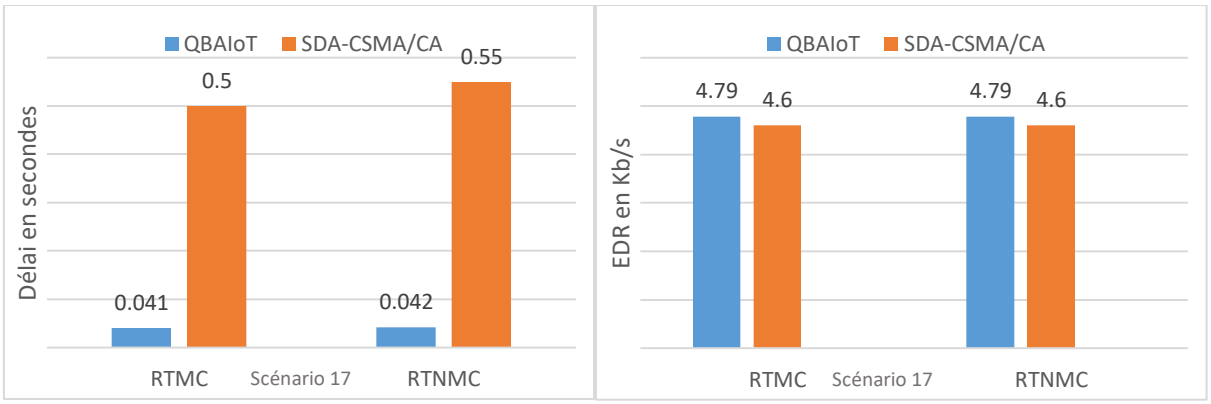

Figure 5.22 : Comparaison du délai moyen et de l'EDR avec QBAIoT et SDA-CSMA/CA (scénario 17)

5.5. Conclusion

A travers ce chapitre, nous avons décrit notre contribution, à savoir *QBAIoT*, concernant une

méthode de contrôle d'accès à un canal partagé entre différents objets IoT de la couche sensing de notre architecture IoT tout en prenant en considération la *QoS* exigée par les trafics générés par ces objets. Cette méthode de contrôle d'accès basée sur la *QoS* permet d'assurer un traitement différencié entre les différents flux pour respecter les besoins de chacun. Ces besoins sont définis dans un contrat de niveau de service (*iSLA*) souscrit entre un *IoT-C* et un *IoT-SP*. L'évaluation des performances de *QBAIoT*, en termes de délai moyen, *PDR*, *EDR* et *PPR*, montre que notre méthode permet de respecter les exigences de *QoS* des différents types de trafics même dans les cas de congestion. Ainsi, différents scénarios de simulation ont montré que *QBAIoT* offre aux trafics temps réel (i.e., *RTMC* et *RTNMC*) de meilleurs délais comparé à ceux observés en utilisant la méthode du standard IEEE 802.15.4. De même, *QBAIoT* assure dans la majorité des simulations un meilleur taux de livraison des paquets (i.e., *PDR*) pour tous les types de trafics existants dans l'environnement IoT. En comparant *QBAIoT* à une autre méthode basée sur la *QoS*, à savoir *SDA-CSMA/CA*, nous avons obtenu de meilleures performances pour *QBAIoT* en termes de délai moyen malgré une utilisation moins efficace du débit effectif du canal dans certains scénarios. La flexibilité de l'architecture IoT que nous proposons, permet de remédier aux limites de l'accès au canal avec *QBAIoT* dans le cas d'une saturation de ce canal suite à la mise en place d'un nombre important d'objets par classe de *QoS* dans le cadre d'un contrat de type *iSLA*. Ceci est rendu possible grâce à l'implémentation d'une nouvelle passerelle *LL-Gw* utilisant *QBAIoT* afin de respecter les exigences des trafics. *QBAIoT* permet d'assurer un traitement différencié des trafics dans un environnement IoT pour respecter leurs exigences en termes de *QoS*. Afin d'assurer un niveau de service global dans l'IoT, notre Framework doit être étendu afin de prendre en compte un deuxième volet du niveau de service à savoir la sécurité. Nous spécifions dans le chapitre suivant cette extension permettant de garantir un niveau de sécurité associé à un service IoT offert dans le cadre de notre architecture IoT.

Chapitre 6 – Extension du Framework de garantie de niveau de service à la sécurité dans l'Internet des Objets

6.1. Introduction

La sécurité est un enjeu important dans l'environnement IoT permettant de préserver la vie privée des utilisateurs et de sécuriser l'échange d'informations. Par conséquent, une sécurité de bout en bout est essentielle dans ce type d'environnement et nécessite des mécanismes spécifiques au niveau de chaque couche de l'architecture IoT. Afin de caractériser les mécanismes de sécurité à implémenter au niveau des couches de l'IoT et permettant de respecter les exigences des clients, il faut étendre les *SLA* de *QoS* que nous avons spécifiés dans le chapitre 4 en y intégrant des paramètres de sécurité spécifiques à l'IoT. Dans ce contexte, nous étendons les contrats de niveau de service de type *cSLA*, *gSLA* et *iSLA* en prenant en considération des mécanismes de sécurité relatifs aux différentes couches de notre architecture IoT sauf pour la couche réseau où la sécurité peut être assurée grâce à un tunnel établi entre la couche cloud et la *HL-Gw* de la couche sensing sans étendre le contrat de niveau de service de type *nSLA*.

Ainsi, nous spécifions dans la section 2 de ce chapitre les besoins de sécurité au niveau des différentes couches de l'IoT et nous décrivons la nouvelle l'architecture de notre Framework étendu à la sécurité grâce à différents contrats de niveau de service de sécurité (*SecSLA*: Security Service Level Agreement) permettant de respecter les besoins définis pour les différentes couche de l'IoT. Ensuite, nous détaillons dans la section 3 les différents contrats *SecSLA* de notre Framework tout en mettant l'accent sur le processus d'établissement de ces contrats relatifs aux couches de l'IoT. Enfin, la section 4 nous permet de conclure ce chapitre.

6.2. Framework de sécurité dans l'IoT

Afin de garantir la sécurité dans l'environnement IoT, les besoins en termes de sécurité au niveau de chaque couche IoT doivent être étudiés. Nous présentons dans cette section notre étude concernant les besoins de sécurité des différentes couches de l'IoT (i.e., sensing, réseau et cloud) et nous étendons l'architecture de notre Framework en y intégrant de nouveaux contrats de niveau de service portant sur la sécurité dans l'IoT. Cette dernière sera utilisée comme base pour notre contribution de sécurité dans l'IoT.

6.2.1. Besoins de sécurité pour les couches de l'IoT

Notre proposition d'architecture IoT (voir chapitre 4) comprend trois couches, à savoir la couche sensing, la couche réseau et la couche cloud. Nous classifions dans ce qui suit les besoins de sécurité en fonction de ces trois couches.

6.2.1.1. Besoins de sécurité pour la couche sensing

Au niveau de la couche sensing, les données sont récoltées par les objets IoT et ensuite transférées aux passerelles *LL-Gw* et *HL-Gw* à travers des canaux sans fil. Par conséquent, différents besoins de sécurité s'avèrent important au niveau de cette couche.

Premièrement, les données doivent être protégées à travers les services de confidentialité et d'intégrité en utilisant des mécanismes de chiffrement et de hachage légers. En effet, les objets IoT ont des capacités limitées en termes de mémoire et de traitement induisant un besoin d'utiliser de nouveaux algorithmes de chiffrement et de hachage adaptés à cet environnement.

Dans ce contexte, la gestion et l'échange des clés doivent être pris en compte au niveau de la couche sensing afin d'assurer une communication sécurisée entre les objets IoT et les passerelles de bas niveau. De plus, l'intégrité de l'objet IoT est essentielle au niveau de la couche sensing car ces objets peuvent être implémentés dans différents environnements présentant des risques de sécurité variables. Ainsi, les objets IoT peuvent être sujets aux attaques physiques ayant une incidence directe sur l'intégrité des logiciels et des systèmes d'exploitation implémentés sur ces derniers. Pour assurer cette intégrité, différentes techniques de protection de l'intégrité des objets IoT ont été proposées par différents organismes et projets comme le module *IMA* [104] [105] et la libraire des fonctions de hachage « Cryptosuite » [119] (cf. chapitre 3).

D'autre part, la traçabilité des actions effectuées par les objets IoT est nécessaire au niveau de la couche sensing. Cette traçabilité assure les services de sécurité de type non répudiation et authentification grâce à la signature des actions. En effet, les signatures envoyées via les applications IoT permettent à différentes entités (i.e., passerelles ou objets) de s'assurer de l'identité des sources des actions. Cette traçabilité doit être complétée par un contrôle de l'identité des objets IoT et des passerelles à travers l'utilisation des jetons, des certificats, etc. L'identification et l'authentification doivent être couplées avec un contrôle d'accès des objets IoT et des passerelles en utilisant par exemple des politiques basées sur différents modèles (i.e., *RBAC, ABAC, CapBAC* [101]).

Finalement, la couche sensing nécessite l'utilisation des mécanismes de contrôle d'intrusion (*IDS* : Intrusion Detection System) pour assurer la disponibilité des services IoT à travers la disponibilité de l'infrastructure d'objets connectés via les passerelles. Le Tableau 6.1 résume les différents besoins de sécurité de la couche sensing ainsi que les services et les mécanismes qui peuvent les assurer dans un environnement IoT. Ces mécanismes ont été étudiés en détails dans le

chapitre 3 à travers les algorithmes de chiffrements et de hachage légers, les techniques de vérification d'intégrité des objets IoT, les politiques de contrôle d'accès, etc.

En plus des besoins de sécurité classiques, il est primordial de penser aussi à la protection de la vie privée des clients IoT. Par conséquent, des réponses appropriées doivent être fournies aux utilisateurs des services IoT concernant différentes garanties à apporter au niveau de la couche sensing concernant les questions suivantes :

- Où sont stockées les données et les sauvegardes de secours, s'il y a besoin, au niveau de la couche sensing?
- Comment les données sont-elles stockées dans les passerelles ?
- Comment transitent les données à l'intérieur de l'infrastructure sensing ?
- Comment les identités des objets et des passerelles sont-elles gérées ?
- Comment la correspondance entre les objets et leurs utilisateurs est-elle gérée ?
- Qui a accès aux données stockées au niveau la couche sensing ?

Tableau 6.1 : Besoins, services et mécanismes de sécurité au niveau de la couche sensing

Besoins	Services	Mécanismes
Protection des données	Confidentialité / Intégrité	Chiffrement / Hachage
Gestion des clés	Confidentialité / Authentification / Intégrité	Chiffrement des clés avec une infrastructure PKI
Protection de l'intégrité de l'objet	Intégrité	Hachage des liste des logiciels / TPM
Traçabilité des actions effectuées	Non répudiation / Authentification	Signature des actions des passerelles
Contrôle d'identité des nœuds et des passerelles	Identification / Authentification	Login/MDP – Jeton – Certificats
Contrôle d'accès des nœuds et des passerelles	Contrôle d'accès	Politique de sécurité (RBAC/ABAC/CapBAC, etc.)
Disponibilité des services	Disponibilité	IDS

6.2.1.2. Besoins de sécurité pour la couche réseau

Au niveau de la couche réseau, différents services et mécanismes de sécurité sont nécessaires pour assurer une sécurité de bout en bout dans un environnement IoT. Ainsi, les données acheminées depuis la couche sensing vers la couche cloud doivent être protégées durant leur transmission au niveau de la couche réseau.

Cette protection est assurée grâce aux services de confidentialité et d'intégrité. Au niveau de cette couche réseau, les algorithmes de chiffrement et de hachage traditionnels peuvent être utilisés comme les équipements de cette couche sont performants en termes de capacités de calcul et de stockage. De plus, la gestion des clés symétriques via une infrastructure à clés publiques est importante pour assurer l'échange des clés secrètes. De même, la traçabilité des actions est nécessaire et elle est fournie via la signature des actions. Le Tableau 6.2 résume les différents besoins, services et mécanismes de sécurité de la couche réseau dans un environnement IoT.

Tableau 6.2 : Besoins, services et mécanismes de sécurité au niveau de la couche réseau

Besoins	Services	Mécanismes
Protection des données	Confidentialité / Intégrité	Chiffrement / Hachage
Gestion des clés	Confidentialité / Authentification / Intégrité	Chiffrement avec une infrastructure PKI
Traçabilité des actions effectuées	Non répudiation / Authentification / Non répudiation	Signature des actions

6.2.1.3. Besoins de sécurité pour la couche cloud

La couche cloud nécessite la mise en place de mesures de sécurité spécifiques pour assurer une sécurité de bout en bout aux services IoT. Dans ce contexte, les services de confidentialité et de hachage permettent de répondre à plusieurs besoins de sécurité pour la couche cloud. En effet, ces services permettent de protéger les données sauvegardées et traités au niveau du cloud tout en protégeant la migration de ces données ainsi que la migration des machines virtuelles et des applications entre les équipements physiques du cloud. Pour ce faire, les algorithmes traditionnels de chiffrement et de hachage sont utilisés comme l'infrastructure cloud comprend des capacités importantes en termes de stockage et de calcul. De plus, une gestion efficace des clés secrètes via une infrastructure à clés publiques est nécessaire.

D'autre part, la traçabilité des actions effectuées par des utilisateurs et des applications doit être prise en considération au niveau de la couche cloud. Ces besoins de sécurité peuvent être satisfaits grâce aux signatures des actions, et au accusés de réception. Enfin, la disponibilité des services cloud est un besoin de sécurité primordial qui peut être offert via des outils de type *IDS* adaptés à la couche cloud. Le Tableau 6.3 résume les différents besoins, services et mécanismes de sécurité de la couche cloud de notre architecture IoT.

Tableau 6.3 : Besoins, services et mécanismes de sécurité au niveau de la couche cloud

Besoins	Services	Mécanismes
Protection des données	Confidentialité / Intégrité	Chiffrement / Hachage
Protection de la migration des données / machines virtuelles / applications	Confidentialité / Intégrité	Chiffrement / Hachage
Protection des sauvegardes	Confidentialité / Intégrité	Chiffrement / Hachage
Gestion des clés	Confidentialité / Authentification / Intégrité	Chiffrement des clés symétriques avec une PKI
Traçabilité des actions effectuées	Non répudiation / Authentification / Accusé de réception	Signature des actions
Disponibilité des services (IaaS, SaaS, PaaS)	Disponibilité	IDS

En plus des besoins de sécurité classiques, il est primordial de penser aussi à la protection de la vie privée des clients IoT. Par conséquent, des réponses appropriées doivent être fournies aux

utilisateurs des services IoT concernant différentes garanties à apporter au niveau de la couche cloud concernant les questions suivantes :

- Où sont stockées les données et les sauvegardes de secours de la couche cloud ?
- Comment les données sont-elles stockées dans le cloud ?
- Comment les données transitent-elles à l'intérieur de la couche cloud ?
- Comment les identités des utilisateurs de services IoT sont-elles gérées au niveau du cloud ?
- Qui a accès aux données stockées au niveau des ressources du cloud ?
- Est que la vente des données des services IoT au niveau de la couche cloud est possible ?

6.2.2. Architecture de sécurité pour l'IoT

Afin d'assurer une sécurité de bout en bout dans un environnement IoT, nous étendons notre architecture IoT en mettant en place des mécanismes de sécurité au niveau de chacune de ces trois couches, à savoir la couche sensing, la couche réseau et la couche cloud. Nous présentons dans la Figure 6.1 notre architecture de sécurité qui permet à l'*IoT-SP* de négocier les besoins et exigences de sécurité avec les fournisseurs cloud et réseau ainsi que les clients de services IoT.

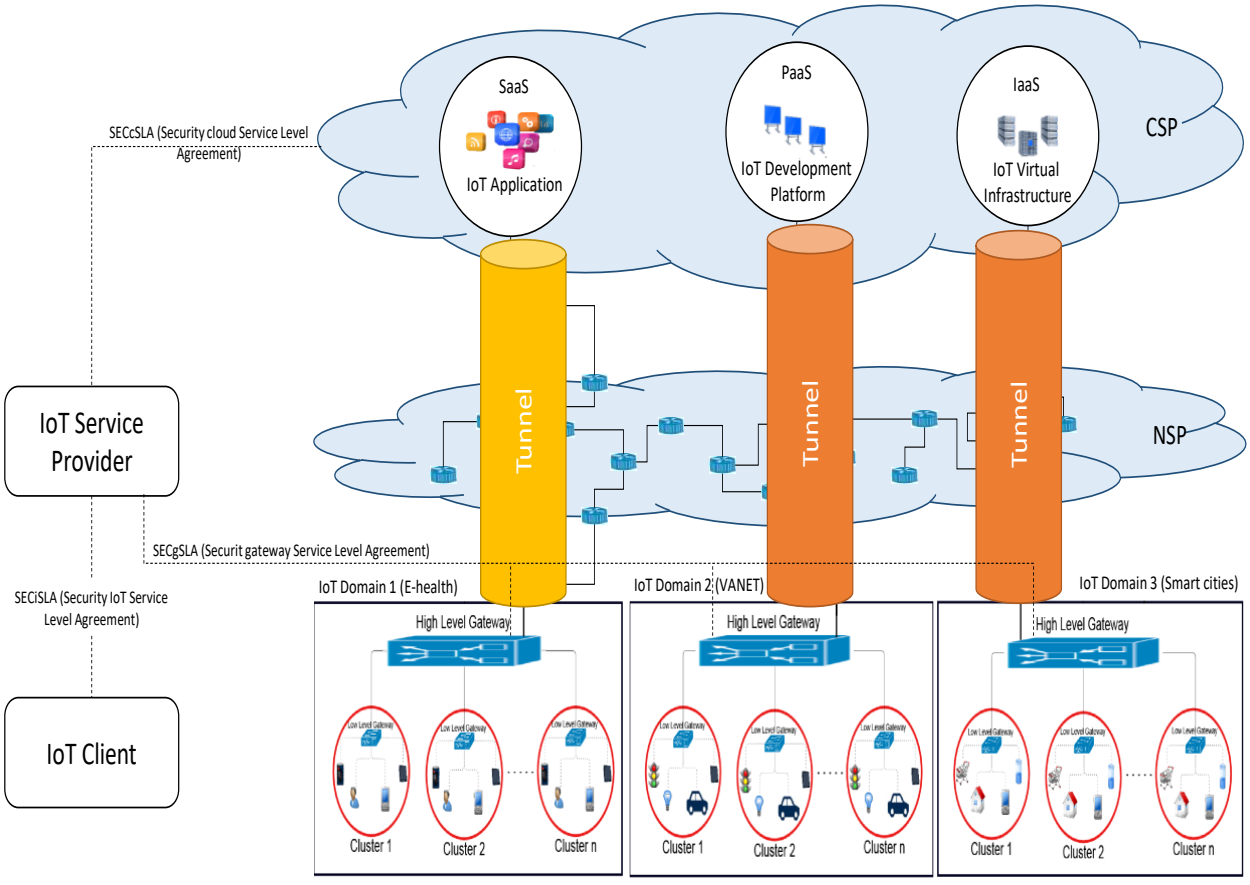

Figure 6.1 : Architecture de sécurité de l'environnement IoT

Dans le cadre de cette architecture de sécurité, nous proposons différents types de contrats de niveau de service portant sur la sécurité. Ainsi, nous spécifions entre l'*IoT-SP* et un *CSP* un contrat de sécurité appelé *SECcSLA* (SECurity cloud Service Level Agreement). De plus nous mettons en place un contrat de sécurité interne à l'*IoT-SP* appelé *SECgSLA* (SECurity gateway Service Level Agreement) afin de prendre en considération les exigences de sécurité des passerelles IoT de la couche sensing.

En ce qui concerne la couche réseau de notre architecture IoT, aucun contrat de niveau de service portant sur la sécurité n'est conclu avec le *NSP* comme des tunnels sont mis en place entre les passerelles de haut niveau (*HL-Gw*) de la couche sensing et le cloud afin d'assurer tous les besoins de sécurité au niveau de la couche réseau. D'une part, ces tunnels, de couleur orange sur la Figure 6.1, sont configurés par l'*IoT-SP* dans le cas de souscription d'un service cloud de type *IaaS* ou *PaaS* auprès du *CSP*. D'autre part, ce tunnel de couleur jaune sur la Figure 6.1, est configuré par le *CSP* dans le cas de souscription par l'*IoT-SP* d'un service cloud de type *SaaS*.

Les deux contrats de niveau de service portant sur la sécurité concernant les couches sensing et cloud (i.e., *SECgSLA* et *SECcSLA*) mais aussi les tunnels déployés au niveau de la couche réseau afin d'interconnecter les *HL-Gw* de la couche sensing et les infrastructures cloud permettent de conclure un contrat de niveau de service global portant sur la sécurité de bout en bout pour un service IoT. Ce contrat, appelé *SECiSLA* (SECurity IoT Service Level Agreement), est établi entre l'*IoT-SP* et un *IoT-C*.

6.3. Extension des SLA du Framework

Les *SLA* de *QoS* que nous avons spécifiés dans le chapitre 4 doivent être étendus pour prendre en considération les besoins de sécurité de l'architecture IoT décrits dans la section précédente. Nous présentons dans ce qui suit notre extension de sécurité des différents contrats de niveaux du Framework. A ce titre le contrat *SECgSLA* concerne la couche sensing, le contrat *SECcSLA* concerne la couche cloud et le contrat *SECiSLA* concerne le service IoT global en prenant en considération l'ensemble des couches de notre architecture de sécurité IoT. *SECgSLA, SECcSLA* et *SECiSLA* sont des extensions des contrats *gSLA, cSLA* et *iSLA* respectivement. Par conséquent, les informations concernant la validité, les partenaires et l'identification des services sont identiques dans le contrat initial et l'extension.

6.3.1. SLA de sécurité pour la couche cloud : SECcSLA

Afin de prendre en considération les besoins de sécurité relatifs à la couche cloud de l'architecture IoT décrits dans la section 6.2 de ce chapitre, nous proposons l'établissement d'un contrat de niveau de service de sécurité, à savoir *SECcSLA*, avec le fournisseur cloud (i.e., *CSP*). Le contrat de type *SECcSLA* définit différents paramètres que nous décrivons dans la section suivante (voir Figure 6.2).

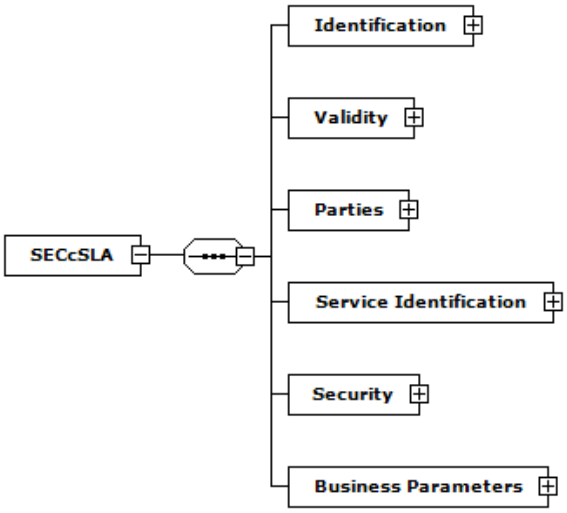

Figure 6.2 : Représentation globale du schéma XML du SECcSLA

6.3.1.1. Paramètres du SECcSLA

Le contrat de sécurité de type *SECcSLA* relatif à la couche cloud de l'architecture IoT comporte des informations qui permettent de l'identifier à travers un *SECcSLA ID* et de l'associer, à travers le *cSLA ID*, au contrat de *QoS* de type *cSLA* qu'il permet d'étendre à la sécurité. D'autre part, le contrat *SECcSLA* dispose des mêmes informations que le *cSLA* (cf. section 4.3.1.2) en ce qui concerne la validité du contrat, les partenaires du contrat (*IoT-SP* étant le client et *CSP* étant le fournisseur) et l'identification du service (voir Figure 6.3).

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- New XML document created with EditiX XML Editor (http://www.editix.com) at Thu Jun 22 19:27:59 CEST 2017 -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

.......

    <xs:element name="SECcSLA" >
        <xs:complexType >
            <xs:sequence >
                <xs:element minOccurs="1" maxOccurs="1" name="Identification">
                    <xs:complexType >
                            <xs:sequence >
                                <xs:element minOccurs="1" maxOccurs="1" name="SECcSLA ID" type="xs:integer" />
                                <xs:element minOccurs="1" maxOccurs="1" name="cSLA ID" type="xs:integer" />
                                </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element minOccurs="1" maxOccurs="1" name="Validity" >
                <xs:element minOccurs="1" maxOccurs="1" name="Parties" >
                <xs:element minOccurs="1" maxOccurs="1" name="Service Identification" >
                <xs:element minOccurs="1" maxOccurs="1" name="Security" >
                <xs:element minOccurs="1" maxOccurs="1" name="Business Parameters" >
                <xs:complexType >
            </xs:element>
            </xs:sequence >
        </xs:complexType >
    </xs:element>
</xs:schema>
```

Figure 6.3 : Schéma XML du SECcSLA montrant la partie identification

De plus, le contrat de sécurité de type *SECcSLA* comporte une partie sécurité permettant de définir les besoins et les caractéristiques qui en découlent au niveau de l'infrastructure cloud. Les paramètres de sécurité dans l'attribut « Security » du *SECcSLA* sont regroupés suivant le service cloud souscrit entre l'*IoT-SP* et le *CSP* à savoir *IaaS*/*PaaS* ou *SaaS*. Pour chaque service ou groupe de service cloud, les garanties de sécurité peuvent être spécifiées d'une manière quantitative ou qualitative (voir Figure 6.4). Les paramètres qualitatifs se déclinent en 3 classes (i.e., Gold, Silver et Bronze). Chaque classe correspond à un mappage bien déterminé d'un ensemble des paramètres de sécurité quantitatifs prédéfinis par le fournisseur de service IoT.

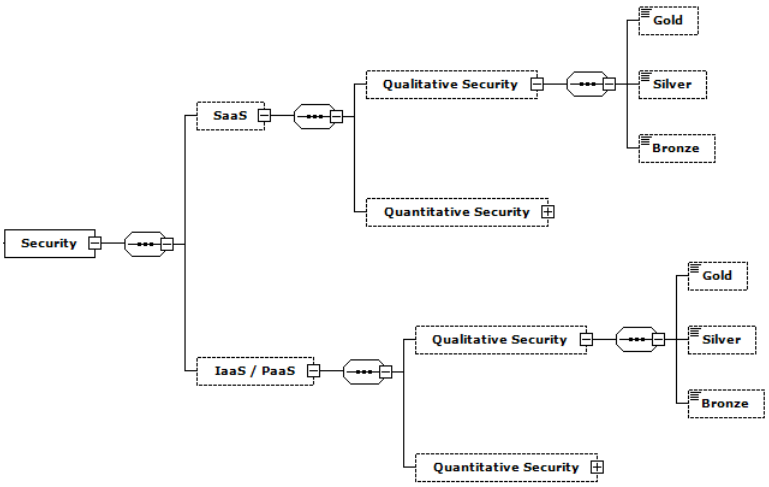

Figure 6.4 : Représentation du schéma XML des paramètres de sécurité du SECcSLA

En ce qui concerne les paramètres de sécurité quantitatifs de l'attribut « Security » du *SECcSLA*, nous spécifions dans un premier temps ceux relatifs au service *SaaS* à travers les Figures 6.5, 6.6 et 6.7. Par la suite, nous spécifions ceux relatifs aux services *IaaS/PaaS* à travers la Figure 6.8.

Ainsi, lors de la souscription d'un service *SaaS*, l'*IoT-SP* délègue l'implémentation de tous les mécanismes de sécurité au *CSP*, qui est dans ce cas responsable de la totalité du service IoT au niveau de la couche cloud. Par conséquent, le contrat *SECcSLA* spécifie dans les paramètres de sécurité quantitatifs relatifs au service *SaaS* les caractéristiques des algorithmes de chiffrement et de hachage permettant d'offrir la confidentialité et l'intégrité des données stockées dans le cloud (voir Figure 6.5). D'une part, la partie confidentialité des données comporte la liste des algorithmes de chiffrement disponibles au niveau de la couche cloud (i.e., *AES*, Blow-Fish, RIVEST Cipher, *3DES*). D'autre part, la partie intégrité des données comporte une liste d'algorithmes de hachage (i.e., *SHA*, Streebog, *MD5*) disponibles au niveau de la couche cloud. De plus, pour chaque algorithme de chiffrement, différentes tailles de clés existent et pour chaque algorithme de hachage différentes versions et longueurs de digest sont disponibles. . D'autres services de sécurité obligatoires sont présents avec une occurrence de [1,1] dans les paramètres de sécurité du *SaaS*. Nous citons à ce titre, l'identification, l'authentification. Dans ce contexte, les utilisateurs des services IoT doivent être

identifiés via un portail de l'application IoT en utilisant un login et un mot de passe, alors que les passerelles *HL-Gw* doivent être authentifiées auprès de l'infrastructure cloud en utilisant des techniques et des solutions plus avancés comme le *SAML* [93], *EAP* (Extensible Authentication Protocol) [169] ou bien *RADIUS* (Remote Authentication Dial-In User Service) [170].

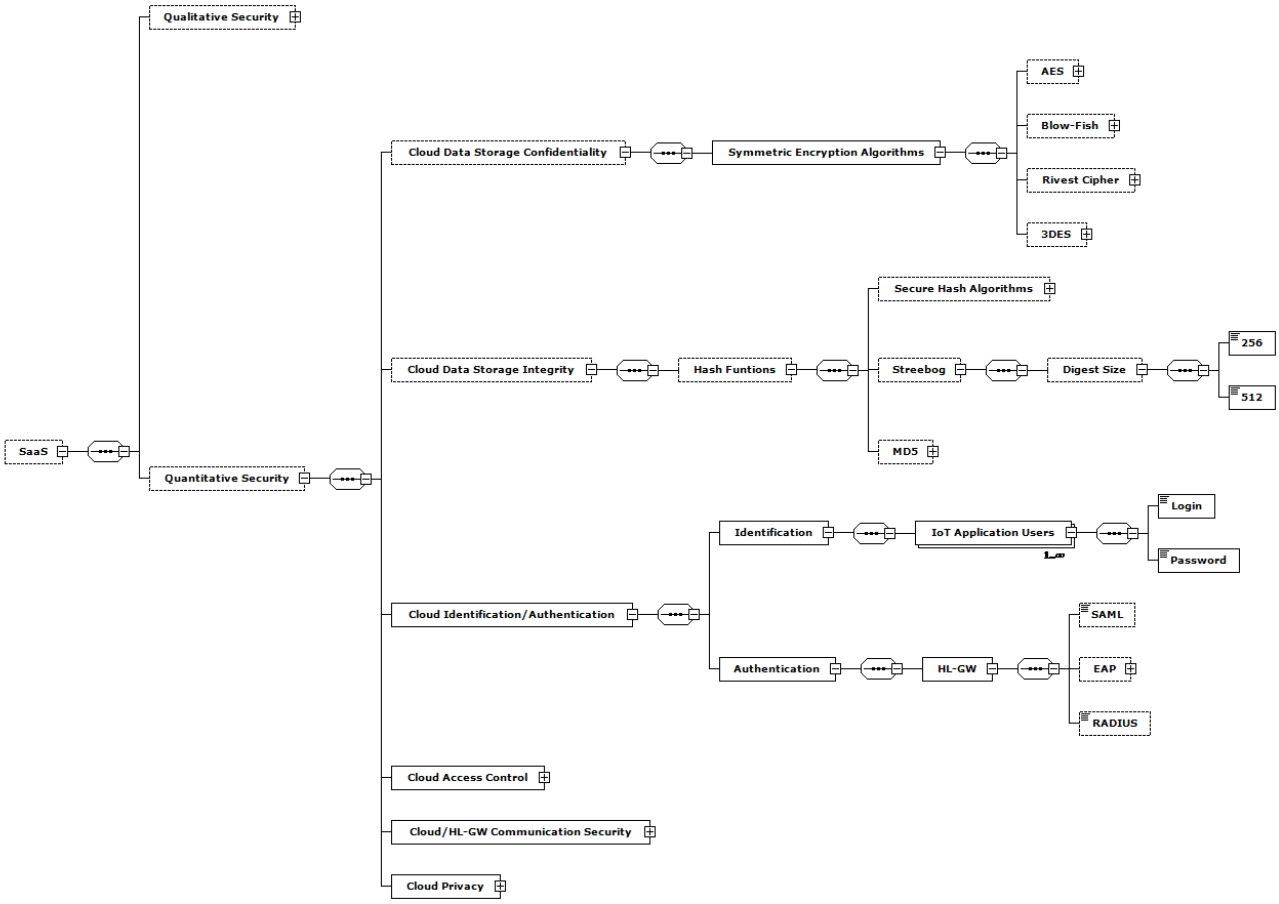

Figure 6.5 : Paramètres de sécurité quantitatifs du SECcSLA (SaaS) : confidentialité, intégrité et identification / authentification

En ce qui concerne le contrôle d'accès dans le cadre d'un service *SaaS* ayant une occurrence de [1,1] (voir Figure 6.6), nous distinguons le contrôle d'accès des utilisateurs IoT et le contrôle d'accès des passerelles *HL-Gw*.

D'une part, le contrôle d'accès des utilisateurs est basé sur les rôles (i.e., utilisateur normal, utilisateur avancé, administrateur) avec différents types de permissions (i.e., lecture, écriture, modification, exécution de tâches, configuration). D'autre part, le contrôle d'accès des passerelles *HL-Gw* est basé sur les attributs. Ces attributs correspondent aux passerelles (i.e., adresse IP, adresse *MAC*, etc.) ou bien à l'environnement (i.e., temps de service, etc.) et permettent de prendre des décisions quant aux actions possibles (i.e. lecture, écriture, modification, exécution) ainsi que les ressources concernées (i.e., Application ID et Database ID).

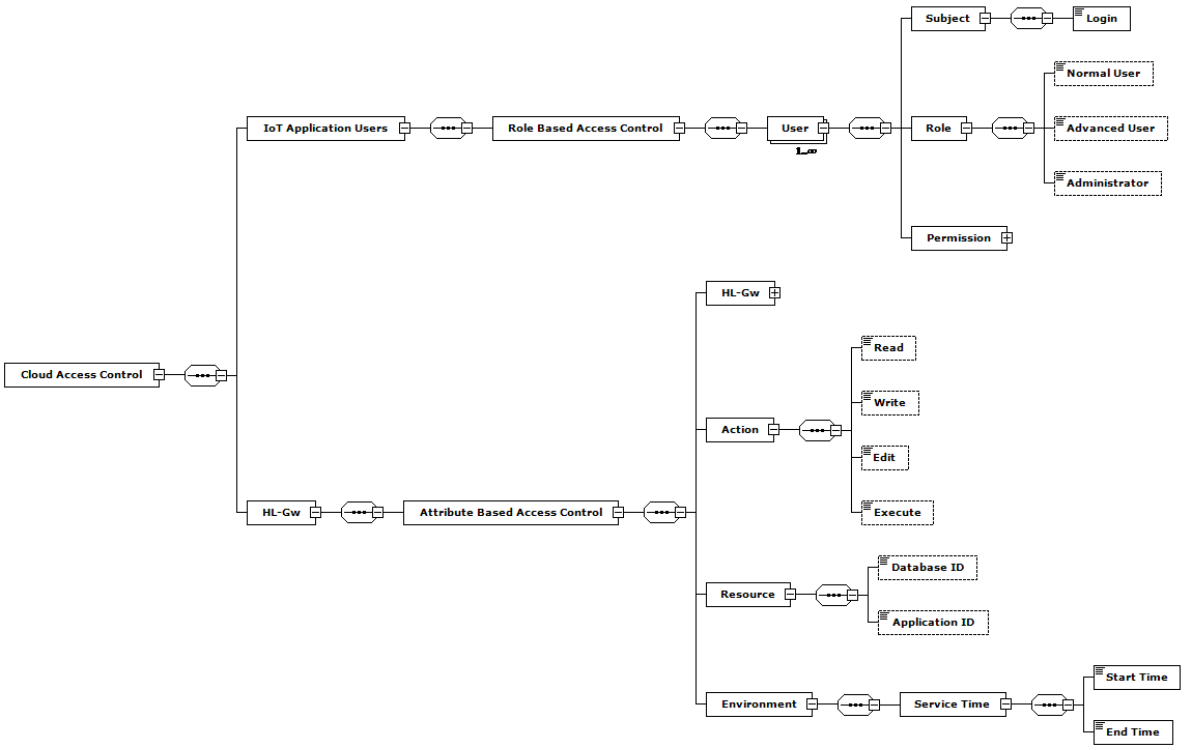

Figure 6.6 : Paramètres de sécurité quantitatifs du SECcSLA (SaaS) : Contrôle d'accès

La sécurité des communications entre le cloud et la *HL-Gw* ainsi que la protection de la vie privée dans le cloud sont aussi des éléments essentiels dans le *SECcSLA*. A ce titre, nous présentons dans la Figure 6.7 les deux paramètres « HL-Gw Communication Security » et « Cloud Privacy ». Le premier paramètre de sécurité spécifie un tunnel implémenté par le *CSP* entre la *HL-Gw* et l'infrastructure cloud en utilisant une technique de tunneling parmi une liste (i.e., *IPSec* (Internet Protocol Security), *L2TP* (Layer 2 Tunneling Protocol), *SSTP* (Secure Socket Tunneling Protocol), *PPTP* (Point-to-Point Tunneling Protocol)).

Le deuxième paramètre concernant la protection de la vie privée dans le cloud spécifie la localisation du stockage des informations récoltées et des sauvegardes de secours, la possibilité de vente des données des utilisateurs à des partenaires, l'utilisation des techniques d'anonymisation des données, etc. Après avoir défini les paramètres de sécurité quantitatifs du *SECcSLA* relatif à un service de type *SaaS*, nous décrivons dans ce qui suit ceux relatifs aux services *PaaS* et *IaaS* que nous avons regroupé dans le contrat *SECcSLA* étant donné que les mécanismes de sécurité qui incombent au *CSP* lors de la souscription par l'*IoT-SP* d'un service *IaaS* ou *PaaS* sont identiques. En effet, la responsabilité du *CSP* en termes de sécurité dans le cadre d'un service *IaaS* ou *PaaS* est limitée à la confidentialité et l'intégrité des données stockées dans le cloud ainsi que la protection de la vie privée concernant ces données alors que le reste des services de sécurité (contrôle d'accès, authentification/identification, etc.) est à la charge de l'*IoT-SP* étant donné qu'il est responsable de l'infrastructure (cas du service *IaaS*) ou bien de la plateforme (cas du service *PaaS*) hébergeant le service IoT.

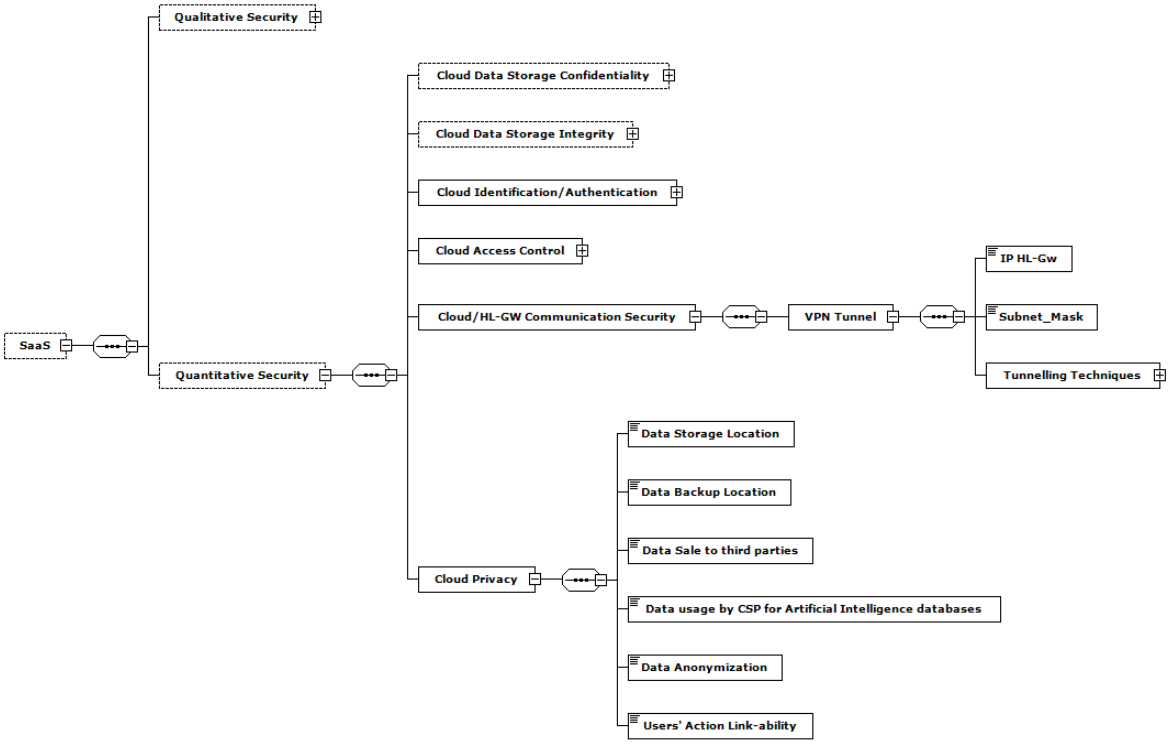

Figure 6.7 : Paramètres de sécurité quantitatifs du SECcSLA (SaaS) : communication cloud/HL-Gw et protection de la vie privée dans le cloud

La Figure 6.8 montre les paramètres de sécurité quantitatifs relatifs aux services *IaaS* et *PaaS* dans le *SECcSLA*. Nous présentons à ce titre, la confidentialité des données à travers une liste d'algorithmes de chiffrement et leur intégrité dans le cloud à travers une liste d'algorithmes de hachage. Les paramètres de sécurité quantitatifs de l'*IaaS*/*PaaS* incluent aussi obligatoirement la protection de la vie privée dans le cloud avec des attributs identiques à ceux utilisés dans le cadre d'un service de type *SaaS*.

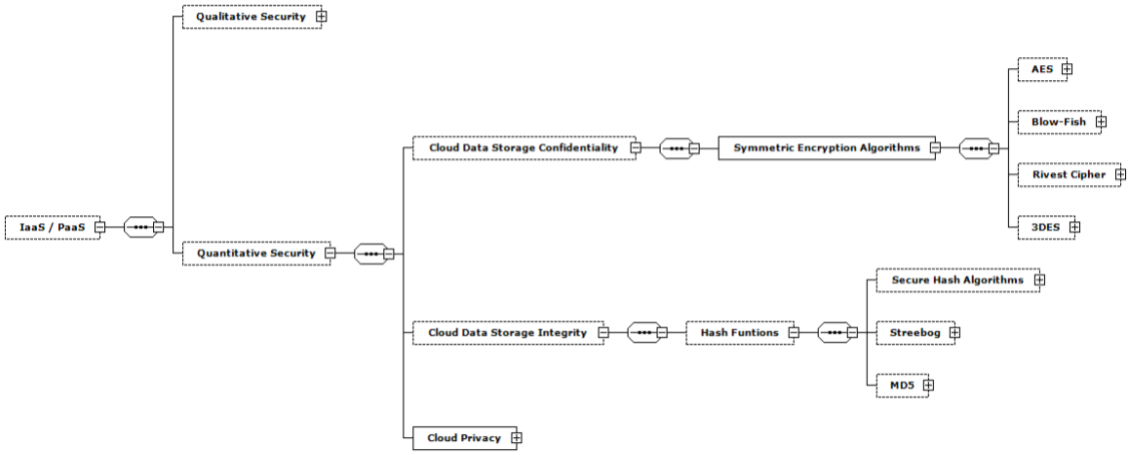

Figure 6.8 : Paramètres de sécurité quantitatifs du SECcSLA (IaaS/PaaS)

La dernière partie du *SECcSLA* comporte les paramètres commerciaux qui se limitent au coût contrairement au *cSLA* qui inclut la gestion des violations et des pénalités. En effet, les *SLA* de sécurité permettent de choisir des algorithmes, des mécanismes et des services de sécurité à appliquer dans l'environnement IoT et ne sont pas associés à des seuils de performance qui déclenchent des violations comme c'est le cas avec les paramètres de *QoS* dans les *SLA* de *QoS*. La Figure 6.9 présente l'attribut « Business Parameters » du *SECcSLA* qui définit le coût de sécurité en fonction du type de service cloud concerné (i.e., *SaaS*, *PaaS* et *IaaS*).

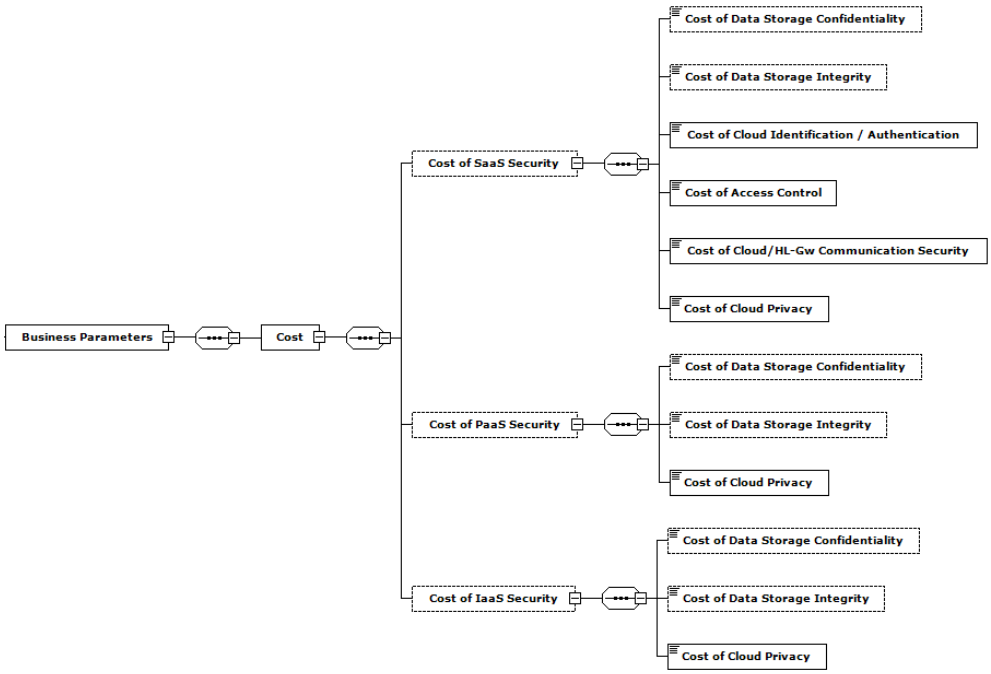

Figure 6.9 : Représentation du schéma XML des paramètres commerciaux du SECcSLA

En effet, suivant le service cloud souscrit et la spécificité des mécanismes choisis au sein de chaque service de sécurité, la tarification varie. Par exemple, le choix d'un algorithme de chiffrement complexe avec une longueur de clé importante implique un coût du *SECcSLA* plus élevé. Ainsi, pour un service de type *SaaS* souscrit, le coût peut dépendre de la nature et de la taille de la clé de l'algorithme de chiffrement, de la nature de la taille du digest de l'algorithme de hachage, du type d'authentification de la *HL-Gw*, du contrôle d'accès des *HL-Gw* et des utilisateurs des services IoT, de la nature du tunnel choisi et des mécanismes de protection de la vie privée.

6.3.1.2. Procédure d'établissement du SECcSLA

Différents échanges doivent avoir lieu entre l'*IoT-SP* et le *CSP* pour établir le *SECcSLA*. Ces échanges permettent aux différentes parties de se mettre d'accord sur les paramètres de sécurité demandés par l'*IoT-SP*. Nous spécifions dans la Figure 6.10 l'ensemble des états de l'automate fini (*FSM* : Finite State Machine) concernant l'établissement du *SECcSLA* au niveau de l'*IoT-SP* .

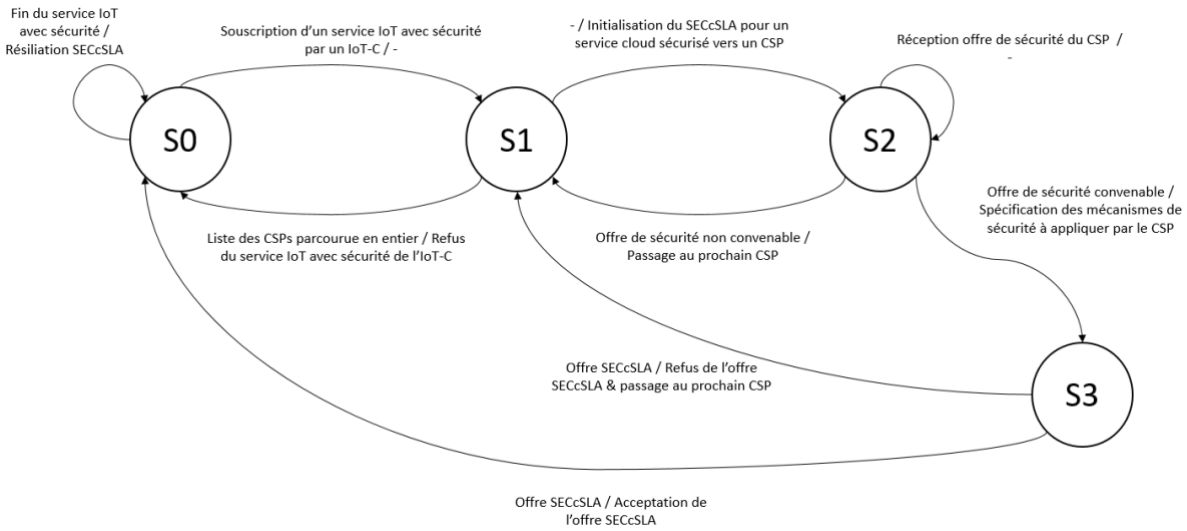

Figure 6.10: FSM de l'IoT-SP pour l'établissement du SECcSLA

Dans l'état S0, l'*IoT-SP* attend la demande de souscription d'un nouveau service IoT avec garantie de sécurité émanant d'un *IoT-C*. Suite à la réception de cette demande, l'*IoT-SP* passe à l'état S1. Dans cet état, il envoie une requête d'initialisation d'un *SECcSLA* vers un *CSP* avant de passer vers l'état S2. L'*IoT-SP* reste dans l'état S2 en recevant l'offre de sécurité du *CSP* correspondant. Si l'offre de sécurité reçue du *CSP* n'est pas convenable, l'*IoT-SP* revient à l'état S1 et change de *CSP*. Si, dans l'état S1, l'*IoT-SP* a déjà refusé toutes les offres de sécurité de tous les *CSP* disponibles, l'*IoT-SP* repasse à l'état S0 en refusant le service IoT demandé par l'*IoT-C*. Sinon, si dans l'état S2, l'offre de sécurité reçue du *CSP* est convenable, l'*IoT-SP* envoie la spécification des mécanismes de sécurité à appliquer vers le *CSP* et passe à l'état S3. Dans cet état S3, l'*IoT-SP* reçoit l'offre *SECcSLA* du *CSP* et passe vers l'état S1 en changeant de *CSP* dans le cas où l'offre *SECcSLA* n'a pas été acceptée (à cause d'un problème de tarification par exemple). Par contre, dans l'état S3, si l'offre *SECcSLA* est accepté par l'*IoT-SP*, ce dernier passe vers l'état S0 et attend la fin du service IoT avec garantie de sécurité pour résilier le *SECcSLA* correspondant.

6.3.2. SLA de sécurité pour la couche sensing : SECgSLA

Afin de prendre en considération les besoins de sécurité relatifs à la couche sensing de l'architecture IoT décrits dans la section 6.2 de ce chapitre, nous spécifions le contrat de type *SECgSLA* avec différents paramètres que nous décrivons dans la section suivante.

6.3.2.1. Paramètres du SECgSLA

Le contrat *SECgSLA* est un contrat interne à l'*IoT-SP* permettant de spécifier les mesures de sécurité à mettre en place au niveau de la couche sensing afin de contribuer à l'offre de sécurité globale exigée par l'*IoT-C*. Ce contrat comprend 5 parties à savoir : « Identification », « Validity », « Parties », « Service Identification » et « Security » (voir Figure 6.11). Ainsi, le contrat de sécurité

de type *SECgSLA* comporte des informations qui permettent de l'identifier à travers un *SECgSLA ID* et de l'associer, à travers le *gSLA ID*, au contrat de *QoS* de type *gSLA* qu'il permet d'étendre à la sécurité. D'autre part, le contrat *SECgSLA* dispose des mêmes informations que le *gSLA* (cf. section 4.3.3.2) en ce qui concerne la validité du contrat, les partenaires du contrat (*IoT-SP* étant le client et *HL-Gw* étant le fournisseur) et l'identification du service (voir Figure 6.11). Le contrat *SECgSLA* ne comprend pas de paramètres commerciaux comme il s'agit d'un contrat interne à l'*IoT-SP*.

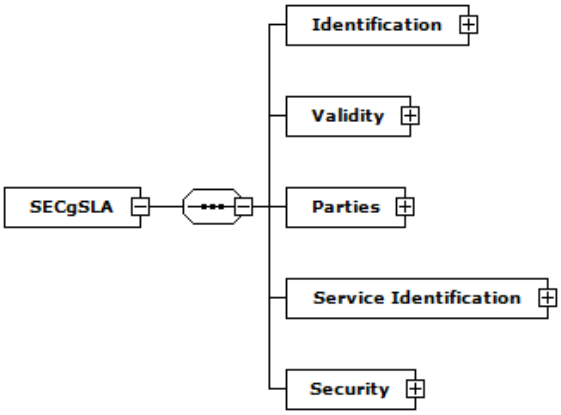

Figure 6.11 : Représentation globale du schéma XML du SECgSLA

Les paramètres de sécurité de l'attribut « Security » du *SECgSLA* (voir Figure 6.12) peuvent être spécifiés d'une façon qualitative (i.e., Gold, Silver, Bronze) ou quantitative sachant qu'un niveau de sécurité décrit d'une façon qualitative est mappé à un ensemble de paramètres de sécurité quantitatifs spécifiés par l'*IoT-SP*.

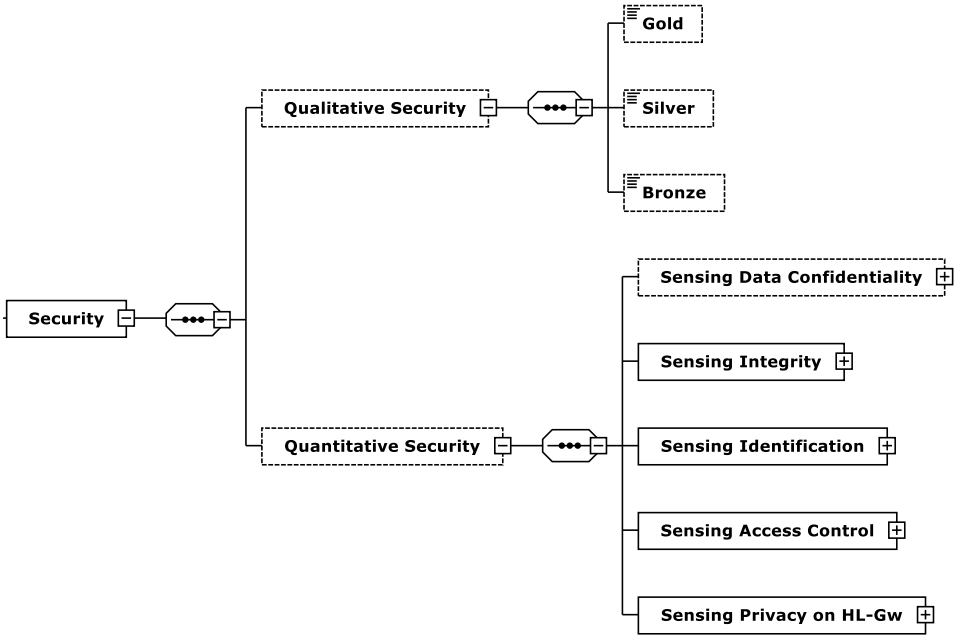

Figure 6.12 : Représentation du schéma XML des paramètres de sécurité du SECgSLA

Les paramètres de sécurité quantitatifs du *SECgSLA* concernent la confidentialité, l'intégrité, l'identification, le contrôle d'accès et la vie privée au niveau de la couche sensing à travers les attributs « Sensing Data Confidentiality », « Sensing Integrity », « Sensing Identification », « Sensing Access Control » et « Sensing Privacy on HL-Gw ». Ainsi, la confidentialité et l'intégrité des données au niveau de la couche sensing sont spécifiées à travers un ensemble d'algorithmes de chiffrement et de hachage adaptés aux objets ayant des contraintes en termes de capacités de traitement et de mémoire (voir Figure 6.13).

```
<xs:element minOccurs="0" maxOccurs="1" name="Sensing Data Confidentiality">
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="Symmetric Encryption Algorithms for Constrained Objects">
                <xs:complexType >
                    <xs:sequence >
                        <xs:element minOccurs="0" maxOccurs="1" name="Curupira" >
                        <xs:element minOccurs="0" maxOccurs="1" name="Katan" >
                        <xs:element minOccurs="0" maxOccurs="1" name="Ktantan" >
                        <xs:element minOccurs="0" maxOccurs="1" name="Present" >
                        <xs:element minOccurs="0" maxOccurs="1" name="Simon" >
                        <xs:element minOccurs="0" maxOccurs="1" name="Speck">
                        <xs:element minOccurs="0" maxOccurs="1" name="RECTANGLE" >
                        <xs:element minOccurs="0" maxOccurs="1" name="HIGHT">
                        <xs:element minOccurs="0" maxOccurs="1" name="CLEFIA" >
                        <xs:element minOccurs="0" maxOccurs="1" name="CAMELLIA">
                        <xs:element minOccurs="0" maxOccurs="1" name="TWINE">
                        <xs:element minOccurs="0" maxOccurs="1" name="XTEA">
                        <xs:element minOccurs="0" maxOccurs="1" name="mCrypton">
                        <xs:element minOccurs="0" maxOccurs="1" name="DESLX">
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 6.13 : Paramètres de sécurité quantitatifs du SECgSLA : Confidentialité

L'intégrité des objets IoT de la couche sensing est préservée en définissant des groupes d'objets avec les mêmes caractéristiques (logiciels installés par exemple) dont le digest est calculé grâce au Trusted Platform Module (*TPM*) ou encore à l'Integrity Measurement Architecture (*IMA*). De même, nous préservons l'intégrité des passerelles de bas niveau et de haut niveau (*LL-Gw* et *HL-Gw*) en utilisant le module *TPM* ou l'architecture *IMA* pour vérifier leurs digests (voir Figure 6.14). Ce digest nous permet de déterminer l'empreinte digitale utilisée dans d'autres services de sécurité telle que la vérification d'identité.

De plus, les objets et les passerelles doivent être identifiés au niveau de la couche sensing. Par conséquent, nous définissons dans l'attribut « Sensing Identification » (voir Figure 6.15) du *SECgSLA* les paramètres utilisés pour leurs identifications. Ainsi, l'identification des objets IoT est basée sur l'appartenance à un groupe d'objets ayant chacun un identifiant et un type de service IoT (i.e., *RTMC*, *RTNMC*, *Streaming*, *NRT*).Le groupe d'objets comporte aussi les identifiants des passerelles *LL-Gw* impliquées dans l'offre d'un service IoT ainsi que son empreinte digitale. De même, les passerelles *LL-Gw* et *HL-Gw* sont identifiées via leurs identifiants (*LL-Gw ID* et *HL-Gw ID*) et leurs empreintes digitales grâce au même attribut « Sensing Identification ».

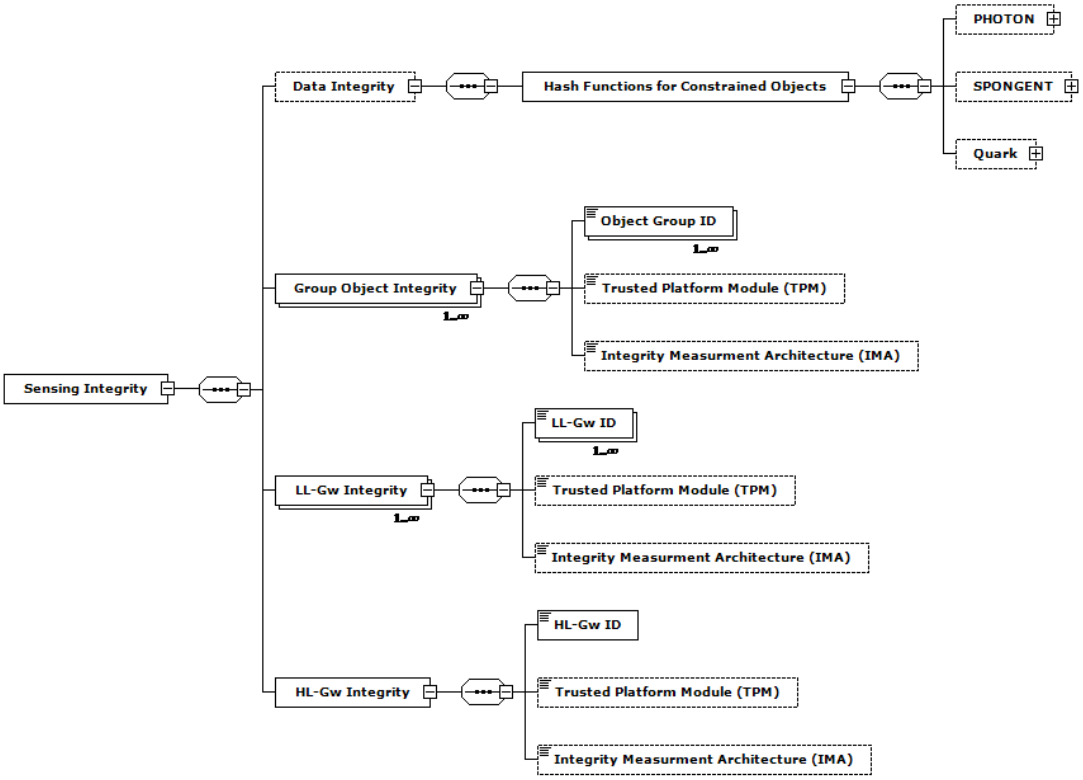

Figure 6.14 : Paramètres de sécurité quantitatifs du SECgSLA : Intégrité

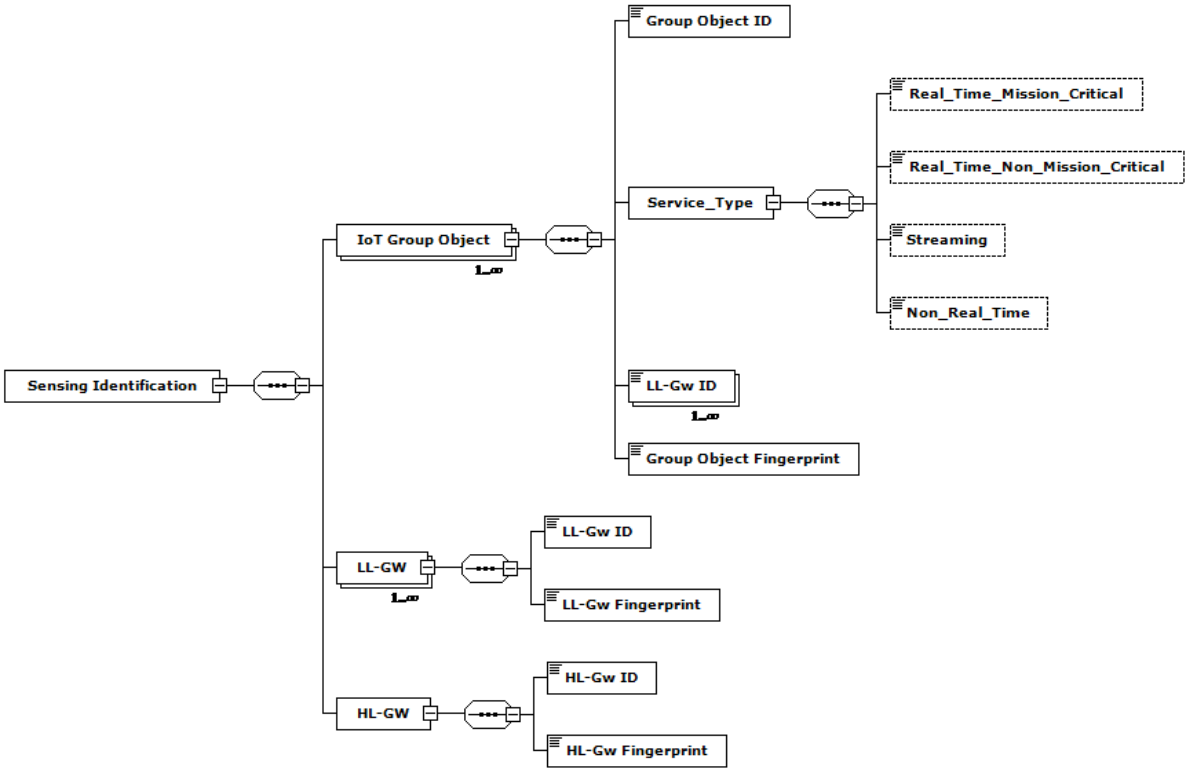

Figure 6.15 : Paramètres de sécurité quantitatifs du SECgSLA : Identification

D'autre part, le contrôle d'accès des objets IoT à la passerelle *LL-Gw* est défini dans l'attribut « Sensing Access Control » (voir Figure 6.16) du *SECgSLA* via un modèle basé sur les attributs. Ce modèle prend en considération les attributs suivants : identifiant du groupe et niveau de confiance de base pour les objets du groupe. Le niveau de confiance est utilisé afin de donner accès aux objets IoT ayant un niveau de confiance supérieur ou égal au niveau de confiance spécifié dans le *SECgSLA*. La méthode que nous avons spécifiée pour la détermination du niveau de confiance des objets IoT est détaillée dans le chapitre 7.

Le modèle basé sur les attributs prend en considération aussi le temps de service comme paramètre environnemental afin de prendre une décision relative à différentes actions possibles (lecture, écriture et modification) dans le cadre du contrôle d'accès de l'objet IoT à la ressource de type *LL-Gw*. De même, un contrôle d'accès pour les *LL-Gw* auprès des *HL-Gw* est spécifié dans ce même attribut « Sensing Access Control » du *SECgSLA* en utilisant un modèle basé sur les attributs identique à celui du modèle du contrôle d'accès des objets.

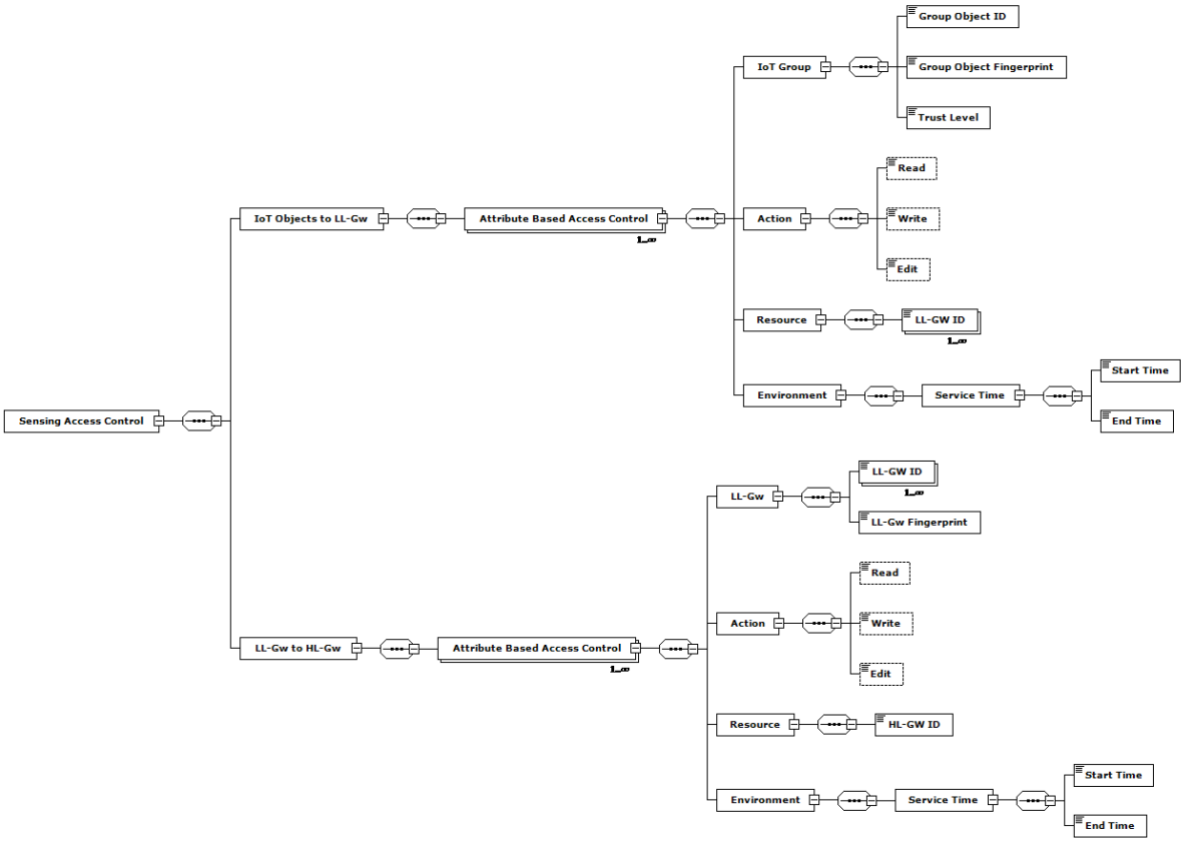

Figure 6.16 : Paramètres de sécurité quantitatifs du SECgSLA : Contrôle d'accès

Enfin, la protection de la vie privée des utilisateurs au niveau de la couche sensing est prise en compte grâce à l'attribut « Sensing Privacy on HL-Gw » (Voir Figure 6.17) du *SECgSLA* en

spécifiant la localisation du stockage des données et des sauvegardes de secours, la possibilité de vente des données à des tierces parties et l'anonymisation des données.

```
<xs:element minOccurs="1" maxOccurs="1" name="Sensing Privacy on HL-Gw" >
    <xs:complexType >
        <xs:sequence >
            <xs:element minOccurs="1" maxOccurs="1" name="Data Storage Location" type="xs:string"/>
            <xs:element minOccurs="1" maxOccurs="1" name="Data Backup Location" type="xs:string"/>
            <xs:element minOccurs="1" maxOccurs="1" name="Data Sale to third parties" type="xs:boolean"/>
            <xs:element minOccurs="1" maxOccurs="1" name="Data Anonymization" type="xs:boolean"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Figure 6.17 : Schéma XML du SECgSLA : Protection de la vie privée

6.3.2.2. Procédure d'établissement du SECgSLA

Différents échanges doivent avoir lieu entre l'*IoT-SP et la HL-Gw* pour établir le *SECgSLA*. Ces échanges permettent aux différentes parties de se mettre d'accord sur les paramètres de sécurité demandés par l'*IoT-SP*. Nous spécifions dans la Figure 6.18 l'ensemble des états de l'automate fini *FSM* concernant l'établissement du *SECgSLA* au niveau de l'*IoT-SP*.

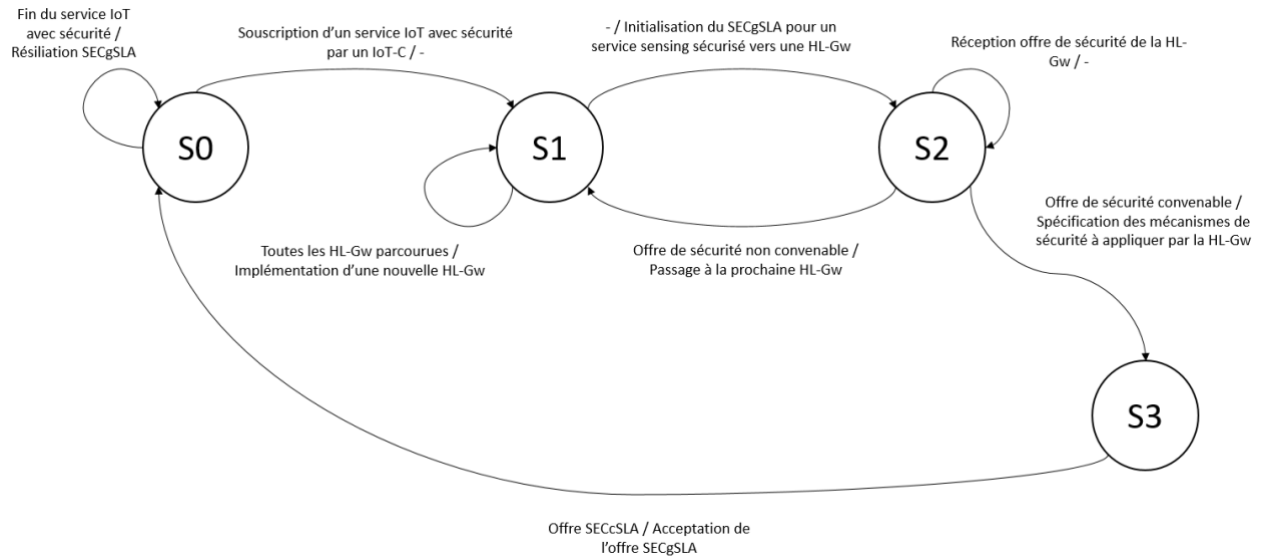

Figure 6.18 : FSM de l'IoT-SP pour l'établissement du SECgSLA

Dans l'état S0, l'*IoT-SP* attend la demande de souscription d'un nouveau service IoT avec sécurité émanant d'un *IoT-C*. Suite à la réception de cette demande, l'*IoT-SP* passe à l'état S1. Dans cet état, il envoie une requête d'initialisation d'un *SECgSLA* vers une *HL-Gw* avant de passer vers l'état S2. L'*IoT-SP* reste dans l'état S2 en recevant l'offre de sécurité de la *HL-Gw* correspondante. Si l'offre de sécurité reçue n'est pas convenable, l'*IoT-SP* revient à l'état S1 et change de *HL-Gw*. Dans ce cas (i.e. dans l'état S1), si l'*IoT-SP* a déjà refusé toutes les offres de sécurité émanant de toutes les *HL-Gw* disponibles, l'*IoT-SP* reste dans l'état S1 et implémente une nouvelle *HL-Gw*. Sinon, dans l'état S2, si l'offre de sécurité reçue de la *HL-Gw* est convenable, l'*IoT-SP* envoie la

spécification des mécanismes de sécurité à appliquer par la *HL-Gw* et passe à l'état S3. Dans cet état, l'*IoT-SP* reçoit l'offre *SECgSLA* de la *HL-Gw* qu'il accepte avant de passer vers l'état S0 en attendant la fin du service IoT avec garantie de sécurité correspondant pour résilier ce contrat.

6.3.3. SLA global de sécurité IoT : SECiSLA

L'établissement du *SECcSLA* et du *SECgSLA* permet à l'*IoT-SP* de fournir une offre de sécurité globale à l'*IoT-C* à travers un *SLA* de sécurité de type *SECiSLA*. Ce contrat permet de spécifier les mécanismes de sécurité à adopter dans toutes les couches de notre architecture IoT dans le cadre de l'offre d'un service IoT. Ainsi, le contrat *SECiSLA* comporte différentes parties basées sur les deux contrats que nous avons décrits dans les sections précédentes, à savoir *SECgSLA* pour la couche sensing et *SECcSLA* pour la couche cloud.

6.3.3.1. Paramètres du SECiSLA

Le *SLA* de type *SECiSLA* est un contrat de niveau de sécurité global pour l'architecture IoT. Ce contrat est établi entre l'*IoT-SP* et l'*IoT-C* et permet de regrouper les paramètres de sécurité existants dans le *SECcSLA* et le *SECgSLA* en plus d'autres paramètres spécifiques. Ainsi, nous spécifions pour le *SECiSLA* 6 attributs : « Identification », « Validity », « Parties », « Service Identification », « Security » et « Business Parameters » (voir Figure 19). Ainsi, le contrat *SECiSLA* est identifié par le *SECiSLA ID* et associé à l'*iSLA ID* qui permet de le mapper vers le contrat *iSLA* correspondant (voir Figure 6.19). Comme pour l'*iSLA*, l'attribut « Validity » est défini à travers une date de mise en service et d'arrêt du service. L'attribut « Parties » définit l'*IoT-SP* en tant que fournisseur et l'*IoT-C* en tant que client. Dans ce contexte, l'attribut « Service Identification » permet de définir la partie source à travers l'adresse *IP* des clients du service IoT et la partie destination à travers l'identificateur de l'application, l'adresse *IP* de l'interface de l'application et le numéro de port de l'application avec les identifiants des *LL-Gw* et de la *HL-Gw* impliqués dans l'offre du service en question.

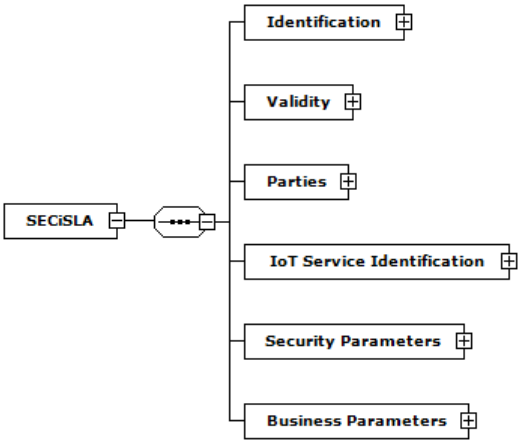

Figure 6.19 : Représentation globale du schéma XML du SECiSLA

Les paramètres de sécurité du *SECiSLA* (i.e., attribut « Security ») sont spécifiés d'une façon qualitative (i.e., Gold, Silver, Bronze) ou quantitative (voir Figure 6.20) sachant qu'un niveau de sécurité décrit d'une façon qualitative est mappé à un ensemble de paramètres de sécurité quantitatifs. De plus, les paramètres de sécurité quantitatifs du *SECiSLA* sont classés selon deux catégories obligatoires : paramètres de sécurité pour la couche sensing et paramètres de sécurité pour la couche cloud. D'une part, pour les paramètres quantitatifs de sécurité du *SECiSLA* relatifs à la couche sensing, nous reprenons ceux que nous avons défini dans le *SECgSLA* (cf. section 6.3.2) à travers les attributs suivants : « Sensing Data Confidentiality », « Sensing Integrity », « Sensing Identification », « Sensing Access Control » et « Sensing Privacy on HL-Gw » (voir Figure 6.20). D'autre part, pour les paramètres quantitatifs de sécurité du *SECiSLA* relatifs à la couche cloud, nous reprenons ceux que nous avons défini dans le *SECcSLA* (cf. section 6.3.1) à travers les attributs suivants : « Cloud Data Storage Confidentiality », « Cloud Data Storage Integrity », « Cloud Identification / Authentication », « Cloud Access Control », « Cloud HL-Gw Communication Security », « Cloud Privacy » (voir Figure 6.20).

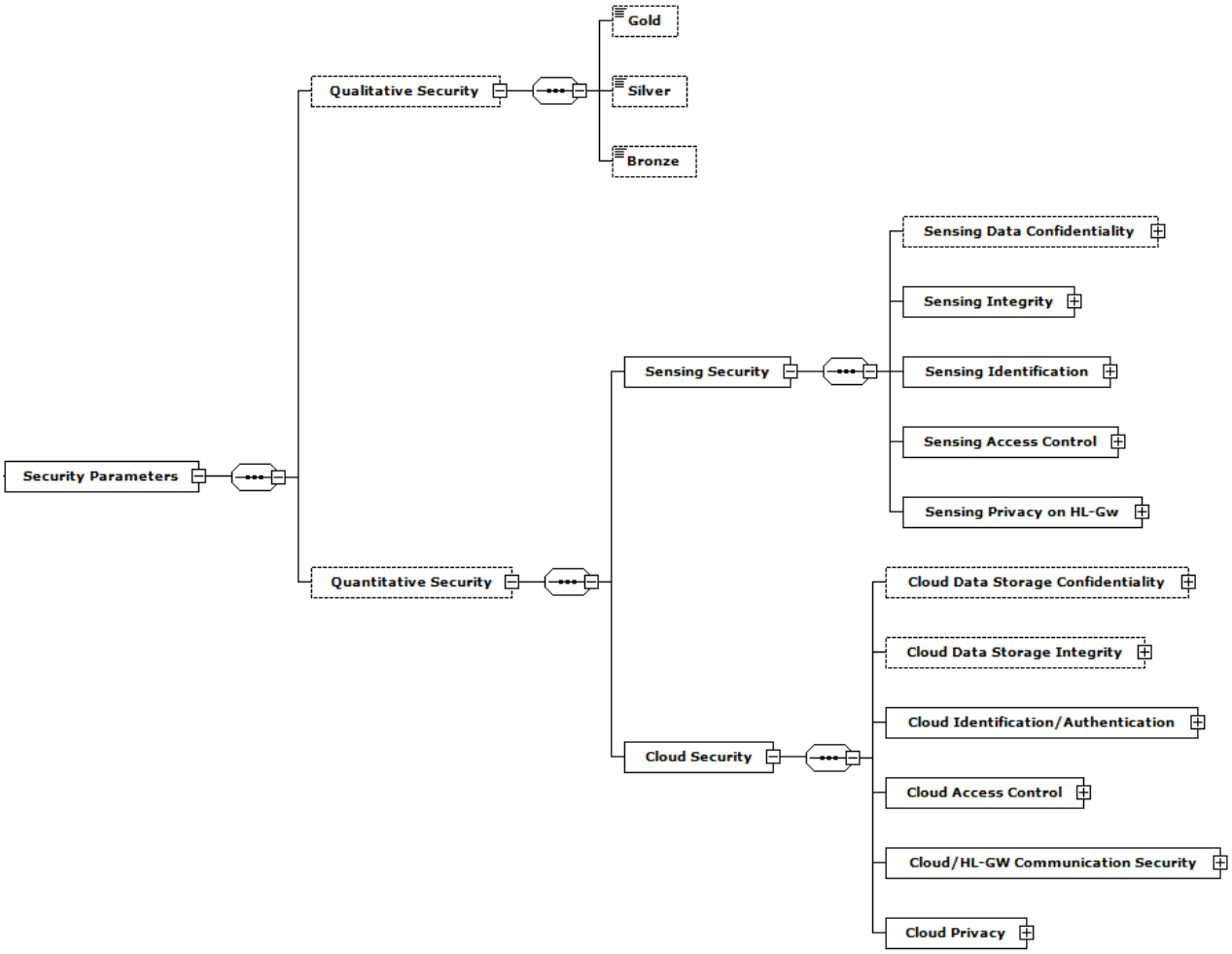

Figure 6.20 : Représentation du schéma XML des paramètres de sécurité du SECiSLA

Il est à noter que même si le contrat *SECcSLA* établi avec un *CSP* ne comprend pas tous ces attributs (cas d'un service IaaS par exemple), le contrat *SECiSLA* établi avec le client doit comprendre tous ces attributs qu'ils soient fournis par le *CSP* ou par l'*IoT-SP* au niveau de a couche cloud.

La dernière partie du *SECiSLA* comporte l'attribut « Business Parameters » qui spécifie les paramètres commerciaux permettant de définir le coût de l'implémentation des mécanismes de sécurité au niveau des différentes couches de l'architecture IoT dans le cadre d'une offre de service IoT sécurisé. Les paramètres commerciaux se limitent au coût et ne spécifient pas la gestion des violations et des pénalités. En effet, le *SECiSLA* permet de choisir des algorithmes, des mécanismes et des services de sécurité à appliquer dans l'environnement IoT sans les associer à des seuils de performance qui déclenchent des violations comme c'est le cas avec les paramètres de *QoS* dans le *iSLA*. Le coût du *SECiSLA* dépend des algorithmes et méthodes choisis pour assurer différents services de sécurité (i.e., confidentialité, intégrité, contrôle d'accès, protection de la vie privée, etc.) au niveau des différentes couches de l'architecture IoT et en particulier la couche sensing et la couche cloud (voir Figure 6.21).

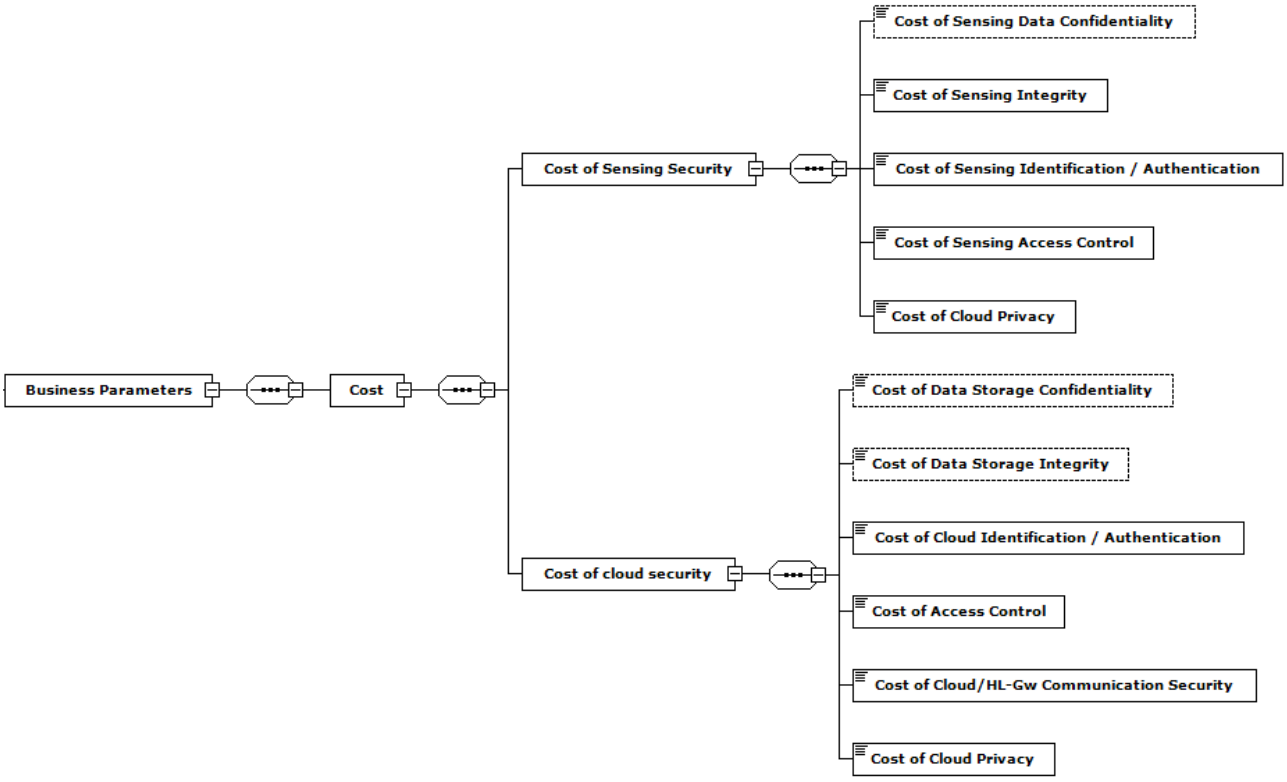

Figure 6.21 : Représentation du schéma XML des paramètres commerciaux du SECiSLA

6.3.3.2. Procédure d'établissement du SECiSLA

Différents échanges doivent avoir lieu entre l'*IoT-SP*, l'*IoT-C*, le *CSP* et la *HL-Gw* pour établir le *SECiSLA*. Ces échanges permettent aux différentes parties de se mettre d'accord sur les

paramètres de sécurité relatifs à un service IoT. Nous spécifions dans la Figure 6.22 l'ensemble des états de l'automate fini *FSM* concernant l'établissement du SEC*iSLA* au niveau de l'*IoT-SP*.

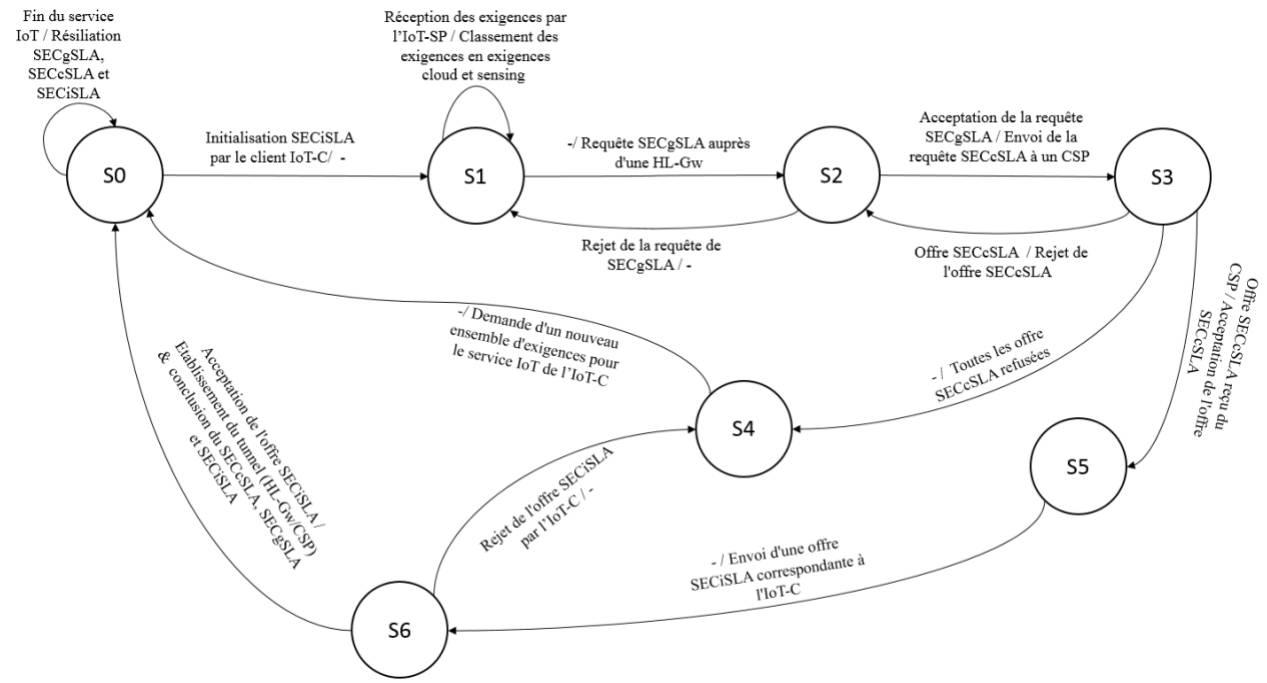

Figure 6.22 : FSM de l'IoT-SP pour l'établissement du SECiSLA

L'*IoT-SP* passe de l'état initial S0 vers S1 lorsqu'il reçoit une demande d'initialisation d'un *SECiSLA* provenant d'un *IoT-C*. Dans l'état S1, l'*IoT-SP* classe les exigences de sécurité du service IoT correspondant selon les couches sensing et cloud de notre architecture IoT. Par la suite, l'*IoT-SP* initie l'établissement du *SECgSLA* avec une *HL-Gw* et passe à l'état S2. Dans le cas de refus du *SECgSLA* de la part de la *HL-Gw* à cause de ressources insuffisantes, l'*IoT-SP* revient à l'état S1 et initie l'établissement du *SECgSLA* avec une nouvelle *HL-Gw* en passant de nouveau à l'état S2. Par contre, dans le cas de l'acceptation du *SECgSLA, l'IoT-SP* envoie une requête *SECcSLA* à un *CSP* et passe à l'état S3. Dans cet état, l'*IoT-SP* attend la réponse du *CSP* qui va proposer une offre *SECcSLA*. Si cette offre est rejetée par l'*IoT-SP* alors ce dernier passe de nouveau à l'état S2 et la requête *SECcSLA* est envoyée à un nouveau *CSP*. Dans le cas où toutes les offres des différents *CSP* sont rejetées par l'*IoT-SP*, ce dernier passe de l'état S3 à l'état S4 afin de demander au client d'adapter ses exigences de sécurité et passer à l'état S0 de nouveau. Sinon, si l'offre *SECcSLA* est acceptée, l'*IoT-SP* passe de l'état S4 à l'état S5. Depuis cet état, l'*IoT-SP* envoie une offre *SECiSLA* à l'*IoT-C* et passe à l'état S6 dans l'attente de sa réponse. Si l'*IoT-C* rejette l'offre *iSLA*, l'*IoT-SP* passe à l'état S4 pour demander à l'*IoT-C* d'adapter ses exigences de sécurité et faire une transition vers l'état S0. Par contre, dans l'état S6, si l'offre *SECiSLA* est acceptée par l'*IoT-C*, l'*IoT-SP* revient à l'état S0 après avoir établi le tunnel (*HL-Gw*/*CSP*) et conclu les contrats *SECcSLA, SECgSLA* et *SECiSLA* avec le *CSP*, la *HL-Gw* et l'*IoT-C* respectivement.

6.4. Conclusion

A travers ce chapitre, nous avons spécifié notre Framework de sécurité basé sur une architecture et différents *SLA* de sécurité. Ces contrats du niveau de service de sécurité sont des extensions des *SLA* de *QoS* définis dans le chapitre 4. Ainsi, nous avons étendu les contrats *cSLA* avec les fournisseurs cloud grâce à des garanties de sécurité qui concernent la couche cloud de notre architecture IoT. De plus, nous avons étendu les contrats internes de type *gSLA* avec les *HL-Gw* grâce à des garanties de sécurité qui concernent la couche sensing de notre architecture IoT. Quant à la couche réseau de notre architecture IoT, les garanties de sécurité seront assurées grâce à un tunnel établi entre la couche sensing et la couche cloud. Nous avons utilisé ces contrats afin de spécifier un *SLA* global de sécurité appelé *SECiSLA* entre l'*IoT-C* et l'*IoT-SP* permettant de prendre en considération les garanties de sécurité relatives aux différentes couches de l'architecture IoT. Dans ce contexte, nous avons spécifié et présenté les processus d'établissements de ces différents contrats (i.e., *SECcSLA*, *SECgSLA* et *SECiSLA)*. Après avoir étendu dans ce chapitre les *SLA* de *QoS* , définis dans le chapitre 4, pour inclure les garanties de sécurité dans un environnement IoT, nous proposons dans le chapitre suivant un nouveau mécanisme de sécurité permettant d'offrir des services garantis par ces contrats étendus, à savoir le contrôle d'accès des objets IoT au niveau de la couche sensing de l'architecture IoT.

Chapitre 7 – Spécification d'IoT-MAAC : une méthode de contrôle d'accès sécurisé dans l'IoT

7.1. Introduction

La garantie d'un niveau de sécurité spécifié à travers un *SLA* tel que *SECiSLA* dans l'environnement IoT est rendue possible grâce à l'implémentation de différents mécanismes de sécurité au niveau des différentes couches de l'architecture IoT. Dans ce contexte, nous proposons dans ce chapitre un mécanisme de contrôle d'accès au niveau de la couche sensing de l'architecture IoT. Ce mécanisme de sécurité assure le contrôle d'accès des objets IoT aux passerelles de bas niveau *LL-Gw* avec comme objectif de garantir la fiabilité des informations remontées en limitant le nombre d'objets IoT autorisés à les communiquer aux passerelles.

Ainsi, nous présentons dans la section 2 de ce chapitre un Framework pour notre proposition de méthode de contrôle d'accès dans un environnement IoT appelée *IoT-MAAC* (IoT Multiple Attribute Access Control). Nous spécifions dans ce Framework les besoins relatifs à un mécanisme de contrôle d'accès des objets IoT à la passerelle *LL-Gw* et nous présentons les attributs et l'architecture de notre méthode basée sur les deux standards *XACML* (eXtensible Access Control Markup Language) et *SAML* (Security Assertion Markup Language). Un des attributs du contrôle d'accès *IoT-MAAC* est la confiance définie en détails dans la section 3. A ce titre, nous présentons dans cette section le modèle de logique floue utilisé pour quantifier la confiance ainsi que les fonctions d'appartenance et les règles d'inférence utilisées par ce modèle. Ensuite, nous spécifions notre algorithme de prise de décision *IoT-MAAC* dans la section 4. Dans cette même section, nous décrivons les échanges effectués entre les composants du Framework pour appliquer l'algorithme *IoT-MAAC* ainsi que la stratégie de contrôle d'accès correspondante. Enfin, la section 5 nous permet de conclure ce chapitre.

7.2. Framework de contrôle d'accès dans l'IoT

Nous définissons dans le cadre de notre Framework de contrôle d'accès dans l'IoT les besoins de mettre en place un service de sécurité de type contrôle d'accès dans un environnement formé par les objets IoT et les passerelles de bas niveau (*LL-Gw*) de notre architecture IoT. De plus, nous spécifions dans le cadre de ce Framework une architecture adaptée à notre méthode de contrôle d'accès, à savoir *IoT-MAAC,* qui se base sur les standards *SAML* et *XACML* tout en définissant les attributs de cette méthode.

7.2.1. Besoins de contrôle d'accès dans l'IoT

L'environnement IoT se caractérise par différentes contraintes rendant difficile l'adoption des mécanismes de sécurité conçus pour les systèmes traditionnels. En effet, les objets IoT ayant des contraintes en termes de capacités de mémoire et de traitement doivent être en mesure de prendre en considération les différents services de sécurité d'une manière adaptée. A ce titre, plusieurs services de sécurité ont été traités dans la littérature afin de les adapter à l'environnement IoT. En particulier, le service de sécurité de type contrôle d'accès est considéré comme étant un besoin primordial dans ce type d'environnement. Ce service doit faire appel dans l'IoT à des mécanismes d'identification, d'authentification et d'autorisation mutuelle entre les objets et les utilisateurs des services selon des politiques prédéfinies. L'importance du contrôle d'accès dans l'IoT a été mise en avant par différents organismes et travaux de recherche [11] [123] [171].

Dans l'environnement de notre Framework (voir Figure 7.1), les informations relatives à un service IoT sont remontées depuis les objets IoT vers les passerelles de bas niveau qui doivent les authentifier. Dans ce contexte, il est primordial de proposer une méthode de contrôle d'accès des objets auprès des passerelles *LL-Gw* pour assurer un niveau de sécurité adéquat lors de la récolte de ces informations. En effet, dans un environnement IoT ouvert, les passerelles peuvent récolter des informations émanant de différents types d'objets avec des niveaux de sécurité hétérogènes et il est donc important de prévoir des règles et des politiques permettant de limiter l'accès dans ce type d'environnement ouvert. Les travaux de recherche portant sur le contrôle d'accès dans l'IoT que nous avons présenté dans le chapitre 3 (cf. section 3.3.2.2), ne s'intéressent qu'à l'accès des utilisateurs de service IoT aux objets IoT et ne prennent pas en compte le contrôle de l'accès des objets IoT aux passerelles de la couche sensing de l'architecture IoT. D'où le besoin de proposer un mécanisme de sécurité assurant un contrôle d'accès des objets IoT à la passerelle *LL-Gw* dans le cadre de notre Framework. Ainsi, nous nous concentrons dans notre Framework sur le contrôle d'accès concernant les objets et passerelles IoT. Dans ce contexte, nous considérons un environnement (voir Figure 7.1) dans lequel différents objets IoT sont connectés à une passerelle *LL-Gw* sous le contrôle d'une passerelle de haut niveau *HL-Gw* afin de fournir des informations relatives à un service IoT.

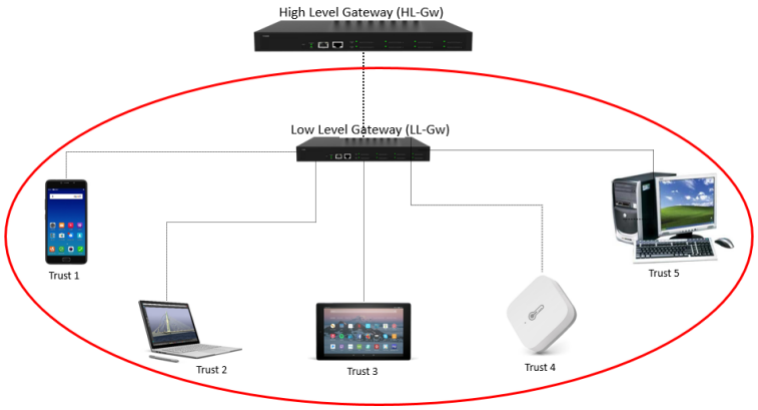

Figure 7.1 : Environnement de contrôle d'accès IoT

Dans cet environnement de contrôle d'accès IoT, la *LL-Gw* peut recevoir des informations critiques d'un objet IoT qui a subi une attaque physique, voir même un objet IoT malveillant. Par conséquent, il est nécessaire de contrôler l'accès des objets IoT capables d'envoyer des données concernant un service IoT à la *LL-Gw*. En effet, un objet IoT corrompu ou encore malveillant peut envoyer des données erronées concernant un événement critique et mettre ainsi en péril le bon fonctionnement du service ainsi que le système correspondant.

7.2.2. Utilisation des standards XACML et SAML

Différentes technologies peuvent être utilisées pour assurer un contrôle d'accès à travers une authentification ou une identification au niveau de l'environnement IoT. Parmi ces technologies, deux standards, à savoir *XACML* et *SAML*, permettent conjointement de réaliser le contrôle d'accès au sein de ce type d'environnement.

XACML, proposé par l'organisme de standardisation *OASIS* (Organization for the Advancement of Structured Information Standards), est un langage de politique de contrôle d'accès fournissant une communication bidirectionnelle de la demande d'accès et de la réponse correspondante. Ce langage de politique est très répandu et peut fonctionner avec différents modèles de contrôle d'accès (tels que *RBAC*, *ABAC*, etc.). *XACML* comprend un groupe de composants *XML* standard et spécifie des points d'extension standard pour des règles individuelles, des types de données et des procédures [172]. Il existe de nombreuses implémentations pour *XACML* (i.e., AT&T XACML 3.0 Implementation [173], Sun's XACML Implementation [174], etc.).

XACML est un langage standardisé, générique, puissant et pouvant être déployé sur plusieurs plateformes (i.e., AuthZForce [175] assurant une implémentation sur une plateforme Java, ndg-xacml [176] assurant une implémentation sur une plateforme python, etc.). Ce langage supporte les modèles de contrôle d'accès basés sur les attributs *ABAC* et il est compatible avec plusieurs standards parmi lesquels nous pouvons citer *SAML* et *LDAP* (Lightweight Directory Access Protocol) [172].

Les différents composants et modules du standard *XACML* sont les suivants (voir Figure 7.2) :

- *PAP* (Policy Administration Point) : le point d'administration des politiques est une interface pour la définition des politiques mises à disposition du module point de décision de politiques (*PDP*).
- *PEP* (Policy Enforcement Point) : le point d'application des politiques reçoit la demande d'accès de l'utilisateur de service et applique la décision d'accès reçue par l'intermédiaire du gestionnaire de contexte (i.e., Context Handler).
- *PDP* (Policy Decision Point) : le point de décision de politiques évalue la demande d'accès par rapport aux politiques applicables et renvoie la décision d'accès au PEP en passant par le gestionnaire de contexte (i.e., Context Handler).
- Context Handler : le gestionnaire de contexte traduit les demandes d'accès pour les transmettre au PDP. Il agit comme un proxy entre le PEP et le *PDP* et il est chargé de

récupérer les valeurs des attributs nécessaires à l'évaluation de la politique correspondante à la demande d'accès.

- *PIP* (Policy Information Point) : le point d'information sur la politique fournit les valeurs des attributs permettant la prise de décision. Il interagit directement avec le gestionnaire de contexte et les sujets (i.e., Subjects) du contrôle d'accès.

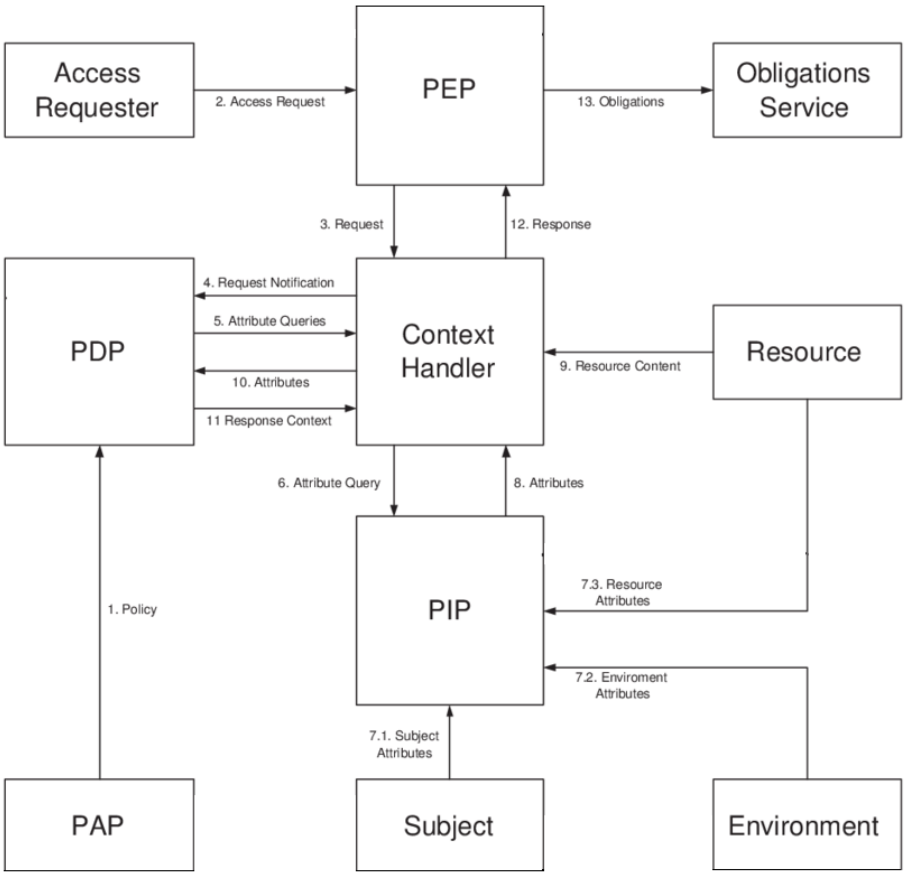

Figure 7.2 : Composants et modules de XACML [177]

Une stratégie *XACML* comporte des politiques et chaque politique est formée par un ensemble de règles regroupées à l'aide d'un algorithme de combinaison de règles. Ces règles sont composées de trois éléments: une cible, un effet et une condition. D'une part, la cible permet de déterminer la règle à considérer. D'autre part, l'effet est le résultat à fournir (accepter ou refuser) lorsqu'une règle est satisfaite alors que la condition est la fonction à exécuter lorsqu'une règle est applicable [172]. Dans ce contexte, le moteur *XACML* évalue une requête *XACML* donnée par rapport à plusieurs règles indépendamment. Le moteur *XACML* est un composant logiciel qui nécessite deux entrées afin de trouver un résultat en guise de sortie. Les deux entrées sont les politiques *XACML* et la demande d'accès, tandis que le résultat concerne la décision d'accès. Le résultat relatif à une demande d'accès aux données peut se présenter sous l'une des formes suivantes : Autorisation, Refus, Non applicabilité [177].

Afin d'assurer le contrôle d'accès, une étape d'authentification est nécessaire. Ce service de sécurité peut être fourni à travers le langage *SAML*. *SAML* [178], standardisé par OASIS, est largement utilisé pour mettre en place une connexion unique inter-domaines (*SSO* : Single Sign On). Il peut être mappé vers différents autres protocoles tels que *SOAP* et *HTTP*, etc. En effet, *SAML* est un standard permettant l'échange de données d'authentification et d'autorisation entre des partenaires, en particulier entre un fournisseur d'identité (*IdP* : Identity Provider) et un fournisseur de services (*SP* : Service Provider). L'*IdP* crée et gère les informations d'identité des sujets et fournit l'authentification aux différents *SP*. Le *SP* est une entité qui fournit des services aux sujets authentifiés. L'IdP peut demander des informations aux sujets (i.e., noms d'utilisateurs, mots de passe, etc.) afin de l'authentifier. Pour authentifier un sujet, *SAML* utilise la notion d'assertion qui est transmise de l'*IdP* aux *SP*. Les assertions *SAML* permettent de diffuser des informations relatives à la sécurité pouvant être utilisées à diverses fins. L'un des objectifs les plus importants consiste à contribuer aux décisions de contrôle d'accès. *SAML* ne spécifie pas comment ces informations doivent être utilisées ni comment les stratégies de contrôle d'accès doivent être traitées. En effet, *SAML* ne spécifie pas la méthode d'authentification chez l'*IdP*. L'*IdP* peut utiliser un nom d'utilisateur et un mot de passe, ou une autre forme d'authentification, y compris une authentification à plusieurs facteurs.

SAML et *XACML* sont utilisés conjointement pour assurer une authentification suivie d'un contrôle d'accès (voir Figure 7.3).

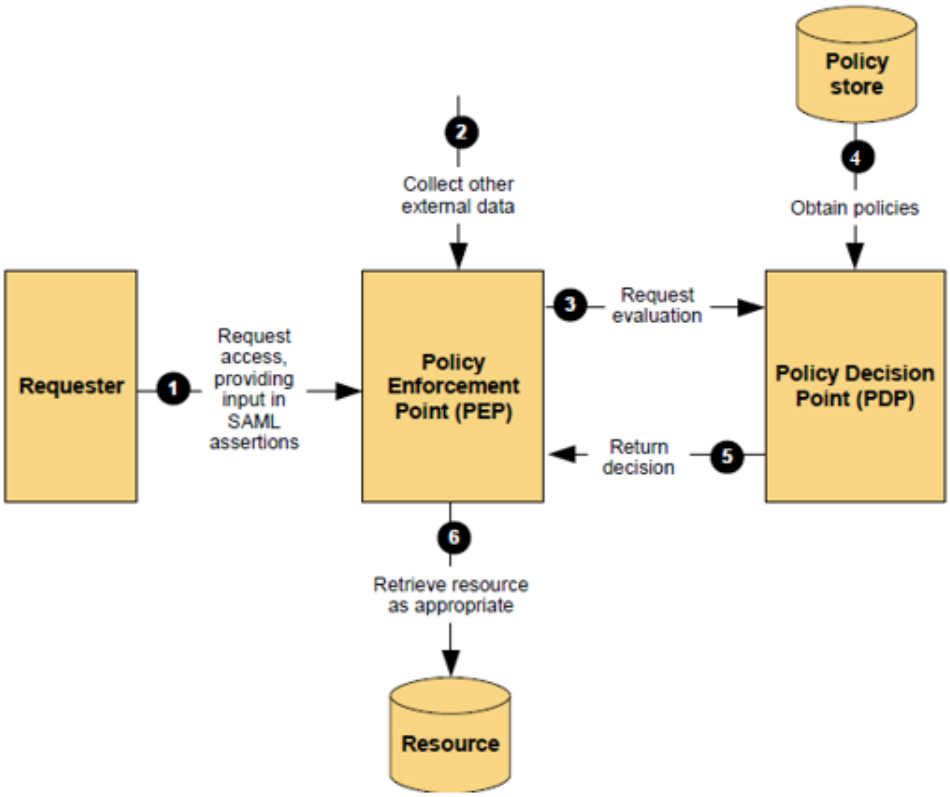

Figure 7.3 : Intégration de XACML et SAML [179]

Dans ce contexte, le standard *SAML* définit la méthode de mappage des différents attributs utilisés par *SAML* lors de l'authentification, vers des attributs *XACML* utilisés lors du contrôle d'accès via le profil d'attribut *XACML*. Ce profil d'attribut définit également un nouveau type de requête de décision d'autorisation spécialement conçu pour une utilisation dans un environnement *XACML*. Le profil d'attribut *XACML* étend l'utilisation du standard *SAML* dans le contexte *XACML* en permettant à *SAML* d'être utilisé pour extraire une politique *XACML* pouvant s'appliquer pour une prise de décision d'accès mais aussi en permettant à l'assertion *SAML* d'être utilisée pour envelopper une politique *XACML* ou encore d'échanger les informations nécessaires à l'identification et l'authentification. L'utilisation combinée de *SAML* et de *XACML* implique généralement les étapes suivantes:

- Le *PEP* reçoit une demande d'accès à une ressource et obtient des assertions *SAML* contenant des informations sur les demandeurs d'accès et les ressources concernées par la demande d'accès. Ces assertions peuvent accompagner la demande d'accès ou peuvent être obtenues directement auprès d'une autorité *SAML*.
- Le *PEP* obtient d'autres informations pertinentes relatives à la demande d'accès, telles que l'heure, la date, l'emplacement, les propriétés de la ressource, etc.
- Le *PEP* présente toutes les informations obtenues grâce entre autres à *SAML* au *PDP* pour décider si l'accès doit être autorisé.
- Le *PDP* obtient toutes les politiques pertinentes concernant la demande d'accès et les évalue.
- Le *PDP* informe le *PEP* de la décision relative au contrôle d'accès.
- Le *PEP* applique la décision en autorisant l'accès demandé ou en indiquant que l'accès n'est pas autorisé

Une assertion *SAML* comprend divers types d'attributs (i.e., identité, signature numérique, période de validité, etc.). *SAML* permet aussi de créer des requêtes pour interroger une autorité d'attributs en ligne et conserver la réponse à cette requête. En effet, *SAML* définit différents éléments tels que <*samlp: AttributeQuery*> et <*samlp: Response*>. D'autre part, afin de permettre à un *PEP* d'utiliser le standard *SAML* avec une prise en charge complète de la syntaxe *XACML*, la syntaxe SAML a été étendue pour prendre en charge les requêtes *XACML* telles que <*xacml-saml: XACMLAuthzDecisionStatementType*> qui inclut un élément *XACML* de type <*xacml-context: Response*> ainsi que d'autres informations facultatives. Enfin, <*xacml-samlp: XACMLAuthzDecisionQuery*> est un autre élément d'extension *SAML* pouvant être utilisé par un *PEP* pour soumettre un contexte de requête *XACML*, en l'associant à d'autres informations facultatives, en tant que requête *SAML* à un gestionnaire de contexte *XACML* [178].

7.2.3. Attributs du contrôle d'accès des objets IoT

Afin d'assurer le contrôle d'accès des objets IoT au niveau de la *LL-Gw*, différents paramètres doivent être pris en considération pour vérifier l'identité et authentifier ces objets. En effet, le contrôle

d'accès s'applique sur des objets IoT identifiés et authentifiés. Dans ce contexte, nous proposons dans le cadre de notre Framework de contrôle d'accès IoT au niveau de la couche sensing une nouvelle méthode, appelée *IoT-MAAC*, permettant le contrôle d'accès des objets IoT auprès de la passerelle de bas niveau (*LL-Gw*) en se basant sur 4 attributs. Ainsi, nous spécifions comme attributs de contrôle d'accès pour la méthode *IoT-MAAC* : les identifiants de l'objet, l'empreinte digitale de l'objet, le niveau de confiance de l'objet et les paramètres environnementaux (voir Tableau 7.1).

Ainsi, l'identification des objets est effectuée à travers l'identifiant du groupe d'objets (*Group Object ID*) et l'identifiant de l'objet lui-même (*IoT Object ID*). De plus, l'objet doit transmettre son empreinte digitale comportant le hachage des caractéristiques communes du groupe d'objets pour s'assurer que son intégrité n'est pas atteinte et qu'il n'a pas été sujet à une attaque. Cette empreinte digitale de l'objet est comparée à l'empreinte renseignée au niveau du contrat de sécurité de type *SECgSLA* défini dans le chapitre 6 (cf. 6.3.2). Nous spécifions comme autre attribut pour notre méthode de contrôle d'accès *IoT-MAAC* le niveau de confiance de l'objet pour lequel nous proposons, dans la section 3 de ce chapitre, une méthode d'évaluation basée sur la logique floue. Enfin, nous utilisons avec *IoT-MAAC* un attribut portant sur des informations environnementales telles que l'heure et la date de la demande d'accès.

Tableau 7.1 : Attributs IoT-MAAC

Attributs IoT-MAAC	Besoins d'utilisation
Identifiants (*IoT Object ID, Group Object ID)*	Identification des objets
Empreinte Digitale	Vérification de l'intégrité des objets
Niveau de confiance de l'objet	Evaluation du niveau de confiance des objets
Paramètres environnementaux (Date et temps de la requête d'accès)	Evaluation de l'environnement de la requête

7.2.4. Architecture de contrôle d'accès basée sur XACML/SAML

L'architecture que nous proposons (voir Figure 7.4) pour notre Framework de contrôle d'accès dans l'IoT permet de prendre en considération les composants des standards *XACML* et *SAML* au niveau de la couche sensing de l'environnement composée par les passerelles de type *LL-Gw* et *HL-Gw* mais aussi les objets IoT. Nous détaillons dans ce qui suit les composants de notre architecture permettant d'utiliser la méthode *IoT-MAAC* afin de contrôler l'accès des objets IoT aux passerelles de bas niveau dans le cadre de l'offre d'un service IoT associé à un niveau de service de sécurité décrit dans un *SecSLA*.

Un premier composant de notre architecture de contrôle d'accès est la passerelle de haut niveau (i.e., *HL-Gw*) qui nous permet d'implémenter les fonctions du point de décision des politiques (*PDP*) et le point d'administration des politiques (*PAP*) du standard *XACML* ainsi qu'une base des stratégies de contrôle d'accès, un répertoire des *SLA* et une autorité d'attributs. Dans ce contexte, le

PAP nous permet de sauvegarder dans le répertoire des *SLA* les *SECgSLA* relatifs au niveau de sécurité de la couche sensing. Ces contrats de type *SECgSLA* sont utilisés pour alimenter non seulement l'Autorité d'attributs qui comporte les informations concernant le niveau de confiance des objets mais aussi la base de stratégies relatives aux passerelles de bas niveau. En effet, les stratégies concernant l'accès des objets IoT aux différentes *LL-Gw* sont déduites des *SECgSLA* , et en particulier de la partie « Access Control » de ce contrat de sécurité (cf. section 6.3.2.1). De plus, le *PDP* utilise *XACML* pour interroger l'Autorité d'attributs afin d'obtenir le niveau de confiance correspondant à l'objet IoT demandant l'accès à la *LL-Gw*. Les détails de la méthode d'évaluation du niveau de confiance des objets IoT en se basant sur la logique floue avec différents critères de sécurité sont spécifiés dans la section suivante (cf. section 7.3).

Les valeurs de ces critères utilisés pour le calcul du niveau de confiance sont récupérées des *SECgSLA* sauvegardés au niveau du répertoire des *SLA* de la *HL-Gw*. Dans le cadre de notre architecture de contrôle d'accès, le *PDP* utilise *XACML* pour interroger la base des stratégies afin d'obtenir la stratégie et par conséquent la liste de politiques à appliquer pour le contrôle d'accès à la *LL-Gw* concernée par la demande de l'objet IoT. En effet, la stratégie comporte différentes politiques concernant une *LL-Gw* bien spécifique. Chaque politique concerne un groupe d'objets identifié par un *Group Object ID* et comporte différentes règles à appliquer.

Un deuxième composant de notre architecture de contrôle d'accès est la passerelle de bas niveau (i.e., *LL-Gw*) qui nous permet d'implémenter les fonctions du point d'application des politiques (*PEP*). Ce dernier, permet d'une part de traduire les requêtes d'accès *IoT-MAAC* des objets IoT (i.e., assertion *SAML*) en des requêtes *IoT-MAAC XACML* (i.e., requêtes d'autorisation *XACML)*. D'autre part, il permet de traduire les réponses d'autorisation *IoT-MAAC XACML* reçues du *PDP* suite à la prise de décision en des réponses de contrôle d'accès *IoT-MAAC,* encapsulées suivant le protocole de communication utilisé dans la couche sensing et envoyées vers les objets IoT.

Un dernier composant de notre architecture de contrôle d'accès est l'objet IoT. Ce dernier utilise une assertion *SAML* pour envoyer à la *LL-Gw*, jouant le rôle de *PEP*, une demande d'accès *IoT-MAAC*. Cette assertion comprend différentes informations formant une partie des attributs de notre méthode d'accès *IoT-MAAC* (voir Tableau 7.1). Nous citons à ce titre, l'*IoT Object ID,* le *Group Object ID* et l'empreinte digitale de l'objet. Le calcul de l'empreinte se fait grâce à la technique spécifiée dans le *SECgSLA* correspondant (i.e., *TPM, IMA*). Les échanges entre l'objet IoT et la passerelle *LL-Gw* sont sécurisés via une clé de chiffrement symétrique établie entre ces deux composants suite à la requête de demande d'accès de l'objet IoT à la passerelle *LL-Gw*. Ainsi, les échanges concernant la demande d'accès seront sécurisés grâce à cette clé. Enfin, il est à noter que dans notre architecture de contrôle d'accès, le fournisseur de service (*SP*) *SAML* est la *HL-Gw* qui vérifie l'identité des objets et le fournisseur d'identité (*IdP*) *SAML* est l'objet IoT lui-même.

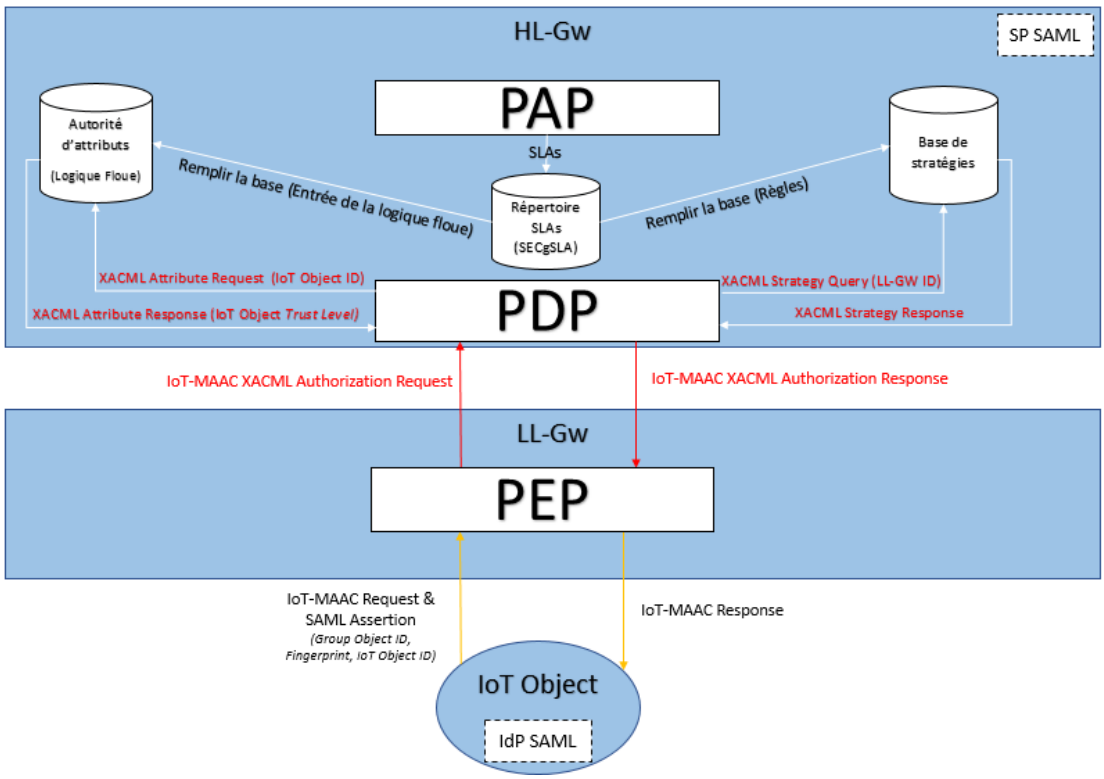

Figure 7.4 : Architecture de contrôle d'accès IoT-MAAC

7.3. Système d'évaluation du niveau de confiance des objets IoT

Après avoir défini dans la section précédente notre Framework de contrôle d'accès *IoT-MAAC*, nous spécifions dans cette section un système d'évaluation basé sur la logique floue, de l'un des attributs, à savoir le niveau de confiance, utilisé par la méthode *IoT-MAAC* permettant de contrôler l'accès des objets IoT aux passerelles de bas niveau *LL-Gw*.

7.3.1. Utilisation de la logique floue

La logique floue est un outil d'aide à la décision permettant de mettre en place un système de prise de décision proche du système humain en étendant la logique booléenne crée par Lotfi Zadeh en 1965 [180]. La logique floue se base sur la théorie des ensembles flous en représentant un ensemble de connaissances via des degrés de vérification de conditions (i.e., une condition aura une degré de vérification). En effet, l'état d'une condition n'est plus décrit par vrai ou faux mais plutôt par un degré d'exactitude. L'ultime intérêt de la logique floue est la description du raisonnement humain à travers un ensemble de règles énoncées selon un langage naturel [180]. La logique floue a été utilisée dans plusieurs domaines d'application tels que la stabilisation d'image, le contrôle des valeurs de PH, le choix du handover dans les réseaux mobiles, etc. [181]. La logique floue fait appel à différents éléments importants qui permettent d'assurer un processus de prise de décision automatisé. Nous décrivons dans ce qui suit certains de ces éléments que nous utilisons dans notre système d'évaluation

de niveau de confiance à savoir : les sous-ensembles flous, les fonctions d'appartenance, les variables linguistiques, la fuzzification, la base d'inférences et la défuzzification.

Les **sous-ensembles flous** permettent de sortir de la logique booléenne et d'introduire la notion d'appartenance pondérée. Cette notion permet d'avoir des niveaux dans l'appartenance d'un élément à un sous-ensemble. Soit *X* un ensemble classique, nous appelons *A* un sous ensemble flou de *X* caractérisé par une fonction d'appartenance $f_A : X \rightarrow [0,1]$. Celle–ci est l'équivalent de la fonction caractéristique d'un ensemble classique. Ainsi, à chaque point *x* de *X* nous associons une valeur réelle $f_A(x)$ entre 0 et 1 qui représente le degré d'appartenance de *x* à *A*. En introduisant la notion de degré d'appartenance dans la vérification d'une condition, la logique floue confère une souplesse de raisonnement permettant de prendre en compte des imprécisions et des incertitudes.

Les **fonctions d'appartenance** peuvent prendre différentes formes (formes sigmoïde, tangente hyperbolique, exponentielle, gaussienne, etc.) choisies arbitrairement en suivant les conseils de l'expert ou des études statistiques. Dans le cadre de notre Framework, une allure triangulaire a été utilisée pour l'évaluation du niveau de confiance des objets IoT, en raison de sa simplicité et de l'efficacité des calculs [182] [183]. Une fonction d'appartenance caractérise un ou plusieurs sous-ensembles d'une variable d'entrée. Elle est représentée en logique floue par la variable $\mu(x)$. Par exemple, la fonction d'appartenance du sous ensemble flou *A* de l'ensemble classique *X* est représentée par $\mu_A(x)$. Nous représentons dans la Figure 7.5 une fonction d'appartenance de l'ensemble flou « moyenne » de *A* de l'ensemble classique *X* qui est dans ce cas la vitesse en Km/h. Dans ce contexte, nous remarquons par exemple que la vitesse 100 Km/h a un degré d'appartenance de 0.2 pour l'ensemble flou « moyenne », alors que la vitesse 90 Km/h a un degré d'appartenance de 0.

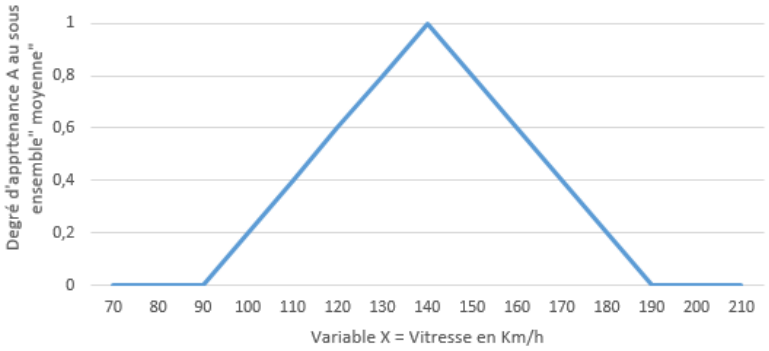

Figure 7.5 : Fonction d'appartenance caractérisant l'ensemble flou « moyenne »

Les **variables linguistiques** dans un sous-ensemble flou sont constituées par le triplet *(X, V,* T_x*)*. X est la variable (i.e., ensemble classique de valeurs), *V* est la plage de valeurs de la variable et T_x est le vocabulaire choisi pour définir linguistiquement la variable *X*. T_x contient les sous-ensembles flous utilisés pour caractériser l'ensemble classique *X*. Par exemple pour la vitesse, nous

pouvons créer 3 variables linguistiques : faible, moyenne, élevée chacune caractérisée par une fonction d'appartenance [180].

Le système flou nécessite le passage par quatre étapes que nous définissons dans ce qui suit. La **fuzzification** permet de passer d'une variable réelle à une variable floue en utilisant les fonctions d'appartenance permettant de déterminer le degré d'appartenance d'une valeur réelle à une ou plusieurs variables linguistiques. La **base d'inférence** comprend différentes règles formulées à l'aide de Si et Alors. La définition des règles est dédiée à l'expert. Il n'existe pas de directives précises pour l'établissement de ces règles. Afin de calculer le nombre de règles nécessaires pour le système de logique floue, nous utilisons l'équation 7.1 qui prend en compte le nombre d'entrées existantes et le nombre de sous-ensembles flous de chaque variable d'entrée. K_i représente le nombre de sous ensemble flous de l'entrée et n_i représente le nombre d'entrées.

$$\text{Nombre de règles} = \prod_{i=1}^{n} K_i \qquad (7.1)$$

Pour appliquer les règles spécifiées dans la base d'inférence, une matrice de décision est définie en considérant les implications floues basées sur des opérateurs logiques (OU, ET, NON) et permettant de concaténer les règles déclenchées. La défuzzification permet de remonter une valeur précise à partir des ensembles flous pour faciliter la prise de décision. Plusieurs méthodes de defuzzification peuvent être utilisées. Nous citons à titre d'exemple la méthode moyenne des maxima (*MOM* : Mean Of Maxima) et la méthode de centre de gravité (*COG* : Center of Gravity). La méthode de type *MOM* définit la sortie comme étant la moyenne des abscisses des maxima de l'ensemble flou issu de l'agrégation des valeurs de sortie. La méthode de defuzzification de type *COG* définit la sortie comme étant l'abscisse du centre de gravité de la surface de la fonction d'appartenance caractérisant l'ensemble flou issu de l'agrégation des conclusions [180].

Différents modèles flous existent dans la littérature. Nous utilisons dans le cadre de notre Framework de contrôle d'accès, le modèle flou Takagi-Sugeno-Kang (*TSK*) pour évaluer le niveau de confiance des objets IoT qui est un des attributs de notre méthode *IoT-MAAC*.

7.3.2. Le modèle TSK

Le modèle *TSK* repose sur un système à *r* entrées et une sortie. Il est constitué d'une base de règles de la forme : Si x_1 est A_1 et x_2 est A_2, et x_r est A_r alors $y = f_i (x_1, x_2, \ldots, x_r)$ avec $x_1, x_2, \ldots\ldots, x_r$ des variables numériques d'entrée du modèle flou et f_i est la fonction qui relie les entrées à la sortie numérique.

Dans le modèle *TSK*, aucun sous-ensemble flou n'est utilisé pour la sortie [184] et par conséquent, le module de défuzzification n'est pas utilisé. Néanmoins, afin d'évaluer cette sortie qui résulte des règles déclenchées, nous calculons une moyenne pondérée. En effet, le modèle *TSK* permet de calculer pour chaque règle déclenchée une valeur de sortie (voir équation 7.2) et un poids pour cette valeur. Le poids est calculé en appliquant l'opérateur arithmétique ET aux différents degrés

d'appartenance des paramètres d'entrée de la règle. Chaque fonction d'appartenance dans chaque règle a son degré d'appartenance ou de vérité égal à la valeur de l'axe des ordonnées correspondant à la valeur du paramètre d'entrée sur l'axe des abscisses. Les paramètres d'entrée peuvent déclencher une ou plusieurs règles et par conséquent différents poids et valeurs de sortie sont calculés.

$$y = f_i(x_1, x_2, \dots\dots, x_r) = a_1.x_1 + a_2.x_2 + \ \dots\dots + \ a_r.x_r \qquad (7.2)$$

Afin d'évaluer la sortie global de notre modèle, la moyenne pondérée des sorties de toutes les règles déclenchées est calculée selon l'équation 7.3 sachant que *N* représente le nombre de règles déclenchées et w_i représente le poids d'une sortie z_i.

$$\text{Sortie} = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} w_i} \qquad (7.3)$$

Le modèle *TSK* offre une flexibilité dans la conception de notre système d'évaluation en utilisant des règle simples (i.e., si x_1 est A_1 et x_2 est A_2 alors $y = f_i\ (x_1, x_2)$) et il est efficace en termes de calcul comme il utilise une moyenne pondérée pour quantifier la sortie contrairement à d'autres modèles qui utilisent la méthode *COG* ou *MOM* [184][185]. Ce modèle permet de quantifier les valeurs de sortie grâce à des valeurs numériques et non des ensembles flous afin de les utiliser dans des fonctions d'optimisation. Le modèle *TSK* est utilisé dans les systèmes où la sortie correspond à un niveau numérique et non pas à un ensemble flou.

Le modèle de logique floue basé sur *TSK* que nous spécifions dans le cadre de l'évaluation de l'un des attributs de la méthode de contrôle d'accès *IoT-MAAC*, à savoir le niveau de confiance des objets IoT de notre Framework, prend en considération différents paramètres d'entrés avec des fonctions d'appartenance spécifiques à chaque paramètre et un ensemble de règles d'inférences que nous décrivons dans les sections suivantes.

7.3.3. Paramètres d'entrée du système d'évaluation du niveau de confiance

Notre modèle de logique floue de type *TSK* utilise trois paramètres d'entrée pour évaluer le niveau de confiance des objets IoT, à savoir la sécurité physique (*DPS* : Device Physical Security), le niveau de sécurité de l'objet (*DSL* : Device Security Level) et la confiance accordée au propriétaire de l'objet (*DOT* : Device Ownership Trust). Différents systèmes de notation sont proposés pour quantifier ces trois paramètres d'entrée (i.e., *DPS*, *DSL* et *DOT*). Chaque système de notation, peut être personnalisé avec différentes valeurs, en fonction du scénario d'utilisation IoT et des caractéristiques de l'offre de service IoT correspondante.

Les objets IoT sont implémentés dans divers environnements physiques avec des caractéristiques de sécurité différentes. En effet, l'objet IoT peut être implémenté dans des environnements avec des niveaux de sécurité hétérogènes tels qu'une zone protégée, une zone contrôlée, une zone non contrôlée, etc. De plus, les objets IoT peuvent être protégés avec un bouclier sécurisé ou peuvent être déployés librement sans protection. Par conséquent, en fonction de leur

environnement physique d'implémentation, les objets IoT peuvent subir des attaques physiques pouvant affecter l'intégrité des systèmes d'exploitation et de leurs fonctionnalités. Par conséquent, le *DPS* est un paramètre d'entrée essentiel dans l'évaluation du niveau de confiance des objets IoT. Nous spécifions un système de notation afin de déterminer le *DPS* relatif à un objet. Ce système de notation (voir Figure 7.6) permet de déterminer une note sur 100 calculée à l'aide de 2 sous-paramètres, à savoir environnement physique et bouclier de sécurité physique. La valeur du paramètre *DPS* relatif à un objet IoT résulte de la somme des notes affectées à ces deux sous-paramètres selon le barème décrit dans la Figure 7.6.

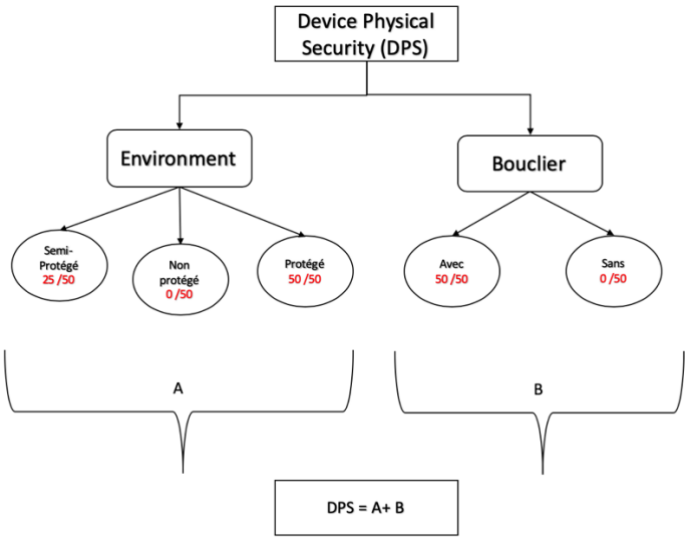

Figure 7.6 : Système de notation DPS

Les objets IoT fixes ont une valeur *DPS* qui ne change pas, étant donné que les propriétés de sécurité de l'environnement de l'objet et l'existence du bouclier de sécurité ne varient pas. Par contre, si les objets IoT sont mobiles, la valeur *DPS* peut varier en fonction de la trajectoire de cet objet. Dans ce contexte, la zone de couverture de la passerelle connectant les objets IoT peut être divisée en sous-zones de différents types (i.e., non protégée, semi-protégée ou protégée). Pour créer les sous-zones, nous prenons en compte la distance et l'orientation de l'objet IoT par rapport à la passerelle grâce à des données *GPS* par exemple. Dans le cas d'un objet IoT mobile, le paramètre *DPS* est calculé à chaque intervalle de temps en fonction de la vitesse de déplacement.

Le niveau de sécurité de l'objet, appelé *DSL*, est le deuxième paramètre d'entrée de notre système de logique floue d'évaluation du niveau de confiance des objets IoT. Nous déterminons le *DSL* de l'objet IoT en se basant sur les caractéristiques des algorithmes de chiffrement et de hachage utilisés par cet objet. De même, nous spécifions un système de notation sur 100 pour ce paramètre *DSL* dont la valeur résulte de la somme des notes affectées à son algorithme de chiffrement et son algorithme de hachage (voir Figure 7.7)

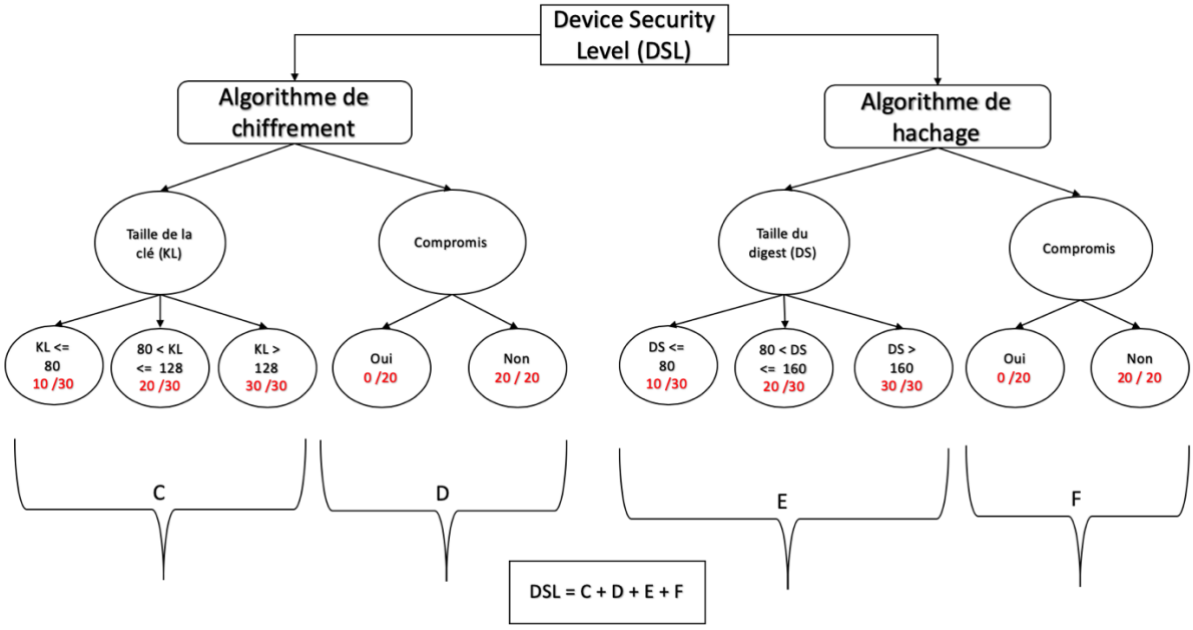

Figure 7.7 : Système de notation DSL

Différents algorithmes de chiffrement et de hachage sont utilisés dans l'environnement IoT (cf. chapitre 3). Plus la longueur de la clé de l'algorithme de chiffrement est importante, plus le *DSL* est élevé. Par conséquent, nous avons classé, dans le cadre du système de notation *DSL*, les algorithmes de chiffrement en trois classes en fonction de la longueur de la clé et nous avons affecté à chaque classe une note sur 30. De plus, nous consacrons une note sur 20 pour indiquer si l'algorithme de chiffrement a été compromis. Nous adoptons un raisonnement similaire pour évaluer l'algorithme de hachage en se basant la longueur du digest dans le cadre du système de notation *DSL*.

Le troisième paramètre d'entrée de notre système d'évaluation du niveau de confiance des objets IoT est la confiance accordée au propriétaire de l'objet (i.e., *DOT*). Ce paramètre a un impact direct sur le niveau de confiance global des objets IoT. Ainsi, différents objets IoT peuvent être utilisés pour collecter différents types d'informations. Ces objets peuvent appartenir à plusieurs types de propriétaires dont l'*IoT-SP*, l'*IoT-C* ou à des tiers. Dans ce contexte, des propriétaires d'objets non fiables peuvent manipuler les objets IoT et modifier par conséquent leurs fonctionnalités et caractéristiques, ce qui a une incidence sur la fiabilité et la précision des informations collectées. Afin de quantifier le paramètre *DOT* relatif à la confiance accordée au propriétaire de l'objet IoT, nous avons utilisé un système de notation sur 100 selon le barème décrit dans la Figure 7.8. Dans notre système de notation *DOT*, les objets de l'*IoT-SP* se voient attribuer 100 points et ceux de l'*IoT-C* 90 points, alors que les objets appartenant à d'autres parties se voient attribuer une note en fonction de l'historique de leurs utilisations. Nous rappelons que les trois systèmes de notation que nous venons de spécifier pour les trois paramètres d'entrée de notre modèle d'évaluation de logique floue peuvent être modifiés ou encore adaptés sans impacter le principe de fonctionnement de notre modèle.

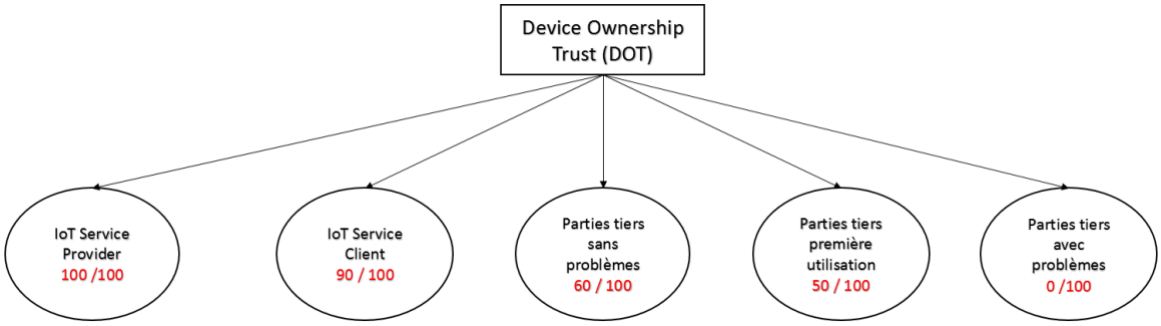

Figure 7.8 : Système de notation DOT

7.3.4. Fonctions d'appartenance du système d'évaluation du niveau de confiance

Dans notre système d'évaluation du niveau de confiance des objets IoT, basé sur la logique floue, nous utilisons trois paramètres d'entrée (i.e., *DPS*, *DSL* et *DOT*). Chacun de ses paramètres possède un ensemble flou formé de trois variables linguistiques offrant trois fonctions d'appartenance triangulaires (Low, Mid et High) pour chaque paramètre. Les fonctions d'appartenance que nous utilisons pour les trois paramètres d'entrée sont représentées par la Figure 7.9 : la courbe bleue (i.e., valeurs des abscisses appartenant à [0, 50]) correspond à la fonction d'appartenance de la variable linguistique « Low », la courbe rouge (i.e., valeurs des abscisses appartenant à [0, 100]) correspond à la fonction d'appartenance de la variable linguistique « Mid » et la courbe verte (i.e., valeurs des abscisses appartenant à [50, 100]) correspond à la fonction d'appartenance de la variable linguistique « High ».

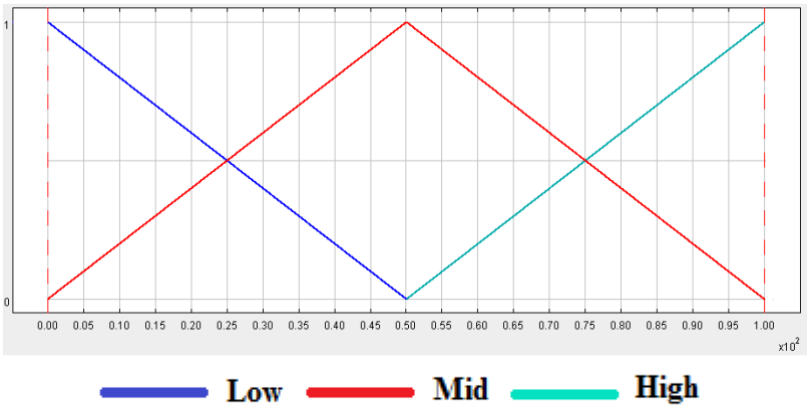

Figure 7.9 : Fonctions d'appartenance de DPS, DSL et DOT

7.3.5. Règles et base d'inférence du système d'évaluation du niveau de confiance

Les trois paramètres d'entrée de notre système d'évaluation ont le même poids, et par conséquent la même importance dans l'évaluation du niveau de confiance des objets IoT. Nous

spécifions dans le Tableau 7.2 la base d'inférence de notre modèle *TSK* grâce à des règles d'inférence en fonction de l'occurrence des niveaux de fonction d'appartenance (i.e., Low, Mid, High). Nous associons à chaque règle une valeur de sortie de confiance bien définie (de 1 à 10, avec 10 le meilleur niveau de confiance possible). Par exemple, une règle avec une fonction d'appartenance Low pour les trois paramètres d'entrée correspond à la première entrée du tableau 7.2 associée à la valeur de sortie égale à 1.

Dans le cadre de notre système d'évaluation du niveau de confiance basé sur la logique floue avec 3 paramètres d'entrée et 3 fonctions d'appartenance pour chaque paramètre, nous spécifions 27 règles dans la base d'inférence correspondante. Le nombre de règles est déterminé à l'aide de l'équation 7.1. Le Tableau 7.2 présente une synthèse de notre base d'inférence.

Tableau 7.2 : Synthèse de la base d'inférence de notre modèle TSK

Occurrences	Valeur de sortie de confiance
3 Low	1
2 Low & 1 Mid	2
2 Low & 1 High	3
2 Mid & 1 Low	4
3 Mid	5
1 Low & 1 Mid & 1 High	6
2 Mid & 1 High	7
2 High & 1 Low	8
2 High & 1 Mid	9
3 High	10

7.3.6. Modèle global d'évaluation du niveau de confiance basé sur la logique floue

Suite à la définition des différents paramètres d'entrée, fonctions d'appartenance et règles de la base d'inférence, nous décrivons dans ce qui suit notre modèle global d'évaluation du niveau de confiance des objets IoT à travers un exemple d'utilisation. La Figure7.10 montre une instanciation des différents composants de notre modèle global de logique floue *TSK* d'évaluation du niveau de confiance. Ce modèle utilise les valeurs des paramètres d'entrée, le composant de fuzzification, les règles de la base d'inférence, le moteur d'inférence *TSK* afin de calculer une valeur de confiance de sortie pour un scénario d'utilisation particulier.

Le scénario d'utilisation de notre modèle d'évaluation du niveau de confiance que nous considérons correspond à un objet IoT implémenté dans un environnement semi-protégé avec un bouclier de sécurité et par conséquent une valeur du paramètre d'entrée *DPS* égale à 75/100. L'objet IoT utilise un algorithme de chiffrement non compromis ayant une longueur de clé de 80 bits et un algorithme de hachage non compromis avec un digest de 160 bits. Dans ce contexte, le *DSL* de cet

objet est de 80/100. Finalement, l'objet IoT est la propriété d'un *IoT-C* et il se voit donc attribuer un paramètre d'entrée *DOT* égal à 90/100.

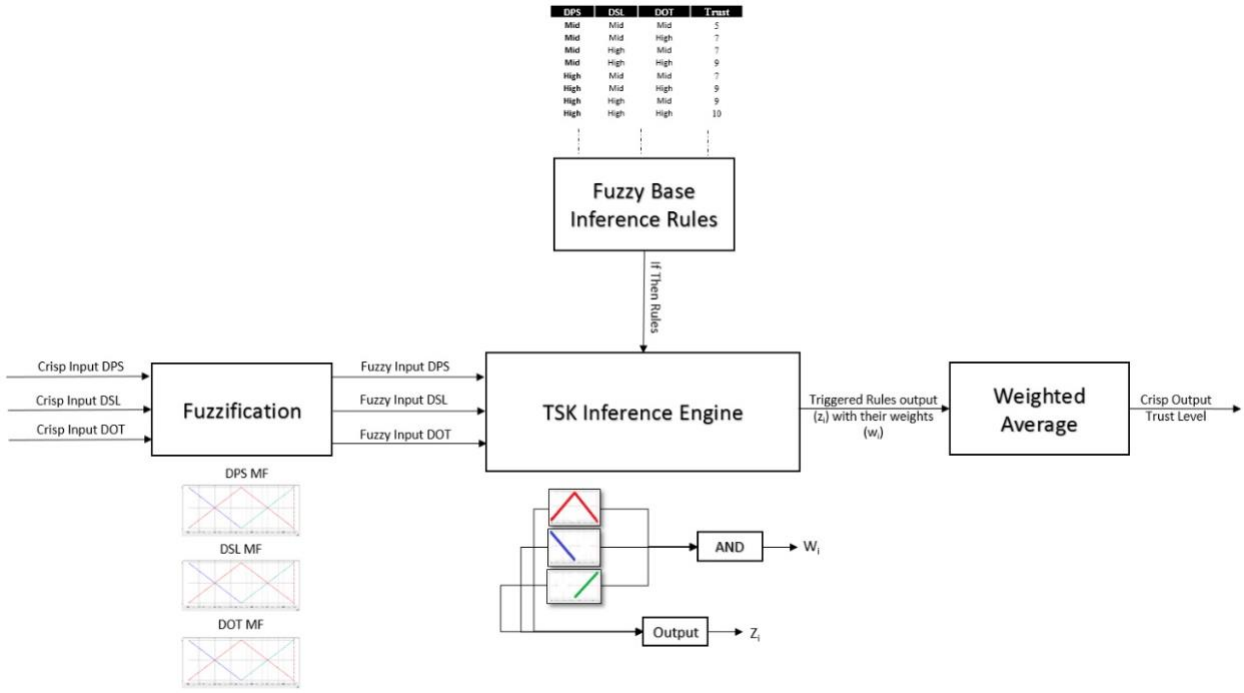

Figure 7.10 : Le modèle de logique floue TSK d'évaluation du niveau de confiance

Les caractéristiques de cet objet IoT avec les valeurs correspondantes des paramètres d'entrée *DPS*, *DSL* et *DOT*, permettent de déclencher 8 règles de la base d'inférence de notre modèle que nous décrivons dans le Tableau 7.3 en les associant au poids et valeur de sortie (i.e., niveau de confiance) de chaque règle. Le poids de la règle 1 du tableau 7.3 est calculé comme suit : le degré de vérité de la fonction d'appartenance *DPS* Mid est égal à 0,5, le degré de vérité de la fonction d'appartenance *DSL* Mid est égal à 0,4 et le degré de vérité de la fonction d'appartenance *DOT* Mid est égal à 0,2 (voir Figure 7.11).

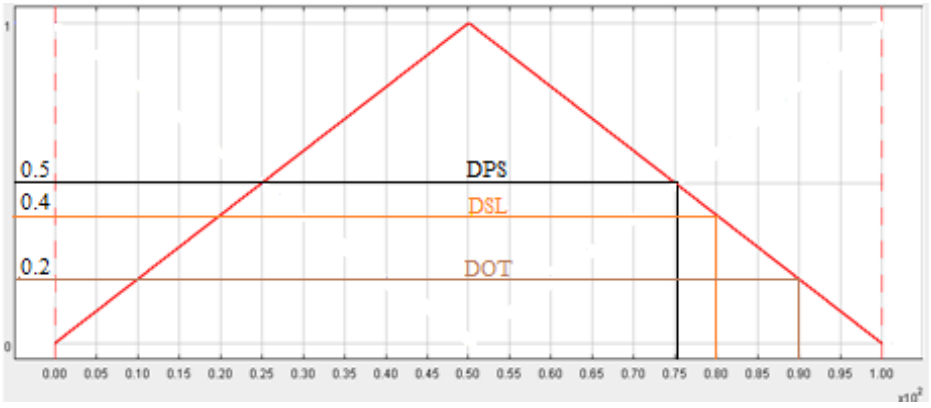

Figure 7.11 : Degré de vérité de DPS, DSL et DOT pour la fonction d'appartenance Mid

Ainsi, nous utilisons la méthode ET pour obtenir un poids égal à 0,2 pour la règle 1 déclenchée comme le montre l'équation 7.4.

$$\text{Poids}_{\text{Règle}_i} = \text{Degré vérité}_{\text{DPS}} \text{ ET Degré vérité}_{\text{DSL}} \text{ ET Degré vérité}_{\text{DOT}} = 0.5 \text{ ET } 0.4 \text{ ET } 0.2 = 0.2 \quad (7.4)$$

La sortie de la règle est déterminée suivant les occurrences des niveaux « Low », « Mid », et « High » comme définie dans le tableau 7.3. Enfin, nous utilisons les valeurs de sortie des règles déclenchées afin de calculer grâce à l'équation 7.3 le niveau de confiance global relatif à cet objet IoT. Dans notre scénario d'utilisation, nous obtenons un niveau de confiance global égal à 8.3 pour l'objet IoT considéré comme le montre l'équation 7.5.

$$Niveau\ de\ confiance\ global = \frac{(0.2*5)+(0.4*7)+(0.2*7)+(0.5*9)+(0.2*7)+(0.4*9)+(0.2*9)+(0.5*10)}{0.2+0.4+02+0.5+0.2+0.4+0.2+0.5} = 8.3 \quad (7.5)$$

Tableau 7.3 : Règles déclenchées du scénario d'utilisation

Règle	DPS	DSL	DOT	Poids	Sortie
1	Mid	Mid	Mid	0.2	5
2	Mid	Mid	High	0.4	7
3	Mid	High	Mid	0.2	7
4	Mid	High	High	0.5	9
5	High	Mid	Mid	0.2	7
6	High	Mid	High	0.4	9
7	High	High	Mid	0.2	9
8	High	High	High	0.5	10
Niveau de confiance global					8.3

7.4. IoT-MAAC : IoT Multiple Attribute Access Control

Nous proposons dans le cadre de notre Framework une nouvelle méthode de contrôle d'accès basée sur les attributs, à savoir *IoT-MAAC*, permettant de contrôler l'accès des objets IoT à la passerelle de bas niveau *LL-Gw*.

Nous présentons dans cette section l'algorithme de prise de décision relatif à la méthode *IoT-MAAC* ainsi que les échanges *IoT-MAAC* à travers les *FSM* de l'objet IoT et la passerelle *LL-Gw* ainsi que le *MSC* relatif à un scénario d'utilisation de cette méthode. Enfin, nous spécifions la stratégie de contrôle d'accès utilisée avec *IoT-MAAC* qui comprend différentes politiques de contrôle d'accès. Chaque politique est spécifique à un groupe d'objets. De plus, chaque politique est composée de deux règles. La première règle permet d'autoriser l'accès si tous les attributs du contrôle d'accès sont respectés alors que la deuxième refuse l'accès si la première règle n'a pas été respectée.

7.4.1. Algorithme de prise de décision IoT-MAAC

Nous spécifions pour notre méthode de contrôle d'accès *IoT-MAAC* un algorithme de prise de décision de type First Applicable (c'est à dire, la première règle vérifiée est appliquée) (voir Figure 7.12).

```
Decision Making Algorithm [LL-Gw_ID]

Strategy ← Retrieve_Strategy_from_Strategy_Directory (LL-Gw_ID)
Number_Policies ← Retrieve_Number_Policies (Strategy);
for i ∈ (1, Number_Policies) do
        if ( Group_Object_ID (IoT Object) = Group_Object_ID (Policy(i)) ) then
                Number_Rules ← 2;
                for i ∈ (1, Number_Rules) do
                        if ( Attributes (Rule (i) = 1) ) then
                                return Deny;
                        else
                             if [( (IoT Object ID) ∈ (List_Object [Group Object ID]) )
                                &( Fingerprint (IoT Object) = Fingerprint (Rule(i)) )
                                    & ( Trust_Level (IoT Object) >= Trust_Level (Rule(i)) )
                                        & ( Request_Access_Time (Start) >= Start_Time(Rule(i)))
                                            & ( Request_Access_Time(End) < End_Time(Rule(i)))] then
                                                return Permit;
                             end if
                        end if
                end for
        end if
end for
return NotApplicable
```

Figure 7.12 : Algorithme de prise de décision IoT-MAAC

L'algorithme de prise de décision *IoT-MAAC* implémenté au niveau du *PDP* dans la *HL-Gw* (voir Figure 7.12) relatif à l'accès d'un objet IoT à une passerelle *LL-Gw* nous permet de déterminer dans un premier temps la stratégie associée à cette passerelle de bas niveau ainsi que le nombre de politiques formant cette stratégie. Par la suite, l'algorithme nous permet de parcourir ces politiques afin de déterminer, grâce à la valeur du *Group Object ID*, celle relative à l'objet IoT demandant l'accès. La politique associée à cet objet, comme chacune des autres politiques de la stratégie, comprend toujours deux règles qui seront testés successivement. Ainsi, l'algorithme de prise de décision commence par tester dans l'ordre les différents attributs de la première règle. Pour ce faire, l'algorithme teste le nombre d'attributs dans la règle. Si un seul attribut existe dans la règle en question, alors une décision de refus d'accès est prise comme les attributs de la requête reçue ne respectent pas les attributs définis dans la première règle de la politique. Sinon si plusieurs attributs existent, la première règle est testée et par conséquent les différents attributs seront évalués en respectant l'ordre suivant : test de l'inclusion de l'*IoT Object ID* dans la liste des objets du *Group Object ID*, test de l'empreinte digitale de l'objet (*Fingerprint*), test du niveau de confiance de l'objet (*Trust Level*) et test du temps d'accès (*Request Access Time*). Ainsi, l'algorithme de prise de décision

commence par vérifier les attributs dans cet ordre. Si un attribut de l'objet ne correspond pas à l'attribut de la règle, alors l'algorithme nous permet de passer directement à la règle suivante (i.e., deuxième règle correspondant à un refus d'accès). En revanche, si tous les attributs de la règle sont respectés alors une autorisation d'accès est accordée à l'objet IoT. Il est à noter que l'algorithme *IoT-MAAC* prévoit, dans le cas où aucune politique ne correspond au *Group Object ID* remontée par l'objet, qu'une décision de non applicabilité est remontée ce qui correspond à un refus d'accès implicite. Cette décision de non applicabilité résulte du fait que la stratégie de la *LL-Gw* ne comprend pas une politique et par conséquent des règles concernant le *Group Object ID* de l'objet demandant l'accès.

7.4.2. Spécification des échanges IoT-MAAC

Les échanges d'informations concernant le contrôle d'accès des objets IoT aux passerelles *LL-Gw* en utilisant la méthode *IoT-MAAC* sont présentés dans cette section à travers les automates à états finis (*FSM*) des différentes entités du contrôle d'accès (i.e., objet IoT et *LL-Gw*) ainsi qu'un *MSC* relatif à un scénario d'utilisation *IoT-MAAC*. Le processus de contrôle d'accès *IoT-MAAC* nécessite le passage de l'objet IoT par différents états présentés dans la Figure 7.13. L'objet IoT passe de l'état S0 à l'état S1 en envoyant une requête de contrôle d'accès *IoT-MAAC* à la passerelle *LL-Gw*. Dans l'état S1, l'objet IoT communique avec la *LL-Gw* pour établir de la clé symétrique et passe à l'état S2 suite à la création de cette dernière. Ensuite, il passe à l'état S3 en envoyant une assertion *SAML* chiffrée par la clé symétrique et comportant entre autres un des attributs *IoT-MAAC*. Suite à cet envoi, l'objet IoT reçoit la réponse *IoT-MAAC*. Si la réponse est positive (Access Control Success), l'objet IoT passe à l'état S4 et initie l'envoi des données chiffrés vers la *LL-Gw* jusqu'à la fin de la validité de la clé symétrique. Si la réponse est négative (Access Control Failure), l'objet IoT repasse à l'état S0 tout en éliminant la clé symétrique de chiffrement.

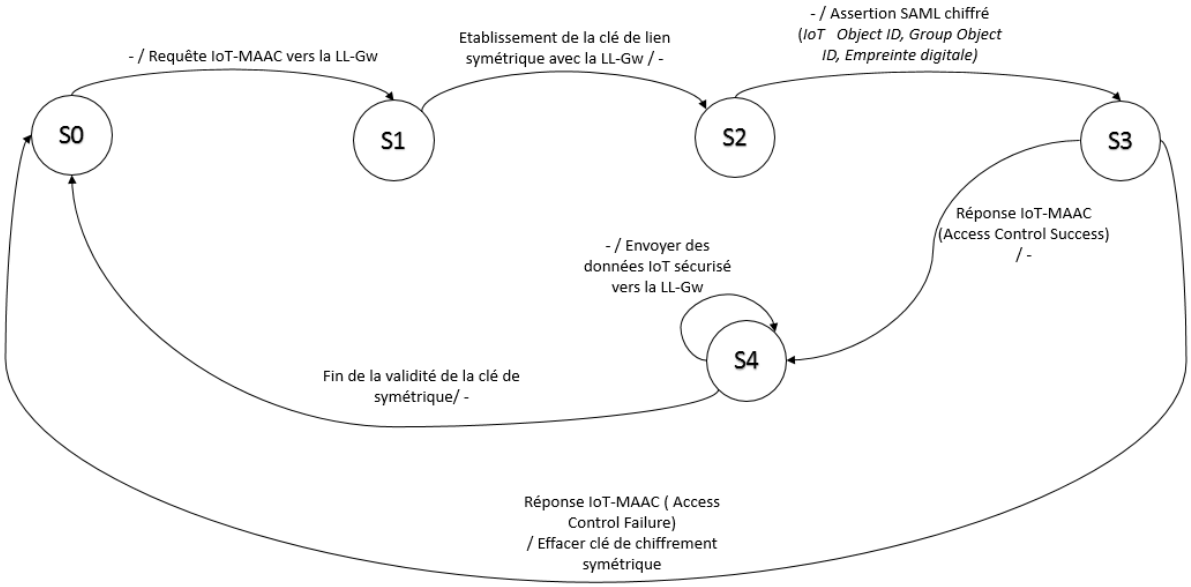

Figure 7.13 : FSM de l'objet IoT concernant IoT-MAAC

De même, le processus de contrôle d'accès *IoT-MAAC* nécessite le passage de la *LL-Gw* par différents états présentés dans la Figure 7.14. La *LL-Gw* passe de l'état S0 à l'état S1 suite à la réception d'une requête *IoT-MAAC* émanant d'un objet IoT. Dans cet état, la *LL-Gw* initialise l'établissement de la clé symétrique avec l'objet IoT et passe à l'état S2 suite à la création de cette dernière. Ensuite, la *LL-Gw* reçoit une assertion *SAML* chiffrée de l'objet IoT qu'il traduit en une requête *XACML* afin de l'envoyer vers la *HL-Gw* et passer à l'état S3. A la réception d'une réponse *XACML* positive, la *LL-Gw* traduit la réponse d'autorisation *IoT-MAAC XACML* en une réponse *IoT-MAAC* afin de l'envoyer à l'objet IoT et passer à l'état S4. Ainsi, la *LL-Gw* va rester dans le même état S4 et commencer à recevoir les données chiffrés de l'objet IoT jusqu'à la fin de la validité de la clé de chiffrement symétrique. Dès la fin de la validité de cette la clé, la *LL-Gw* repasse à l'état S0 pour recommencer le processus de contrôle d'accès *IoT-MAAC*. Il est à noter que dans le cas d'une réponse *IoT-MAAC XACML* mentionnant un refus d'autorisation d'accès, la passerelle *LL-Gw* passe directement de l'état S3 à l'état S0 en envoyant à l'objet IoT la réponse *IoT-MAAC* correspondante au refus d'accès. De plus, la clé de chiffrement symétrique est supprimée de la *LL-Gw*. Ainsi, si la *LL-Gw* reçoit dans l'état S0, des données chiffrées par des clés non reconnues par cette dernière, alors ces données sont ignorées et éliminées.

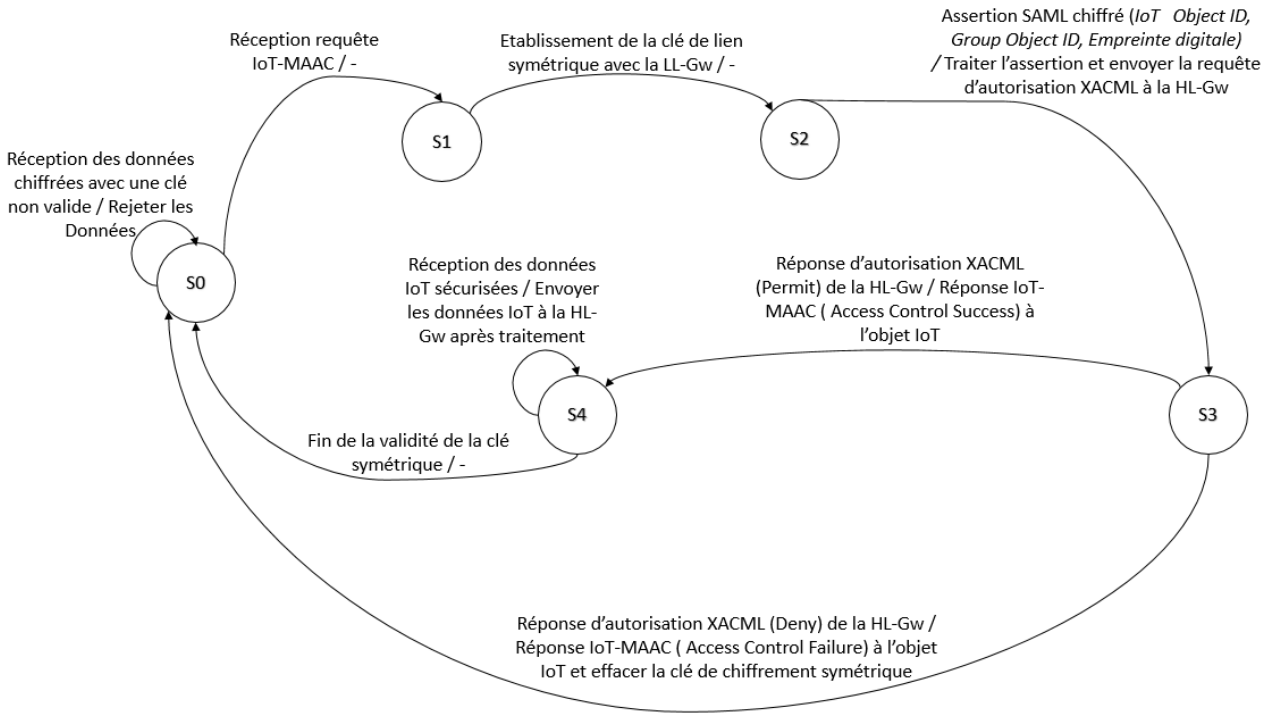

Figure 7.14 : FSM de la LL-Gw concernant IoT-MAAC

Les différents échanges entre les composants de l'architecture de contrôle d'accès *IoT-MAAC* sont spécifiés à travers le *MSC* de la Figure 7.15. Premièrement, une requête de contrôle d'accès *IoT-MAAC* est envoyée par l'objet vers la passerelle *LL-Gw*. Dès la réception de cette requête, le processus d'établissement de la clé de session entre l'objet IoT et la passerelle *LL-Gw* est initialisé. Différents

algorithmes et protocoles adaptés à un environnement IoT peuvent être utilisés afin de permettre l'établissement de cette clé. A titre d'exemple, nous proposons l'utilisation du protocole *SKKE* (Symmetric-Key key Establishment) utilisé dans ZigBee, etc. La création de la clé symétrique à ce stade permet de sécuriser les échanges concernant le contrôle d'accès entre les objets IoT et les passerelles. Suite à l'établissement de la clé de session, l'objet IoT envoie dans le cadre du contrôle d'accès *IoT-MAC* une assertion *SAML* chiffré par cette clé secrète. Cette assertion est déchiffrée au niveau de la passerelle *LL-Gw* et elle est traduite dans le cadre du processus *IoT-MAAC* en une requête *IoT-MAAC XACML* de demande d'autorisation envoyée vers la *HL-Gw*. La réception de cette demande d'autorisation *IoT-MAAC XACML* par la *HL-Gw*, jouant le rôle de *PDP* dans l'architecture *IoT-MAAC*, déclenche une première requête interne à la *HL-Gw* envoyée vers l'autorité d'attributs pour récupérer l'attribut niveau de confiance de l'objet IoT (cf. section 7.3). Une deuxième requête interne à la *HL-Gw* permet d'interroger le répertoire des stratégies afin récupérer la stratégie de contrôle d'accès relative à la passerelle *LL-Gw* en question. Ainsi, le *PDP* utilise l'algorithme de prise de décision *IoT-MAAC* (cf. section 7.5.1) afin d'envoyer une réponse d'autorisation *IoT-MAAC XACML*, contenant la décision, vers la *LL-Gw* jouant le rôle de *PEP* dans l'architecture *IoT-MAAC*. Cette réponse *XACML* sera traduite par la *LL-Gw* en une réponse de contrôle d'accès *IoT-MAAC* compréhensible par l'objet IoT. D'une part, une réponse de contrôle d'accès *IoT-MAAC* positive permet à l'objet IoT de commencer à envoyer les données en les chiffrant avec la clé secrète partagée avec la *LL-Gw* et créée au début du processus de contrôle d'accès. Les données seront ensuite déchiffrées au niveau de la *LL-Gw* et envoyées vers la *HL-Gw* après les traitements nécessaires et leur chiffrement par la clé de la *HL-Gw*. D'autre part, une réponse négative s'accompagne de la suppression de la clé secrète de la *LL-Gw*.

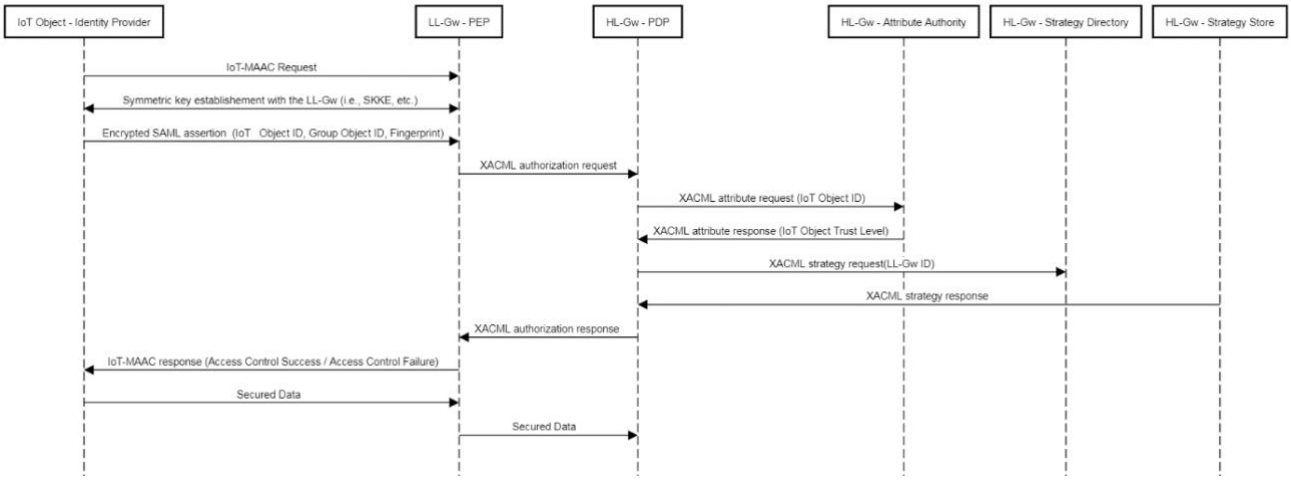

Figure 7.15 : MSC de contrôle d'accès IoT-MAAC

Nous spécifions dans le cadre de la méthode de contrôle d'accès *IoT-MAAC* différents échanges dont le format respecte les standards *SAML* et *XACML*. Ainsi, l'objet IoT utilise une assertion *SAML* (voir Figure 7.16) afin de communiquer entre autres l'un des attributs de contrôle d'accès IoT-MAAC à la passerelle *LL-Gw*.

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  Version="2.0"
  IssueInstant="2019-02-12T15:26:12Z">
    <saml:Issuer Format="urn:oasis:names:SAML:2.0:nameid-format:entity">
        IoT Object 1
    </saml:Issuer>
    <saml:Subject>
        <saml:NameIdentifier> IoT Object 1</saml:NameIdentifier>
    </saml:Subject>
    <saml:Conditions NotBefore="2019-02-12T15:26:12Z" NotOnOrAfter="2019-02-12T15:28:12Z"/>
    <saml:AttributeStatement>
        <saml:Attribute Name="Fingerprint">
            <saml:AttributeValue>$oàikj^plnrusdn=çkdjd21587</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="I
            <saml:AttributeValue>11:12:13:14:15:16</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="Group_Object_ID">
            <saml:AttributeValue>1</saml:AttributeValue>
        </saml:Attribute>
    </saml:AttributeStatement>
</saml:Assertion>
```

Figure 7.16 : Assertion SAML de l'objet IoT

Nous présentons dans la Figure 7.16 un exemple d'assertion *SAML* envoyé par un objet IoT qui contient des informations relatives à la validité de cette assertion (i.e., NotBefore, etc) ainsi que l'attribut de contrôle d'accès de type « Fingerprint » mais aussi des informations d'indentification de l'objet telles que « IoT_Object_ID » et « Group_Object_ID ».

Dans le cadre du contrôle d'accès *IoT-MAAC*, cette assertion *SAML* est traduite par le *PEP* (i.e., *LL-Gw*) en une requête d'autorisation *IoT-MAAC XACML* (voir Figure 7.17) afin de l'envoyer au *PDP* (i.e., *HL-Gw*) qui doit prendre une décision en exécutant l'algorithme de prise de décision *IoT-MAAC*. La requête *XACML* comprend alors des informations récupérées de l'assertion *SAML* mais aussi des informations concernant la ressource demandée (i.e., identificateur de la passerelle *LL-Gw*) et les droits d'accès à la ressource (i.e., lecture, écriture, modification, etc.) spécifiés par la ressource elle-même. En effet, la *LL-Gw* spécifie dans la requête les droits d'accès de l'objet en question afin que ces droits soient étudiés par le *PDP* au niveau de la *HL-Gw*. De plus, un intervalle de temps est spécifié dans cette requête *XACML* pour définir le temps d'accès demandé. La Figure 7.17 montre un exemple de requête *XACML* envoyée par la *LL-Gw* à la *HL-Gw*.

Suite à la prise de décision par la *HL-Gw* jouant le rôle de *PDP* dans l'architecture I*oT-MAAC*, une réponse *XACML* est ennoyée à la *LL-Gw* jouant le rôle de *PEP*. Nous représentons dans la Figure 7.18 un exemple de réponse d'autorisation *IoT-MAAC XACML* positive de type « Permit ». Cette réponse sera ensuite traduite par le *PEP* en une réponse finale d'accès *IoT-MAAC* compréhensible par l'objet suivant la technologie de communication utilisée entre cet objet et la passerelle *LL-Gw*.

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>IoT Object 1</AttributeValue>
    </Attribute>
    <Attribute AttributeId="Fingerprint" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>$oàikj^plnrusdn=ckdjd21587</AttributeValue>
    </Attr
    <Attribute AttributeId="IoT_Object_ID" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>11:12:13:14:15:16</AttributeValue>
    </Attribute>
    <Attribute AttributeId="Group_Object_ID" DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>1</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#string"
      <AttributeValue>11:15:16:18:17:16</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>read</AttributeValue>
    </Attribute>
    <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeValue>write</AttributeValue>
    </Attribute>
  </Action>
  <Environment>
     <Attribute AttributeId="NotBefore" DataType="http://www.w3.org/2001/XMLSchema#datetime">
    <AttributeValue>2019-02-12T15:26:12Z</AttributeValue>
    </Attribute>
    <Attribute AttributeId="NotAfter" DataType="http://www.w3.org/2001/XMLSchema#datetime">
    <AttributeValue>2019-02-12T15:26:12Z</AttributeValue>
    </Attribute>
</Request>
```

Figure 7.17 : Requête d'autorisation XACML de la LL-Gw dans IoT-MAAC

```
<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Result>
        <Decision>Permit</Decision>
        <Status>
            <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
        </Status>
    </Result>
</Response>
```

Figure 7.18 : Réponse d'autorisation XACML de la HL-Gw dans IoT-MAAC

7.4.3. Stratégie de contrôle d'accès IoT-MAAC

Nous spécifions dans le cadre de la méthode de contrôle d'accès *IoT-MAAC* une stratégie relative à chaque *LL-Gw*. Cette stratégie comprend différentes politiques de contrôle d'accès. Chacune de ces politiques est spécifique à un groupe d'objets identifié par *Group Object ID*. Le nombre de *Group Object ID* associés à une *LL-Gw* détermine le nombre de politiques dans une stratégie. Nous spécifions pour chaque politique et par conséquent pour chaque groupe d'objets deux règles. Une première règle contient une liste d'attributs et leurs valeurs respectives à vérifier lors du contrôle d'accès pour avoir une autorisation d'accès. La deuxième règle contient juste l'identificateur du groupe d'objets comme attribut et donne lieu à une décision de refus d'accès. Ainsi, si tous les attributs de la première règle sont respectés, un accès est accordé (Decision = Permit). Par contre, si un des paramètres n'est pas respecté la deuxième règle est testée ce qui correspond à la règle avec un seul attribut donnant une décision de refus d'accès systématique (Decision = Deny). Dans le cas où

le *Group Object ID* de l'objet IoT ne correspond à aucune politique de la stratégie, une décision de non applicabilité est prise (Decision = Not Applicable) qui correspond à une décision de refus implicite.

Afin de spécifier les règles de nos politiques de contrôle d'accès formant nos stratégies *IoT-MAAC*, nous utilisons un outil, appelé (*ACPT* : Access Control Policy Testing), développé par le NIST (National Institute of standards and Technology) qui permet de créer des stratégies formées par des politiques incluant des règles de contrôle d'accès et de tester leurs conformités. *ACPT* est un outil qui vérifie de manière automatique les fautes syntaxiques et sémantiques des stratégies de contrôles d'accès avant de les déployer. De plus, cet outil fournit un modèle d'interface graphique pour la composition des stratégies de contrôle d'accès, un vérificateur *SMV* (Symbolic Model Verification) des propriétés du modèle de contrôle d'accès, un ensemble de tests complet généré par l'outil *ACTS* (Automated Combinatorial Testing for Software) du NIST [186], et une génération automatique de politiques XACML en sortie du modèle vérifié [187]. Ainsi, nous avons utilisé *ACPT* pour créer les différents attributs du contrôle d'accès *IoT-MAAC*, les sujets, les ressources, les actions possibles et les détails concernant l'environnement. Ensuite, nous avons créé les règles concernant les groupes d'objet (deux règles par groupe d'objet formant une politique d'accès pour ce groupe) et nous avons généré automatiquement les politiques *XACML* correspondantes dont un exemple est illustré par la Figure 7.19 et ce grâce à cet outil *ACPT* (voir Figure 7.20).

```
<?xml version="1.0" encoding="UTF-8"?>
<ACPT>
    <Target>
        <Root Type="Root" Value="MAAC">
            <Subjects Type="Subjects" Value="Subjects">
                <Subject_Attributes Type="Subject Attributes" Value="String;Fingerpirnt">
                <Subject_Attributes Type="Subject Attributes" Value="String;Group_Object_ID">
                <Subject_Attributes Type="Subject Attributes" Value="String;Service_Time">
                <Subject_Attributes Type="Subject Attributes" Value="Integer;Trust_Level">
            </Subjects>
            <Resources Type="Resources" Value="Resources">
                <Resource_Attributes Type="Resource Attributes" Value="Integer;LL_GW_ID">
            </Resources>
            <Actions Type="Actions" Value="Actions">
            <Environments Type="Environments" Value="Environments"/>
            <Inheritance Type="Inheritance" Value="Inheritance"/>
        </Root>
    </Target>
    <Model>
        <ROOT Type="ROOT" Value="Model">
            <RBAC Type="RBAC" Value="RBAC"/>
            <MULTILEVEL Type="MULTILEVEL" Value="MULTILEVEL"/>
            <WORKFLOW Type="WORKFLOW" Value="WORKFLOW"/>
            <ABAC Type="ABAC" Value="ABAC">
                <ABAC Type="ABAC" Value="LL_GW_123456">
                    <ABACRULES Type="ABACRULES" Value="Rules: First-applicable">
                        <ABACRULES Type="ABACRULES" Value="Rule 1#Subject Attributes;Fingerpirnt;String;sfjlifbzeeifnzifnziofu#Subject Attributes;Group_Object_ID;String;G1
                        #Subject Attributes;Service_Time;String;All_day#Subject Attributes;Trust_Level;Integer;7#Resource Attributes;LL_GW_ID;Integer;123456
                        #Action Attributes;MLSDefaultAction;String;write->Permit"/>
                        <ABACRULES Type="ABACRULES" Value="Rule 2#Resource Attributes;LL_GW_ID;Integer;123456->Deny"/>
                    </ABACRULES>
                </ABAC>
                <ABAC Type="ABAC" Value="LL_GW_456789">
                    <ABACRULES Type="ABACRULES" Value="Rules: First-applicable"/>
                </ABAC>
            </ABAC>
        </ROOT>
    </Model>
    <SecurityConstraints>
        <ROOT Type="ROOT" Value="SOD"/>
    </SecurityConstraints>
    <Combinations>
        <ROOT Type="ROOT" Value="Policies: First-applicable">
            <COMBINATION Type="COMBINATION" Value="ABAC#LL_GW_123456"/>
            <COMBINATION Type="COMBINATION" Value="ABAC#LL_GW_456789"/>
        </ROOT>
    </Combinations>
    <Property>
        <ROOT Type="ROOT" Value="PROPERTY"/>
    </Property>
</ACPT>
```

Figure 7.19 : Extrait d'une politique de contrôle d'accès IoT-MAAC

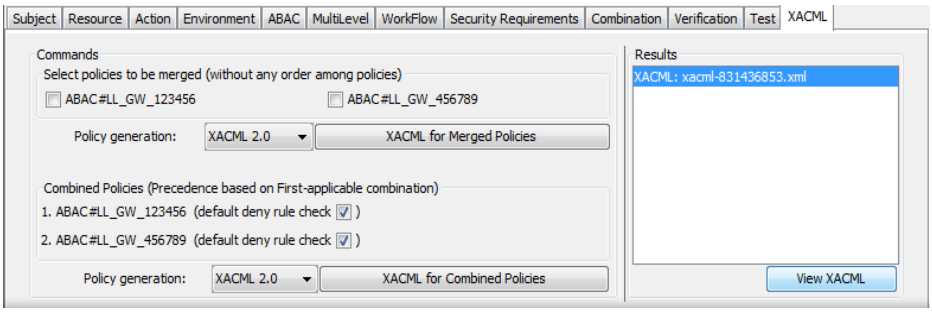

Figure 7.20 : Utilisation de l'Outil ACPT du NIST

7.5. Conclusion

Nous avons proposé dans ce chapitre un Framework relatif à une nouvelle méthode de contrôle d'accès des objets IoT à la passerelle de bas niveau (*LL-Gw*) de la couche sensing de l'architecture IoT, appelée *IoT-MAAC*. Cette méthode est basée sur différents attributs dont l'intégrité des objets, le niveau de confiance et l'identité de l'objet IoT. L'implémentation de la méthode de contrôle d'accès *IoT-MAAC* permet uniquement aux objets authentiques ayant un niveau de confiance supérieur au seuil spécifié par l'utilisateur de participer à un service IoT. Par conséquent, les données utilisées pour le service IoT correspondant sont plus fiables et sécurisées, de sorte à satisfaire les utilisateurs. De plus, *IoT-MAAC* est modulaire grâce à la capacité de l'*IoT-C* à spécifier le seuil de niveau de confiance d'objet IoT à prendre en compte pour l'octroi de l'accès de ces objets aux *LL-Gw* en fonction de la caractéristique de service IoT. Nous avons évalué ce niveau de confiance grâce à un système basé sur la logique floue.

Ainsi, grâce à notre méthode *IoT-MAAC*, les objets IoT doivent être autorisés suite à un contrôle d'accès avant de se connecter à la passerelle *LL-Gw* et remonter des informations relatives à un service IoT. Nous avons spécifié pour la méthode *IoT-MAAC* des politiques de contrôle d'accès que nous avons générées automatiquement grâce à l'outil *ACPT* du *NIST* mais aussi un algorithme de prise de décision. Après avoir défini dans ce chapitre ainsi que les chapitres précédents notre Framework de garantie de niveau de service intégrant des mécanismes de QoS (i.e. *QBAIoT*) et de sécurité (i.e. *IoT-MAAC*), nous spécifions dans les deux chapitres suivants son adaptation à la gestion autonome.

Chapitre 8 – Gestion autonome de QBAIoT

8.1. Introduction

La gestion autonome de la *QoS* dans un environnement Internet des objets est un enjeu primordial permettant de minimiser les coûts de gestion des infrastructures tout en respectant les exigences et les recommandations des organismes de standardisation. Cette gestion autonome permet d'assurer un service IoT géré tout en limitant l'intervention humaine et les erreurs qui en découlent mais aussi en facilitant la tâche de gestion des administrateurs. Dans ce contexte, nous apportons à notre méthode de contrôle d'accès au canal basée sur la *QoS*, à savoir *QBAIoT* (cf. chapitre 5), deux fonctions de gestion autonome qui sont l'auto-configuration et l'auto-optimisation.

Ainsi, nous présentons dans la section 2 les besoins de gestion autonome de la méthode de contrôle d'accès au canal *QBAIoT*. Dans la section 3, nous décrivons une adaptation de notre Framework de garantie de niveau de service (cf. chapitre 4) qui nous permet d'inclure les principes de la gestion autonome. Par la suite nous spécifions dans les sections 4 et 5, respectivement, les fonctions d'auto-configuration et d'auto-optimisation de *QBAIoT* tout en détaillant le principe de fonctionnement, le design et les algorithmes utilisés pour réaliser ces fonctions de gestion autonome. Nous associons dans la section 6 ces deux fonctions de gestion autonome pour une meilleure utilisation de la méthode *QBAIoT* dans notre Framework de garantie de niveau de service dans l'IoT. Enfin, nous évaluons dans la section 7 les performances des fonctions de gestion autonome de la méthode *QBAIoT* selon différents scénarios avant de conclure ce chapitre dans la section 8.

8.2. Besoins de la gestion autonome de QBAIoT

QBAIoT est une méthode de contrôle d'accès, basée sur la *QoS*, à un canal partagé entre différents objets IoT connectés à une passerelle de bas niveau *LL-Gw*. Cette méthode de contrôle d'accès est basée sur une configuration qui dépend des caractéristiques de l'environnement IoT. En fait, cette configuration dépend du type des données remontées par les objets et par conséquent des classes de *QoS* existantes dans l'environnement IoT. De plus, cette configuration doit aussi s'adapter en temps réel aux changements des caractéristiques de l'environnement IoT. Ainsi, une gestion autonome de la configuration de *QBAIoT* est nécessaire pour rendre possible son adaptation à l'environnement IoT sans intervention humaine et par conséquent réduire le coût de gestion de l'infrastructure IoT. En effet, les coûts opérationnels (i.e., *OPEX* : Operational Expenditure) liés à la gestion de l'infrastructure IoT sont très importants à cause d'un nombre élevé d'objets IoT dans la couche sensing mais aussi l'hétérogénéité des technologies utilisées. Dans ce contexte, le paradigme de l'Autonomic Computing (*AC*) ou encore gestion autonome caractérisé par le concept d'auto-configuration permet de faire face à la complexité croissante de la gestion de l'infrastructure IoT. Cette auto-configuration permet aussi de réduire les erreurs humaines de configuration et son impact

sur la *QoS* du service IoT rendu à l'utilisateur comme par exemple les conséquences d'une configuration non adaptée sur le temps d'attente des trafics avant l'accès au canal.

D'autre part, une auto-optimisation de la configuration initiale de notre méthode d'accès *QBAIoT* est nécessaire pour une utilisation efficace des ressources (i.e., canaux d'accès sans fil entre les objets IoT et la passerelle *LL-Gw* de la couche sensing) en temps réel. En effet, la configuration initiale de *QBAIoT* est basée sur le nombre de classes de trafics IoT qui existent dans l'environnement de la couche sensing sans prendre en compte la charge de trafic dans chaque classe de *QoS*. Une classe peut comprendre un ou plusieurs objets IoT qui peuvent utiliser la totalité ou une part des ressources affectés à cette classe. Pour minimiser le gaspillage de ressources, une méthode de réaffectation des *IT* selon les besoins des différents trafics est nécessaire afin d'optimiser l'allocation des ressources et d'améliorer la *QoS* de toutes les classes. Dans ce qui suit nous définissons les fonctions d'auto-configuration et d'auto-optimisation que nous proposées pour la méthode d'accès au canal *QBAIoT* en se basant sur la boucle *MAPE-K* (cf. chapitre 3) de gestion autonome et une base de connaissances des configurations possibles.

8.3. Framework de gestion autonome de QBAIoT

Le processus de gestion autonome *QBAIoT* est mis en œuvre via l'implémentation de la boucle de contrôle fermée *MAPE-K* sur la *HL-Gw* et la *LL-Gw* de notre architecture IoT (voir Figure 8.1).

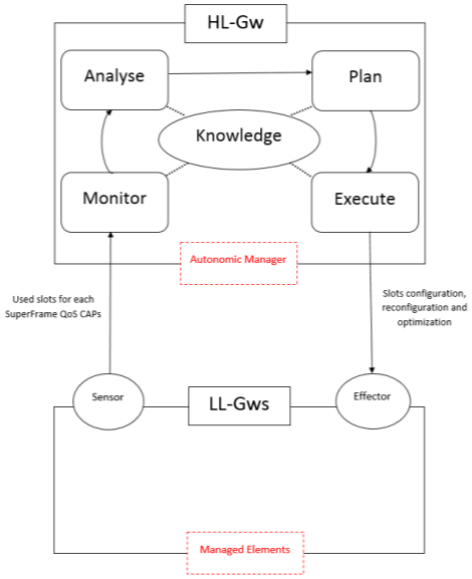

Figure 8.1 : Implémentation de la boucle MAPE-K sur la HL-Gw et la LL-Gw

Notre Framework de gestion autonome (voir Figure 8.1) se base sur les composants de la couche sensing de notre architecture IoT. En effet, la *HL-Gw* correspond au gestionnaire autonome (*AM*) qui comprend 4 fonctionnalités (i.e., surveiller (Monitor), analyser (Analyse), planifier (Plan) et exécuter (Execute) ainsi que la base de connaissances (Knowledge Base). Les spécifications de

chaque fonctionnalité concernant les processus d'auto-optimisation et d'auto-configuration sont détaillées dans la section suivante. La *LL-Gw* dispose d'une interface de type effecteur (Effector) qui reçoit les configurations envoyées par la *HL-Gw* afin de mettre à jour les informations de configurations des *IT* dans la balise *QBAIoT*. L'interface de type capteur (Sensor) de la *LL-Gw* est utilisée pour remonter des informations en temps réel relatives à l'environnement IoT afin de permettre une adaptation de la configuration suivant les caractéristiques de cet environnement et assurer ainsi une auto-optimisation de la configuration de *QBAIoT*. Ainsi, la gestion autonome de *QBAIoT* assure 2 fonctions importantes parmi les 4 fonctions de gestion autonome (i.e., *Self CHOP*). La première fonction est l'auto-configuration permettant de choisir une configuration adaptée en fonction du nombre et types de classes de *QoS* qui existent dans l'environnement IoT et la deuxième concerne l'auto-optimisation de cette configuration pour assurer une utilisation efficace des ressources. Dans ce qui suit, nous présentons ces deux fonctions de gestion autonome *QBAIoT*.

8.4. Auto-configuration de QBAIoT

8.4.1. Principe de fonctionnement

La nature des trafics existants dans l'environnement IoT influence les valeurs de *BO*, *SO* et les configurations des *IT* des différentes *QoS CAP*. Nous spécifions dans le Tableau 8.1 différents scénarios d'existence de classes de trafic, ainsi que la configuration des *IT* et les valeurs *BO* / *SO* correspondantes. Ainsi, le Tableau 8.1 représente la base de connaissances de la fonction d'auto-configuration de *QBAIoT*. Dans ce contexte, nous considérons que *QoS CAP1* est attribué à la classe de *QoS* la plus prioritaire, tandis que *QoS CAP4* est attribué à la classe de *QoS* la moins prioritaire dans le cas où quatre classes de *QoS* existent dans l'environnement IoT considéré.

Tableau 8.2 : Configurations disponibles dans la base de connaissances QBAIoT

Existence trafics	**BO/SO**	**Durée IT**	**Slots (QoS_CAP _Initial$_1$)**	**Slots (QoS_CAP _Initial$_2$)**	**Slots (QoS_CAP _Initial$_3$)**	**Slots (QoS_CAP _Initial$_4$)**
1 *RTC* ou 1 *NRTC*	14	15.72 s	16	N/A	N/A	N/A
2 *NRTC*	3	7.68 ms	13	3	N/A	N/A
2 *RTC*	2	3.84 ms	9	7	N/A	N/A
1 *RTC* et 1 *NRTC*	2	3.84 ms	12	4	N/A	N/A
1 *RTC* et 2 *NRTC*	2	3.84 ms	8	5	3	N/A
2 *RTC* et 1 *NRTC*	2	3.84 ms	7	6	3	N/A
2 *RTC* et 2 *NRTC*	2	3.84 ms	6	5	3	2

La classe temps réel (*RTC*) inclut le trafic des classes *RTNMC* et *RTNMC*, tandis que la classe non temps réel (*NRTC*) inclut le trafic des classes de *Streaming* et *NRT*. La priorité des différents trafics IoT selon un ordre descendant est comme suit: *RTMC*, *RTNMC* , *Streaming*, *NRT*. Par exemple, considérons le cas où dans l'environnement IoT existent des trafics appartenant à deux classes de

QoS: 1 *RTC* et 1 *NRTC*. Selon le Tableau 8.1, 12 *IT* sont attribués à *QoS CAP1* qui correspond à la priorité la plus élevée (c'est-à-dire au trafic *RTC*), tandis que *QoS CAP2* se vo*it* attribuer 4 *IT* et correspond au trafic *NRTC*.

En se basant sur les différents *gSLA* stockés au niveau de la *HL-Gw*, cette dernière calcule le nombre de classes de *QoS* qui existent dans l'environnement d'une *LL-Gw*. Par conséquent, la *HL-Gw*, jouant le rôle de gestionnaire autonome, récupère de la base de connaissance (voir Tableau 8.1) la configuration correspondante des *IT* de la supertrame *QBAIoT* à annoncer à la *LL-Gw* via l'interface de type « Effector ». En effet, les différentes configurations disponibles dans la base de connaissances sont les configurations initiales à définir au niveau de la *LL-Gw* sachant que cette dernière va par la suite les communiquer aux objets IoT grâce aux balises *QBAIoT*. Ces configurations initiales peuvent être adaptées en fonction de l'utilisation des *QoS CAP* par les objets IoT et ce grâce à la fonction d'auto-optimisation que nous définissons dans la section 8.5. Par contre, dans le cas d'une modification concernant l'existence de classes de *QoS* (suppression ou ajout de *gSLA*), une mise à jour des valeurs de *BO* et de *SO* ainsi que de la configuration des *IT* est réalisée grâce à la fonction d'auto-configuration en se basant sur la base de connaissances du Tableau 8.1. Dans ce contexte, les nouvelles valeurs mises à jour de *BO*, *SO* et de la configuration des *IT* sont communiquées à la LL-Gw qui les envoie avec la balise *QBAIoT*, d'une façon périodique à chaque *BI*, à tous les objets IoT. En effet, la prise en compte de l'évolution du nombre de classes de trafics dans l'environnement IoT de manière autonome via la gestion des *gSLA* sur la *HL-Gw* correspond à la capacité d'auto-configuration de *QBAIoT*.

Le processus d'auto-configuration de *QBAIoT* est mis en œuvre via l'implémentation de la boucle de contrôle *MAPE-K* sur la *HL-Gw* et la *LL-Gw* de notre architecture IoT. En effet, la fonction « **Monitor** » de la *HL-Gw* permet de surveiller les modifications de l'environnement IoT en prenant en compte les différents *gSLA* stockés dans la base de connaissances au niveau de la *HL-Gw*. Pour chaque *LL-Gw*, la fonction de surveillance regroupe les *gSLA* correspondants. Par conséquent, les caractéristiques de l'environnement de la *LL-Gw* sont disponibles à travers le groupe de *gSLA* correspondant à cette passerelle. La fonction « **Analyse** » permet de retrouver le nombre de classes de *QoS* (*Nb_QoS_Classes*) et le nombre de *RTC* (*RT_Classes*) dans chaque groupe de *gSLA* relatif à une *LL-Gw* ; et elle retrouve également le nombre de *NRTC* (*NRT_Classes*). Ensuite, la fonction « **Plan** » permet de planifier la configuration correspondante en recherchant dans la base de connaissances l'entrée correspondante au nombre de *RTC* et *NRTC* récupérés par la fonction « **Analyse »**. Enfin, la fonction « **Execute »** extrait les valeurs de configuration de la base de connaissances et les envoie à l'interface de type « **Effector** » de la *LL-Gw*. Cette dernière met à jour les valeurs de configurations des *IT* afin de les envoyer dans la balise *QBAIoT* aux objets IoT. Cette boucle de contrôle fermée *MAPE-K* est appelée à chaque fois que l'*IoT-SP* apporte des modifications concernant l'environnement IoT. En effet, si un *gSLA* est ajouté, supprimé ou modifié, la boucle de contrôle est exécutée afin d'adapter la configuration de la supertrame *QBAIoT* au niveau de la *LL-Gw* sans intervention humaine. La Figure 8.2 illustre le processus d'auto-configuration *QBAIoT* déployé sur la passerelle *HL-Gw* avec les différentes fonctionnalités de la boucle *MAPE-K*.

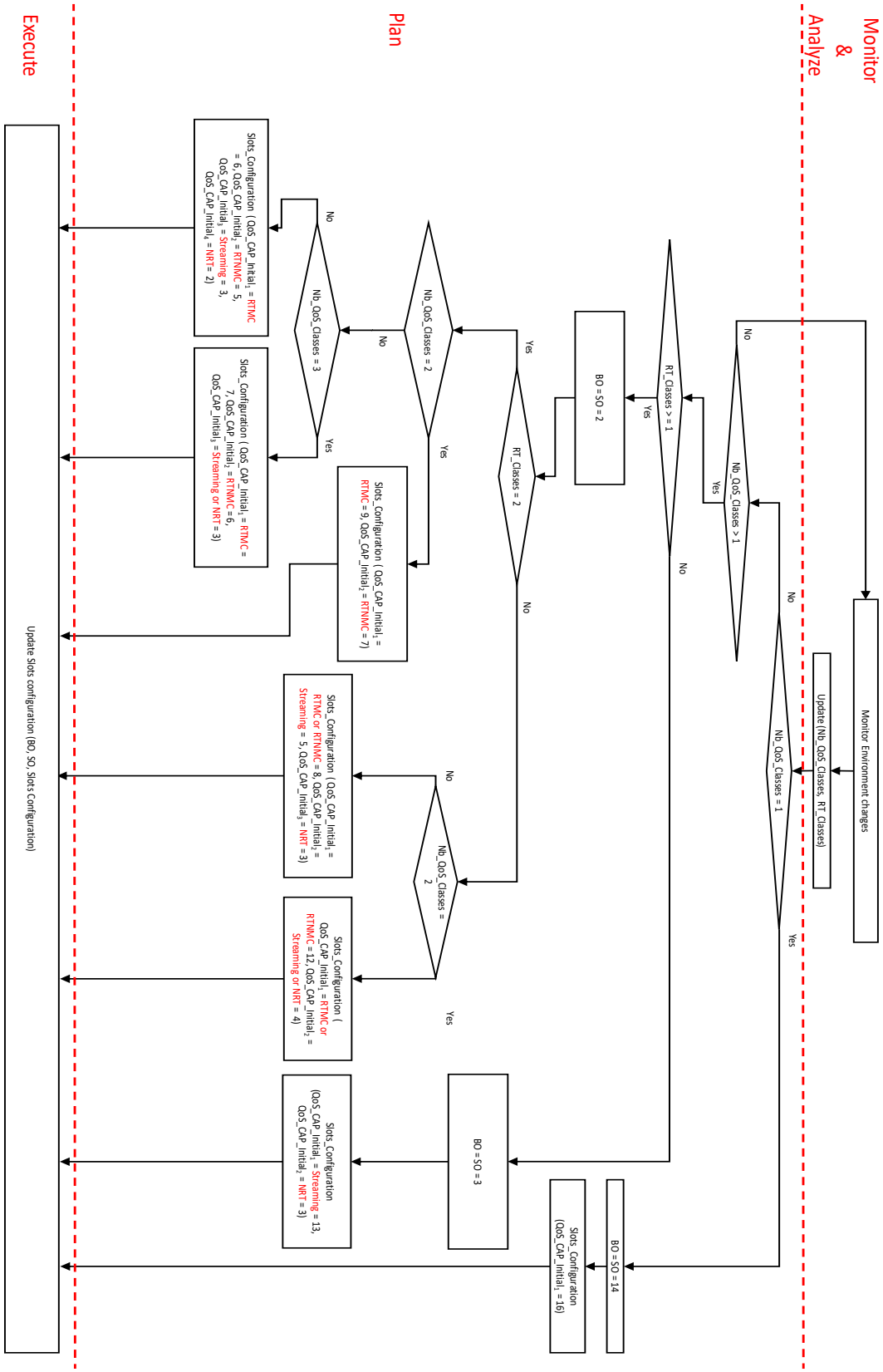

Figure 8.2 : Processus de l'auto-configuration de QBAIoT sur la HL-Gw

Ainsi, dans le cadre de l'auto-configuration *QBAIoT (*voir Figrue 8.2), la *HL-Gw* surveille les modifications de l'environnement IoT et met à jour les valeurs de *Nb_QoS_Classes* et *RT_Classes*. Lorsque *Nb_QoS_Classes* est égal à zéro, l'*IoT-SP* n'a pas de *gSLA* concernant la *LL-Gw*. Dans ce cas, aucune classe de *QoS* n'est disponible dans l'environnement de la *LL-Gw* et aucune configuration n'est nécessaire. Par contre, si *Nb_QoS_Classes* est égal à 1, il existe une seule classe de *QoS* dans l'environnement de la *LL-Gw*, alors *BO* et *SO* auront la valeur 14, et la seule *QoS CAP* existante dans la supertrame *QBAIoT* se vo*it* attribuer les 16 *IT*. S'il existe plus qu'une classe de *QoS* dans l'environnement IoT (i.e., *Nb_QoS_Classes* est supérieur à 1) alors la configuration dépend de l'existence des classes temps réel dans l'environnement de la *LL-Gw*. D'une part, si *RT_Classes* est égal à 0 alors il existe deux classes *NRTC*. Dans ce cas, *BO* et *SO* ont une valeur de 3 et la supertrame *QBAIoT* comprend deux *QoS CAP*. La première correspond au trafic *Streaming* et se vo*it* attribuer 13 *IT* alors que la deuxième *QoS CAP* correspond au trafic *NRT* et se vo*it* attribuer 3 IT. D'autre part, si *Nb_QoS_Classes* est supérieur à 1 et que *RT_Classes* est égal ou supérieur à 1, une boucle imbriquée est parcourue afin de permettre à la *HL-Gw* de choisir la configuration des *IT* correspondante pour l'environnement IoT considéré.

8.4.2. Design de l'auto-configuration de QBAIoT

Différents paramètres sont initialisés durant le processus d'auto-configuration *QBAIoT* pour choisir la configuration adaptée en fonction du nombre de classes de *QoS*. Certains de ces paramètres sont utilisés par la suite dans la fonction d'auto-optimisation pour adapter la configuration initiale suivant l'utilisation effective des *QoS CAP* en temps réel. Nous définissons ces paramètres dans le Tableau 8.2. Il est à noter que les paramètres soulignés sont initialisés par la fonction d'auto-configuration et utilisés par la fonction d'auto-optimisation *QBAIoT*.

Tableau 8.2 : Paramètres du processus d'auto-configuration QBAIoT

Paramètres	**Définitions**
Nb_QoS_Classes	Nombre de classes *QoS* existantes
RT_Classes	Nombre de classes temps réel existantes
BO / SO	Valeurs de *BO* et *SO*
Slots (QoS_CAP_Initial$_i$)	Nombre initial d'*IT* (i.e. Slots) affectés à *QoS CAP$_i$*
Minimal_QoS_CAP_Slots$_i$	Nombre minimal d'*IT* à affecter à *QoS CAP$_i$*
N$_i$_Reference	*QoS CAP(s)* à observer avant la réallocation des *IT* non utilisés
Slot_Duration	Durée d'un IT
Nb_Packet_Max$_i$	Nombre maximal de paquets pouvant être reçus durant la *QoS CAP$_i$*

Nous spécifions dans le *FSM* de la Figure 8.3 les différents états de la *HL-Gw* durant ce processus d'auto-configuration qui fait appel aux paramètres du Tableau 8.2 que nous spécifions grâce aux équations de 8.1 à 8.9 .

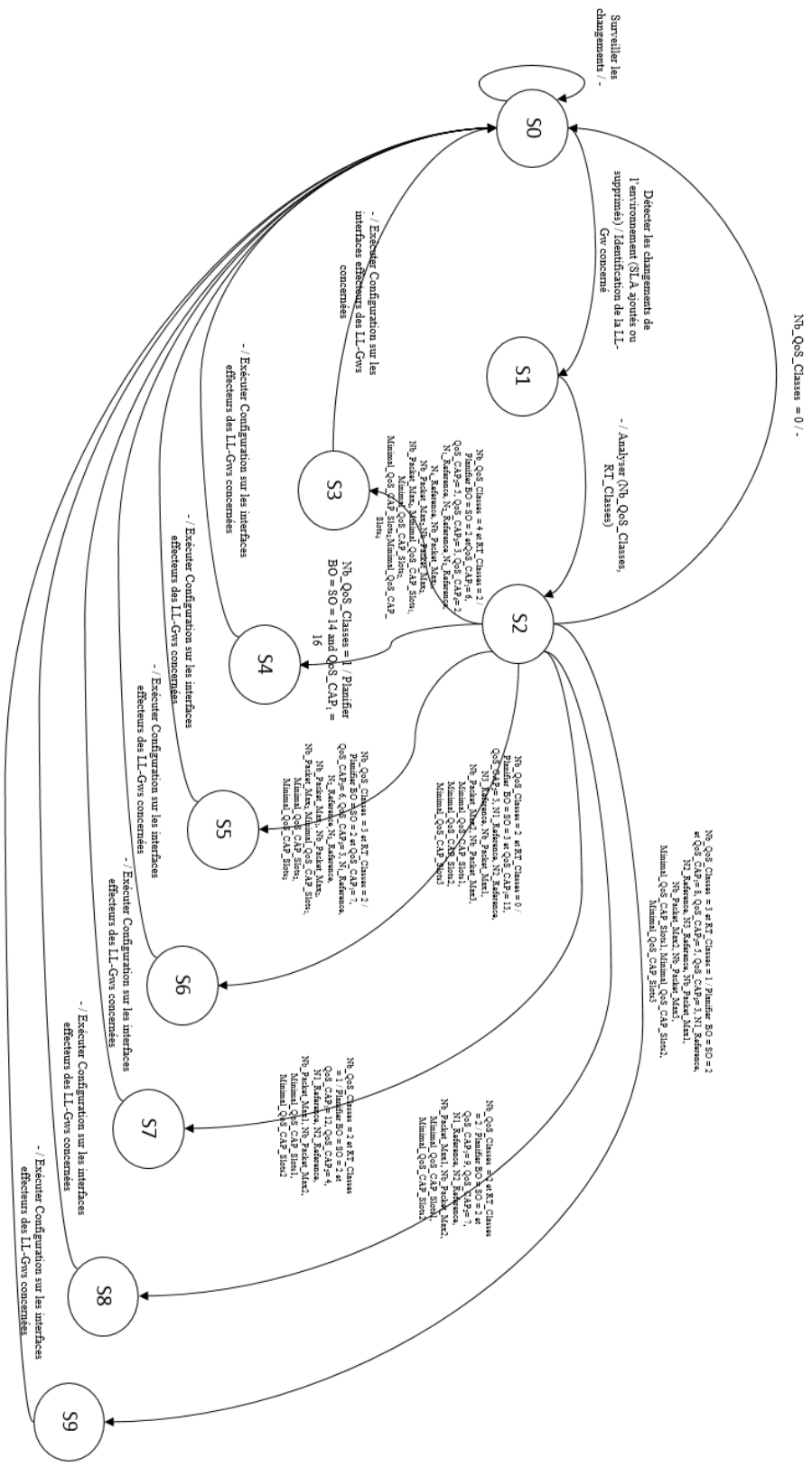

Figure 8.3 : FSM de la HL-Gw concernant l'auto-configuration QBAIoT

Ainsi, dans l'état S0, la *HL-Gw* est dans l'attente de changements au niveau de l'environnement IoT pour cela elle boucle sur l'état S0 tant qu'aucun changement ne survient dans l'environnement IoT. Elle passe à l'état S1 lorsqu'elle détecte l'ajout ou la suppression d'un *gSLA*. En passant à l'état S1, la *HL-Gw* identifie la *LL-Gw* concernée. Par la suite, la *HL-Gw* analyse le nombre de classes de *QoS* ainsi que le nombre de classes *RTC* et passe à l'état S2. Dans cet état S2, la *HL-Gw* se base sur le nombre de classes de *QoS* et le nombre de classes *RTC* pour passer vers l'état S0 si aucune classe de *QoS* n'existe (suite à la suppression de tous les *gSLA*) ou vers l'un des états de S3 à S9. Dans l'un de ces états (i.e., S3 à S9), la *HL-Gw* planifie la configuration à envoyer à l'interface de type effecteur de *la LL-Gw* concernée (*BO*, *SO*, *QoS CAP*$_i$, *N*$_i$*_Reference*, *Nb_Packet_Max*$_i$, *Minimal_QoS_CAP*$_i$). Ensuite, la *HL-Gw* repasse de l'un des états S3 à S9 vers l'état S0 en exécutant les configurations sur l'interface de type effecteur de la *LL-Gw*. Dans l'état S0, la *HL-Gw* surveille de nouveau l'environnement IoT pour détecter un éventuel changement.

Nous décrivons dans ce qui suit les méthodes de calcul des paramètres *QBAIoT* spécifiés dans le cadre de la fonction d'auto-configuration décrite par le *FSM* de la Figure 8.3 et que nous allons respecter dans le cadre de la fonction d'auto-optimisation. Ainsi, pour calculer le nombre minimal d'*IT* à affecter à une *QoS CAP*$_i$, nous utilisons l'équation 8.1 qui détermine la partie entière de la moitié du nombre initial d'*IT* affectés à la *QoS CAP*$_i$. Nous avons opté pour cette formule afin de laisser à une classe de trafics au moins la moitié des *IT* suite à une réaffectation durant le processus d'auto-optimisation.

$$Minimal\_QoS\_CAP\_Slots_i = \lfloor Slots\ (QoS\_CAP\_Initial_i) / 2 \rfloor \qquad (8.1)$$

Le compteur du nombre de *QoS CAP* à observer avant la réaffectation des *IT* (i.e., *N*$_i$*_Reference*) est calculé en utilisant l'équation 8.2. Ainsi, nous multiplions par 10 le nombre d'*IT* affectés initialement à la *QoS CAP* en question pour déterminer la valeur de ce compteur. Notre choix de la valeur 10 est aléatoire et peux être optimiser suivant les cas de figures et les scénarios dans les travaux futurs. Cette valeur de compteur nous permet d'observer suffisamment de temps l'utilisation des *IT* dans les *QoS CAP* avant de prendre une décision relative à leur réaffectation. Ainsi, nous minimisons le nombre de réaffectations inutiles pouvant se produire durant le déroulement du processus d'auto-optimisation.

$$N_i\_Reference = 10 * Slots\ (QoS\_CAP\_Initial_i) \qquad (8.2)$$

La durée de chaque *IT* (i.e., *Slot_Duration*) est utilisée par le processus d'auto-optimisation pour évaluer la durée de chaque *QoS CAP*. Nous utilisons l'équation 8.3 pour calculer en secondes cette durée et ce en divisant la durée de la supertrame *QBAIoT* par 16 (i.e., nombre d'*IT* formant une supertrame).

$$Slot\_Duration = [(aBaseSuperframeDuration * 2^{SO}) / Bit\_Rate] / 16 \qquad (8.3)$$

Le nombre maximal de paquets pouvant être reçus par chaque *QoS CAP*$_i$ (i.e., *Nb_Packet_Max*$_i$) est calculé en divisant la durée de la *QoS CAP* par la durée globale d'un paquet (voir Equation 8.4). Afin de calculer la durée globale d'un paquet (i.e., *Packet_Time*), nous utilisons

l'équation 8.5 qui prend en compte le temps de transmission du paquet (i.e., $Packet_{transmission}$) le temps d'attente avant envoi de l'accusé de réception (t_{ack}), le temps de transmission de l'accusé de réception ($ACK_{transmission}$) et l'espacement de trame après la réception de l'*ACK* et avant envoi du prochain paquet tel que défini dans le standard IEEE 802.15.4 (*LIFS* : Long Inter-Frame Spacing).

$$Nb\_Packet\_Max_x = Slot\_Duration * Slots\ (Actual\_QoS\_CAP) / Packet\_Time \quad (8.4)$$

$$Packet\_Time = Packet_{transmission} + t_{ack} + ACK_{transmission} + LIFS \quad (8.5)$$

Le temps de transmission du paquet vers le coordinateur (voir équation 8.6) est calculé en divisant la taille du paquet comportant la charge utile et l'entête (i.e., *Packet_Size*) par le débit binaire. De plus, nous utilisons l'équation 8.7 pour calculer le temps d'attente avant envoi de l'accusé de réception, à savoir t_{ack}, qui correspond au temps *SIFS* (Short Inter-Frame Spacing) tel que défini dans le standard IEEE 802.15.4. Dans ce contexte, la variable *macSIFS* spécifie une période de 12 symboles de 4 bits.

$$Packet_{transmission} = Packet\_Size / Bit\_Rate \quad (8.6)$$

$$t_{ack} = SIFS = macSIFS / Bit\_Rate \quad (8.7)$$

Le temps de transmission de l'accusé de réception ($ACK_{transmission}$) est calculé en divisant la taille de l'acquittement (i.e., *ACK_Size*) par le débit (voir équation 8.8). Enfin, nous utilisons l'équation 8.9 pour calculer la valeur de la variable *LIFS*. Pour ce faire, nous prenons en considération une période *macLIFS* de 40 symboles de 4 bits tel que défini dans le standard IEEE 802.15.4.

$$ACK_{transmission} = ACK\_Size / Bit\_Rate \quad (8.8)$$

$$LIFS = macLIFS / Bit\_Rate \quad (8.9)$$

8.4.3. Algorithme d'auto-configuration QBAIoT

L'algorithme d'auto-configuration de *QBAIoT* (voir Figure 8.4) permet de choisir les configurations initiales des *IT* suivant les caractéristiques de l'environnement IoT, en termes de nombre de classes de *QoS* existantes, mais aussi les nouvelles configurations d'une façon autonome suite à un changement induisant la modification de ce nombre de classes de *QoS* . De plus, l'algorithme d'auto-configuration permet d'initialiser différents paramètres qui seront ensuite utilisés dans la fonction d'auto-optimisation présentée dans la section suivante (i.e., durée des *IT*, compteur de référence, nombre de paquet à recevoir durant les *QoS CAP*, nombre minimal d'*IT* à affecter à une *QoS CAP*, etc.).

L'algorithme d'auto-configuration *QBAIoT* spécifié dans la Figure 8.4 permet de récupérer les configurations à appliquer depuis la base de connaissances (i.e. *Config_table[]*) en utilisant la fonction *Retrieve_from_Knowledge_Base()*. Les entrées de cette base de connaissances ont été représentées en partie dans le Tableau 8.1. Par exemple, la première entrée de cette base de connaissances correspondant à *Config_table[1]* dans l'algorithme spécifie le cas où une seule classe

de *QoS* existe dans l'environnement IoT. Il est à noter que l'algorithme d'auto-configuration fait appel à la fonction *Update_Configuration_on_effectors()* afin de communiquer les paramètres de configuration à la passerelle de bas niveau *LL-Gw* à travers l'interface de type effecteur implémentée sur cette dernière. L'algorithme d'auto-configuration *QBAIoT* avec des commentaires détaillés est présent dans l'annexe 1.

```
Algorithme  Auto-configuration de QBAIoT au niveau de la HL-Gw
1:  While (true)
2:    if (Environment_Changes = True) then
3:      if (Nb_QoS_Classes = 1) then
4:        Retrieve_from_Knowledge_Base(Config_table[1])
5:        Environment_Changes ← False & Update_Configuration_on_effectors
6:      else
7:        if (Nb_QoS_Classes < 1) then
8:            Environment_Changes ← False
9:        else
10:           if (RT_Classes = 0) then
11:               Retrieve_from_Knowledge_Base(Config_table[2])
12:               Environment_Changes ← False & Update_Configuration_on_effectors
13:           else
14:               if (RT_Classes = 1) then
15:                   if (Nb_QoS_Classes = 2) then
16:                       Retrieve_from_Knowledge_Base(Config_table[4])
17:                       Environment_Changes ← False & Update_Configuration_on_effectors
18:                   else
19:                       Retrieve_from_Knowledge_Base(Config_table[5])
20:                       Environment_Changes ← False & Update_Configuration_on_effectors
21:                   end if
22:               else
23:                   if (Nb_QoS_Classes = 4) then
24:                       Retrieve_from_Knowledge_Base(Config_table[7])
25:                       Environment_Changes ← False & Update_Configuration_on_effectors
26:                   else
27:                       if (Nb_QoS_Classes = 3) then
28:                           Retrieve_from_Knowledge_Base(Config_table[6])
29:                           Environment_Changes ← False & Update_Configuration_on_effectors
30:                       else
31:                           Retrieve_from_Knowledge_Base(Config_table[3])
32:                           Environment_Changes ← False & Update_Configuration_on_effectors
33:                       end if
34:                   end if
35:               end if
36:           end if
37:       end if
38:     end if
39:   end if
40: end while
```

Figure 8.4 : Algorithme d'auto-configuration QBAIoT

8.5. Auto-optimisation de QBAIoT

8.5.1. Principe de fonctionnement

La charge de trafic existante dans chaque classe de *QoS* de l'environnement IoT influence l'utilisation des ressources affectées à cette classe (i.e., *IT* affectés à chaque classe). Afin de maximiser l'utilisation des ressources tout en respectant les besoins de *QoS* des classes prioritaires, nous proposons une fonction d'auto-optimisation de *QBAIoT* permettant de réaffecter les *IT* non

utilisés par une classe de *QoS* aux classes qui en ont besoin. Cette fonction d'auto-optimisation est implémentée à travers la boucle *MAPE-K* sur la *HL-Gw* et la *LL-Gw*.

Pour ce faire, la *HL-Gw* va assurer les 4 fonctionnalités de la boucle de contrôle fermée : surveiller (i.e., Monitor), analyser (i.e., Analyse), planifier (i.e., Plan) et exécuter (i.e., Execute) en utilisant la base de connaissances (i.e., Knowledge base). Tout d'abord, la fonctionnalité « **Monitor** » reçoit les informations d'environnement relatives à l'utilisation des *IT* en provenance des interfaces de type capteurs (i.e., sensors) implémentées sur les éléments gérés (i.e., *LL-Gw*). Ces interfaces de type capteurs envoient les informations concernant le nombre de paquets reçus durant chaque *QoS CAP* (relative à une classe de *QoS*) des supertrames *QBAIoT*. Ainsi, la *HL-Gw* surveille le système en recevant l'évolution de l'utilisation des *IT* pour chaque *QoS CAP* dans chaque supertrame. Deuxièmement, la fonctionnalité « **Analyse »** traite les données obtenues à partir de la fonction « **Monitor** » et décide si des modifications doivent être appliquées à la configuration des *QoS CAP* des prochaines supertrames. La décision est basée sur l'évolution de l'utilisation des *IT* mais aussi les informations existantes dans la base de connaissances qui stocke différents ensembles de scénarios et de règles. Troisièmement, la fonctionnalité « **Plan »**, se base sur la décision reçue de la fonction « **Analyse »** pour déterminer les valeurs de configuration des *IT*, de *BO* et de *SO*, ainsi que d'autres valeurs (telles que le nombre d'*IT* non utilisés, le nombre de *QoS CAP* non utilisées en entier, etc.) nécessaires pour une éventuelle réaffectation des *IT* dans les *QoS CAP*. Enfin, la fonctionnalité « **Execute »** envoie les actions à appliquer à l'interface de type effecteur. Cette interface de type effecteur implémentée sur l'élément géré (i.e., *LL-Gw*), applique, dans le cadre du processus d'auto-optimisation *QBAIoT*, la reconfiguration des *IT* si leur utilisation n'est pas effective dans plusieurs supertrames. La base de connaissances est implémentée sur la *HL-Gw* et comprend entre autres plusieurs paramètres présentés dans les Tableaux 8.2 et 8.3.

Tableau 8.3 : Paramètres du processus d'auto-optimisation QBAIoT

Paramètres	Définitions
Slots (Actual_QoS_CAP$_i$)	Nombre d'*IT* actuel affectés à *QoS CAP$_i$*
N$_i$	Nombre de *QoS CAP$_i$* observées avec des *IT* non utilisés
Nb_Packet$_i$	Nombre de paquets reçus durant *QoS CAP$_i$*
Nb_Packet_QoS_Classes	Nombre de paquets reçus durant toutes les *QoS CAP*
Packet_Time	Durée globale d'acheminement d'un paquet

Pour assurer la fonction d'auto-optimisation au niveau de la *HL-Gw*, nous implémentons un processus spécifique (voir la Figure 8.5) qui surveille l'utilisation des *IT* et évalue la nécessité de procéder à une réallocation des *IT* entre les *QoS CAP*. Dans ce contexte, ce processus utilise l'équation 8.10 afin de déduire le nombre de slots non utilisés (*Slots_Not_Used*) et décider par la suite d'une éventuelle réallocation de ces *IT* non utilisés afin de maximiser l'utilisation des ressources au niveau des différentes supertrames.

$$Slots\_Not\_Used = (Nb\_Packet\_Max_x - Nb\_Packet_x) * Packet\_Time / Slot\_Duration \quad (8.10)$$

La Figure 8.5 spécifie les détails du processus d'auto-optimisation *QBAIoT* implémenté au niveau de la *HL-Gw*. Différents paramètres existants dans la base de connaissance sont utilisés dans ce processus et sont définis dans les Tableaux 8.2 et 8.3. De même, nous définissons dans le Tableau 8.4 des paramètres additionnels ainsi que différentes fonctions que nous utilisons dans la spécification du processus d'auto-optimisation.

Tableau 8.4 : Paramètres et fonctions du processus d'auto-optimisation QBAIoT

Paramètres/ Fonctions	Définitions
i	Compteur pour parcourir les différentes *QoS_CAP*$_i$ existantes
Receiving_Class	La classe de *QoS* qui reçoit les *IT* réalloués de la classe de *QoS* qui n'utilise pas tous ses IT
Serving_Class	La classe de *QoS* qui fournit les *IT* à réallouer à la classe de *QoS* qui en a besoin
Lowest_Priority()	Fonction qui permet de déterminer la classe de *QoS* la moins prioritaire ayant plus d'*IT* que sa configuration initiale
Update(Slots_Configuration)	Fonction de mise à jour des configurations des *IT* des *QoS_CAP*$_i$ sur l'interface de type effecteur de la *LL-Gw*
Candidate []	Table contenant la liste des candidats (i.e., classe de *QoS*) qui sont éligibles à la réallocation des *IT* non utilisées par d'autres classes *QoS*. Cette table est classée selon la priorité des classes en utilisant les fonctions Sort() et Reorder()
ind	Indice de la table *Candidat []*
Slots_Not_Used ()	Fonction permettant de déterminer le nombre d'*IT* non utilisés par la *QoS_CAP*$_i$ selon l'équation 8.10
Slots_Not_Used	Valeur retournée par la fonction Slots_Not_Used ()
QoS_CAP_Not_Used$_i$[]	Table contenant les différentes valeurs du nombre d'*IT* non utilisés dans les différentes supertrames observées pour chaque *QoS_CAP*$_i$. La taille maximale de cette table correspond à la valeur N$_i$_Reference
L$_i$ []	Table permettant de préserver un *IT* non utilisé pour les différentes *QoS_CAP*$_i$ lors de la réallocation

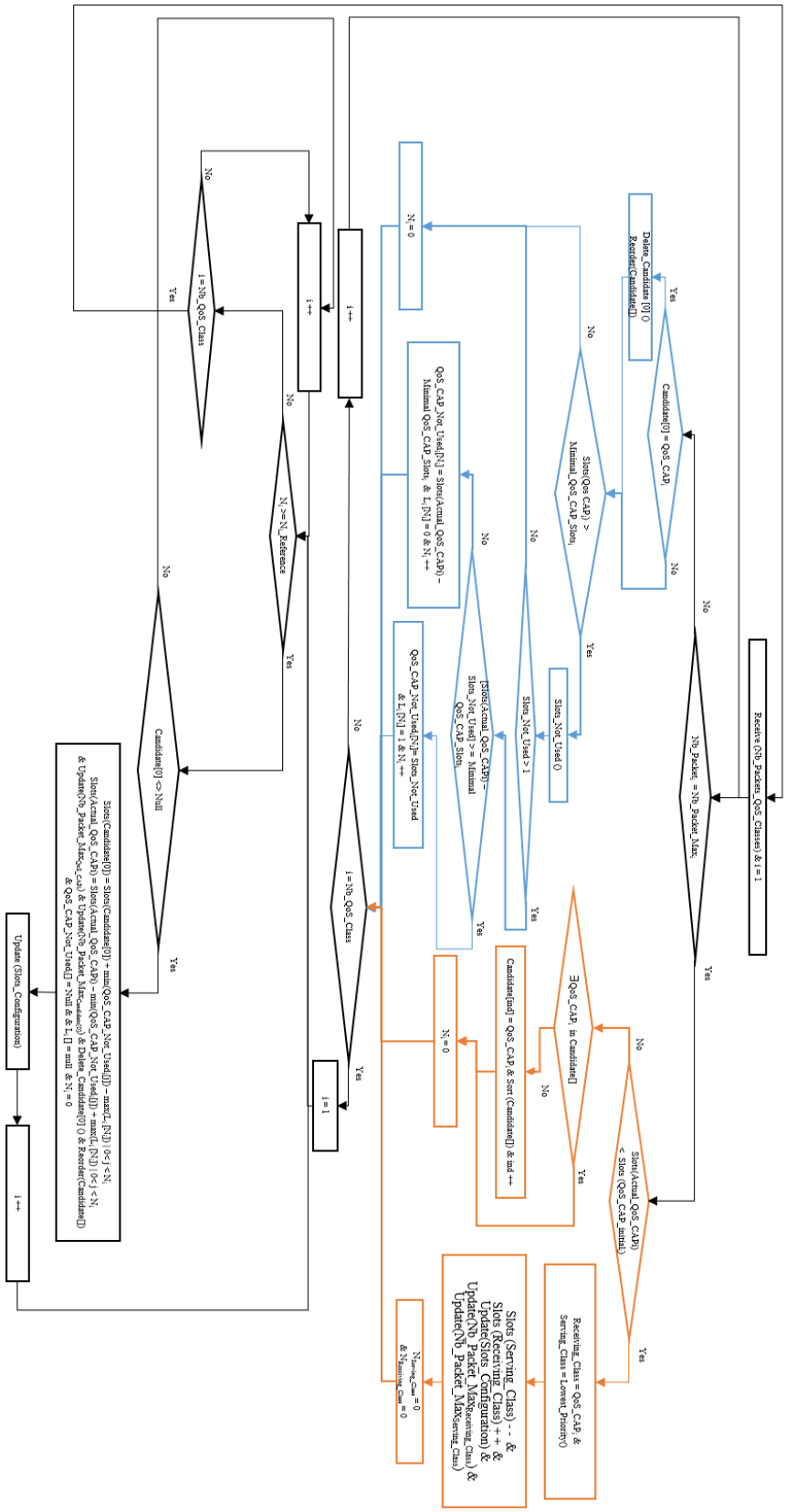

Figure 8.5 : Processus d'auto-optimisation de QBAIoT sur la HL-Gw

Comme le montre la Figure 8.5, la *HL-Gw* initie le processus d'auto-optimisation en récupérant de la *LL-Gw* le nombre de paquets reçus au cours de chaque *QoS CAP*. Ensuite, la *HL-Gw* compare le nombre de paquets reçus à la valeur maximale calculée pour chaque *QoS CAP*. D'une part, si le nombre de paquets reçus est égal au maximum, la *HL-Gw* teste si la *QoS CAP* a déjà cédé certains de ses *IT* (Partie orange sur la Figure 8.5). Si tel est le cas, la *QoS CAP* reprend un seul *IT* de la *QoS CAP* ayant la priorité la plus basse et ayant plus d'*IT* que dans sa configuration initiale. Si ce n'est pas le cas, la *QoS CAP* sera identifiée comme candidate à la réallocation d'*IT* et elle sera ajoutée à la table contenant la liste des candidats. D'autre part, si le nombre de paquets reçus est inférieur au maximum (Partie bleu sur Figure 8.5), la *HL-Gw* vérifie d'abord que la *QoS CAP* correspondante n'est pas étiquetée comme candidate. Ensuite, le nombre d'*IT* de la *QoS CAP* est comparé au nombre minimal d'*IT* devant être alloués pour s'assurer que le nombre d'*IT* restants est conforme aux exigences minimales de la classe de QoS. Si le nombre actuel des *IT* correspond au minimum de la *QoS CAP*, tous les *IT* disponibles pour la réallocation ont été déjà réaffectés et la *QoS CAP* ne peut plus céder d'autres *IT*. Par contre, s'il y a des *IT* disponibles pour la réallocation dans la *QoS CAP*, le nombre des *IT* non utilisés est déterminé à travers la fonction (*Slots_Not_Used()*). Dans ce cas, s'il y a plus d'un seul *IT* inutilisé, la *HL-Gw* détermine le nombre d'*IT* pouvant être réaffectés. Ensuite, la valeur N_i est comparée à une valeur de référence (i.e., *Ni_Reference*). Si N_i est égal ou supérieur à la valeur de référence, la *HL-Gw* doit réaffecter les *IT* à la *QoS CAP* candidate et prioritaire. Enfin, à l'issue d'une réallocation, la *HL-Gw* envoie les nouvelles configurations à la *LL-Gw* qui va les renseigner dans la balise *QBAIoT* et les transmettre à tous les objets IoT de son environnement.

8.5.2. Design de l'auto-optimisation QBAIoT

Après la définition des différents paramètres et le principe de fonctionnement de l'auto-optimisation *QBAIoT*, nous spécifions dans le *FSM* de la Figure 8.6 les différents états de la HL-Gw durant ce processus d'auto-optimisation. Ainsi, dans l'état S0, la *HL-Gw* est dans l'attente du nombre de paquets reçus durant les *QoS CAP* qui lui permet de passer à l'état S1 afin d'analyser les valeurs reçues. Pour chaque *QoS CAP*, la HL-Gw compare le nombre de paquets reçus au nombre maximal de paquets pouvant être reçus.

Premièrement, si le nombre de paquets reçus a atteint le maximum, la *HL-Gw* passe à l'état S2. Dans cet état, la *HL-Gw* évalue le nombre d'*IT* affectés à chaque *QoS CAP*. D'une part, si le nombre d'*IT* affectés est inférieur à la configuration initiale, la *HL-Gw* réattribue les *IT* déjà cédés, remet les compteurs N_i à zéro des *QoS CAP* qui ont cédé (i.e., *Serving_Class* du Tableau 8.4) ou reçu (i.e., *Receiving_Class* du Tableau 8.4) des *IT*, réinitialise le nombre de paquets maximal et repasse à l'état S1 pour évaluer l'utilisation des ressources par une nouvelle classe de *QoS* (i.e., *QoS CAP* suivante). D'autre part, si le nombre d'*IT* affectés est supérieur ou égal à la configuration initiale, la *QoS CAP* sera identifiée comme candidate. Par conséquent, la liste des candidats est réordonnée selon les priorités des classes de *QoS* correspondantes, le compteur N_i de la classe considéré sera mis à 0 et la *HL-Gw* repasse à l'état S1 pour évaluer la *QoS CAP* suivante.

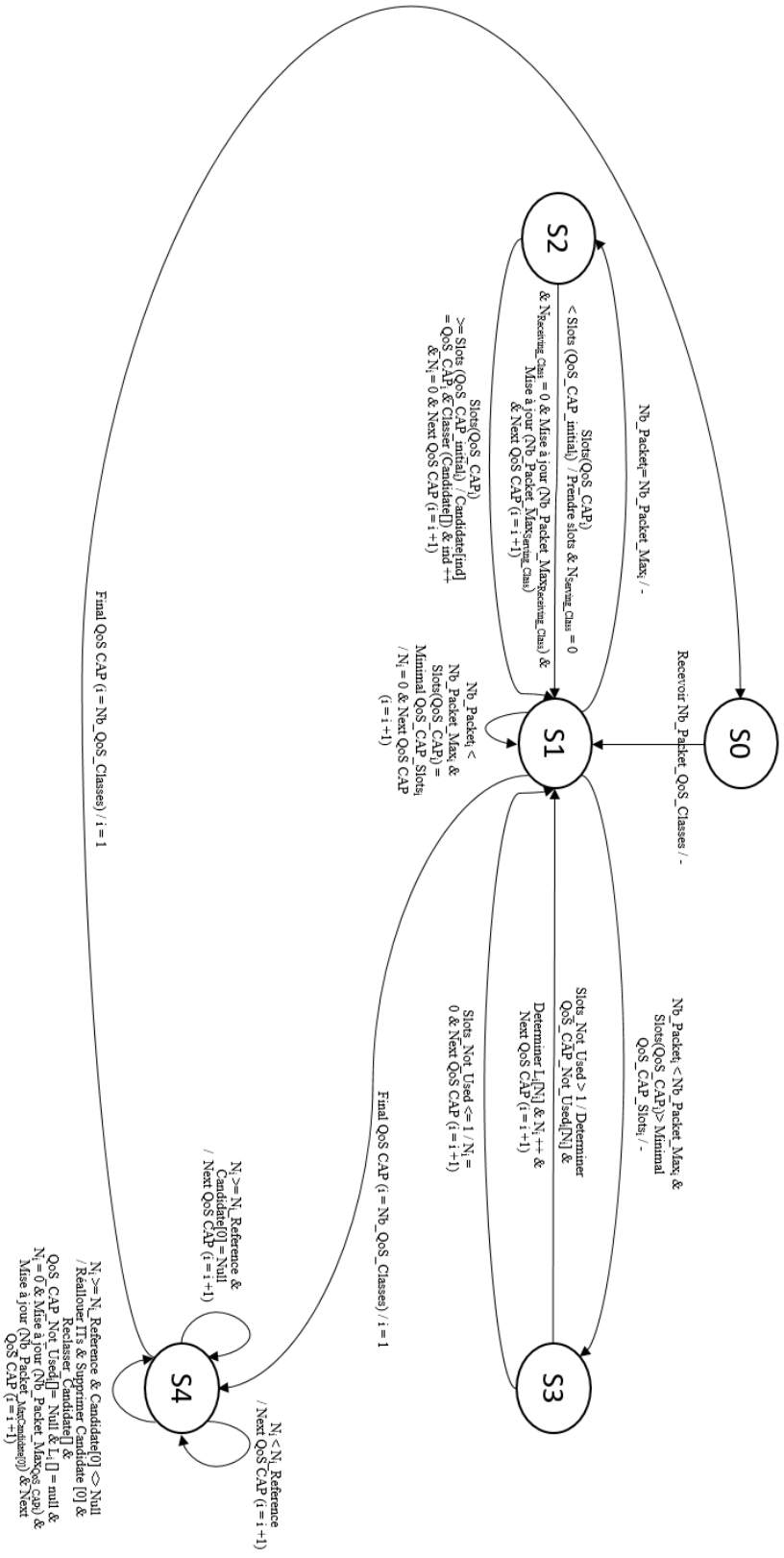

Figure 8.6 : FSM de la HL-Gw concernant l'auto-optimisation QBAIoT

Deuxièmement, si le nombre de paquets reçus n'a pas atteint le maximum dans la *QoS CAP* et que le nombre d'*IT* affectés est supérieur au nombre minimal d'*IT* à affecter à la classe de *QoS*, la *HL-Gw* passe à l'état S3. D'une part, si le nombre d'*IT* non utilisés est supérieur à 1, la *HL-Gw* détermine le nombre d'*IT* à réaffecter, met à jour les variables (i.e., $QoS\_CAP\_Not\_Used_i[]$ et $L_i[]$) et passe à la prochaine *QoS CAP* en revenant vers l'état S1. D'autre part, si le nombre d'*IT* non utilisés est inférieur ou égal à 1, le compteur N_i de la *QoS* de classe est remis à 0 et la *HL-Gw* passe vers la prochaine *QoS CAP* en revenant à l'état S1.

Troisièmement, si le nombre de paquets reçus n'a pas atteint le maximum dans la *QoS CAP* et que le nombre d'*IT* affectés à la *QoS CAP* est égal au nombre minimal d'*IT* de la classe de QoS, le compteur N_i de la classe *QoS* est remis à 0 et la *HL-Gw* passe à la *QoS CAP* suivante en bouclant sur S1. Dans cet état S1, si toutes les *QoS CAP* ont été parcourues, la *HL-Gw* passe à l'état S4 en mettant le compteur *i* des *QoS CAP* à 1 afin de tester les compteurs N_i de toutes les *QoS CAP*. D'une part, si N_i est inférieur à la valeur de référence (i.e., N_i *_Reference* du Tableau 8.2), la *HL-Gw* passe à la *QoS CAP* suivante en incrémentant le compteur *i*. D'autre part, si la valeur de N_i est supérieure ou égale à la valeur de référence et que la liste des candidats est nulle, la *HL-Gw* incrémente le compteur *i* et passe à la *QoS CAP* suivante. Par contre, si la valeur de N_i est supérieure ou égale à la valeur de référence et que la liste des candidats n'est pas nulle, la *HL-Gw* réalloue les *IT* à la classe candidate (i.e., la classe la plus prioritaire ayant besoin d'*IT* supplémentaire), supprime le candidat de la liste, réinitialise les listes et tableaux (i.e., $QoS\_CAP\_Not\_Used_i[]$ et $L_i[]$) de la classe réceptrice d'*IT* et met à jour le nombre de paquets maximal pouvant être reçus dans la classe réceptrice et émettrice d'*IT* avant de passer à la *QoS CAP* suivante. Enfin, si dans l'état S4 toutes les *QoS CAP* ont été parcourues, la *HL-Gw* repasse à l'état S0 pour attendre à nouveau la réception des nombres de paquets reçus par *QoS CAP* tout en réinitialisant le compteur *i* à 1.

8.5.3. Algorithme d'auto-optimisation QBAIoT

Nous spécifions dans la Figure 8.7 l'algorithme d'auto-optimisation de *QBAIoT* permettant d'optimiser d'une façon autonome, sans intervention de l'administrateur et en temps réel les configurations initiales des *QoS CAP de QBAIoT* selon l'utilisation effective des ressources par les classes de *QoS* correspondantes dans l'environnement IoT de la passerelle de bas niveau *LL-Gw*. Cette auto-optimisation est rendu possible grâce à une réallocation dynamique des *IT* non utilisés aux classes de *QoS* les plus prioritaires qui en ont besoin tel que décr*it* dans l'algorithme d'auto-optimisation (voir Figure 8.7) implémenté au niveau de la passerelle de haut niveau *HL-Gw*.

L'algorithme d'auto-optimisation *QBAIoT* de la Figure 8.7 utilise différents paramètres et fonctions décrit dans les Tableaux 8.2, 8.3 et 8.4 mais aussi les fonctions *min()* et *max()* qui permettent de retrouver la valeur minimale du tableau $QoS\_CAP\_Not\_Used_i[]$ et la valeur maximale du tableau $L_i[]$.

Algorithme Auto-optimisation de QBAIoT au niveau de la HL-Gw

```
1:  Receive_from_Sensor_LL-Gw (Nb_Packet_QoS_Classes)
2:  for i ∈ [1, Nb_QoS_Classes] do
3:      if (Nb_Packet_i - Nb_Packet_Max_i) then
4:          if (Slots(Actual_QoS_CAPi) < Slots(QoS_CAP_Initial_i)) then
5:              Receiving_QoS_Class ← QoS_CAP_i
6:              Serving_QoS_Class ← Lowest_Priority ()
7:              Slots(Actual_Receiving_Class) ← Slots(Actual_Receiving_Class) + 1
8:              Slots(Actual_Serving_Class) ← Slots(Actual_Serving_Class) - 1
9:              Update(Slots_Configuration) & Update_Configuration_on_effector
10:             Nb_Packet_Max_Receiving_Class ← Slot_Duration * Slots(Actual_Receiving_Class) / Packet_Time
11:             Nb_Packet_Max_Serving_Class ← Slot_Duration * Slots(Actual_Serving_Class) / Packet_Time
12:             N_Receiving_Class ← 0
13:             N_Serving_Class ← 0
14:         else
15:             QoS_CAP_Existence ← false
16:             for j ∈ [0, ind] do
17:                 if (Candidate[j] = QoS_CAP_i) then
18:                     QoS_CAP_Existence ← true
19:                 end if
20:             end for
21:             if (QoS_CAP_Existence = true) then
22:                 N_i ← 0
23:             else
24:                 Candidate[ind] ← QoS_CAP_i
25:                 Sort (Candidate [])
26:                 N_i ← 0
27:                 ind ← ind + 1
28:             end if
29:         end if
30:     else
31:         if (Candidate[0] = QoS_CAP_i) then
32:             for j ∈ [0, ind] do
33:                 if (j < ind) then
34:                     Candidate [j] ← Candidate [j+1]
35:                 else
36:                     Candidate [j] ← Null
37:                 end if
38:             end for
39:             ind ← ind - 1
40:         end if
41:         if (Slots(Actual_QoS_CAPi) > Minimal_QoS_CAP_Slots_i) then
42:             Slots_Not_Used ← Slots_Not_Used (QoS_CAP_i , Nb_Packet_i)
43:             if (Slots_Not_Used > 1) then
44:                 if (Slots(Actual_QoS_CAPi) - Slots_Not_Used >= Minimal_QoS_CAP_Slots_i) then
45                      QoS_CAP_Not_Used_i[N_i] ← Slots_Not_Used
46:                     L_i [N_i] ← 1
47:                     N_i ← N_i + 1
48:                 else
49:                     QoS_CAP_Not_Used_i[N_i]←Slots(Actual_QoS_CAPi) - Minimal_QoS_CAP_Slots_i
50:                     L_i [N_i] ← 0
51:                     N_i ← N_i + 1
52:                 end if
53:             else
54:                 N_i ← 0
```

```
55:                         end if
56:                 else
57:                         Ni ← 0
58:                 end if
59:             end if
60:         end for
61:         for i ∈ [1,Nb_QoS_Classes] do
62:             if (Ni >= Ni_Reference) then
63:                 if (Candidate[0] <> Null) then
64:                         Min ← min(QoS_CAP_Not_Usedi[])
65:                         Max ← max(Li [])
66:                         Slots(Candidate[0]) ← Slots(Candidate[0]) + Min – Max
67:                         Slots(Actual_QoS_CAPi) ← Slots(Actual_QoS_CAPi) – Min + Max
68:                         Nb_Packet_Max QoS_CAPi ← Slot_Duration * Slots(Actual_QoS_CAPi) / Packet_Time
69:                         Nb_Packet_Max Candidate[0] ← Slot_Duration * Slots(Candidate[0]) / Packet_Time
70:                         for j ∈ [0, ind] do
71:                                 if (j < ind) then
72:                                         Candidate [j] ← Candidate [j+1]
73:                                 else
74:                                         Candidate [j] ← Null
75:                                 end if
76:                         end for
77:                         ind ← ind - 1
78:                         for j ∈ [0, Ni] do
79:                                 QoS_CAP_Not_Usedi[j] ← Null
80:                                 Li[j] ← Null
81:                         end for
82:                         Ni ← 0
83:                         Update(Slots_Configuration) & Update_Configuration_on_effector
84:                 end if
85:             end if
86:         end for
```

Figure 8.7 : Algorithme d'auto-optimisation QBAIoT

8.6. Gestion autonome globale de QBAIoT

8.6.1. Design de la gestion autonome globale de QBAIoT

Pour assurer une gestion autonome globale de *QBAIoT*, nous associons les deux fonctions d'auto-configuration et d'auto-optimisation afin d'adapter *QBAIoT*, non seulement aux changements du nombre de classes de *QoS* dans l'environnement IoT de la passerelle *LL-Gw* suite à l'évolution des contrats de type *gSLA* au niveau de la *HL-Gw*, mais aussi adapter *QBAIoT* à l'évolution de l'utilisation effective des ressources allouées à chaque classe de service (i.e., *IT* par *QoS CAP*). Pour ce faire, nous commençons par spécifier dans le *FSM* de la Figure 8.8 les différents états de la *HL-Gw* durant le processus de gestion autonome globale de *QBAIoT* relative à une *LL-Gw* particulière.

Ainsi, dans l'état S0, la *HL-Gw* est dans l'attente de l'établissement du premier *iSLA* induisant un ajout d'un *gSLA* à son niveau afin de passer à l'état S1 en lançant le processus d'auto-configuration. Dans l'état S1, la *HL-Gw* récupère via l'interface capteur de la *LL-Gw* le nombre de paquets reçus au niveau de chaque *QoS CAP* et passe à l'état S2 en lançant le processus d'auto-

optimisation. Dans l'état S2, la *HL-Gw* relance le processus d'auto-optimisation à chaque collecte du nombre de paquets reçus de la part de la *LL-Gw* tout en restant dans l'état S2 si aucun changement de l'environnement IoT n'a été détecté en termes de *gSLA*. Par contre, dans l'état S2, la *HL-Gw* peut passer à l'état S1 si un *gSLA* a été supprimé et que le nombre de *gSLA* concernant la *LL-Gw* en question n'est pas nul ou encore si un *gSLA* a été ajouté. Durant cette transition de l'état S1 à l'état S2, le processus d'auto-configuration *QBAIoT* est appelé de nouveau. Enfin, dans l'état S2, la suppression au niveau de la *HL-Gw* du dernier *gSLA* relatif à une *LL-Gw* fait passer la *HL-Gw* à l'état S0 avec l'envoi de l'information (*gSLA_existence* = False) à la *LL-Gw* concerné via l'interface de type effecteur.

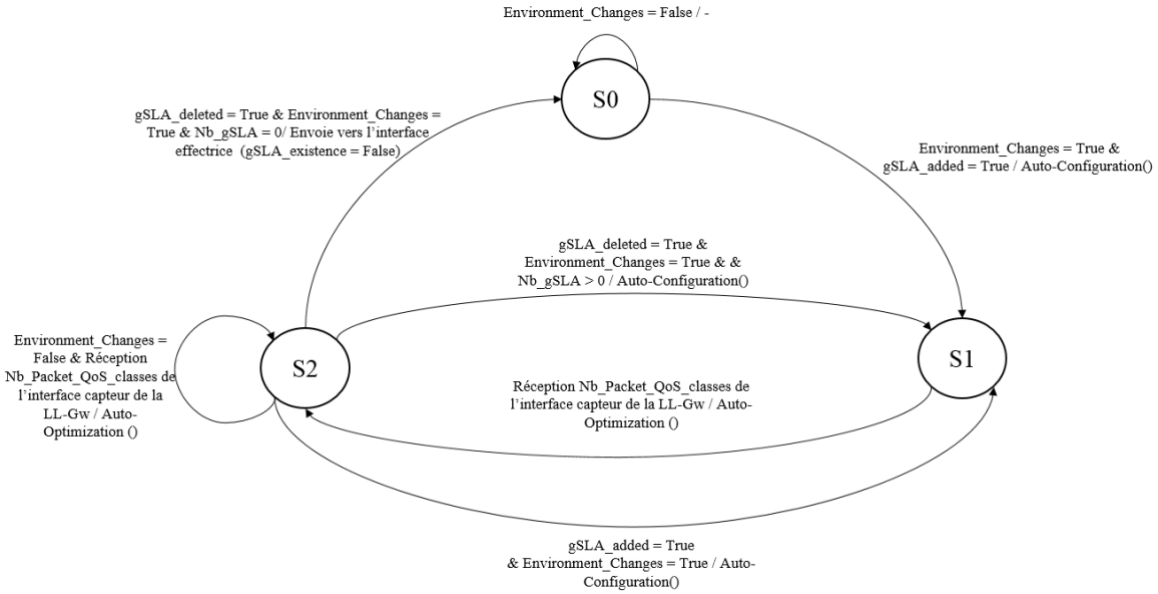

Figure 8.8 : Machine à états finis de la HL-Gw concernant la gestion autonome

8.6.2. Algorithme de la gestion autonome globale de QBAIoT

Dans le cadre de la gestion autonome globale *QBAIoT*, nous proposons un algorithme au niveau de la *HL-Gw* qui fait appel aux deux fonctions d'auto-configuration et d'auto-optimisation *QBAIoT* (voir Figure 8.9).

Algorithme Gestion autonome de QBAIoT au niveau de la HL-Gw

1: ind ← 0
2: Candidate[] ← Null
3: $QoS\_CAP\_Not\_Used_1[]$ ← Null
4: $QoS\_CAP\_Not\_Used_2[]$ ← Null
5: $QoS\_CAP\_Not\_Used_3[]$ ← Null
6: $QoS\_CAP\_Not\_Used_4[]$ ← Null
7: $L_1[]$ ← Null
8: $L_2[]$ ← Null

```
9 :   L3[] ← Null
10 :  L4[] ← Null
11:   N1 ← 0
12:   N2 ← 0
13:   N3 ← 0
14:   N4 ← 0
15:   Environment_Changes ← true
16:   while ( Environment_Changes = true) do
17:       Determine(Nb_QoS_Classes)
18:       Determine(RT_Classes)
19:       HL-Gw_Self_Configuration ()
20:       for i ∈ [1,Nb_QoS_Classes] do
21:           Slots(Actual_QoS_CAPi) ← Slots(QoS_CAP_Initiali)
22:           for j ∈ [0, Ni] do
23:               QoS_CAP_Not_Usedi[j] = Null
24:               Li[j] = Null
25:           end for
26:           Ni ← 0
27::          Nb_Packeti ← 0
28:           for j ∈ [0, ind] do
28:               Candidate [j] ← Null
29:           end for
30:       end for
31:       ind ← 0
32:       Environment_Changes ← false
33:       while ( Environment_Changes = false) do
35:           HL-Gw_Self_Optimization ()
35:           if (gSLA_added = true) then
37:               Nb_gSLA ← Nb_gSLA + 1
38:               Environment_Changes ← true
39:           else
40:               if (gSLA_deleted = true) then
41:                   Nb_gSLA ← Nb_gSLA - 1
42:                   Environment_Changes ← true
43                    if (Nb_gSLA = 0) then
44                        Send_To_LL-Gw_Effector (gSLA_existence = False)
45                    end if
46:               end if
47:           end if
48:       end while
49:   end while
```

Figure 8.9 : Algorithme de gestion autonome globale de QBAIoT

L'algorithme de gestion autonome globale de *QBAIoT* que nous implémentons au niveau de la passerelle de haut niveau *HL-Gw* appelle la fonction d'auto-configuration (i.e., *HL-Gw_Self_Configuration()* qui correspond à l'algorithme de la Figure 8.4) lors d'un changement du nombre de classes de *QoS* dans l'environnement IoT de la *LL-Gw*. De plus, cet algorithme de gestion autonome globale appelle la fonction d'auto-optimisation (i.e., *HL-Gw_Self_Optimization()* qui

correspond à l'algorithme de la Figure 8.7) suite à la réception du nombre de paquets reçus par *QoS CAP* de la *LL-Gw*. Les fonctions, variables et paramètres utilisés dans cet algorithme de gestion autonome globale sont définis dans les Tableaux 8.2, 8.3 et 8.4.

8.7. Validation et évaluation de la gestion autonome de QBAIoT

Dans le cadre de la validation et l'évaluation des performances des fonctions d'auto-configuration et d'auto-optimisation spécifiées pour la gestion autonome de la méthode de contrôle d'accès au canal *QBAIoT* dans un environnement IoT, nous définissions dans ce qui suit des ensembles de scénarios de simulations permettant de montrer l'efficacité de ces nouvelles fonctions ajoutées à *QBAIoT*. Ainsi, l'évaluation de la gestion autonome de *QBAIoT* consiste à considérer dans un premier temps la fonction d'auto-configuration selon un premier ensemble de scénarios de simulations et par la suite de se baser sur cette première fonction d'auto-configuration et enrichir nos scénarios de simulation pour évaluer la deuxième fonction d'auto-optimisation. Dans ce contexte, nous évaluons l'impact de l'utilisation de différentes configurations de *QoS CAP* et l'adaptation de ces configurations en fonction des caractéristiques de l'environnement IoT. Pour ce faire, nous utilisons le même environnement OMNET++ défini dans le chapitre 5 (cf. section 5.4.1) pour la mise en place des différents scénarios de simulation avec les paramètres présentés dans le Tableau 8.5.

Tableau 8.5 : Paramètres des simulations

Paramètres	Valeurs
Temps de simulation	100 s
Taille du paquet (couche MAC)	50 Octets
Topologie	Etoile
Nombre de coordinateur par WPAN	1
Fréquence	2.4 Ghz
Débit du canal	250 kb/s

8.7.1. Résultats de l'auto-configuration QBAIoT

Afin mettre en place le processus d'auto-configuration au niveau de la passerelle de haut niveau *HL-Gw*, nous implémentons en premier lieu la base de connaissances en créant une base de données des configurations disponibles (cf. Tableau 8.1). Cette base de données est consultée par la *HL-Gw* pour déterminer la configuration à adopter en se basant sur le nombre de classes *RTC* et *NRT*C déduit des *gSLA* relatifs à une *LL-Gw*. Pour ce faire, nous avons procédé à des changements sur le code du modèle du standard IEEE 802.15.4 développé dans OMNET++ par [167] afin d'y intégrer les fonctionnalités et la base de connaissances de la boucle de contrôle fermée *MAPE-K* relative au processus d'auto-configuration.

Tableau 8.6 : Caractéristiques du scénario de simulation 1

Temps	Nombre objets RTNMC	Nombre objets RTNMC	Nombre objets Streaming	Nombre objets NRT	DGI
0s < t < 15 s	3	3	3	3	0.25 s
15 s < t < 100s	3	3	0	0	0.25 s

Nous spécifions le scénario 1 (voir Tableau 8.6) pour évaluer les performances de la configuration autonome de *QBAIoT*. Dans ce contexte, l'environnement IoT de la *LL-Gw* contient initialement les 4 classes de *QoS*. Après un temps $t_{0=15s}$ du début de la simulation, l'*IoT*-SP supprime des *gSLA* de la *HL-Gw* suite à l'annulation de deux contrats *iSLA* et par conséquent, il ne reste que deux classes de *QoS* générant des trafics *RTC* dans l'environnement de la *LL-Gw*. Ainsi, au début de la simulation, chacune des 4 classes de *QoS* dispose de 3 objets générant des paquets avec un intervalle de 0.25 s. Au bout de 15 secondes, les *gSLAs* concernant les trafics *Streaming* et *NRT* sont supprimés et les trafics correspondants sont arrêtés. Nous comparons dans Figure 8.10 le délai moyen observé par les trafics en utilisant ou non la capacité de configuration autonome *QBAIoT* pour les trafics *RTMC* et *RTNMC*. Dans ce scénario, sans la capacité de configuration autonome de *QBAIoT*, la *HL-Gw* n'est pas en mesure de modifier les configurations initiales sans intervention humaine. Par conséquent, les créneaux attribués aux trafics *Streaming* et *NRT* ne sont pas utilisés pendant chaque supertrame après la suppression des *gSLA* correspondants. Ainsi, pour chaque supertrame, 5 *IT* sont non utilisés, ce qui entraîne une utilisation de 68% (11/16 *IT*) des *IT* de la supertrame. Les résultats obtenus illustrés dans la Figure 8.11 montrent qu'en utilisant *QBAIoT* avec configuration autonome l'utilisation des *IT* perdus en raison de modifications de l'environnement permet une minimisation des délais observés par les trafics *RTMC* et *RTNMC*. Ainsi *QBAIoT* est capable de mieux répondre aux exigences du trafic en temps réel.

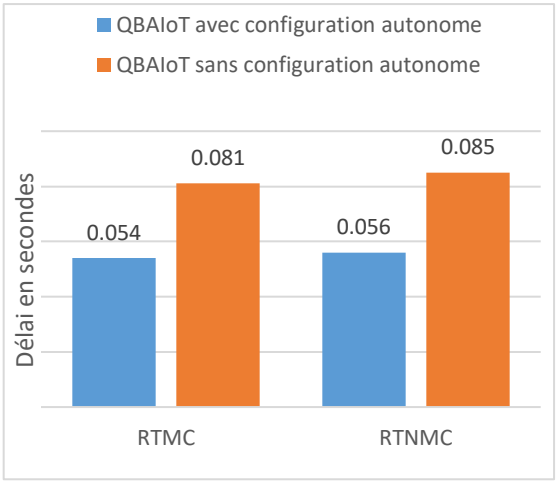

Figure 8.10 : Evaluation des délais en utilisant QBAIoT avec et sans auto-configuration (scénario 1)

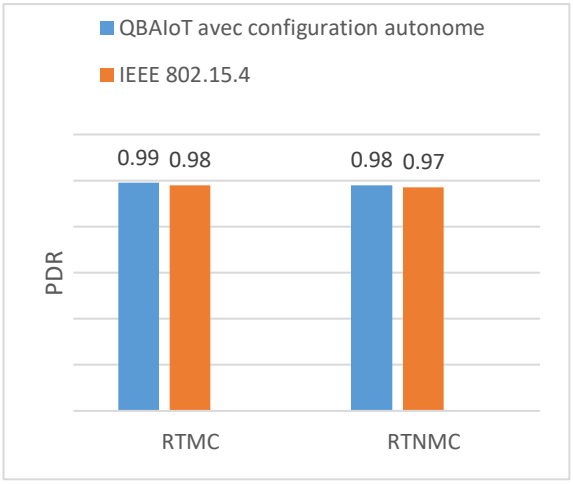

Figure 8.11 : Evaluation des PDR en utilisant QBAIoT avec et sans auto-configuration (scénario 1)

De plus, nous démontrons à travers la Figure 8.11 que l'utilisation de la fonction d'auto-configuration *QBAIoT* dans le scénario 1 permet une meilleure utilisation des ressources de la LL-Gw. La Figure 8.11 montre les résultats obtenus concernant le paramètre de performance PDR pour les trafics *RTMC* et *RTNMC* lors de l'utilisation de *QBAIoT* avec et sans capacité de configuration autonome selon le même scénario 1. Nous pouvons remarquer qu'avec la capacité de configuration autonome, les *IT* des trafics *Streaming* et *NRT* sont immédiatement réaffectés aux classes *QoS* restantes. Par conséquent, les *PDR* des trafics *RTMC* et *RTNMC* profitent d'une légère amélioration, comme une réaffectation des créneaux a été attribuée à leurs classes *QoS*.

8.7.2. Résultats de l'auto-optimisation QBAIoT

Afin mettre en place le processus d'auto-optimisation au niveau de la passerelle de haut niveau HL-Gw, nous enrichissons en premier lieu la base de connaissances, que nous avons implémentée dans le cadre de l'auto-configuration, avec de nouveaux paramètres définis dans les Tableaux 8.3 et 8.4. Cette base de connaissances est consultée par la *HL-Gw* pour optimiser la configuration des *QoS CAP* en se basant sur l'utilisation des *IT* correspondants. Pour ce faire, nous avons modifié le code du modèle du standard IEEE 802.15.4 développé dans OMNET++ par [167] afin d'y intégrer les fonctionnalités *MAPE-K* relative au processus d'auto-optimisation mais aussi enrichir la base de connaissances implémentée dans le cadre de la fonction d'auto-configuration.

Nous évaluons les performances de l'auto-optimisation *QBAIoT* en se basant sur un ensemble de simulations comprenant deux scénarios (voir Tableau 8.7 pour scénario 2 et 8.8 pour scénario 3). Le processus d'auto-configuration *QBAIoT* est exécuté avant l'auto-optimisation afin d'établir les configurations initiales en termes d'*IT* alloués aux 4 *QoS CAP* dans les deux scénarios de simulation considérés (i.e., 6 *IT* pour *RTNMC* / 5 *IT* pour *RTNMC* / 3 *IT* pour *Streaming* / 2 *IT* pour *NRT*).

Tableau 8.7 : Caractéristiques des trafics du scénario 2

Classes de QoS	***RTMC***	***RTNMC***	***Streaming***	***NRT***
Nombre d'objets IoT par classe	1	3	3	3
Intervalle de génération de paquets par objet	0.25	0.125	0.25	0.25
Nombre total de paquets générés	400	2400	1200	1200

Dans le scénario 2 (voir Figure 8.12), le processus d'auto-optimisation détermine que le trafic *RTNMC* généré par les objets IoT n'a pas besoin de tous les *IT* affectés à cette classe de *QoS* alors que le trafic *RTNMC* génère un trafic ayant besoin de plus d'*IT* que ceux affectés par le processus d'auto-configuration. Dans ce contexte, nous évaluons les performances de l'auto-optimisation de *QBAIoT* en termes de délai moyen. Les résultats obtenus (voir Figure 8.12) montrent que l'auto-optimisation *QBAIoT* améliore considérablement le délai moyen pour les flux *RTNMC* et *Streaming* grâce à la réallocation d'*IT* non utilisés par le trafic *RTMC* tout en respectant les contraintes temps réel de cette classe.

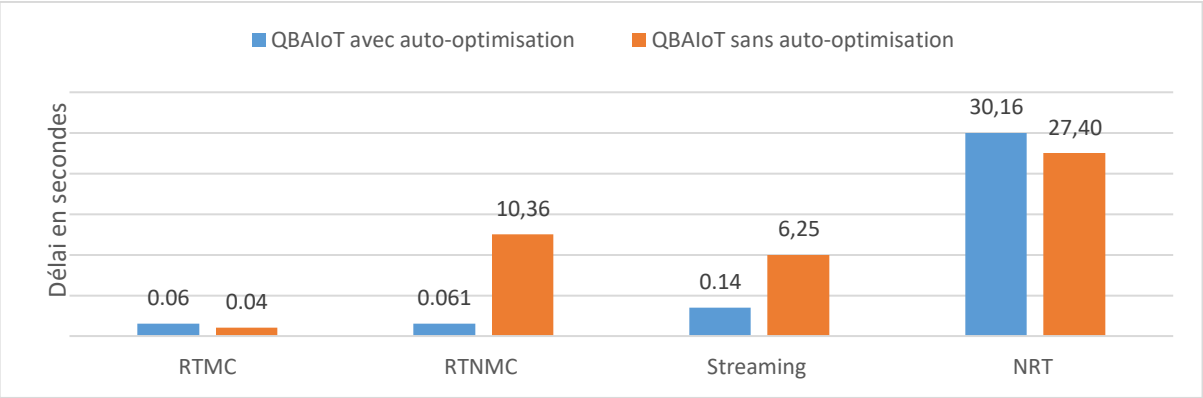

Figure 8.12 : Evaluation du délai moyen avec et sans auto-optimisation (scénario 2)

En effet, la Figure 8.12 montre que le délai moyen du trafic de la classe *RTNMC* passe de l'ordre de secondes à l'ordre de millisecondes. En ce qui concerne le délai moyen de trafic *RTMC*, nous pouvons noter un délai approximativement similaire avec les deux approches (avec et sans auto-optimisation). Avec une capacité d'auto-optimisation, le trafic *RTMC* observe un délai supplémentaire négligeable par rapport au gain de délai moyen observé par le trafic *RTNMC*. Ce délai supplémentaire n'entraîne pas un non-respect des exigences du trafic *RTMC* et ce grâce à la définition d'un nombre minimal d'*IT* pour chaque *QoS CAP* lors de la réaffectation des *IT* dans le processus d'auto-optimisation *QBAIoT*. En ce qui concerne le trafic *NRT*, un délai supplémentaire est observé (non critique pour le trafic non temps réel) dû à un nombre plus élevé de paquets *NRT* reçus par la *LL-Gw* (voir le *PDR* du trafic *NRT* avec capacité d'auto-optimisation dans la Figure 8.13). En effet, pendant la simulation, des *IT* supplémentaires sont alloués à cette classe de *QoS* de type *NRT* pendant une certaine période lorsque les autres classes ne sont pas en train d'utiliser tous les *IT* qui leur sont affectés, ce qui permet de servir davantage de paquets retardés et augmenter ainsi le délai moyen des paquets de cette classe.

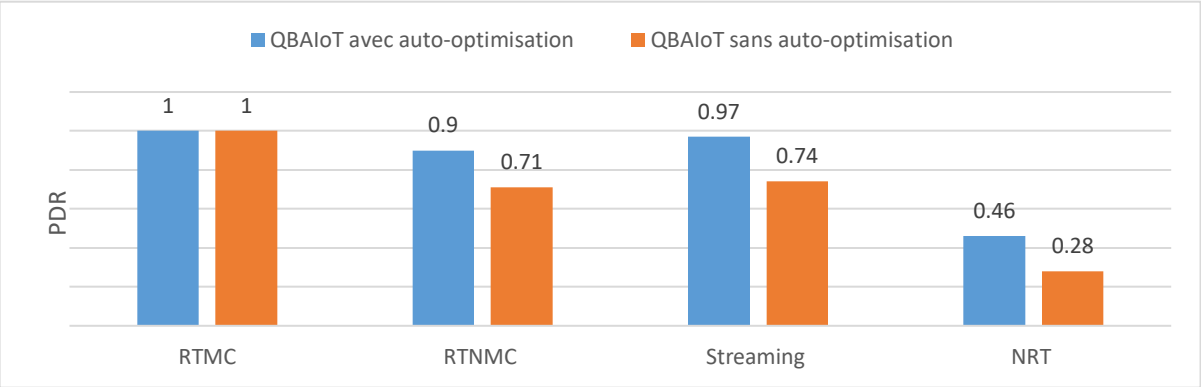

Figure 8.13 : Evaluation du PDR avec et sans auto-optimisation (scénario 2)

L'évaluation des *PDR* (voir Figure 8.13) concernant les différents types de trafic lors de l'utilisation de *QBAIoT* avec et sans capacité d'auto-optimisation montrent que nous obtenons de meilleurs résultats avec la capacité d'auto-optimisation pour les différents types de trafic du scénario

1. Ces résultats démontrent que l'auto-optimisation *QBAIoT* améliore l'utilisation des *IT* par différents types de trafic tout en respectant les exigences de chacun. La réallocation des *IT* inutilisés à d'autres *QoS CAP* offre un meilleur taux de réception de paquets dans chaque *QoS CAP*.

Tableau 8.8 : Caractéristiques des trafics du scénario 3

Classes de QoS	***RTMC***	***RTNMC***	***Streaming***	***NRT***
Nombre d'objets IoT par classe	3	3	3	3
Intervalle de génération de paquets par objet	0.25	0.25	0.25	0.25
Nombre total de paquets générés	1200	500	1200	1200

Le scénario de simulation 3 que nous considérons pour évaluer l'auto-optimisation de *QBAIoT* est basé sur un environnement IoT incluant un trafic *RTNMC* intermittent. En effet, dans ce scénario, le trafic *RTNMC* n'est pas généré d'une façon continue pendant toute la durée de la simulation puisqu'il est interrompu pendant 25 s. De plus, les objets *RTNMC* ne sont pas tous actifs en même temps. Au début de ce scénario de simulation, un seul objet est actif pendant 25 secondes, par la suite l'interruption du trafic *RTNMC* se produ*it* pendant 25 secondes suivie par l'activation de 2 objets jusqu'à la fin de la simulation. Par conséquent, le trafic *RTNMC* dans le scénario 3 n'occupe pas les 5 *IT* qui lui sont attribués à travers la configuration initiale. Les caractéristiques des différents trafics du scénario 3 sont présentées dans le Tableau 8.8. Dans ce contexte, nous commençons par évaluer le délai des paquets des différents trafics de ce scénario.

D'une part, nous observons pour le trafic *RTNMC* (voir Figure 8.14) et le trafic *RTNMC* (voir Figure 8.15) un délai approximativement similaire lors de l'utilisation de *QBAIoT* avec auto-optimisation (courbe bleue) et sans-optimisation (courbe orange). En effet, dans ce scénario, l'auto-optimisation *QBAIoT* n'alloue pas plus d'*IT* pour le trafic *RTNMC* et *RTNMC* car ils sont bien desservis par la configuration initiale. De plus, dans ce scénario, les *IT* non utilisés par le trafic *RTNMC* sont réaffectés aux classes de *QoS* de type *Streaming* et *NRT* dans le cadre du processus d'auto-optimisation *QBAIoT*.

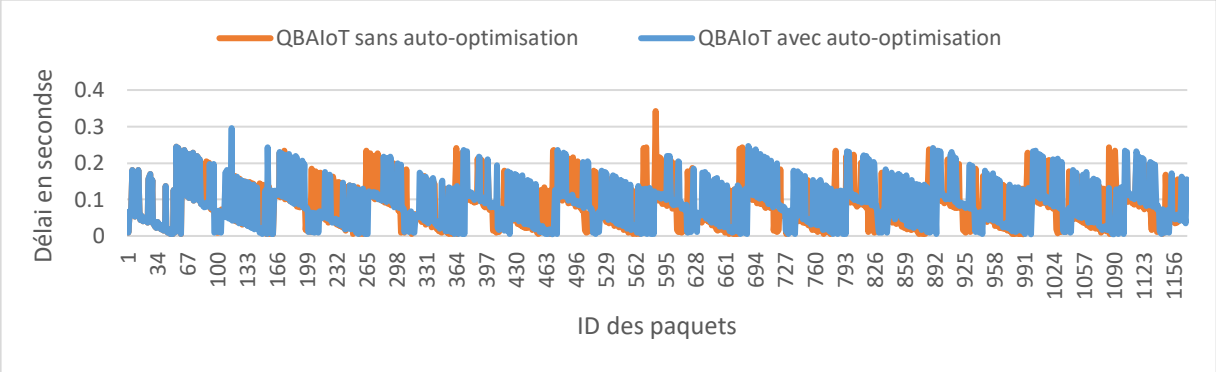

Figure 8.14 : Evaluation des délais des paquets RTMC en utilisant QBAIoT avec et sans auto-optimisation (scénario 3)

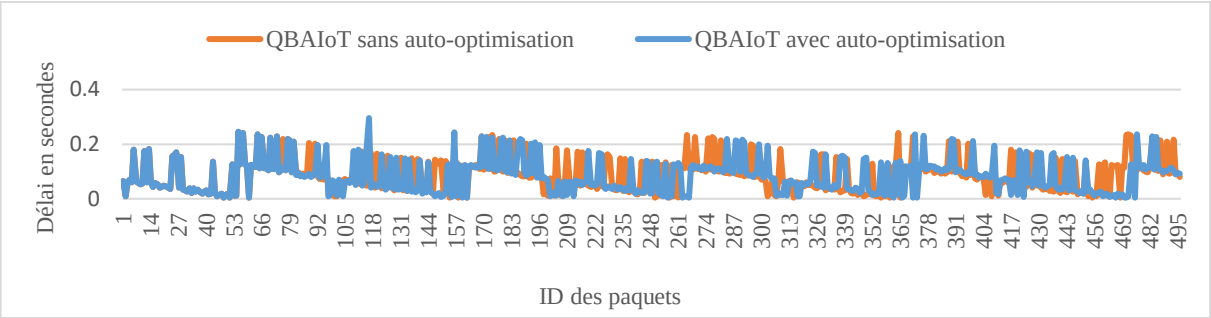

Figure 8.15 : Evaluation des délais des paquets RTNMC en utilisant QBAIoT avec et sans auto-optimisation (scénario 3)

D'autre part, le trafic de *Streaming* n'est pas desservi efficacement avec la configuration initiale des *IT*. Dans ce scénario, le processus d'auto-optimisation *QBAIoT* permet de remédier à ce problème en réaffectant les *IT* non utilisés de la classe *RTNMC* à la classe *Streaming*. Ainsi, nous pouvons observer dans Figure 8.16 que sans capacité d'auto-optimisation (courbe orange), le délai des paquets *Streaming* augmente à mesure que le nombre de paquets à servir dans la mémoire tampon augmente sachant que le nombre d'*IT* affectés à cette classe est fixe pendant toute la durée de la simulation. Par contre, pour ce même trafic *Streaming*, lors de l'utilisation de *QBAIoT* avec une capacité d'auto-optimisation (courbe bleue de la Figure 8.16), nous pouvons observer que le délai augmente pour les premiers paquets et par la suite le délai des paquets diminue avant d'augmenter à nouveau. La diminution du délai des paquets de *Streaming* correspond à la période de 25 secondes d'interruption du trafic *RTNMC* qui a engendré la réaffectation des *IT* inutilisés par ce trafic à la *QoS CAP* de la classe *Streaming* grâce à l'auto-optimisation *QBAIoT*. A la reprise du trafic *RTNMC*, les *IT* cédés à la classe *Streaming* sont récupérés ce qui explique l'augmentation du délai des paquets de *Streaming* après leur diminution. Nous pouvons également noter que certains *IT* de la classe *RTNMC* ont été réaffectés au trafic *Streaming* par le processus d'auto-optimisation depuis le début de la simulation. Cette réaffectation des *IT* montre qu'avant même l'interruption du trafic *RTNMC*, tous les *IT* attribués à cette classe ne sont pas utilisés. De plus, cette réaffectation explique le fait que la courbe orange de la Figure 8.16 représentant le délai *Streaming* lors de l'utilisation de *QBAIoT* sans capacité d'auto-optimisation présente des valeurs plus élevées que la courbe bleue (c'est-à-dire *QBAIoT* avec auto-optimisation).

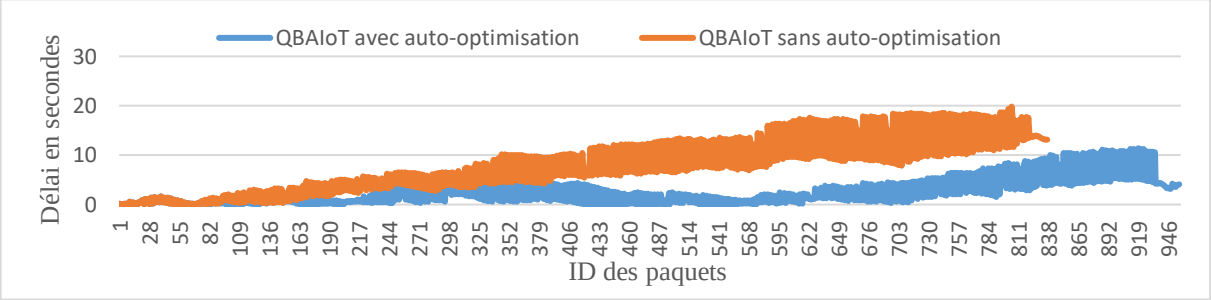

Figure 8.16 : Evaluation des délais des paquets Streaming en utilisant QBAIoT avec et sans auto-optimisation (scénario 3)

L'auto-optimisation *QBAIoT* permet d'allouer dans ce scénario de simulation plus d'*IT* à la classe de trafic *Streaming* ce qui permet de servir plus de paquets de cette classe. Dans ce contexte, nous évaluons l'*EDR* des différents trafics de ce scénario et nous observons un meilleur *EDR* indiquant un meilleur *PDR* aussi pour les trafics *Streaming* et *NRT*. En effet, un meilleur *EDR* du trafic *NRT* indique que certains *IT* ont été réaffectés à ce dernier durant le processus d'auto-optimisation pour une période assez courte étant donné la légère amélioration de l'EDR en utilisant *QBAIoT* avec la capacité d'auto-optimisation. En effet, comme le montre la Figure 8.17, un *EDR* supérieur de 14% est observé par le trafic *Streaming*, et un *EDR* supérieur de 10% pour le trafic *NRT*. L'*EDR* des trafics *RTMC* et *RTNMC* est le même, car aucun *IT* supplémentaire n'a été affecté à *RTNMC* et la réaffectation temporaire des *IT* inutilisés par *RTNMC* avec l'auto-optimisation *QBAIoT* n'a pas d'incidence sur ce trafic. Ainsi, l'amélioration du débit effectif de données (i.e., *EDR*) de ces trafics grâce à la fonction d'auto-optimisation *QBAIoT* contribue à une utilisation efficace de la bande passante du canal sans fil dans la couche sensing de l'environnement IoT.

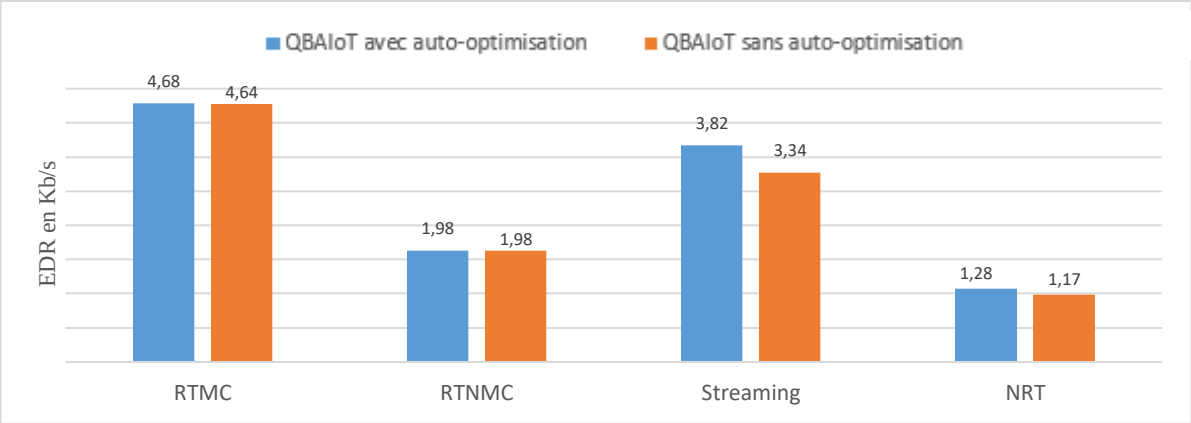

Figure 8.17 : Evaluation de l'EDR en utilisant QBAIoT avec et sans auto-optimisation (scénario 3)

8.8. Conclusion

A travers ce chapitre, nous avons défini deux processus assurant la gestion autonome de *QBAIoT*. Premièrement, l'auto-configuration permet de choisir la configuration adaptée aux caractéristiques de l'environnement IoT et de modifier cette configuration suite à un changement qui concerne le nombre de classes de *QoS* dans l'environnement de la *LL-Gw*. Deuxièmement, l'auto-optimisation permet d'adapter cette configuration en temps réel en prenant en considération l'utilisation effective des ressources allouées initialement à chaque classe de *QoS* par la fonction d'auto-configuration. Dans ce contexte, nous avons détaillé le principe de fonctionnement, le design et les algorithmes à utiliser au sein de notre Framework de gestion autonome de *QBAIoT* afin de mettre en place ces deux fonctions d'auto-configuration et d'auto-optimisation. De plus, l'évaluation des performances que nous avons menée pour ces deux processus a montré leur efficacité à travers l'amélioration des paramètres de *QoS* relatifs aux trafics de l'environnement IoT en termes de délai,

le taux de livraison des paquets et de débit effectif de données. En effet, cette gestion autonome de *QBAIoT* a permis de choisir la meilleure configuration des *QoS CAP* en fonction du nombre de classes de *QoS* qui existent dans l'environnement de la *LL-Gw* et d'adapter cette configuration en temps réel en fonction de l'utilisation effective des ressources par les différents trafics afin d'améliorer les paramètres de *QoS* dans l'environnement IoT. Ainsi, les simulations que nous avons menés ont montré que la gestion autonome de *QBAIoT* permet d'améliorer les délais lors de l'acheminement des trafics et d'obtenir de meilleurs taux de livraison de paquets grâce une utilisation efficace des ressources du système IoT. La gestion autonome dans un environnement IoT ne se limite pas à la méthode d'accès au canal et peut prendre en compte d'autres aspects spécifiques aux caractéristiques de cet environnement tels que l'optimisation de la consommation énergétique que nous allons traiter dans le chapitre suivant.

Chapitre 9 – Gestion autonome de l'efficacité énergétique dans l'IoT

9.1. Introduction

La durée de vie du système IoT est considérée comme un critère de *QoS* important dans ce type d'environnement permettant d'assurer la disponibilité du service IoT offert par ce système. Afin de minimiser la consommation énergétique des objets IoT alimentés par des batteries au niveau de la couche sensing, nous proposons dans ce chapitre de réduire la consommation énergétique des objets IoT dans un environnement utilisant notre méthode d'accès au canal *QBAIoT*. En effet, *QBAIoT* élimine la période inactive du standard IEEE 802.15.4 pour réduire le délai des trafics temps réel mais ceci risque d'augmenter la consommation énergétique des objets IoT d'où la nécessité de trouver un remède. Nous proposons dans ce chapitre de gérer d'une manière autonome la consommation énergétique des objets IoT grâce à une méthode qui permet de choisir d'une façon périodique, un sous-ensemble des objets IoT associés à la *LL-Gw* pour remonter les informations relatives à un service IoT et ce en se basant sur différents critères propres à l'environnement IoT et en utilisant un système de logique floue.

Ainsi, nous présentons dans la section 2 de ce chapitre notre Framework d'auto-optimisation de la consommation énergétique. Ensuite, nous décrivons dans la section 3 le modèle de logique floue que nous avons spécifié pour réaliser cette fonction d'auto-optimisation énergétique. Ce modèle permet à la LL-Gw de choisir les objets IoT à activer en prenant en compte différents paramètres en entrée. Nous présentons dans cette section non seulement les paramètres d'entrée (i.e., distance, énergie résiduelle et précision des informations) de notre modèle, mais aussi le paramètre de sortie (i.e., chance de l'objet IoT) et la base d'inférence incluant les règles à appliquer. Par la suite, nous spécifions dans la section 4 les détails de conception de cette fonction d'auto-optimisation ainsi que l'algorithme de prise de décision. La section 5 est consacrée à l'évaluation des performances de notre proposition. Enfin, la section 6 nous permet de conclure ce chapitre.

9.2. Framework d'auto-optimisation de la consommation énergétique

9.2.1. Besoins d'auto-optimisation de la consommation énergétique dans l'IoT

Nous avons démontré dans le chapitre 4, que la durée de vie d'un système IoT est un paramètre de *QoS* important pour l'offre d'un service dans ce type d'environnement. En effet, le système IoT ne peut remplir sa mission d'offre de service que lorsqu'il est toujours en vie ou encore opérationnel. Dans ce contexte, la durée de vie du système indique son utilité maximale. Par conséquent, la durée

de vie du système définit le temps opérationnel pendant lequel le système est en mesure d'exécuter les tâches qui lui incombent. Dans le cadre des réseaux de capteurs avec des caractéristiques similaires aux objets de la couche sensing de l'architecture IoT, certains travaux de recherche estiment que la durée de vie des réseaux de capteurs doit être mesurée jusqu'à ce que le premier capteur ou groupe de capteurs soit à court d'énergie. Pour ces travaux, perdre un capteur ou un groupe de capteurs entraine la perte de certaines fonctionnalités du système et par conséquent toutes les opérations de ce système. D'autres travaux considèrent que la durée de vie du réseau de capteurs est définie comme étant la durée maximale pendant laquelle les capteurs vivants déployés ont la capacité de surveiller les phénomènes d'intérêt [188]. Dans le cadre de notre environnement IoT, nous considérons que notre système, formé par des objets IoT, est opérationnel (i.e., en vie) lorsque les objets déployés sont capables de remonter des données précises. En d'autres termes, nous considérons que notre système IoT fonctionne pleinement lorsqu'il est capable de renvoyer des données précises selon le type de service IoT offert par ce système. Ainsi, pour les services IoT critiques, le système est considéré comme étant opérationnel à condition qu'il y ait au moins deux objets IoT actifs (i.e., énergie résiduelle non nulle) capables de générer des données précises. Dans le cas de services IoT non critiques, le système est considéré comme étant opérationnel à condition qu'il y ait au moins deux objets IoT actifs indépendamment de la précision des données remontées.

Afin d'étendre la durée de vie du système IoT, il faut maximiser la durée de vie de chaque composant de ce système. En particulier, il faut optimiser la durée de vie des composants ayant des contraintes en termes de capacités énergétiques comme les objets IoT fonctionnant avec des batteries. De plus, l'amélioration des performances du système IoT dépendent fortement de l'optimisation de la consommation d'énergie, qui améliore la durée de vie des objets IoT et par conséquent du système qui offre un service IoT. La gestion de la consommation énergétique est un sujet très vaste qui s'étend aux logiciels et aux matériels. Il est important de considérer la gestion de la consommation énergétique dans un environnement IoT. Dans notre étude, nous nous concentrons sur l'optimisation de la consommation énergétique au niveau des objets IoT étant donné que les autres composants du système IoT de notre architecture sont alimentés en continue. Dans ce contexte, l'optimisation peut porter sur plusieurs facteurs et paramètres tels que le temps d'activation des objets IoT (i.e., un objet non actif préserve son énergie), la fréquence de collecte des données (i.e., l'envoi des données consomme de l'énergie), la consommation due à l'utilisation de certains technologies de communications, la consommation énergétique du microprocesseur, la complexité des algorithmes et logiciels implémentés, etc. [189].

Différentes approches ont été proposées à travers des travaux de recherche pour tenter de minimiser la consommation d'énergie dans les systèmes IoT. Ainsi, les auteurs dans [190] ont présenté une nouvelle méthode de routage pour les réseaux de capteurs afin de maximiser la durée de vie du système en combinant une approche floue et un algorithme « A-star » dans des environnements IoT à sauts multiples. Dans leur contribution, les auteurs ont utilisé deux paramètres d'entrée essentiels pour leur système flou, à savoir l'énergie restante dans les objets et la charge de trafic, et un seul paramètre de sortie appelé coût du nœud et ce pour acheminer le trafic suivant une route bien

définie en utilisant les meilleurs objets. Dans le travail de recherche présenté dans [191], les auteurs proposent un nouvel algorithme de transmission de paquets tenant compte de la consommation énergétique et fonctionnant au niveau de la sous couche *MAC* IEEE 802.15.4 dans des conditions variables (i.e., différentes charges de trafic). L'algorithme proposé est appelé *ABSD* (Adaptive Beacon Order, Superframe Order and Duty cycle). À travers *ABSD*, les auteurs cherchent à minimiser le temps d'écoute avant l'envoi des données et à prolonger la période de sommeil inactive des objets IoT afin d'améliorer leurs énergies résiduelles. D'autre part, les auteurs dans [192] ont présenté une approche permettant de maximiser la durée de vie du système IoT en sélectionnant le meilleur objet IoT pour prendre le rôle du Cluster Head (*CH*). Pour ce faire, ils ont utilisé un système de logique floue prenant en compte quatre paramètres d'entrée (énergie résiduelle, puissance, mobilité et centralité) et un paramètre de sortie unique appelé la chance pour choisir le meilleur objet pour être le *CH*. Le meilleur objet assure une durée de vie importante pour le *CH* et permet ainsi au système IoT de rester fonctionnel plus longtemps.

Les travaux d'optimisation de la consommation énergétique que nous avons étudiés dont les trois approches que nous venons de présenter ne peuvent pas convenir à tous les scénarios d'utilisation et à toutes les architectures IoT. En particulier, notre architecture est basée sur une topologie à un seul saut. Par conséquent, l'algorithme présenté dans [190] n'est pas efficace pour ce type de topologie. De plus, notre méthode d'accès au canal *QBAIoT* fonctionne sans période inactive. Par conséquent, le protocole *ABSD*, qui se base sur la période inactive, ne peut pas être utilisé conjointement avec notre méthode pour minimiser la consommation d'énergie. Enfin, la troisième approche basée sur le choix d'un objet IoT pour jouer le rôle d'un *CH* n'est pas adaptée à notre proposition d'architecture IoT. En effet, nos passerelles de bas niveau *LL-Gw* jouent le rôle de *CH* dans notre architecture et ne sont pas concernées par l'optimisation de la consommation énergétique étant donné qu'elles sont alimentées d'une façon continue. Ainsi, toutes ces approches montrent que nous avons besoin de proposer une nouvelle approche afin d'optimiser la consommation énergétique dans un environnement IoT basé sur notre architecture à 3 couches mais aussi sur notre méthode d'accès basée sur la *QoS*, à savoir *QBAIoT*, au niveau de la couche sensing de cette architecture.

Notre méthode *QBAIoT* est basée sur l'élimination de la période inactive du standard IEEE 802.15.4. Cette élimination se traduit par une activation permanente de tous les objets IoT et par conséquent une consommation énergétique plus importante et une réduction de la durée de vie du système IoT. Ainsi, nous proposons une méthode qui permet de minimiser la consommation énergétique de *QBAIoT* en choisissant les objets qui participent à l'offre de service IoT. En effet, pour chaque intervalle de temps spécifique à chaque type d'application IoT, certains objets IoT dans l'environnement de la *LL-Gw* sont activés pour remonter des informations alors que d'autres sont en mode de préservation d'énergie. Le choix des objets à activer est effectué d'une manière autonome en se basant sur la boucle de contrôle fermée *MAPE-K* et une approche de logique floue. L'activation des objets doit se faire grâce à la configuration d'un intervalle de remontée d'informations exigées par le service IoT. Tandis que la non activation des objets se fait en les configurant avec un intervalle de remontée d'informations plus important et multiple de l'intervalle d'activation.

9.2.2. Framework de gestion autonome pour la consommation énergétique

Le processus de gestion autonome de l'optimisation de la consommation énergétique des objets IoT utilisant QBAIoT est mis en œuvre via l'implémentation de la boucle de contrôle fermée *MAPE-K* au niveau de la *LL-Gw* et des objets IoT de notre architecture IoT (voir Figure 9.1).

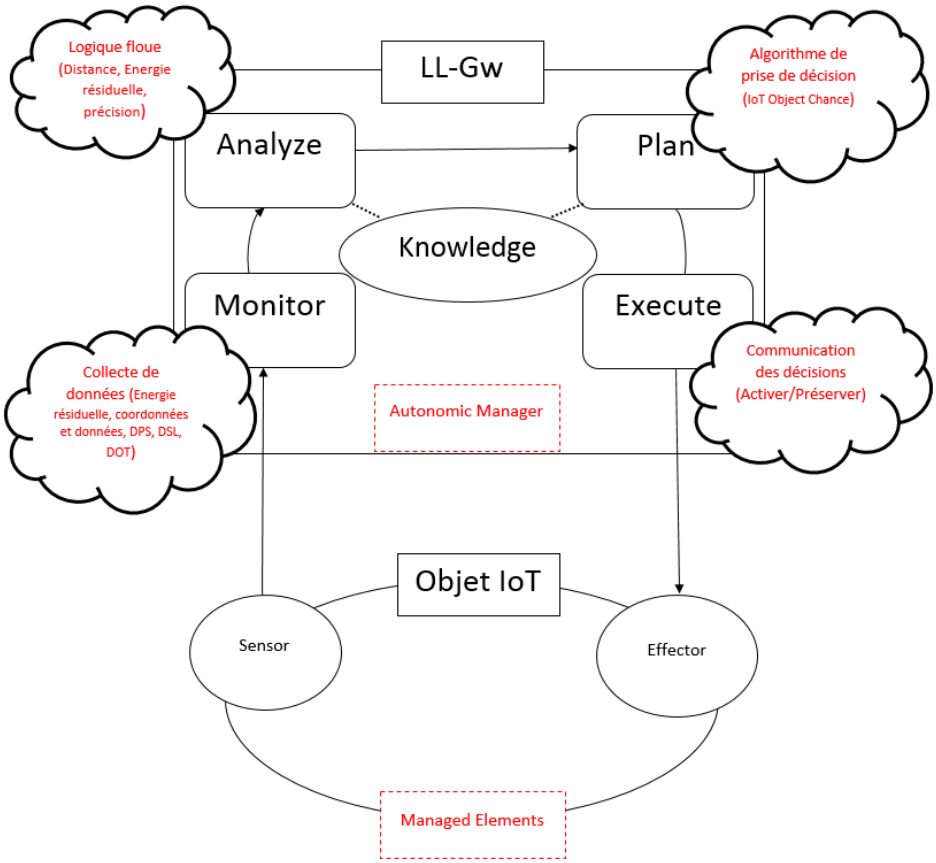

Figure 9.1 : Boucle MAPE-K pour l'auto-optimisation énergétique des objets IoT

Notre Framework d'auto-optimisation de la consommation énergétique illustré par la Figure 9.1 montre que la passerelle de bas niveau *LL-Gw* de la couche sensing de notre architecture IoT joue le rôle de gestionnaire autonome (*AM*) en mettant en œuvre les 4 fonctionnalités de la boucle de contrôle (i.e., surveiller (Monitor), analyser (Analyze), planifier (Plan) et exécuter (Execute)) ainsi que la base de connaissances (Knowledge Base). La fonctionnalité « **Monitor** » de la *LL-Gw* permet de récolter d'une façon régulière les informations relatives aux objets IoT afin d'alimenter le processus d'auto-optimisation de la consommation énergétique. L'interface de type « **Sensor** » de l'objet IoT est utilisée pour remonter ces informations en temps réel. Ensuite, la fonctionnalité « **Analyze** » permet de lancer le processus d'évaluation des objets IoT associés à la *LL-Gw* en se basant sur les informations récoltés par la fonctionnalité « **Monitor** » et des informations disponibles dans la base de connaissance. Cette évaluation est basée sur un modèle de la logique floue. Par la suite, la fonctionnalité « **Plan** » permet de prendre la décision concernant le(s) objet(s) à activer en

se basant sur l'évaluation effectuée par la fonctionnalité « **Analyze** » et un algorithme de prise de décision. Cette décision sera mise en place par la fonctionnalité « **Execute** » en la communiquant aux objets IoT concernés à travers leurs interfaces de type « **Effector** ». Ainsi, cette interface reçoit les décisions envoyées par la *LL-Gw* afin de configurer l'activité de l'objet et optimiser ainsi sa consommation énergétique d'une façon autonome. Les détails des différentes fonctionnalités de la boucle de contrôle fermée et en particulier le système de logique floue pour la fonctionnalité « **Analyze** » et l'algorithme de prise de décision pour la fonctionnalité « **Plan** » sont présentés dans les sections 9.3 et 9.4 respectivement.

9.3. Système de logique floue d'évaluation des objets IoT

Afin de minimiser la consommation énergétique du système IoT et étendre ainsi sa durée de vie, nous optons pour une approche permettant de choisir périodiquement les objets IoT qui doivent remonter à la passerelle *LL-Gw* les informations nécessaires pour l'offre d'un service IoT. Dans le cadre de cette approche, nous proposons un algorithme de sélection pour ces objets IoT en utilisant la valeur de sortie, appelée chance de l'objet IoT, d'un système de logique floue basé sur le modèle de Mamdani que nous spécifions dans la section suivante.

Afin de prolonger la durée de vie du système IoT, notre objectif consiste à minimiser la consommation d'énergie des objets IoT. Dans ce contexte, nous modélisons la consommation d'énergie des objets IoT en utilisant l'équation 9.1 et nous proposons de minimiser l'un des composants de cette équation à savoir l'énergie de transmission des données.

$$\text{Energie consommée (Objet IoT)} = \text{Enérgie (OS et applications)} + \text{Energie (transmission des données)} + \text{Energie (réception des données)} \quad (9.1)$$

9.3.1. Modèle Mamdani de logique floue

Nous commençons dans cette section par définir les caractéristiques du modèle de logique floue de Mamdani et nous spécifions par la suite son utilisation dans notre système d'évaluation des objets IoT. Le modèle de logique floue de Mamdani permet une description linguistique du système grâce à une base de règles floues conditionnelles formulées à l'aide de Si et Alors. Dans ce modèle, les ensembles d'entrée et de sorites sont des ensembles flous. Par conséquent, afin de retrouver une valeur numérique au niveau de la sortie, une étape de défuzzification est nécessaire. De plus, le modèle Mamdani peut correspondre à un système *MISO* (Multiple Input Single Output) ou bien à un système *MIMO* (Multiple Input Multiple Output). Par conséquent, nous pouvons avoir en sortie un ou plusieurs paramètres flous permettant le calcul d'une valeur numérique grâce à la fonction de défuzzification. Le modèle Mamdani est bien adapté pour utiliser des paramètres d'entrée et de sortie représentés par des expressions et des adjectifs linguistiques [193]. A titre d'exemple, nous pouvons utiliser ce modèle avec le paramètre d'entrée distance décrit par des variables linguistiques comme proche, loin et moyen mais aussi avec le paramètre de sortie chance décrit par des variables linguistiques comme haute, moyenne, faible.

Le modèle Madani de logique floue utilise plusieurs composants représentés par la Figure 9.2. Ce modèle permet d'avoir un ou plusieurs paramètres d'entrée et un ou plusieurs paramètres de sortie. Les paramètres de sortie sont déterminés par un système automatisé (i.e., inference engine) suivant un raisonnement humain inféré via un ensemble de règles prédéfinies et des fonctions d'appartenance. Le modèle Mamdani comporte un composant additionnel, à savoir celui de défuzzification, par rapport au modèle *TSK* défini dans le chapitre 7. Ce composant de défuzzification permet de convertir la valeur floue de sortie du moteur d'inférence en une valeur numérique. Deux méthodes sont principalement utilisées pour convertir cette valeur floue en une valeur numérique. La première méthode est celle du centre de gravité / de surface (*COG* : Center of Gravity) / (*COA* : Center of Area) alors que la deuxième est celle de la moyenne maximale (*MOM*) [194].

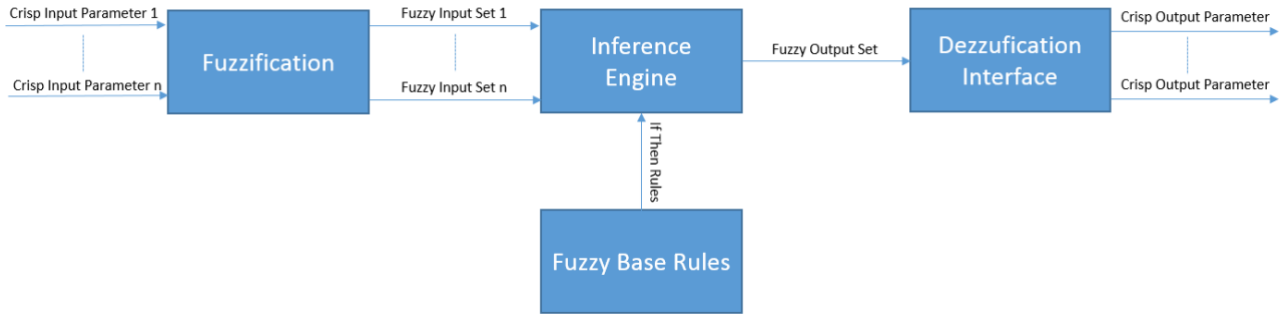

Figure 9.2 : Composants du système logique floue basé le modèle Mamdani

Dans notre proposition d'auto-optimisation de la consommation énergétique des objets IoT, nous spécifions un système d'évaluation des objets basé sur le modèle de logique floue Mamdani avec trois paramètres d'entrée (i.e., distance, Pourcentage d'énergie résiduelle et précision) et un paramètre de sortie (i.e., chance de l'objet IoT*)* associés à différentes fonctions d'appartenance mais aussi des règles et une base d'inférence. Nous détaillons dans ce qui suit les paramètres et la base d'inférence de notre système de logique floue Mamdani d'évaluation des objets IoT.

9.3.2. Paramètres d'entrée du système d'évaluation

Dans notre système d'évaluation basé sur le modèle de logique floue Mamdani, nous prenons en considération trois paramètres d'entrée pour sélectionner les objets IoT à activer en fonction du paramètre de sortie. Ces paramètres d'entrée concernent la distance qui sépare l'objet IoT de la passerelle *LL-Gw* mais aussi le pourcentage d'énergie résiduelle de l'objet IoT ou encore la précision des données remontées par l'objet IoT.

Le premier paramètre d'entrée que nous prenons en considération dans notre système de logique floue est la distance entre les objets IoT et le coordinateur (i.e., la passerelle de bas niveau *LL-Gw*). En effet, la distance a un impact direct sur la consommation d'énergie. Comme mentionné dans [195], la consommation d'énergie de chaque objet dépend directement de la distance qui le sépare du coordinateur. La distance entre les objets et le coordinateur ne peut pas dépasser la portée

spécifiée dans le standard IEEE 802.15.4. Par conséquent, la plage de valeurs possibles pouvant être prises en compte est comprise entre 0 et 100 mètres. Dans [196], les auteurs ont étudié la consommation d'énergie en fonction de la distance entre le coordinateur et les objets pour le standard IEEE802.15.4. Cette étude a démontré que la consommation d'énergie est inférieure à 100 mJ pour une distance comprise entre 0 et 30 mètres et elle est comprise entre 100 et 110 mJ pour une distance comprise entre 30 et 60 mètres alors qu'elle est supérieure à 110 mJ pour une distance comprise entre 60 et 100 mètres. Afin d'évaluer la distance entre les objets IoT et le coordinateur qui est la *LL-Gw* dans notre architecture, cette dernière peut recevoir les coordonnées de l'objet IoT à travers l'interface capteur implémentée sur ce dernier. Suite à la réception de ces coordonnées, la *LL-Gw* calcule la distance en utilisant l'équation 9.2., où (*Xa*, *Ya*, *Za*) et (*Xb*, *Yb*, *Zb*) sont les coordonnées respectives de la *LL-Gw* et de l'objet.

$$Distance = \sqrt{(Xa - Xb)^2 + (Ya - Yb)^2 + (Za - Zb)^2} \qquad (9.2)$$

Dans ce contexte, nous spécifions les variables linguistiques suivantes pour l'ensemble flou du paramètre distance : *Close*, *Mid* et *Far*. Nous spécifions dans la Figure 9.3 des fonctions d'appartenance trapézoïdales pour ces trois variables linguistiques relatives au paramètre d'entrée distance. En prenant en considération la distance en tant que paramètre d'entrée, nous visons à favoriser l'activation des objets IoT les plus proches de la *LL-Gw* afin de minimiser la consommation énergétique nécessaire pour la transmission des données

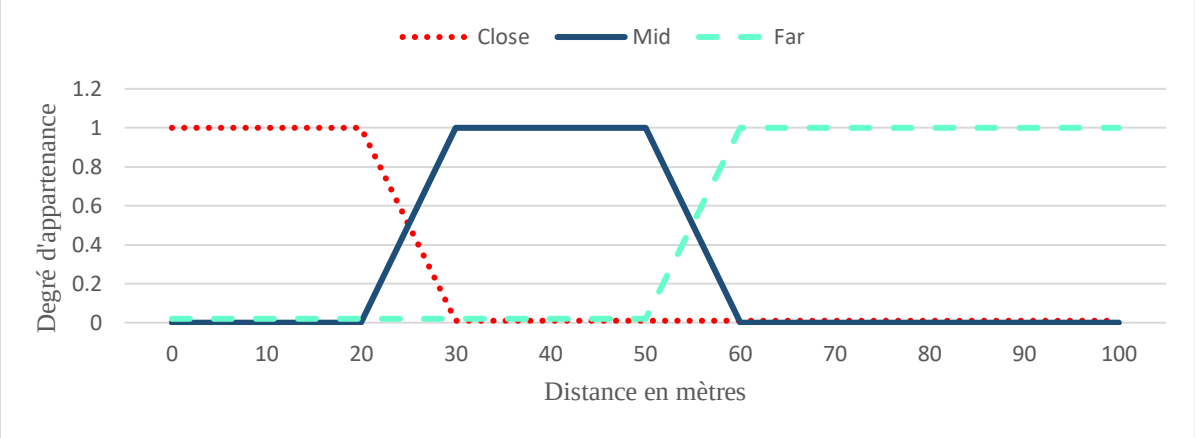

Figure 9.3 : Fonctions d'appartenance du paramètre distance

Le deuxième paramètre d'entrée que nous prenons en considération dans notre système de logique floue d'évaluation des objets IoT est le pourcentage d'énergie résiduelle de ces objets IoT. Afin de maximiser la durée de vie du système IoT, nous devons optimiser l'utilisation de l'énergie résiduelle dans les objets IoT. La valeur de ce paramètre correspond à un pourcentage (entre 0 et 100%) calculé selon l'équation 9.3. En effet, l'objet IoT communique, via son interface de type capteur, son niveau d'énergie résiduelle à la *LL-Gw* et cette dernière calcule le pourcentage de cette énergie résiduelle par rapport à l'énergie initiale communiquée par cet objet lors de son initialisation.

Dans ce contexte, nous spécifions les variables linguistiques suivantes pour l'ensemble flou du paramètre d'entrée pourcentage d'énergie résiduelle : *Low*, *Mid* et *High* sachant que plus le pourcentage d'énergie résiduelle de l'objet est élevé, plus les chances de sélection de cet objet sont grandes. Nous spécifions dans la Figure 9.4 des fonctions d'appartenance triangulaires pour ces trois variables linguistiques relatives au paramètre d'entrée pourcentage d'énergie résiduelle.

$$Energie\ résiduelle\ (\%) = \frac{Energie\ résiduelle}{Energie\ initiale} x100 \qquad (9.3)$$

En prenant en considération le pourcentage d'énergie résiduelle en tant que paramètre d'entrée, nous visons à favoriser l'activation des objets IoT avec un pourcentage d'énergie résiduelle permettant de prolonger la durée de vie des différents objets du système et par conséquent du système IoT formé par ces objets.

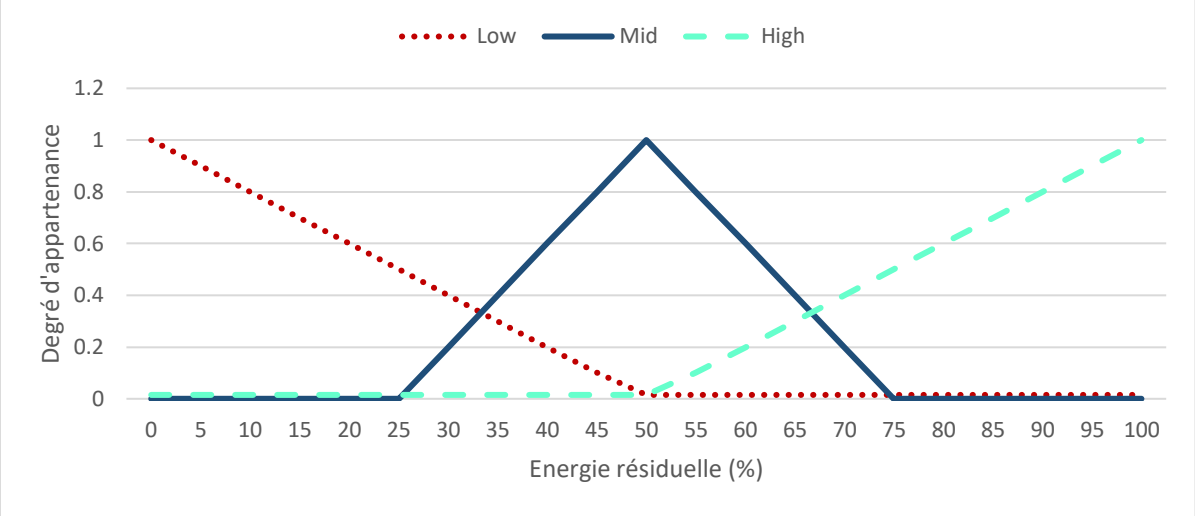

Figure 9.4 : Fonctions d'appartenance du paramètre énergie résiduelle

Le troisième paramètre d'entrée que nous prenons en considération dans notre système de logique floue d'évaluation des objets IoT est la précision des données remontées par l'objet. Lors de la maximisation de la durée de vie du système IoT, formé par les objets et les passerelles, il faut prendre en compte la qualité des données collectées afin de préserver la fiabilité du service IoT correspondant. Plusieurs critères de qualité sont utilisés pour la qualification des données dans différents travaux de recherche. Nous citons à ce titre : la précision (comparaison avec des données de référence), la complétude (pourcentage de données manquantes), etc. [197][198][199]. La norme ISO 5725 [200] utilise la véracité (i.e., trueness) et la précision pour décrire une méthode de mesure de la qualité des données. Selon cette norme, la véracité décrit la précision de la valeur moyenne. Dans notre système d'évaluation des objets IoT, nous utilisons la précision des données en tant que paramètre d'entrée décrivant la qualité des données. La valeur du paramètre précision correspond à un pourcentage (entre 0 et 100%) calculé selon l'équation 9.4. La précision de l'information remontée par chaque objet IoT est calculée par la *LL-Gw* suite à la réception des informations de tous les objets.

Dans ce contexte, nous spécifions les variables linguistiques suivantes pour l'ensemble flou du paramètre d'entrée précision des données : *Low*, *Mid* et *High*. Plus la précision des données générées par l'objet IoT est élevée, plus les chances de sélection de l'objet sont grandes. Nous spécifions dans la Figure 9.5 des fonctions d'appartenance trapézoïdale pour ces trois variables linguistiques relatives au paramètre d'entrée précision des données. Nous remarquons dans la figure 9.5 que la fonction d'appartenance *High* atteint la valeur maximale de degré d'appartenance pour une précision à partir de 97% en supposant que cette valeur a été précisée dans le contrat de type *iSLA*.

$$Précision\ (\%) = (1 - \frac{|Valeur\ données\ -\ Valeur\ moyenne\ |}{Valeur\ moyenne}) * 100 \qquad (9.4)$$

En prenant en considération la précision des données remontées par les objets IoT en tant que paramètre d'entrée, nous visons à favoriser l'activation des objets avec une précision permettant de maintenir une certaine fiabilité du service IoT correspondant.

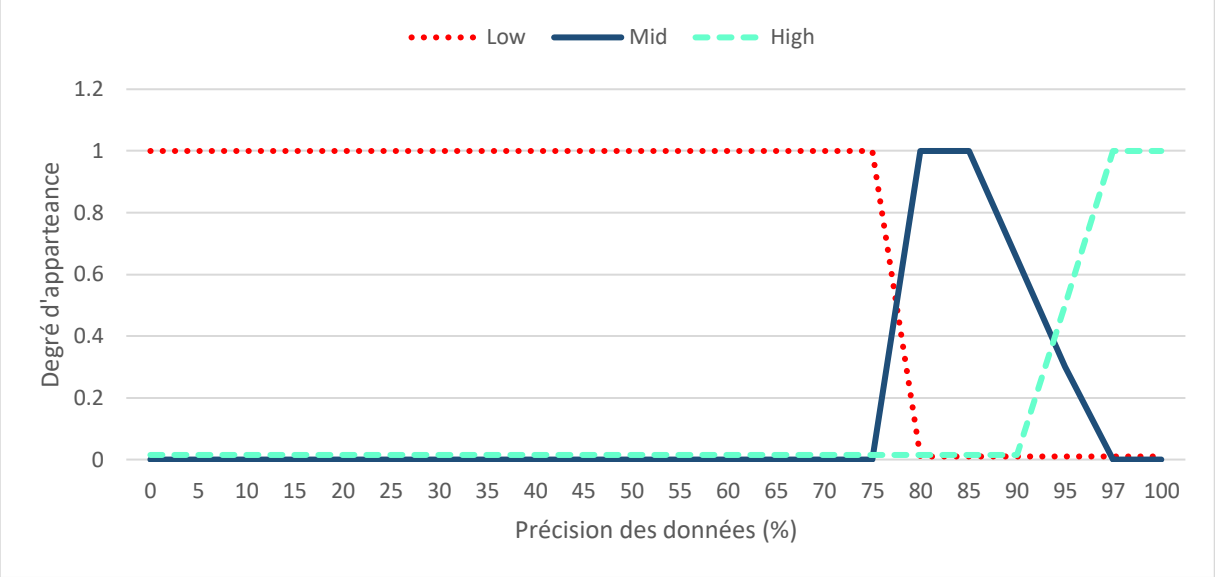

Figure 9.5 : Fonctions d'appartenance du paramètre précision

9.3.3. Paramètre de sortie du système d'évaluation

La sortie de notre système d'évaluation basé sur un modèle de logique floue permet de déterminer la chance de sélection de l'objet IoT (i.e., chance de l'objet IoT*)* à travers une note dans l'intervalle [0, 10]. En effet, suite à l'injection des trois paramètres d'entrée dans le système de logique floue, une sortie floue est produite. La sortie floue est caractérisée à l'aide des fonctions d'appartenance de sortie représentées par la figure 9.6. Ainsi, 9 niveaux sont pris en compte pour la chance de l'objet IoT: *Ultra Low, Very Low, Low, Average Low, Average, Average High, High, Very High, Ultra High*. Des fonctions d'appartenance triangulaires sont utilisées pour caractériser chaque niveau. La defuzzification permet de calculer la valeur numérique de la chance de l'objet IoT

correspondante à la sortie floue en déterminant le centre de gravité (*COG*) de la zone floue désigné par le système de logique floue de Mamdani.

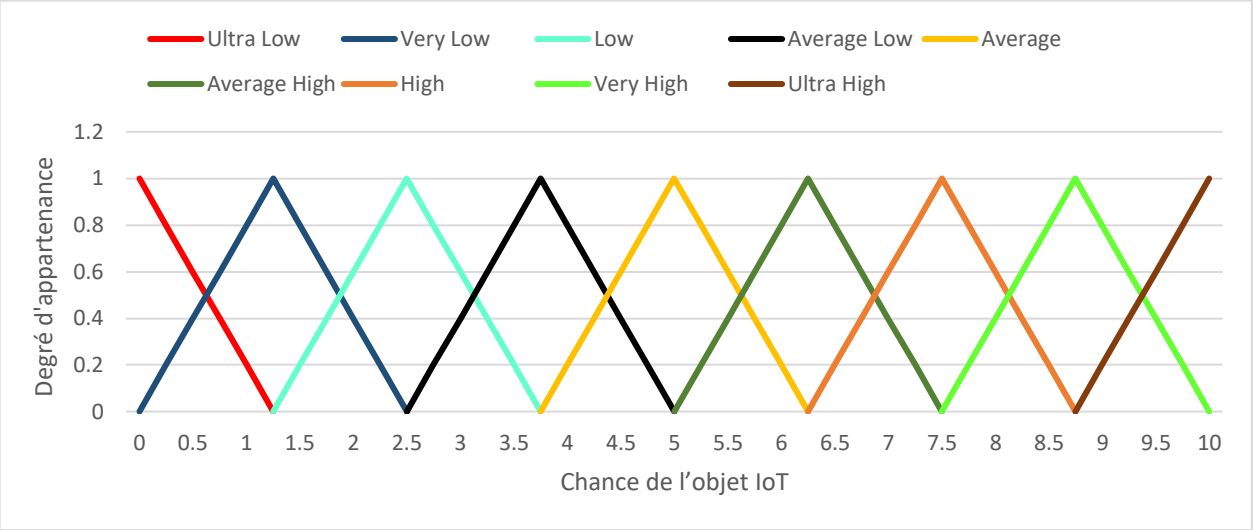

Figure 9.6 : Fonctions d'appartenance de la chance de l'objet IoT

9.3.4. Bases d'inférence et règles

Afin de déterminer la sortie floue du système, des règles prédéfinies sont utilisées à partir d'une base d'inférences. Ces règles sont décrites selon un format conditionnel de type Si et Alors. Notre système d'évaluation des objets IoT basé sur la logique floue comporte trois entrées ayant chacune 3 fonctions d'appartenance. Par conséquent, notre base d'inférences comporte 27 règles différentes.

Dans le cadre de notre système permettant d'évaluer des objets qui vont offrir des services IoT critiques ou non, nous spécifions deux bases d'inférences. La première correspond aux services IoT critiques en privilégiant le paramètre d'entrée précision alors que la deuxième correspond aux services IoT non critiques où tous les paramètres d'entrée sont équivalents. Le résultat de chaque règle correspond à un des 9 ensembles flou de la sortie. Afin de valider l'efficacité de nos règles, nous avons effectué différents jeux de simulations pour les adapter de manière à obtenir des résultats cohérents. Nous présentons respectivement dans le Tableau 9.1 et le Tableau 9.2 les bases d'inférences correspondant aux services IoT critiques et non critiques.

Tableau 9.1 : Base d'inférence des services IoT critiques

Précision	RE	Distance	Chance de l'objet IoT
Low	Low	Far	Ultra-Low
Mid	Low	Far	Average Low
High	Low	Far	Average
Low	Mid	Far	Very Low
Mid	Mid	Far	Average Low
High	Mid	Far	Average High
Low	High	Far	Low
Mid	High	Far	Average
High	High	Far	Very High
Low	Low	Mid	Very Low
Mid	Low	Mid	Average Low
High	Low	Mid	Average High
Low	Mid	Mid	Very Low
Mid	Mid	Mid	Average Low
High	Mid	Mid	High
Low	High	Mid	Low
Mid	High	Mid	Average
High	High	Mid	Very High
Low	Low	Close	Low
Mid	Low	Close	Average
High	Low	Close	High
Low	Mid	Close	Low
Mid	Mid	Close	Average
High	Mid	Close	Very High
Low	High	Close	Low
Mid	High	Close	Average
High	High	Close	Ultra-High

Tableau 9.2 : Base d'inférence des services IoT non critiques

Précision	***RE***	***Distance***	***Chance de l'objet IoT***
Low	Low	Far	Ultra-Low
Mid	Low	Far	Very Low
High	Low	Far	Low
Low	Mid	Far	Very Low
Mid	Mid	Far	Average Low
High	Mid	Far	Average
Low	High	Far	Low
Mid	High	Far	Average
High	High	Far	High
Low	Low	Mid	Very Low
Mid	Low	Mid	Average Low
High	Low	Mid	Average
Low	Mid	Mid	Average Low
Mid	Mid	Mid	Average
High	Mid	Mid	Average High
Low	High	Mid	Average
Mid	High	Mid	Average High
High	High	Mid	Very High
Low	Low	Close	Low
Mid	Low	Close	Average
High	Low	Close	High
Low	Mid	Close	Average
Mid	Mid	Close	Average High
High	Mid	Close	Very High
Low	High	Close	High
Mid	High	Close	Very High
High	High	Close	Ultra-High

9.3.5. Scénario d'utilisation du système d'évaluation

Par définition, une fonction d'appartenance détermine le degré d'appartenance de chaque paramètre à un ensemble flou. Le degré d'appartenance correspond à un intervalle variant de 0 à 1. Pour chaque entrée, une valeur numérique est injectée dans le système flou, ce qui donne un degré d'appartenance spécifique à l'ensemble flou déclenché par cette valeur. Chaque combinaison d'entrées floues (i.e., les sorties du composant de fuzzification) permet de déclencher une ou plusieurs règles de la base d'inférence. Le résultat de chaque règle permet de choisir une des fonctions d'appartenance de l'ensemble floue de sortie (i.e., *Ultra-Low*, *Very High*, etc.).

Le degré d'appartenance à l'ensemble flou de sortie déclenché par la règle correspond au minimum des degrés d'appartenance des entrées. L'ensemble des zones formées par la concaténation des zones déclenchées par les différentes règles considérées forment la zone de la chance de l'objet IoT résultant. Par conséquent, comme nous utilisons la méthode *COG* de Mamdani pour la défuzzification, le centre de gravité de la zone désignée est calculé et considéré comme étant la chance de l'objet IoT résultant de ces paramètres d'entrée relatifs à l'objet.

Nous illustrons ce processus par la Figure 9.7 qui décrit un scénario d'utilisation de notre système d'évaluation d'un objet dans le cadre de l'offre d'un service IoT critique. Ce scénario correspond à un objet ayant une distance de 80 m qui le sépare de la *LL-Gw* mais aussi un pourcentage d'énergie résiduelle de 31% et une précision de 99%. Dans ce contexte, une distance de 80 m correspond à un degré d'appartenance de 1 pour la fonction d'appartenance relative à la variable linguistique *Far* et 0 pour les fonctions d'appartenance relatives à *Mid* et *Close*. De plus, un pourcentage d'énergie résiduelle de 31% correspond à un degré d'appartenance de 0,36 pour la fonction d'appartenance relative à *Low*, à un degré de 0,18 pour la fonction *Mid* et à un degré 0 pour la fonction *High*. Enfin, une précision de 99% correspond à un degré d'appartenance de 1 pour la fonction d'appartenance relative à *High* et à un degré de 0 pour les fonctions *Mid* et *Low*. Par conséquent, deux règles sont déclenchées avec ces valeurs d'entrée (voir Tableau 9.1).

La première règle (Distance=*Far* & Energie Résiduelle= *Low* & Précision=*High*) permet d'obtenir une chance de l'objet IoT appartenant à la fonction de sortie *Average*. Pour déterminer le degré d'appartenance à la fonction de sortie correspondante, nous considérons, selon le modèle de Mamdani, la valeur minimale des degrés d'appartenance des paramètres d'entrée aux fonctions d'appartenance déclenchées. Pour la première règle, le minimum correspond à *min* (1 ; 0,36 ; 1) = 0,36 (voir Figure 9.7). En ce qui concerne la deuxième règle déclenchée (Distance=*Far* & Energie Résiduelle=*Mid* & Précision=*High*), la chance de l'objet IoT appartient à la fonction de sortie *Average High* avec un degré d'appartenance de *min* (1 ; 0,18 ; 1) = 0,18 (voir Figure 9.7).

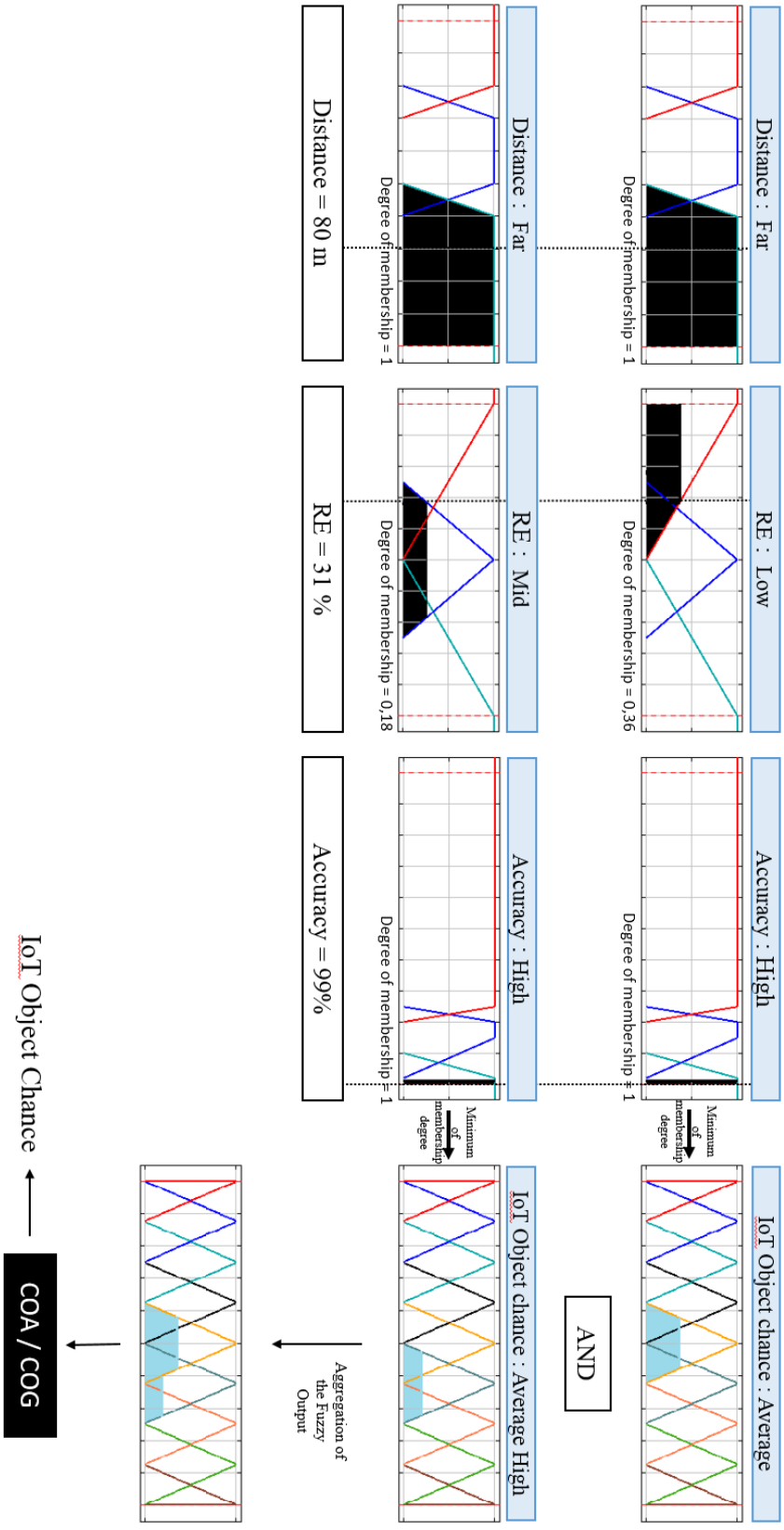

Figure 9.7 : Processus du système d'évaluation pour le calcul de la chance de l'objet IoT

Afin de calculer la valeur numérique globale de la chance de l'objet IoT, nous déterminons le centre de gravité de l'agrégation des zones de sortie correspondantes aux règles déclenchées. Si les fonctions d'appartenance de sortie des règles se chevauchent, le maximum est pris en compte. Nous illustrons l'agrégation de ces zones dans le cas de notre scénario d'utilisation grâce à la Figure 9.8. Nous calculons le centre de gravité de la zone colorée de la Figure 9.8 qui correspond à la valeur de la chance de l'objet IoT en utilisant l'équation 9.5.

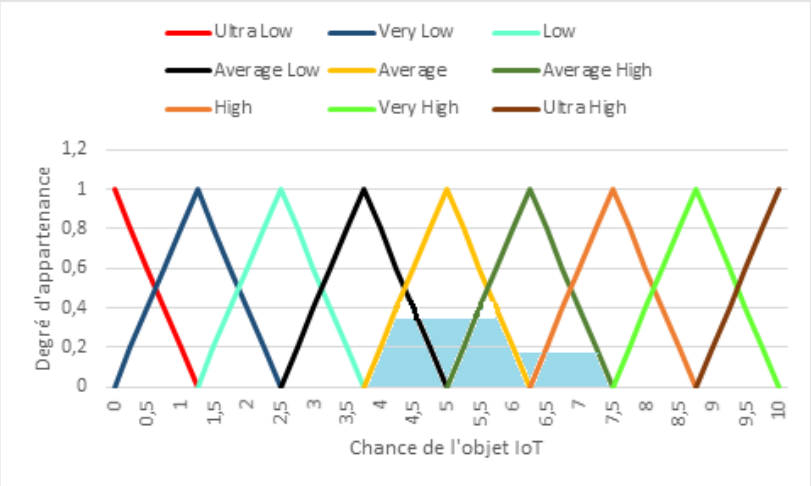

Figure 9.8 : Agrégation des zones de sortie correspondantes aux règles déclenchées

Nous calculons grâce à cette équation 9.5 la valeur globale de la chance de l'objet IoT en déterminant le centre de la surface de la fonction de sortie (*COG*). Dans cette équation, $\mu(x)$ permet de définir la zone de sortie globale (i.e., zone coloriée en bleu dans la Figure 9.8).

$$IoT\ Object\ Chance = \frac{\int x * \mu(x) dx}{\int \mu(x) dx} \qquad (9.5)$$

Ainsi, afin de calculer cette valeur globale de la chance de l'objet IoT, nous devons morceler la courbe μ (x) en des courbes simples ayant des équations bien connues. Par conséquent, $\mu(x)$ correspondant au résultat de notre cas d'utilisation (voir Figure 9.8) peut être représenté selon l'équation 9.6 :

$$\mu(x) = \begin{cases} mx + c & 3.75 \leq x < 4.25 \\ 0.36 & 4.25 \leq x < 5.75 \\ nx + d & 5.75 \leq x < 6.00 \\ 0.18 & 6.00 \leq x < 7.25 \\ ex + k & 7.25 \leq x < 7.50 \end{cases} \qquad (9.6)$$

Les coefficients *m*, *c*, *n* et *d* sont calculés en considérant deux points différents de chaque partie de la courbe. Par conséquent, nous utilisons les valeurs trouvées de ces coefficients pour transformer μ (x) comme le montre l'équation 9.6'.

$$\mu(x) = \begin{cases} 0.72x - 2.7 & 3.75 \leq x < 4.25 \\ 0.36 & 4.25 \leq x < 5.75 \\ -0.72x + 4.5 & 5.75 \leq x < 6.00 \\ 0.18 & 6.00 \leq x < 7.25 \\ -0.72x + 5.4 & 7.25 \leq x < 7.50 \end{cases} \quad (9.6')$$

Par conséquent, nous déterminons, grâce à l'équation 9.5, que l'objet que nous avons considéré dans notre scénario d'utilisation du système d'évaluation obtient une valeur de la chance de l'objet IoT égale à 5.15 comme le montre les détails de calcul suivant :

$$IoT\ Object\ Chance = \frac{\int_{3.75}^{4.25}(0.72\ x-2.7)x\ dx + \int_{4.25}^{5.75} 0.36\ x\ dx + \int_{5.75}^{6}(-0.72x+4.5)x\ dx + \int_{6}^{7.25} 0.18\ x\ dx + \int_{7.25}^{7.5}(-0.72x+5.4)x\ dx}{\int_{3.75}^{4.25}(0.72\ x-2.7)\ dx + \int_{4.25}^{5.75} 0.36\ dx + \int_{5.75}^{6}(-0.72x+4.5)\ dx + \int_{6}^{7.25} 0.18\ dx + \int_{7.25}^{7.5}(-0.72x+5.4)\ dx} = 5.15$$

Cette valeur calculée de la chance de l'objet IoT sera utilisée par l'algorithme d'auto-optimisation énergétique proposé dans la section 9.4.2.

9.4. Design et algorithme de l'auto-optimisation de la consommation énergétique dans l'IoT

9.4.1. Design de l'auto-optimisation énergétique

Après l'utilisation de notre système d'évaluation pour calculer les valeurs de la chance de l'objet IoT relatives aux différents objets concernés par l'offre d'un service IoT particulier, la *LL-Gw* doit prendre une décision concernant les objets à activer afin de récolter les données relatives à ce service tout en optimisant la consommation énergétique. Dans ce contexte, nous spécifions dans le *FSM* de la Figure 9.9 les différents états de la *LL-Gw* durant ce processus d'auto-optimisation énergétique.

Ainsi, dans l'état S0, la *LL-Gw* attend la souscription d'un contrat *iSLA* entre un l'*IoT-SP* et un client concernant un service IoT afin de recevoir de la *HL-Gw* les caractéristiques de ce service y compris le niveau de précision spécifié dans le *gSLA* et passe ainsi à l'état S1. Dans cet état S1, la *LL-Gw* initie le service IoT en utilisant l'identifiant de l'application IoT ainsi que les groupes d'objets IoT concernés par cette application IoT et ce grâce aux informations reçus de la HL-Gw avant de passer à l'état S2. Par la suite, la *LL-Gw* reçoit les paramètres d'entrée du système d'évaluation de logique floue concernant les différents objets participant à ce service, et ce afin de déterminer leurs chances (i.e., Chance de l'objet IoT) et utiliser l'algorithme de sélection des objets (cf. 9.4.2) avant de passer à l'état S3. Ainsi, la *LL-Gw* se base sur les résultats de cet algorithme afin d'envoyer les configurations d'activation ou non aux différents objets évalués en utilisant leurs interfaces de type effecteur. Par la suite, elle passe à l'état S4. Dans cet état, la *LL-Gw* reçoit les données des différents objets activés. Ensuite, si l'intervalle de mise à jour des configurations des objets pour l'auto-optimisation énergétique arrive à expiration, la *LL-Gw* passe à l'état S2 et exécute à nouveau l'algorithme de sélection des objets avec les nouvelles valeurs calculées par notre système

d'évaluation de logique floue. Sinon, si dans l'état S4, la *LL-Gw* reçoit une notification concernant la fin du service IoT, alors elle passe à l'état S0 dans l'attente de la mise en place d'un nouveau service.

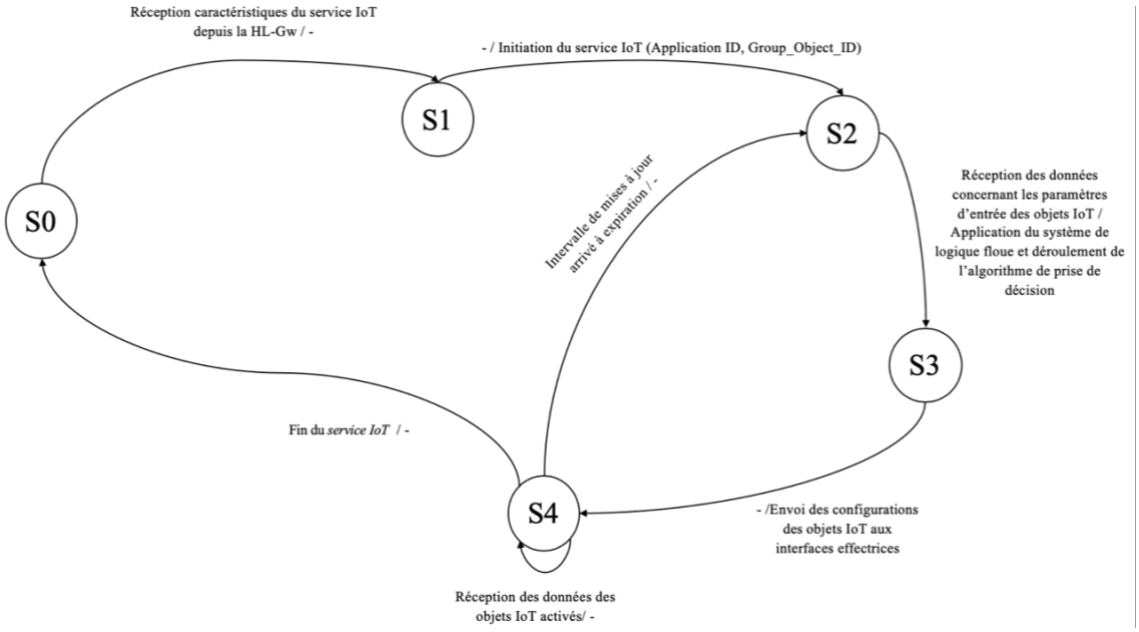

Figure 9.9 : FSM de la LL-Gw concernant l'auto-optimisation énergétique

Les différents échanges entre les composants de la couche sensing de l'architecture IoT (i.e., Objets IoT, *LL-Gw* et *HL-Gw*) permettant de réaliser cette fonctionnalité d'auto-optimisation énergétique sont spécifiés à travers le *MSC* de la Figure 9.10. Ainsi, suite à l'établissement de l'*iSLA* entre l'*IoT-C* et l'*IoT-SP*, la *HL-Gw* envoie à la *LL-Gw* les caractéristiques du service IoT décrits dans le *gSLA* correspondant. Ces caractéristiques sont formées par l'identifiant de l'application permettant d'offrir le service, les groupes d'objets IoT concernés par le service et la qualité des données pour ce service. Dans ce contexte, la qualité des données se traduit par l'écart type entre les informations remontées par les différents objets à travers leurs interfaces de type capteur ainsi que la fréquence de cette remontée des données. Après la réception de ces caractéristiques, la *LL-Gw* initialise le service IoT avec les objets concernés et attend le retour de ces objets concernant les paramètres d'entrée de notre système de logique floue d'évaluation périodiquement selon un intervalle de mise à jour des configurations de l'auto-optimisation énergétique. Cet intervalle dépend du type d'application IoT souscrite au niveau de l'*iSLA*. A la réception de ces informations, la *LL-Gw* détermine les chances de tous les objets IoT concernées et exécute l'algorithme de sélection qui affecte une fréquence de remontée importante aux objets activés et une fréquence de remontée d'information faible aux objets non activées (i.e., une remontée moins importante d'informations à la *LL-Gw*. La fréquence d'application de la logique floue doit être un multiple de la fréquence de remontée des objets non activés. Par la suite, la *LL-Gw* communique les configurations aux différents objets via l'interface de type effecteur et elle commence à recevoir des différents objets les données relatives au service suivant leur fréquence de remontée d'informations jusqu'à la fin de l'intervalle de mise à jours des configurations de l'auto-optimisation énergétique pour recommencer le processus à partir de

l'évaluation des objets en se basant sur des paramètres d'entrée actualisés.

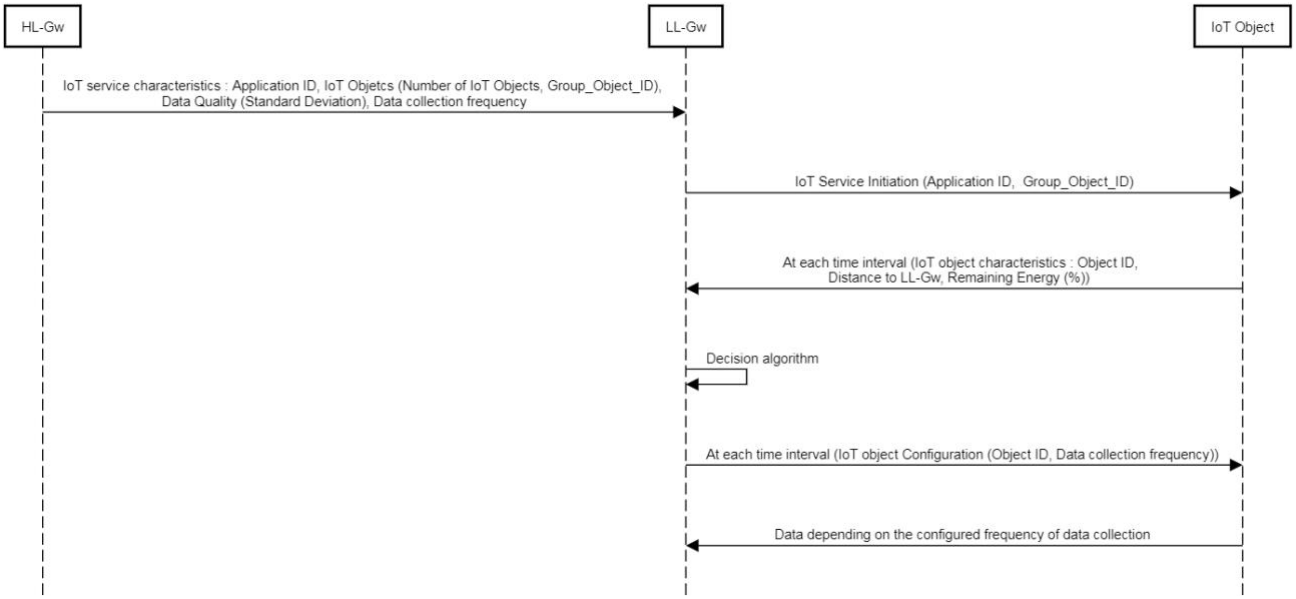

Figure 9.10 : MSC d'auto-optimisation énergétique

9.4.2. Algorithme de l'auto-optimisation énergétique

Afin de choisir les objets IoT à activer ainsi que le nombre d'objets à activer au cours de chaque période d'évaluation (intervalle de mise à jour d'auto-optimisation énergétique), la *LL-Gw* exécute un algorithme de sélection des objets suivant le type du service IoT spécifié dans le contrat de niveau de service. Ainsi, si le service IoT concerne une application critique, la précision des données remontées est un paramètre de *QoS* très important pour ce service. Par conséquent, la *LL-Gw* active autant d'objets que nécessaire pour arriver à la précision demandée. De plus, la *LL-Gw* doit activer au moins un nombre minimal d'objets pour chaque intervalle d'évaluation. Ce fonctionnement est décrit par l'algorithme de la Figure 9.11. Ainsi, la *LL-Gw* active dans un premier temps le nombre minimal d'objets nécessaires pour ce type de service. Pour l'activation de ces objets, la *LL-Gw* les configure avec le paramètre *DGI* (Data Generation Interval) correspondant, appelé *DGI* minimum, qui est dans tous les cas inférieur à l'intervalle d'évaluation. Les objets activés sont les objets ayant les plus hautes valeurs de chance calculées par notre système d'évaluation. Si la précision des données relatives au service IoT critique n'est pas respectée suite à l'activation du nombre minimal d'objets IoT (i.e., la précision moyenne des objets activées est inférieure à la précision demandée par le service), la *LL-Gw* procède à l'activation d'un objet IoT supplémentaire à la fois (i.e., affecter le *DGI* minimum à un objet) pour arriver à la précision demandée par le service. Les objets qui n'ont pas été configurés avec le *DGI* minimum doivent être alors non activés et par conséquent configurés avec le *DGI* maximum. La configuration des objets avec le *DGI* maximum permet à ces objets de recevoir les balises IEEE 802.15.4 et d'envoyer les données selon l'intervalle

de mise à jour des configurations afin d'actualiser les paramètres d'entrée de notre système d'évaluation basé sur la logique floue.

Algorithme Auto-optimisation énergétique – Services IoT critiques

```
1:  Receive data from all objects ()
2:  Mean Value (Data)
3:  if (IoT Objects > Minimum Number of Activated IoT Objects ) then
4:      i ← 0
5:      While ( i < Number of IoT Objects) do
6:          RE [i] ← Retrieve Residual Energy (IoT Object[i])
7:          Accuracy [i] ← Retrieve Accuracy (IoT Object[i])
8:          Distance [i] ← Retrieve Distance (IoT Object[i])
9 :         IoT Object Chance [i] = Fuzzy System (RE[i], Accuracy[i], Distance[i])
10:         i ← i + 1
11:     end while
12:     Order by IoT Object Chance (IoT Objects [], descending Order)
13:     i ← 0
14:     While (i < Minimum number of activated IoT Objects) do
15:         Set Data Generation Interval to minimum (IoT Object[i])
16:         i ← i + 1
17:     end while
18:     Mean Value (Activated IoT Objects)
19:     Tmp ← Accuracy (Activated IoT Objects)
20:     if (Tmp >= Desired Accuracy) then
21:         while (i < Number of IoT Objects) do
22:             Set Data Generation Interval to maximum (IoT Object[i])
23:             i ← i + 1
24:         end while
25:     else
26:         while (Tmp < Desired Accuracy) do
27:             Set Data Generation Interval to minimum (IoT Object[i])
28:             Mean Value (Activated IoT Objects )
29:             Tmp ← Accuracy (Activated IoT Objects )
30:             i ← i + 1
31:         end while
32:         While (i < Number of IoT Objects) do
33:             Set Data Generation Interval to maximum (IoT Object[i])
34:             i ← i + 1
35:         end while
36:     end if
37: end if
```

Figure 9.11 : Algorithme d'auto-optimisation énergétique pour les services IoT critiques

Par contre, si le service IoT concerne une application non critique, l'algorithme d'auto-optimisation énergétique permettant la sélection des objets au niveau de la *LL-Gw* (voir Figure 9.12)

active uniquement le nombre minimal d'objets IoT nécessaire pour ce type de service non critique. Les objets activés sont les objets ayant les plus grandes valeurs de chance et recevront le *DGI* minimum alors que les autres objets seront configurés avec le *DGI* maximum.

Algorithme Auto-optimisation énergétique – Services IoT non critiques

```
1:    Receive data from all objects ()
2:    Mean Value (Data)
3:    if (IoT Objects > Minimum Number of Activated IoT Objects ) then
4:        i ← 0
5:        While ( i < Number of IoT Objects) do
6:            RE [i] ← Retrieve Residual Energy (IoT Object[i])
7:            Accuracy [i] ← Retrieve Accuracy (IoT Object[i])
8:            Distance [i] ← Retrieve Distance (IoT Object[i])
9 :           IoT Object Chance [i] = Fuzzy System (RE[i], Accuracy[i], Distance[i])
10 :          i ← i + 1
11:       end while
12:       Order by IoT Object Chance (IoT Objects [], descending Order)
13:       i ← 0
14:       While (i < Minimum number of activated IoT Objects) do
15:           Set Data Generation Interval to minimum (IoT Object[i])
16:           i ← i + 1
17:       end while
18:       While (i < Number of IoT Objects ) do
19:           Set Data Generation Interval to maximum(IoT Object [i])
20:           i ← i + 1
21:       end while
22:   end if
```

Figure 9.12 : Algorithme d'auto-optimisation énergétique pour les services IoT non critiques

9.5. Validation et évaluation des performances de l'auto-optimisation énergétique

Dans le cadre de la validation et l'évaluation des performances de la fonction d'auto-optimisation spécifiée pour la gestion autonome de la consommation énergétique des objets IoT, nous définissions des scénarios de simulation permettant de montrer l'efficacité de cette nouvelle fonction ajoutée au niveau de la *LL-Gw*. Pour ce faire, nous étendons le simulateur OMNET++ en ajoutant la fonction d'auto-optimisation énergétique au code du modèle IEEE 802.15.4 déjà modifié avec notre implémentation de *QBAIoT*. Les caractéristiques de l'environnement de simulation OMNET++ considéré dans l'évaluation de l'auto-optimisation énergétique sont décrites dans le Tableau 9.3.

Tableau 9.3 : Paramètres des simulations

Paramètres	Valeurs
Taille du paquet (couche MAC)	50 octets
Topologie	Etoile
Nombre de coordinateur par WPAN	1
Fréquence	2.4 Ghz
Débit du canal	250 kb/s
Energie initiale dans l'objet	100 J = 8.5 mAh
Intervalle du système d'évaluation logique floue	10 s

Pour évaluer les performances de notre fonction d'auto-optimisation visant à étendre la durée de vie du système IoT, nous définissons la durée de vie du système IoT en fonction du type de service offert. Ainsi, nous considérons que notre système IoT offrant un service critique est opérationnel à condition qu'il y ait au moins deux objets IoT actifs (i.e., énergie résiduelle non nulle) capables de générer des données précises. Dans le cas de services IoT non critiques, le système est considéré comme étant opérationnel à condition qu'il y ait au moins deux objets IoT actifs indépendamment de la précision des données remontées.

Nous commençons l'évaluation de l'auto-optimisation énergétique en se basant sur les scénarios de simulation définis dans le Tableau 9.4. Ce tableau spécifie différentes caractéristiques de simulation dont la consommation énergétique des objets IoT. Ce paramètre définit la consommation énergétique de chaque objet IoT en fonction de la distance qui le sépare de la passerelle *LL-Gw* en se basant sur l'étude effectuée dans [196]. Ainsi, la consommation énergétique est de 10.1 mA pour l'envoi d'un paquet pour un objet IoT se situant à 80 m de la passerelle, 9.25 mA pour une distance de 60 m, 8.41 mA pour une distance de 40 m et 7.57 mA pour une distance de 20 m. En ce qui concerne l'énergie consommée pour la réception des balises, nous l'avons fixée à 0.33 mA.

Tableau 9.4 : Caractéristiques des scénarios de simulation 1 à 3

Scenario	Nombre d'objets	Précision	Distance	Min DGI	Max DGI	Consommation énergétique
1	4	99%	80 m	0.25 s	2 s	10.1 mA
2	6	99%	80 m	0.25 s	2 s	10.1 mA
3	8	99%	80 m	0.25 s	2 s	10.1 mA

Les scénarios de 1 à 3 correspondent à un système IoT offrant un service critique. Dans ce contexte, l'algorithme d'auto-optimisation énergétique permet de sélectionner deux objets à chaque intervalle d'évaluation comme les objets remontent des informations respectant la précision demandée de 99% par le service IoT et que le nombre minimal d'objets à activer est fixé à 2

Nous comparons dans la Figure 9.13 la durée de vie du système IoT, dans le cadre de ces trois premiers scénarios, pour une utilisation de *QBAIoT* sans auto-optimisation énergétique et avec auto-optimisation énergétique. Les résultats obtenus montrent, qu'avec ou sans auto-optimisation, un nombre plus élevé d'objets relatifs à un même service IoT permet d'avoir une durée de vie plus importante du système IoT correspondant. De plus la durée de vie du système IoT est toujours plus importante pour les trois scénarios lorsque nous utilisons l'auto-optimisation énergétique. Ainsi, sans auto-optimisation énergétique, lorsque le nombre d'objets IoT augmente, la durée de vie du système est légèrement améliorée. En effet, lorsque le nombre d'objets augmente, les risques de collisions sont plus importants et les objets ont un temps d'attente plus long avant la retransmission permettant d'économiser de l'énergie. D'autre part, avec l'auto-optimisation énergétique, la durée de vie du système IoT s'améliore d'une façon plus importante avec l'augmentation du nombre d'objets grâce à la désactivation des certains objets du système IoT.

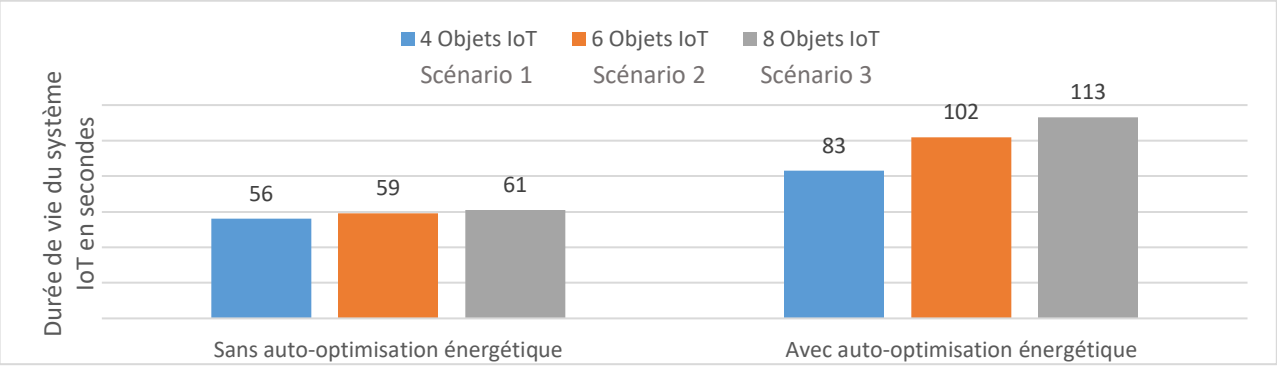

Figure 9.13 : Durée de vie du système IoT (scénarios 1 à 3)

Nous poursuivons les simulations d'évaluation de l'auto-optimisation énergétique avec les scénarios 4 et 5 décrits dans le Tableau 9.5. Ces scénarios correspondent à des objets IoT qui se trouvent à différentes distances de la *LL-Gw* avec différentes valeurs de précision de l'information remontée. La valeur de précision demandée par le service IoT critique considéré dans ces scénarios est fixée à 96%.

Tableau 9.5 : Caractéristiques des scénarios de simulation 4 et 5

Scenario	Nombre d'objets	Précision	Distance	Min DGI	Max DGI	Consommation énergétique
4	4	96% 97% 98% 99%	80m 60m 40m 20m	0.25 s	2 s	10.1 mA 9.25 mA 8.41 mA 7.57 mA
5	4	99% 98% 97% 96%	80 m 60 m 40 m 20 m	0.25 s	2 s	7.57 mA 8.41 mA 9.25 mA 10.1 mA

La Figure 9.14 illustre la comparaison de la durée de vie du système IoT pour les scénarios 4 et 5 avec et sans auto-optimisation énergétique. Nous pouvons noter que *QBAIoT* avec auto-optimisation énergétique assure une meilleure durée de vie du système pour les deux scénarios comparé à la non utilisation de cette fonction. Comme les scénarios 4 et 5 ont des caractéristiques différentes, des objets différents sont sélectionnés à chaque intervalle d'évaluation. Les résultats montrent que nous obtenons une meilleure durée de vie du système IoT dans le scénario 4 car les objets les plus proches possèdent les meilleures précisions de données sachant que l'algorithme d'auto-optimisation a permis la sélection de 2 objets à activer dans les deux scénarios.

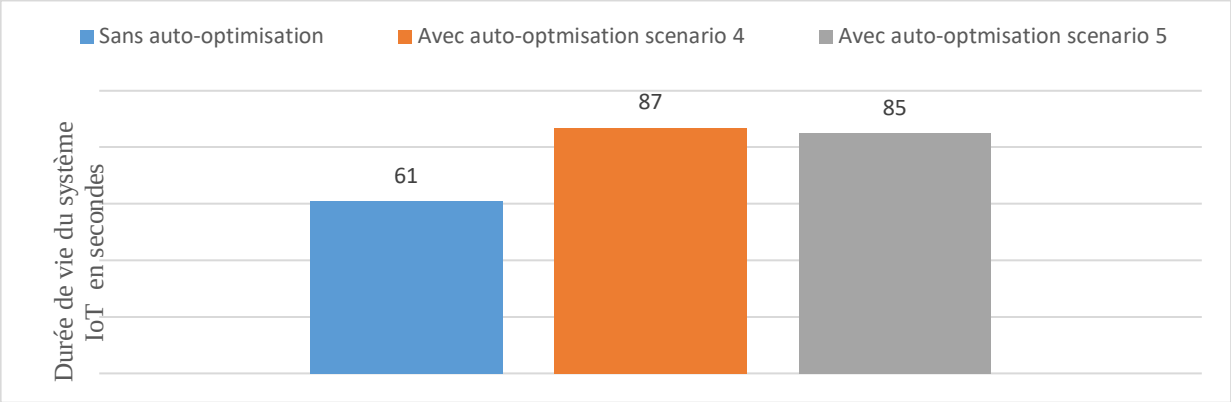

Figure 9.14 : Durée de vie du système IoT (scénarios 4 et 5)

D'autres scénarios de simulation sont présentés dans le Tableau 9.6 avec des objets mobiles qui changent de précisons suivant leurs positions.

Tableau 9.6 : Caractéristiques des scénarios de simulation 6 à 8

Scenario	Nombre d'objets	Précision	Distance	Min / Max DGI	Consommation énergétique
6	4	99% / 95% : 25s 99% / 95% : 50s 95% / 99% : 25s 95% / 99% : 50s	80 m / 40 m : 25 s 80 m / 40 m : 25 s 40 m / 80 m : 25 s 40 m / 80 m : 25 s	0.25 s / 2 s	10.1 mA / 8.41 mA 10.1 mA / 8.41 mA 8.41 mA / 10.1 mA 8.41 mA / 10.1 mA
7	4	99% / 95% : 25s 99% / 95% : 50s 95% / 99% : 25s 95% / 99% : 50s	80 m / 40 m : 25 s 80 m / 40 m : 25 s 40 m / 80 m : 25 s 40 m / 80 m : 25 s	0.25 s / 2 s	10.1 mA / 8.41 mA 10.1 mA / 8.41 mA 8.41 mA / 10.1 mA 8.41 mA / 10.1 mA
8	6	96% 97% 98% 99% 97% 98%	40 m	0.25 s/ 2 s	8.41 mA

Par exemple, les objets 1 et 3 des scénarios 6 et 7 changent de position et donc de précision à 25s et les objets 2 et 4 changent de précision à 50 s. Les scénarios 6 et 7 sont identiques sauf que pour le scénario 6 nous prenons en considération un service non critique et pour le scénario 7 nous prenons en compte un service critique. Par conséquent, deux algorithmes d'auto-optimisation énergétique différents sont utilisés et deux bases de connaissances différentes pour les règles de la logique floue sont considérées. Ainsi, le choix des objets effectué par la *LL-Gw* est différent dans ces deux scénarios. En ce qui concerne le scénario 8, nous simulons 6 objets ayant des précisions différentes mais à une même distance de la *LL-Gw* pour l'offre d'un service IoT critique. Enfin, la précision demandée par le service IoT dans le scénario 7 est fixée à 95% alors que pour le scénario 8, la précision demandée par le service est de 97%.

Nous comparons dans la Figure 9.15 la durée de vie du système pour les scénarios 6 et 7 avec et sans auto-optimisation énergétique. Nous pouvons noter que *QBAIoT* sans auto-optimisation énergétique offre une durée de vie du système IoT inférieure à celle des scénarios 6 et 7. De plus, la durée de vie du système IoT dans le scénario 6 est supérieure à celle du scénario 7.

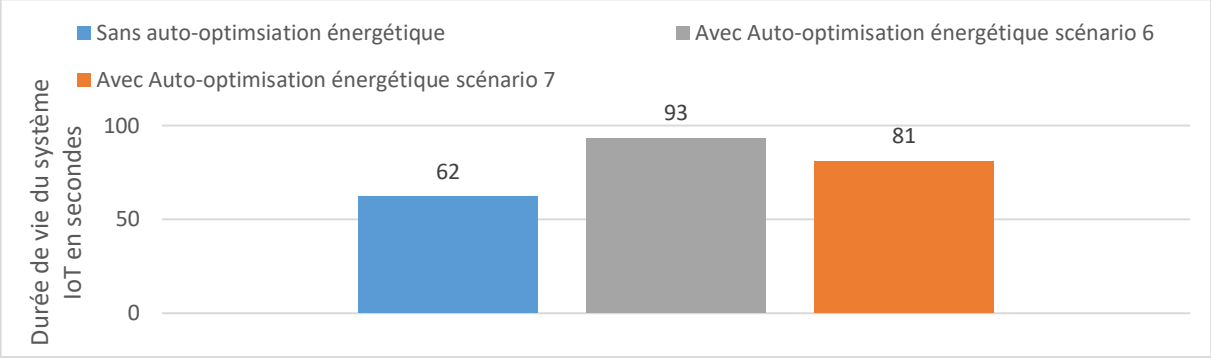

Figure 9.15 : Durée de vie du système IoT (scénarios 6 et 7)

En effet, les objets IoT activés à chaque intervalle d'évaluation de logique floue ne sont pas les mêmes dans les deux scénarios (voir Figure 9.16) étant donné que différents ensembles de règles sont pris en compte donnant lieu à différentes valeurs de chance de l'objet IoT pour les objets.

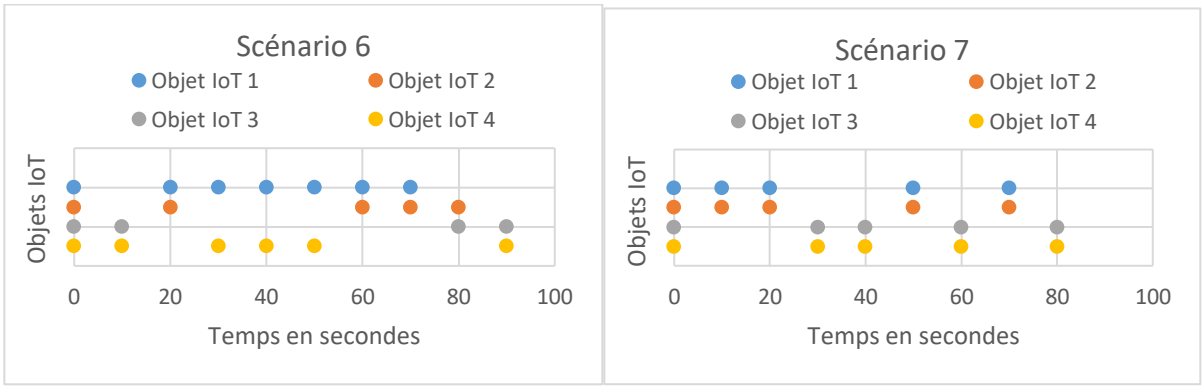

Figure 9.16 : Sélection des objets (scénarios 6 et 7)

Le dernier scénario de simulation, à savoir le scénario 8, a pour objectif de montrer l'efficacité de l'algorithme d'auto-optimisation énergétique dans la sélection de plusieurs objets IoT afin de respecter la précision demandée par un service IoT critique. Dans ce scénario 8 (voir Tableau 9.6), la précision des données émanant de certains objets IoT est inférieure à 97%. Par conséquent, il est possible grâce à cet algorithme d'activer plus que le nombre minimal d'objets (i.e., 2 dans notre cas d'utilisation) pour récupérer les informations relatives au service IoT critique. La Figure 9.17 montre les résultats relatifs au scénario 8 concernant la précision des données émanant des objets IoT activés dans le cas d'utilisation de l'auto-optimisation énergétique et émanant de tous les objets dans le cas de non utilisation de l'auto-optimisation. Nous pouvons noter que *QBAIoT* avec auto-optimisation énergétique permet d'obtenir une meilleure précision des données comparée à *QBAIoT* sans auto-optimisation sauf pour la période d'évaluation entre 50s et 60s.

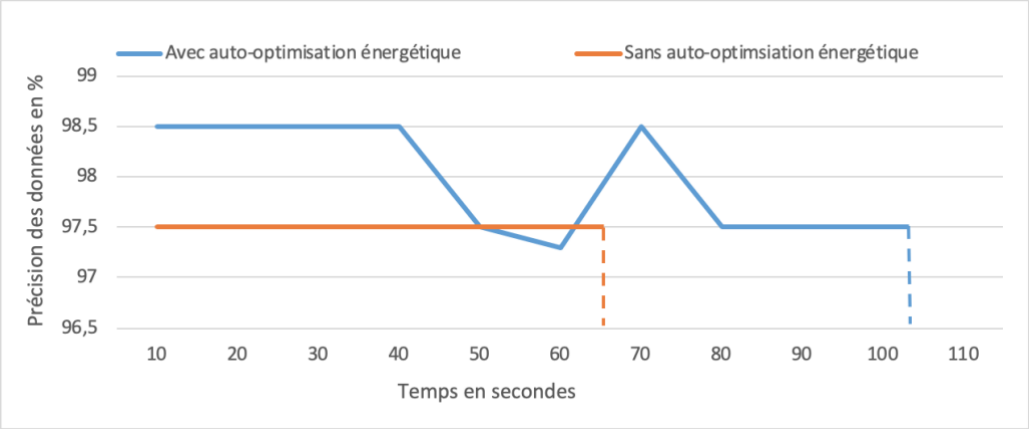

Figure 9.17 : Précision moyenne de l'information remontée (scénario 8)

En effet, à 60 s, nous pouvons noter que la précision des données avec auto-optimisation énergétique est inférieure à celle obtenue sans auto-optimisation énergétique. En fait, à 60 s, 3 objets IoT sont activés (voir Figure 9.18) avec l'algorithme d'auto-optimisation car la précision des objets initialement activés ne respecte pas la précision souhaitée de 97%.

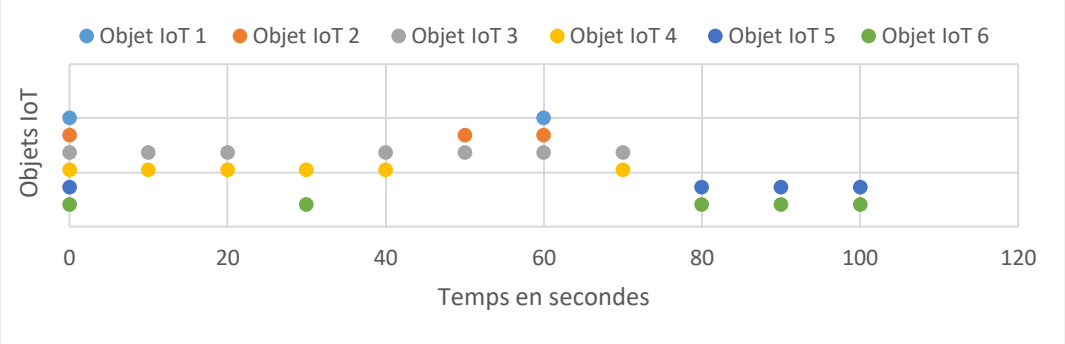

Figure 9.18 : Sélection des objets (scénario 8)

Par conséquent, un objet supplémentaire a été activé pour obtenir une valeur supérieure à 97%. La précision pour *QBAIoT* sans auto-optimisation énergétique est toujours fixe et égale à 97,5 % car tous les objets IoT sont activés jusqu'à l'épuisement de leurs énergies. Ainsi, la durée de vie du système IoT sans auto-optimisation énergétique est de 67 s, tandis qu'elle est de 102 s dans le cas d'utilisation de la fonction d'auto-optimisation énergétique.

En effet, à 102 secondes, plus que deux objets sont vivants (voir Figure 9.19) avec une énergie résiduelle non nulle mais la précision remontée par ces deux derniers objets ne respecte pas la précision souhaitée pour le service IoT. Par conséquent, nous ne pouvons pas considérer le système IoT comme entièrement fonctionnel ce qui explique la valeur de 102s trouvée comme durée de vie du système IoT malgré les deux objets avec une énergie résiduelle non nulle.

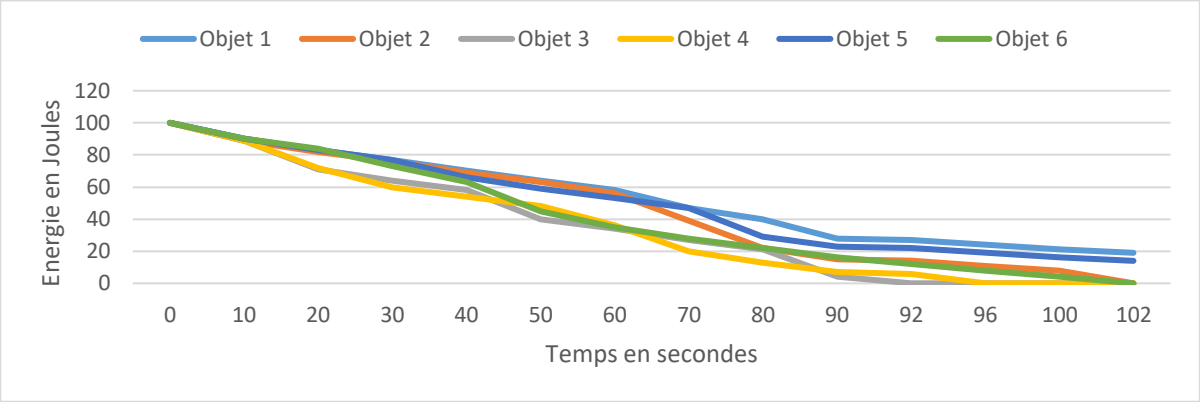

Figure 9.19 : Energie résiduelle des différents objets utilisant l'auto-optimisation énergétique (scénario 8)

9.6. Conclusion

L'auto-optimisation énergétique est nécessaire dans un environnement IoT afin d'étendre la durée de vie du système IoT comportant des objets IoT avec des contraintes énergétiques. Afin de minimiser la consommation énergétique des objets IoT, nous avons proposé dans ce chapitre un système d'évaluation basé sur un modèle de logique floue de type Mamdani ainsi qu'un algorithme d'auto-optimisation énergétique qui permettent de sélectionner les objets à activer pour l'offre d'un service IoT en se basant sur plusieurs critères comme la distance, la précision et l'énergie résiduelle. Nous avons justifié l'utilisation de chacun de ces critères et nous avons détaillé leur méthode de calcul et d'utilisation dans notre système d'évaluation. Nous avons démontré grâce à l'évaluation des performances de plusieurs scénarios de simulation que ce processus d'auto-optimisation énergétique a permis d'étendre la durée de vie du système IoT.

Chapitre 10 – Conclusion générale

10.1. Bilan

Les travaux de recherche menés dans le cadre de cette thèse ont porté sur la conception d'un Framework de gestion autonome de la qualité de service et de la sécurité dans un environnement IoT en accord avec les recommandations des organismes de standardisation. Ce Framework est basé sur une architecture de *QoS* à trois couches (i.e., sensing, réseau, cloud) permettant à un *IoT-SP* d'établir avec un *IoT-C* un contrat de niveau de service, appelé *iSLA*, afin de satisfaire les attentes du client décrits grâce à des paramètres de performance qualitatifs ou quantitatifs. L'*iSLA* établi entre l'*IoT-C* et l'*IoT-SP* se base sur d'autres types de *SLA* établis entre l'I*oT-SP* et le *CSP* (i.e., *cSLA*), l'*IoT-SP* et le *NSP* (i.e., *nSLA*) ainsi qu'un *SLA* interne concernant la couche sensing (i.e., *gSLA*). Afin de respecter les attentes de *QoS* du client, nous avons proposé *QBAIoT*, un mécanisme de contrôle d'accès au canal basé sur la *QoS* au niveau de la couche sensing de notre architecture. Ce mécanisme nous a permis de définir différentes périodes de contention d'accès au canal (i.e., *QoS CAP*) en fonction des types de trafics générés par les objets IoT pour assurer un traitement différencié dans un environnement IoT. *QBAIoT* se base sur le standard IEEE 802.15.4 pour la communication entre les objets IoT et la passerelle de bas niveau de notre architecture de *QoS* (i.e., *LL-Gw*). Cette méthode d'accès au canal a été ensuite gérée d'une manière autonome à travers deux fonctions de la boucle de contrôle fermée *MAPE-K* : l'auto-configuration et l'auto-optimisation. De même, la consommation énergétique des objets IoT utilisant cette méthode a été optimisée d'une manière autonome en utilisant un système de logique floue permettant de choisir les objets actifs dans la récolte d'informations relatives à un service IoT. Ensuite, notre architecture a été étendue pour inclure la garantie d'un niveau de sécurité dans un environnement IoT. Afin d'assurer ce niveau de sécurité dans l'IoT, nous avons proposé un mécanisme, appelé *IoT-MAAC*, permettant le contrôle d'accès des objets IoT aux passerelles *LL-Gw*. Ce mécanisme de sécurité prend en compte plusieurs attributs dont la confiance des objets existants dans l'environnement.

Ainsi, dans une première contribution de cette thèse, nous avons proposé une architecture de garantie de QoS adaptée à un environnement IoT. Dans ce contexte, nous avons spécifié trois types de *SLA* (i.e., *iSLA*, *cSLA*, *nSLA* et *gSLA*) ainsi que la méthode d'accès au canal basé sur la *QoS*, appelée *QBAIoT*, qui permet aux objets IoT d'avoir un accès différencié aux ressources de la passerelle de bas niveau de la couche sensing de notre architecture. Nous avons montré que *QBAIoT* est capable de respecter les exigences des trafics avec différentes contraintes dans l'environnement IoT tout en priorisant les trafics critiques temps réel. Notre évaluation de performances de *QBAIoT* a pris en compte les paramètres de *QoS* tels que le délai, le taux de livraison des paquets, etc. Nous avons comparé les performances de *QBAIoT* au standard IEEE 802.15.4 ainsi que la méthode *SDA-CSMA/CA*. Les résultats obtenus ont montré de meilleures performances pour *QBAIoT* en termes de délai moyen et taux de livraison des paquets pour les trafics temps réels (i.e., *RTMC* et *RTNMC*)

La deuxième contribution de cette thèse porte sur la sécurité dans l'environnement IoT. Ainsi, nous avons étendu notre architecture de *QoS* pour prendre en compte la sécurité en spécifiant des *SLA* de sécurité (i.e., *SECiSLA*, *SECcSLA* et *SECgSLA*) ainsi qu'un mécanisme de sécurité permettant le contrôle d'accès des différents objets IoT aux passerelles de bas niveau de la couche sensing, appelé *IoT-MAAC*. Nous avons utilisé les deux standard *XACML* et *SAML* pour concevoir les échanges et les attributs de notre mécanisme de contrôle d'accès *IoT-MAAC*. Ce dernier est basé sur un ensemble d'attributs spécifiques à l'environnement IoT tels que le niveau de confiance des objets quantifié à travers différents paramètres en utilisant des fonctions d'appartenance et des règles d'un système de logique floue. Nous avons montré l'efficacité d'*IoT-MAAC* grâce à un scénario d'utilisation permettant de mettre en place un contrôle d'accès sécurisé pour des objets IoT au niveau de la couche sensing de l'architecture IoT.

Dans une troisième contribution de cette thèse, nous avons proposé une gestion autonome de notre méthode d'accès QBAIoT. Ainsi, nous avons conçu une première fonction d'auto-configuration permettant de choisir et d'adapter la configuration initiale des *QoS CAP*, spécifiées par *QBAIoT*, en fonction de l'évolution des classes de trafic existantes dans l'environnement IoT de la *LL-Gw*. Cette fonction d'auto-configuration est complétée par une deuxième fonction d'auto-optimisation permettant d'adapter *QBAIoT* en fonction de l'utilisation des *IT* alloués aux *QoS CAP*. Nous avons montré grâce à différents scénarios de simulation que les deux fonctions d'auto-configuration et d'auto-optimisation permettent d'améliorer les performances de *QBAIoT* et d'avoir une meilleure utilisation des ressources disponibles dans la couche sensing de l'architecture IoT. Cette gestion autonome a été étendue pour assurer une optimisation de la consommation énergétique des objets dans l'environnement IoT. Cette optimisation est basée sur un système de logique floue permettant de quantifier la chance de l'objet pour être activé dans le processus de remontée des informations d'un service IoT à la passerelle de bas niveau. Nous avons montré, grâce à différents scénarios de simulation, que le mécanisme d'auto-optimisation de la consommation énergétique des objets IoT nous a permis d'augmenter la durée de vie du système IoT ce qui est primordial pour ce type d'environnement.

10.2. Perspectives

Les contributions présentées dans ce manuscrit de thèse ouvrent plusieurs perspectives pour des travaux futurs afin d'améliorer les performances de nos propositions. Nous mettons en évidence dans ce qui suit quelques perspectives principales.

Le mécanisme *QBAIoT* repose sur une configuration initiale qui dépend des caractéristiques de l'environnement IoT en question. Cette configuration est choisie à travers une fonction de gestion autonome et elle est optimisée grâce à la fonction d'auto-optimisation exécutée selon un intervalle de temps fixe et spécifiée d'une manière statique. Afin d'assurer un mécanisme totalement autonome, il faudrait intégrer un processus d'apprentissage permettant de choisir l'intervalle d'exécution de la boucle de contrôle fermée *MAPE-K* d'une manière autonome et dynamique afin de s'adapter à la fréquence d'occurrence de changements dans l'environnement IoT considéré. D'autre part, notre

contribution *QBAIoT* prend en compte un mode bien spécifique du standard IEEE 802.15.4 à savoir le mode « Slotted CSMA/CA in beacon enabled PAN ». Pour les travaux futurs, il est envisageable d'étudier la possibilité d'extension de notre contribution pour qu'elle soit applicable à d'autres modes du standard IEEE 802.15.4. Enfin, l'évaluation de la métrique de *QoS* concernant la précision des informations est effectuée grâce au calcul de l'écart type. Il serait intéressant de trouver d'autres mécanismes plus spécifiques pour quantifier cette précision et par conséquent quantifier la qualité des données remontées par différents types d'objets IoT.

En ce qui concerne notre contribution relative à l'optimisation de la consommation énergétique des objets IoT, nous pouvons envisager d'enrichir les paramètres que nous prenons en considération dans le choix des objets à activer en ajoutant des paramètres relatifs à la sécurité des objets ce qui permettrait d'améliorer l'adéquation avec différents types d'environnement IoT. De plus, le nombre minimum d'objets IoT à activer est choisi d'une manière statique. Ce choix peut être automatisé en fonction de la valeur de précision demandée par l'application IoT tout en maximisant la durée de vie des systèmes IoT.

Une autre perspective de recherche concerne notre mécanisme de contrôle d'accès des objets à la passerelle IoT. Ce mécanisme est basé sur des attributs propres à l'environnement IoT et il serait possible d'étendre ces attributs afin d'assurer un contrôle d'accès des passerelles *LL-Gw* aux passerelles *HL-Gw* et contribuer ainsi à une sécurité globale au niveau de la couche sensing qui est la plus vulnérable de notre architecture IoT. Enfin, nous pouvons envisager d'évaluer l'impact de la mise en place de la sécurité globale dans la couche sensing sur la durée de vie du système IoT en implémentant ces mécanismes sur une plateforme de tests.

Liste des publications

Revues internationales avec comité de lecture

1. **A. Khalil**, N. Mbarek, O. Togni, "*A self-optimizing QoS Based Access for IoT environments*", Wireless Personal Communications, Springer, [Under Review].
2. **A. Khalil**, N. Mbarek, O. Togni, "*Towards Service Level Guarantee within IoT Sensing Layer*", International Journal On Advances in Internet Technology, IntTech19v12n12, ISSN: 1942-2652, June 2019, IARIA. Accessible via : http://www.iariajournals.org/internet_technology/inttech_v12_n12_2019_paged.pdf
3. **A. Khalil**, N. Mbarek, O. Togni, "*Self-Configuring IoT Service QoS Guarantee Using QBAIoT*", Computers, MDPI Swiss, no 4:64, First online November 2018, [DOI : https://doi.org/10.3390/computers7040064].

Chapitre de livre

4. **A. Khalil**, N. Mbarek, O. Togni, "*Gestion du niveau de service dans l'Internet des Objets*", dans le livre « Gestion du niveau de service dans les environnements émergents », ISTE & Wiley, [En Production].

Conférences Internationales avec actes et comité de lecture

5. **A. Khalil**, N. Mbarek, O. Togni, "*IoT-MAAC: Multiple Attribute Access Control for IoT environments*", IEEE Consumer Communications & Networking Conference (IEEE CCNC 2020), Publisher : IEEE, January 10-13, Las Vegas, USA, 2020.
6. **A. Khalil**, N. Mbarek, O. Togni, "*Fuzzy Logic based security trust evaluation for IoT environments*", IEEE/ACS International Conference on Computer Systems and Applications (IEEE/ACS AICCSA 2019), Publisher : IEEE, November 2 -8, Abu Dhabi, UAE, 2019.
7. **A. Khalil**, N. Mbarek, O. Togni, "*QBAIoT: QoS Based Access for IoT Environments*", Advanced International Conference on Telecomunications (AICT 2018), Publisher: ThinkMind, pp 38-43, July 22 - 26, Barcelona, Spain, 2018.
8. **A. Khalil**, N. Mbarek, O. Togni, "*IoT Service QoS Guarantee Using QBAIoT Wireless Access Method*", International Conference on Mobile, Secure and Programmable Networking (MSPN 2018), Publisher : Springer, pp 157-173, June 18- 20, Paris, France, 2018 [DOI: https://doi.org/10.1007/978-3-030-03101-5_15].
9. **A. Khalil**, N. Mbarek, O. Togni, "*Service level guarantee framework for IoT environments*", International Conference on Internet of Things and Machine Learning (ACM IML 2017), Publisher : ACM, No 50, October 17-18, Liverpool, UK, 2017 [DOI: https://doi.org/10.1145/3109761.3158393].

Bibliographie

[1] « About the Lab | autoid ». [En ligne]. Disponible sur: https://autoid.mit.edu/about-lab. [Consulté le: 25-sept-2019].

[2] K. Ashton, « That “Internet of Things” Thing », RFID Journal, 2009. [En ligne]. Disponible sur : https://www.rfidjournal.com/articles/pdf?4986. [Consulté le: 04-févr-2019].

[3] « [Infographie] Histoire de l’internet des objets au fil du temps », Meilleure Innovation, 2014. [En ligne]. Disponible sur : https://www.meilleure-innovation.com/infographie-internet-objets/. [Consulté le: 04-févr-2019].

[4] A. Dunkels, J. Vasseur, « IP for Smart Objects», Internet Protocol for Smart Objects (IPSO) Alliance, 2008. Disponible sur : http://dunkels.com/adam/dunkels08ipso.pdf. [Consulté le: 04-févr-2019].

[5] D. Evans, « How the Next Evolution of the Internet Is Changing Everything », CISCO, 2011. Disponible sur : https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IB SG_04 11FINAL.pdf. [Consulté le: 04-févr-2019].

[6] J. Fenn, H. LeHong, « Hype Cycle for Emerging Technologies, 2011 », Gartner, 2011. Disponible sur : https://www.gartner.com/doc/1754719/hype-cycle-emerging-technologies. [Consulté le: 04-févr-2019].

[7] ITU Internet Report 2005, « The Internet of Things », ITU-T, 2005. Disponible sur : https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf. [Consulté le: 04-févr-2019].

[8] « Internet of Things 2008 Conference, March 26-28, Zurich ». [En ligne]. Disponible sur : https://iot-conference.org/iot2008/. [Consulté le: 04-févr-2019].

[9] Postscapes, « Internet of Things (IoT) History ». [En ligne]. Disponible sur : https://www.postscapes.com/internet-of-things-history. [Consulté le: 04-févr-2019].

[10] National Intelligence Council, « Disruptive Civil Technologies, Six technologies With Potential Impacts on US Interests Out to 2025 », NIC, 2008. Disponible sur : https://www.hsdl.org/?view&did=485606.[Consulté le: 04-févr-2019].

[11] ITU-T Y.2060, « Overview of the Internet of things », ITU-T, 2012. Disponible sur : https://www.itu.int/rec/T-REC-Y.2060-201206-I. [Consulté le: 04-févr-2019].

[12] ITU-T Y.2066, « Common requirements of the Internet of things», ITU-T, 2014. Disponible sur : https://www.itu.int/rec/T-REC-Y.2066/en. [Consulté le: 04-févr-2019].

[13] Groupe de travail 6Lo IETF. [En ligne]. Disponible sur : https://tools.ietf.org/wg/6lo/. [Consulté le: 04-févr-2019].

[14] Groupe de travail 6LoWPAN IETF. [En ligne]. Disponible sur: https://tools.ietf.org/wg/6lowpan/. [Consulté le: 04-févr-2019].

[15] O. Garcia-Morchon, S. Kumar, M. Sethi, « State of the Art and Challenges for the Internet of Things Security, draft irtf t2trg iot seccons 15 », IETF, 2018. Disponible sur : https://tools.ietf.org/html/draft-irtf-t2trg-iot-seccons-15. [Consulté le: 04-févr-2019].

[16] O. Garcia-Morchon, S. Kumar, M. Sethi, « State of the Art and Challenges for the Internet of Things Security, draft irtf t2trg iot seccons 16 », IETF, 2018. Disponible sur : https://tools.ietf.org/html/draft-irtf-t2trg-iot-seccons-16. [Consulté le: 04-févr-2019].

[17] « IoT: number of connected devices worldwide 2012-2025 », Statista. [En ligne]. Disponible sur: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/. [Consulté le: 09-mai-2019].

[18] A. Nordrum, « Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated », IEEE Spectrum, 2016. [En ligne]. Disponible sur : https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated. [Consulté le: 04-févr-2019].

[19] R. Minerva, A. Biru, D. Rotondi, « Towards a definition of the Internet of Things (IoT) », IEEE, 2015.

[20] ISO/IEC JTC 1, « Internet of Things (IoT) Preliminary Report 2014 », ISO/IEC, 2015.

[21] International Electrotechnical Commission, « IEC role in the IoT, International Electrotechnical Commission », IEC, 2017.

[22] G. M. Lee, J. Park, N. Kong, N. Crespi, I. Chong, « The Internet of Things-Concept and Problem Statement », Internet Research Task Force, 2012.

[23] Groupe de travail T2TRG IRTF, [En ligne]. Disponible sur : http://www.irtf.org/t2trg. [Consulté le: 04-févr-2019].

[24] « P2413 - Standard for an Architectural Framework for the Internet of Things (IoT) ». [En ligne]. Disponible sur: https://standards.ieee.org/project/2413.html. [Consulté le: 09-mai-2019].

[25] « NIST | National Institute of Standards and Technology ». [En ligne]. Disponible sur : https://www.nist.gov/national-institute-standards-and-technology. [Consulté le: 04-févr-2019].

[26] R. Materese, « Internet of Things (IoT) », 2017. [En ligne]. Disponible sur: https://www.nist.gov/topics/internet-things-iot. [Consulté le: 09-mai-2019].

[27] « OASIS | Advancing open standards for the information society ». [En ligne]. Disponible sur : https://www.oasis-open.org/. [Consulté le: 04-févr-2019].

[28] M. Chapman, O. Ireland, « IoT: an OASIS Perspective », 2014. [En ligne]. Disponible sur: http://ec.europa.eu/DocsRoom/documents/5116/attachments/1/translations/en/renditions/native. [Consulté le: 09-mai-2019].

[29] « Internet of Things Initiative | Projects | FP7-ICT | CORDIS | European Commission ». [En ligne]. Disponible sur : https://cordis.europa.eu/project/rcn/95102/factsheet/en. [Consulté le: 04-févr-2019].

[30] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, « Vision and Challenges for Realising the Internet of Things », CERP-IoT, 2010. Disponible sur : http://www.internet-of-things-research.eu/pdf/IoT_Clusterbook_March_2010.pdf. [Consulté le: 04-févr-2019].

[31] « Coordination and Support Action for Global RFID-related Activities and Standardisation | Projects | FP7-ICT | CORDIS | European Commission ». [En ligne]. Disponible sur : https://cordis.europa.eu/project/rcn/85786/factsheet/en. [Consulté le: 04-févr-2019].

[32] I. Smith, K. Sakamura, A. Furness, R. Ma, Y.W. Kim, E. Wlak, C. Harmon, P. Chartier, P. Guillemin, D. Armstrong, « RFID and the Inclusive Model for the Internet of Things », CASAGRAS, 2009. Disponible sur : https://docbox.etsi.org/zArchive/TISPAN/Open/IoT/low%20resolution/www.rfidglobal.eu%20CASAGRAS%20IoT%20Final%20Report%20low%20resolution.pdf. [Consulté le: 04-févr-2019].

[33] S. W. Lin, B. Miller, J. Durand, G. Bleakley, A. Chigani, R. Martin, B. Murphy, M. Crawford, « Industrial Internet Reference Architecture version 1.7 », Industrial Internet Consortium, 2015.

[34] « Webinars - IEEE Internet of Things ». [En ligne]. Disponible sur : https://iot.ieee.org/education/webinars.html. [Consulté le: 27-octo-2016].

[35] Deliverable D1.2 – Initial Architectural Reference Model for IoT, Internet-of-Things Architecture, 2011. Disponible sur : https://www.scribd.com/document/257100955/D1-2-Initial-Architectural-Reference-Model-for-IoT. [Consulté le: 05-févr-2019].

[36] F. Bonomi, « The Smart and Connected Vehicle and the Internet of Things », NIST, 2013. Disponible sur : http://tf.nist.gov/seminars/WSTS/PDFs/1-0_Cisco_FBonomi_ConnectedVehicles.pdf. [Consulté le: 05-févr-2019].

[37] The Intel IoT Platform, « Architecture Specification, Whitepaper Internet of Things IoT. », Intel, 2015. Disponible sur : https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/iot-platform-reference-architecture-paper.pdf. [Consulté le: 06-févr-2019].

[38] « Kaa Enterprise IoT platform », Kaa IoT platform. [En ligne]. Disponible sur: https://www.kaaproject.org/. [Consulté le: 06-févr-2019].

[39] J. R. Stachel, E. Sejdić, A. Ogirala, et M. H. Mickle, « The impact of the internet of Things on implanted medical devices including pacemakers, and ICDs », IEEE International Instrumentation and Measurement Technology Conference (I2MTC), 2013, p. 839‑844.

[40] C. Maple, « Security and privacy in the internet of things», Journal of Cyber Policy, Vol 2, No 2, 2017. Disponible sur : https://www.tandfonline.com/doi/full/10.1080/23738871.2017.1366536. [Consulté le: 06-févr-2019].

[41] « Massive IoT e-health », Ericsson, 2018. [En ligne]. Disponible sur : https://www.ericsson.com/en/news/2018/8/connected-e-health-IoT. [Consulté le: 06-févr-2019].

[42] « WHO | WHO and PATH partner to globalize digital health », WHO, 2018. [En ligne]. Disponible sur : http://www.who.int/ehealth/events/WHO-PATH-partnership/en/. [Consulté le: 06-févr-2019].

[43] « Rapid Expansion Projected for Smart Home Devices, IHS Markit Says | IHS Markit Online Newsroom ». [En ligne]. Disponible sur : https://news.ihsmarkit.com/press-release/technology/rapid-expansion-projected-smart-home-devices-ihs-markit-says. [Consulté le: 06-févr-2019].

[44] « IEC - ISO/IEC JTC 1/SC 25 Dashboard > Scope ». [En ligne]. Disponible sur: https://www.iec.ch/dyn/www/f?p=103:7:0::::FSP_ORG_ID,FSP_LANG_ID:3399,25. [Consulté le: 06-févr-2019].

[45] International Electrotechnical Commission, « Orchestrating infrastructure for sustainable Smart Cities ». IEC, 2017.

[46] « Smart city | Nokia Networks ». [En ligne]. Disponible sur : https://networks.nokia.com/industries/smart-city. [Consulté le: 06-févr-2019].

[47] Alcatel Lucent, « The Internet of Things in Transportation », Alcatel Lucent Enterprise, 2018.

[48] « ParkDC: Penn Quarter/Chinatown », Kittelson & Associates, Inc. [En ligne]. Disponible sur: https://www.kittelson.com/work/parkdc-penn-quarterchinatown/. [Consulté le: 09-mai-2019].

[49] « How the IoT is Changing the Transportation System | NJIT Online ». [En ligne]. Disponible sur : https://graduatedegrees.online.njit.edu/resources/msce/msce-articles/smart-transportation-how-the-iot-is-changing-the-transportation-system/. [Consulté le: 06-févr-2019].

[50] P. Suku, « Intelligent Transport Systems in the UK. », World Scientific, 2012. Disponible sur : https://ec.europa.eu/transport/sites/transport/files/themes/its/road/action_plan/doc/2012-united-kingdom-its-5-year-plan-2012_en.pdf. [Consulté le: 06-févr-2019].

[51] « UK CITE – UK Connected Intelligent Transport Environment ». [En ligne]. Disponible sur: https://www.ukcite.co.uk/. [Consulté le: 06-févr-2019].

[52] « Cellular Technologies Enabling the Internet of Things ». 4G Americas, 2015. Disponible sur : http://www.5gamericas.org/files/6014/4683/4670/4G_Americas_Cellular_Technologies_Enabling_he_IoT_White_Paper_-_November_2015.pdf. [Consulté le: 06-févr-2019].

[53] ETSI TS 136 101, « LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) radio transmission and reception (3GPP TS 36.101 version 10.3.0 Release 10) ». [En ligne]. Disponible sur: https://www.etsi.org/deliver/etsi_ts/136100_136199/136101/10.03.00_60/ts_136101v100300p.pdf. [Consulté le: 10-mai-2019].

[54] GSM Association, « NB-IoT DEPLOYMENT GUIDE to Basic Feature set Requirements », [En ligne]. Disponible sur: https://www.gsma.com/iot/wp-content/uploads/2018/04/NB-IoT_Deployment_Guide_v2_5Apr2018.pdf. [Consulté le: 10-mai-2019].

[55] Technical Report 21.915, « 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Release 15 Description; Summary of Rel-15 Work Items (Release 15) », Mars 2019

[56] « A Technical Overview of LoRa® and LoRaWAN™. ». Lora Alliance, 2015. Disponible sur : https://www.tuv.com/media/corporate/products_1/electronic_components_and_lasers/TUeV_Rheinand_Overview_LoRa_and_LoRaWANtmp.pdf. [Consulté le: 06-févr-2019].

[57] IEEE Standard for Local and Metropolitan Area Networks. « Low-Rate Wireless Personal Area Networks », IEEEComputer Society, 2016.

[58] S. Nath, S. Aznabi, N. Islam, A. Faridi, W. Qarony, « Investigation and performance analysis of some implemented features of the ZigBee protocol and IEEE 802.15.4 mac specification ». Int. J. Online Eng 13, 2017.

[59] P. Thubert, C. Bormann, L. Toutain, R. Cragie, « IPv6 over Low-PowerWireless Personal Area Network (6LoWPAN) Routing Header ». IETF RFC, 2017.

[60] ITU-T G.9959, « Short range narrow-band digital radiocommunication transceivers – PHY, MAC, SAR and LLC layer specifications », ITU-T, 2015. Disponible sur : https://www.itu.int/rec/T-REC-G.9959/en. [Consulté le: 06-févr-2019].

[61] M. Andersson, « Use case possibilities with Bluetooth low energy in IoT applications », u-blox, 2014. Disponible sur : https://www.u-blox.com/sites/default/files/products/documents/BluetoothLowEnergy-IoT-Applications_WhitePaper_%28UBX-14054580%29.pdf. [Consulté le: 06-févr-2019].

[62] X. Feng, L. Jie, H. Ruonan , K. Xiangjie, G. Ruixia, « Service Differentiated and Adaptive CSMA/CA over IEEE 802.15.4 for Cyber-Physical Systems», The Scientific World Journal, 2013.

[63] R. C. Bhaddurgatte, V. Kumar, « Review: QoS Architecture and Implementations in IoT Environment », Research & Reviews: Journal of Engineering and Technology, 2015.

[64] R. Branden, D. Clark, S. Shenker, « Integrated Services in the Internet Architecture: an Overview», Internet Engineering Task Force, Request for Comments (RFC) 1633, 1994.

[65] J. Babiarz, K. Chan, F. Baker, « Configuration Guidelines for DiffServ Service Classes », Internet Engineering Task Force, Request for Comments (RFC) 4594, 2006.

[66] D. Premila, A. Rabara, V. Jerald, « Quality of Service Architecture for Internet of Things and Cloud Computing », International Journal of Computer Applications, vol. 128, n° 7, 2015.

[67] B. Ray, « Benefits of Quality of Service (QoS) in LPWAN for IoT », LinkLabs, 2016.

[58] M. Serrano, « Quality of Service (QoS) for IoT services », OpenIoT Consortium Delivrable D4.6., 2014.

[69] « The French National Research Agency Website », Quasimodo. [En ligne]. Disponible sur: http://www.agencenationale-recherche.fr/Project-ANR-10-INTB-0206. [Consulté le: 31-Oct-2018].

[70] M. Nef, L. Perleps, S. Karagiorgou, G. Stamoulis, « Enabling QoS in the Internet of Things », International Conference on Communication Theory, Reliability, and Quality of Service, 2012.

[71] E. Sayarifah, A. Indrajit, S. Sivaramakrishnan, A. Adnan, « An IoT Environment for WSN Adaptive QoS », IEEE International Conference on Data Science and Data Intensive Systems (DSDIS 2015), 2015.

[72] I. S. AlShawi, L. Yan, W. Pan, et B. Luo, « Lifetime Enhancement in Wireless Sensor Networks Using Fuzzy Approach and A-Star Algorithm », IEEE Sensors Journal, vol. 12, nº 10, p. 3010‑3018, oct. 2012.

[73] M. A. Jawad et F. Mir, « Network lifetime enhancement in wireless sensor networks using secure alternate path », International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), 2017, p. 414‑419.

[74] E. Brose, C.Ngwu, « Performance Test Tools 1st Iteration », F-Interop Consortium Deliverable D3.2., 2016.

[75] M. Hamze, « Autonomie, sécurité et QoS de bout en bout dans un environment de Cloud Computing », Université de Bourgogne, 2015.

[76] L. Grugen., « Updated use cases, requirements and architectur », BigClouT Project Delivrable D1.4., 2018.

[77] H. Khodkari, S. Ghazi-maghrebi, A. Asosheh, « Assurance of QoS in the Integration of Cloud Services and Internet of Things », International Symposium on Networks, Computers and Communications (ISNCC), 2017.

[78] A. Ghowdhurry, S. Mukherjee, S. Souray Banerjee, « Examining of QoS in Cloud Computing Technologies and IoT Services », Advances in Wireless Technologies and Telecommunication (AWTT) Book Serie, 2018.

[79] V. Zarko, P. Skocir, K. Katsaros, « Initial Report on System Requirements and Architecture », SymbIoTe Project Delivrable D1.2., 2015.

[80] R. Duan, X. Chen, T. Xing, « A QoS architecture for IOT », International Conference on Internet of Things and Forth International Conference on Cyber, Physical and Social Computing, 2011.

[81] G. Jain, D. Singh, S. Verma, « Service level agreements on IP networks », Proceedings of the IEEE, vol. 92, 2002.

[82] P. Patel, A. Ranabahu, A. Sheth, « Service Level Agreement in Cloud Computing », 2009. Disponible sur: https://www.researchgate.net/publication/318017907_Service_Level_Agreement_in_Cloud_Computing. [Consulté le: 25-Sept-2019].

[83] G. Gaillard, D. Barthel, F. Theoleyre, F. Valois, « Service Level Agreements for Wireless Sensor Networks: a WSN Operator's Point of View ». IEEE/IFIP NOMS - Network Operations and Management Symposium, 2014.

[84] A. Alqahtani, Y. Li, P. Patel, E. Solaiman, R. Ranjan, « End-to-End Service Level Agreement Specification for IoT Applications », International Conference on High Performance Computing & Simulation (HPCS), 2018.

[85] A. Awatif, P. Pankesh, S.Ellis, R., Rajiv, « Demonstration Abstract: A Toolkit for Specifying Service Level Agreements for IoT applications ».

[86] ITU-T X.800, « Architecture de sécurité pour l'interconnexion en systèmes ouverts d'applications du CCITT», ITU-T, 1991.

[87] S. Hanna, « The untrusted IoT - A Path to Securing Billions of Insecure Devices », Trusted Computing Group, 2015.

[88] « Google Internet of Things | Google's Internet of Things Solutions », Google Developers. [En ligne]. Disponible sur: https://developers.google.com/iot/. [Consulté le: 14-févr-2019].

[89] « Huawei LiteOS ». [En ligne]. Disponible sur: https://www.huawei.com/minisite/liteos/en/index.html. [Consulté le: 14-févr-2019].

[90] « Windows 10 IoT ». [En ligne]. Disponible sur: https://developer.microsoft.com/en-us/windows/iot/. [Consulté le: 14-févr-2019].

[91] S. Li, « IoT Node Authentication, Securing the Internet of Things », Elsevier, 2017. Disponible sur: https://www.oreilly.com/library/view/securing-the-internet/9780128045053/xhtml/chp004.xhtml.

[92] N. Joshi, « Authentication and device identification in IoT security », Allerin 2017. [En ligne]. Disponible sur: https://www.allerin.com/blog/authentication-and-device-identification-in-iot-security. [Consulté le: 14-févr-2019].

[93] « Standards | OASIS ». [En ligne]. Disponible sur: https://www.oasis-open.org/standards. [Consulté le: 14-févr-2019].

[94] « OASIS eXtensible Access Control Markup Language (XACML) TC | OASIS ». [En ligne]. Disponible sur: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml. [Consulté le: 14-févr-2019].

[95] « CORDIS | European Commission ». [En ligne]. Disponible sur: https://cordis.europa.eu/project/rcn/101349/en. [Consulté le: 14-févr-2019].

[96] « IoT Enabling Technologies and Future Developments », BUTLER Consortium Delivrable D.2.5, 2014.

[97] L. Seitz, G. Selander, C. Gehrmann, « Authorization framework for the Internet-of-Things », International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013.

[98] A. Balte, A. Kashid, B. Patil, « Security Issues in Internet of Things (IoT): A Survey », International Journal of Advanced Research in Computer Science and Software Engineering, 2015.

[99] O. Garcia-Morchon, K. Wehrle, « Modular context-aware access control for medical sensor networks », ACM symposium on Access control models and technologies (SACMAT '10), 2010.

[100] A. Ferreria, R. Correia, H. Monterio, M. Brito, L. Antunes, « Usable access control policy and model for healthcare », 24th International Symposium on Computer-Based Medical Systems (CBMS), 2011.

[101] H. Maw, H. Xiao, B. Christianson, J. Malcolm, « A Survey of Access Control Models in Wireless Sensor Networks», Journal Sensor and Actuator Networks, 2014.

[102] « Home », ARMOUR. [En ligne]. Disponible sur: https://www.armour-project.eu/. [Consulté le: 14-févr-2019].

[103] ARMOUR, « Experiments and Requirements version 1.0 », ARMOUR Project Delivrable D1.1., 2016.

[104] B. Pokric, J. Krimmling, A. Soika-Piotrowska, Z. Dyka, P. Lagendorfer, J. Bohli, R. Marin-Perez, « Node Security, Policies, and Automatic Configuration version 1.0 », SMARTIE Project Delivrable D3.2, 2015.

[105] SMARTIE, « IoT information Access and Privacy Preservation », SMARTIE Project Delivrable D4.1, 2014.

[106] SMARTIE, « Components for secure information gathering and storage », SMARTIE Project Delivrable D3.1, 2014.

[107] M. Abdemeziem, « Data Confidentiality in the Internet of Things », Thèse de doctorat, Université de Lorraine, 2016.

[108] « Mirror of AVR-Crypto-Lib », GitHub, [En ligne]. Disponible sur: https://github.com/cantora/avr-crypto-lib. [Consulté le: 25-sept-2019].

[109] « Relic-toolkit», GitHub, 2018. Disponible sur: https://github.com/relic-toolkit. [Consulté le: 25-sept-2019].

[110] Network Simulation Tools, « TOSSIM simulator for TinyOS », TOSSIM. [En ligne]. Disponible sur: http://networksimulationtools.com/tossim/. [Consulté le: 21-févr-2019].

[111] M. Simplicio Jr., P. Barreto, T. Carvalho, C. Margi, M. Naslund, « The CURUPIRA-2 Block Cipher for Constrained Platforms: Specification and Benchmarking », International Workshop on Privacy in Location-Based Applications, 2008.

[112] C. De Cannière, O. Dunkelman, M. Knežević, « KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers », Lecture Notes in Computer Science, vol 5747, Springer, 2009.

[113] W. Zhang, Z. Bao, D. Lin, V. Rijmen, B. Yang, I. Verbauwhede, « RECTANGLE: a bit-slice lightweight block cipher suitable for multiple platforms », Science China Information Sciences, vol. 58, n° 12, 2015.

[114] T. Suzaki, K. Minematsu, S. Morioka, E. Kobayashi, « A Lightweight Block Cipher for Multiple Platforms », Selected Areas in Cryptography, 2013.

[115] C. H. Lim et T. Korkishko, « mCrypton – A Lightweight Block Cipher for Security of Low-Cost RFID Tags and Sensors », Information Security Applications, 2006.

[116] International Organization for Standardization, « ISO/IEC 29192-2:2012 », ISO. [En ligne]. Disponible sur: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/65/56552.html. [Consulté le: 21-févr-2019].

[117] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, « Simon and Speck: Block Ciphers for the Internet of Things », Cryptology ePrint Archive, Report 2015/585, 2015.

[118] International Organization for Standardization, « ISO/IEC 18033-3:2010 », ISO. [En ligne]. Disponible sur: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/45/54531.html. [Consulté le: 21-févr-2019].

[119] I. Kravets, « PlatformIO: An open source ecosystem for IoT development », PlatformIO. [En ligne]. Disponible sur: https://platformio.org. [Consulté le: 25-sept-2019].

[120] J.P. Aumasson, L. Henzen, W. Meier, M. Naya-Plasencia, « Quark: A Lightweight Hash », Cryptographic Hardware and Embedded Systems, 2010.

[121] International Organization for Standardization, « ISO/IEC 29192-5:2016 », ISO. [En ligne]. Disponible sur: https://www.iso.org/standard/67173.html. [Consulté le: 21-févr-2019].

[122] A. Mosenia, N. JHA, « A Comprehensive Study of Security of Internet-of-Things », IEEE Transactions on Emerging Topics in Computing, vol. 5, n° 4, 2017.

[123] GSMA, « IoT Security Guidelines Overview Document Version 1.1», GSMA Association, 2016.

[124] S. Menoret, « Security requirements for the iCore cognitive management and control framework version 1.0 », iCore Project Delivrable D.2.5., 2012.

[125] K. Nagara, K. Aoki, Y. Matsubara, H. Takada, « Portable DoS Test Tool for IoT Devices », Workshop on Internet of Things Security and Privacy, 2017.

[126] SLA Ready, « Making Cloud SLAs readily usable in the EU private sector ». [En ligne]. Disponible sur: http://www.sla-ready.eu/. [Consulté le: 21-févr-2019].

[127] J.Luna, A. Taha, R. Trapero, N. Sur, « Quantitative Reasoning about Cloud Security Using Service Level Agreements », IEEE Trans on Cloud Computing, vol. 5, n° 3, 2017.

[128] B. Monahan, M. Yearworth, « Meaningful Security SLAs », HP Laboratories, 2008.

[129] K. Bernsmed, M. Gilje Jaatun, P. Meland, A. Undheim. « Security SLAs for Federated Cloud Services », Sixth International Conference on Availability, Reliability and Security, IEEE Computer Society, 2011.

[130] S. Sicari, A. Rizzardi, L. Grieco, A. Coen-porisini, « Security, privacy and trust in Internet of Things: The road ahead », Computer Networks, n° 76, 2015.

[131] Z. Yan, P. Zhang, A. Vasilakos, « A survey on trust management for Internet of Things », Journal of Network and Computer Applications, Vol. 42, 2014.

[132] Journal officiel L 281, « Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data », 1995.

[133] Journal officiel de l'Union européenne, « Règlement (UE) 2016/679 du parlement européen et du conseil du 27 avril 2016 », 2016.

[134] Alliance for Internet of Things Innovation, « Report on Workshop on Security & Privacy in IoT », AIOTI, 2017.

[135] « Welcome To Trusted Computing Group », Trusted Computing Group. [En ligne]. Disponible sur: https://trustedcomputinggroup.org/. [Consulté le: 21-févr-2019].

[136] S. Hanna, « The untrusted IoT - A Path to Securing Billions of Insecure Devices », Trusted Computing Group, 2015.

[137] ISO, « ISO/IEC 11889 Information technology -- Trusted platform module library -- Part 1: Architecture », International Organization for Standardization, 2015.

[138] IEEE Standard Association, « IEEE 802.1AR-2018 - IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity ». [En ligne]. Disponible sur: https://standards.ieee.org/standard/802_1AR-2018.html. [Consulté le: 21-févr-2019].

[139] M. Davidson, C. Schmidt, « TAXII Overview », THE MITRE CORPORATION, 2014.

[140] C. Dong, G. Chang, D. Sun, J. Li, J. Lia, X. Wang, « TRM-IoT: A Trust Management Model Based on Fuzzy Reputation for Internet of Things », Computer Science and Information Systems, vol. 8, nº 4, 2011.

[141] M. Salehie, L. Tahvildari, « Autonomic computing: emerging trends and open problems », Workshop on Design and evolution of autonomic application software (DEAS 05), 2005.

[142] D. Sinreich, « An architectural blueprint for autonomic computing », IBM, 2006.

[143] D. Weyns, G. Ramachandran, R. Singh, « Self-managing Internet of Things », SOFSEM 2018: Theory and Practice of Computer Science, 2018.

[144] P. Lalanda, J. Mccann, A. Diaconescu, « Autonomic Computing Principles, Design and Implementations », Springer, 2014.

[145] Mohammad Reza Nami, Koen Bertels, « A Survey of Autonomic Computing Systems », International Conference on Autonomic and Autonomous Systems (ICAS07), 2007.

[146] Q. Ashraf, M. Habaebi, G. Sinniah, M. Ahmed, S. Khan, S. Hameed, « Autonomic protocol and architecture for devices in Internet of Things », IEEE Innovative Smart Grid Technologies - Asia (ISGT ASIA), 2014.

[147] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, F. Zambonelli, « A Survey of Autonomic Communications », ACM Transactions on Autonomous and Adaptive Systems (TAAS), Vol. 1, Issue 2, 2006.

[148] J. Calbimonte, « Self-management and Optimization Framework », OpenIoT Consortium Delivrable D5.1.2., 2014.

[149] G. Pujolle, « An Autonomic-oriented Architecture for the Internet of Things », IEEE John Vincent Atanasoff International Symposium on Modern Computing (JVA 06), 2006.

[150] I. Chatzigiannakis, H. Hasemann, M. Karnstedt , O. Kleine, A. Kröller, M. Leggieri, D. Pfisterer, K. Römer, C. Truong, « True self-configuration for the IoT », IEEE International Conference on the Internet of Things, 2012.

[151] S. Nechifor, D. Puiu, B. Târnaucă, F. Moldoveanu, « Autonomic Aspects of IoT Based Systems: A Logistics Domain Scheduling Example », Interoperability and Open-Source Solutions for the Internet of Things, 2015.

[152] S. Watts, « SaaS vs PaaS vs IaaS: What's The Difference and How To Choose – BMC Blogs », BMC Blogs. [En ligne]. Disponible sur: https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/. [Consulté le: 04-mars-2019].

[153] « Microsoft Azure Cloud Computing Platform & Services ». [En ligne]. Disponible sur: https://azure.microsoft.com/en-us/. [Consulté le: 26-août-2019].

[154] « Logiciel CRM - Gestion de la Relation Client (GRC) », Salesforce.com. [En ligne]. Disponible sur: https://www.salesforce.com/fr/. [Consulté le: 26-août-2019].

[155] « Google - Apps ». [En ligne]. Disponible sur: https://get.google.com/apptips/apps/#!/all. [Consulté le: 26-août-2019].

[156] « Des réunions en ligne faciles en conférence vidéo HD | GoToMeeting ». [En ligne]. Disponible sur: https://www.gotomeeting.com/fr-fr/lp/reunions-en-ligne-simplifiees. [Consulté le: 26-août-2019].

[157] « Compute Engine - IaaS | Compute Engine », Google Cloud. [En ligne]. Disponible sur: https://cloud.google.com/compute/. [Consulté le: 26-août-2019].

[158] « Cloud Application Platform | Heroku ». [En ligne]. Disponible sur: https://www.heroku.com/. [Consulté le: 26-août-2019].

[159] « cisco-metapod », Cisco. [En ligne]. Disponible sur: https://www.cisco.com/c/en/us/obsolete/cloud-systems-management/cisco-metapod.html. [Consulté le: 26-août-2019].

[160] « AWS Elastic Beanstalk – Deploy Web Applications », Amazon Web Services, Inc. [En ligne]. Disponible sur: https://aws.amazon.com/elasticbeanstalk/. [Consulté le: 26-août-2019].

[161] « App Engine - Build Scalable Web & Mobile Backends in Any Language | App Engine », Google Cloud. [En ligne]. Disponible sur: https://cloud.google.com/appengine/. [Consulté le: 26-août-2019].

[162] « Amazon Web Services (AWS) - Cloud Computing Services », Amazon Web Services, Inc. [En ligne]. Disponible sur: https://aws.amazon.com/. [Consulté le: 26-août-2019].

[163] K. AlSharqi, A. Abdelbari, A. Abou-Elnour, M. Tarique, « Zigbee based wearable remote healthcare monitoring system », International Journal of Wireless & Mobile Networks (IJWMN) Vol. 6 , 15 pages, June 2014

[164] H. Fernandez-Lopez, J. A. Afonso, J. H. Coreeia and R. Simoes, « ZigBee-based Remote Patient Monitoring », Studies in health technology and informatics. Septembre 2012.

[165] « What is 6loWPAN and when use it in my IoT project », 09-janv-2019, Zloteria. [En ligne]. Disponible sur: https://zolertia.io/6lowpan-iot-protocol. [Consulté le : 10-Sept-2019].

[166] IEEE Standard for Local and Metropolitan Area Networks, « Low-Rate Wireless Personal Area Networks », IEEE Computer Society, Washington, DC, USA, 2016.

[167] « michaelkirsche/IEEE802154INET-Standalone », GitHub. [En ligne]. Disponible sur: https://github.com/michaelkirsche/IEEE802154INET-Standalone. [Consulté le: 25-sept-2019].

[168] IETF, "Delay Limits for Real-Time Services", draft-suznjevic-tsvwg-delay-limits-00, Transport Area Working Group, 13 June 2016.

[169] S. Willens, A. C. Rubens, C. Rigney, et W. A. Simpson, « Remote Authentication Dial In User Service (RADIUS) ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc2865. [Consulté le: 13-sept-2019].

[170] J. R. Vollbrecht, B. Aboba, L. J. Blunk, H. Levkowetz, et J. Carlson, « Extensible Authentication Protocol (EAP) ». [En ligne]. Disponible sur: https://tools.ietf.org/html/rfc3748. [Consulté le: 13-sept-2019].

[171] A. Balte, A. Kashid, et B. Patil, « Security Issues in Internet of Things (IoT) : A Survey », in International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 4, 2015.

[172] H. F. Atlam, M. O. Alassafi, A. Alenezi, R. Walters, et G. Wills, « XACML for building access control policies in Internet of Things », in Proceedings of the 3rd International Conference on Internet of Things, Big Data and Security: IoTBDS 2018, 2018, p. 253‑260.

[173] « AT&T XACML ». [En ligne]. Disponible sur: https://github.com/att/XACML [Consulté le: 19-sept-2019].

[174] « Sun's XACML Implementation ». [En ligne]. Disponible sur: http://sunxacml.sourceforge.net/. [Consulté le: 19-sept-2019].

[175] « OW2 - Main - AuthZForce (Community Edition) ». [En ligne]. Disponible sur: https://authzforce.ow2.org/. [Consulté le: 19-sept-2019].

[176] P. Kershaw, ndg-xacml: XACML 2.0 implementation for the NERC DataGrid. [En ligne]. Disponible sur: https://pypi.org/project/ndg-xacml/0.5.1/. [Consulté le: 19-sept-2019].

[177] « eXtensible Access Control Markup Language (XACML) Version 3.0 ». [En ligne]. Disponible sur: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. [Consulté le: 23-avr-2019].

[178] « SAML Specifications | SAML XML.org ». [En ligne]. Disponible sur: http://saml.xml.org/saml-specifications. [Consulté le: 23-avr-2019].

[179] « SAML V2.0 Technical Overview ». [En ligne]. Disponible sur: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.pdf. [Consulté le: 19-sept-2019].

[180] F. Dernoncourt, « Introduction à La Logique Floue », Developpez.com, Avril 2011

[181] « Motivations of Fuzzy Logic.doc | Fuzzy Logic | Artificial Neural Network », Scribd. [En ligne]. Disponible sur: https://www.scribd.com/document/172969317/Motivations-of-Fuzzy-Logic-doc. [Consulté le: 04-Apr-2019].

[182] B. Bouchon-Meunier, "La logique floue et ses applications", Addison-Wesley, SA, France, 1995.

[183] D. Dubois, H. Prade, and P. Smets, "Partial Truth is not Uncertainty: Fuzzy Logic versus Possibilistic Logic", IEEE Expert, pp. 15-19, août 1994.

[184] « What Is Sugeno-Type Fuzzy Inference? ». [En ligne]. Disponible sur: https://www.mathworks.com/help/fuzzy/what-is-sugeno-type-fuzzy-inference.html. [Consulté le: 04-Apr-2019].

[185] « Mamdani and Sugeno Fuzzy Inference Systems - MATLAB & Simulink - MathWorks France ». [En ligne]. Disponible sur: https://fr.mathworks.com/help/fuzzy/types-of-fuzzy-inference-systems.html;jsessionid=3d8750953818e96ed6ee074c149b. [Consulté le: 19-sept-2019].

[186] « Automated Combinatorial Testing for Software (ACTS) », NIST, 16-juin-2009. [En ligne]. Disponible sur: https://www.nist.gov/programs-projects/automated-combinatorial-testing-software-acts. [Consulté le: 09-mai-2019].

[187] « ACPT - Access Control Policy Testing | CSRC », CSRC | NIST, 24-mai-2016. [En ligne]. Disponible sur: https://csrc.nist.gov/projects/access-control-policy-tool/acpt. [Consulté le: 25-avr-2019].

[188] « What is Network Lifetime | IGI Global ». [En ligne]. Disponible sur: https://www.igi-global.com/dictionary/network-lifetime/20144. [Consulté le: 11-juin-2019].

[189] « Energy sources and power management in IoT sensors and edge devices », JAXenter, 31-mai-2018. [En ligne]. Disponible sur: https://jaxenter.com/energy-sources-power-management-iot-sensors-edge-devices-145006.html. [Consulté le: 11-juin-2019].

[190] I. S. AlShawi, L. Yan, W. Pan, B. Luo, « Lifetime Enhancement in Wireless Sensor Networks Using Fuzzy Approach and A-Star Algorithm », IEEE Sensors Journal, vol. 12, no. 10, Octobre 2012.

[191] T. D. Nguyen, J. Y. Khan, D. T. Ngo, « An energy and QoS-aware packet transmission algorithm for IEEE 802.15.4 networks », 2015 IEEE Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Hong Kong, 2015.

[192] P. Nayak, A. Devulapalli, « A Fuzzy Logic-Based Clustering Algorithm for WSN to Extend the Network Lifetime », IEEE Sensors Journal, vol. 16, no. 1, Janvier 2016.

[193] S. Zribi Boujelbene, D. Ben Ayed Mezghani, N. Ellouze, « Systèmes à Inférences Floues pour la Classification Phonémique », 2019.

[194] E. H. Mamdani, S. Assilian, « An experiment in linguistic synthesis with a fuzzy logic controller », Int. J. Man-Mach. Stud., vol. 7, no. 1.

[195] M. Masdari, F. Naghiloo, « Fuzzy Logic-Based Sink Selection and Load Balancing in Multi-Sink Wireless Sensor Networks »,, Wireless Personal Communications, vol. 97, no 2, novembre 2017.

[196] Jin-Woo Kim, José Ramón Ramos Barrado, Dong-Keun Jeon, « An Energy-Efficient Transmission Scheme for Real-Time Data in Wireless Sensor Networks », Sensors Journal 2015.

[197] A. Baker, J.E. Lafata, R.E. Ward, F. Whitehouse, G. Dinive, « A web-based diabetes care management support system ». Joint Commission Journal on Quality Improvement, 27(4), 2001

[198] J. Wyatt, F. Sullivan, « What is health information? », Technical report, BMJ 331:568. 2005.

[199] Claudia C. Gutiérrez Rodríguez, Michel Riveill, « e-Health monitoring applications: What about Data Quality? », 2010.

[200] « ISO 5725-1:1994(en), Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions ». [En ligne]. Disponible sur: https://www.iso.org/obp/ui/#iso:std:iso:5725:-1:ed-1:v1:en. [Consulté le: 28-sept-2019].

Annexe

Annexe 1

Algorithme Auto-configuration de QBAIoT au niveau de la HL-Gw

```
1:  While (true)
2:    if (Environment_Changes = True) then   // IoT-SP add or removed gSLA on the HL-Gw
3:    if (Nb_QoS_Classes = 1) then //Analyze functionality of MAPE-K
4:          Retrieve_from_Knowledge_Base(Config_table[1]) // BO=SO= 14,  Slots(QoS_CAP_Initial1)= 16,
            Minimal_QoS_CAP_Slots1= 16
5:          Environment_Changes ← False & Update_Configuration_on_effector //Execute functionality of
            MAPE-K
6:        else
7:          if (Nb_QoS_Classes < 1) then //Analyze functionality of MAPE-K
8:                Environment_Changes ← False
9:          else
10:               if (RT_Classes = 0) then //Analyze functionality of MAPE-K
11:                     Retrieve_from_Knowledge_Base(Config_table[2]) // BO=SO= 3, Slot_Duration=
                        0.00768,  Slots(QoS_CAP_Initial1)= 13, Slots(QoS_CAP_Initial2) = 3,
                        N1_Reference = 130, N2_Reference = 30,  Nb_Packet_Max1 = 22, Nb_Packet_Max2
                        = 5, Minimal_QoS_CAP_Slots1= 7, Minimal_QoS_CAP_Slots2 = 2
12:                     Environment_Changes ← False & Update_Configuration_on_effectors //Execute
                        functionality of MAPE-K
13:               else
14:                     if (RT_Classes = 1) then //Analyze functionality of MAPE-K
15:                           if (Nb_QoS_Classes = 2) then //Analyze functionality of MAPE-K
16:                                 Retrieve_from_Knowledge_Base(Config_table[4]) // BO=SO= 2,
                                    Slot_Duration= 0.00384,  Slots(QoS_CAP_Initial1)= 12,
                                    Slots(QoS_CAP_Initial2) = 4, N1_Reference = 120, N2_Reference
                                    = 40,  Nb_Packet_Max1 = 10, Nb_Packet_Max2 = 3,
                                    Minimal_QoS_CAP_Slots1= 6, Minimal_QoS_CAP_Slots2 = 2
17:                                 Environment_Changes ← False &
                                    Update_Configuration_on_effectors //Execute functionality of
                                    MAPE-K
18:                           else
19:                                 Retrieve_from_Knowledge_Base(Config_table[5]) // BO=SO= 2,
                                    Slot_Duration= 0.00384,  Slots(QoS_CAP_Initial1)= 8,
                                    Slots(QoS_CAP_Initial2) = 5, Slots(QoS_CAP_Initial3) = 3,
                                    N1_Reference = 80, N2_Reference = 50, N3_Reference = 30,
                                    Nb_Packet_Max1 = 6, Nb_Packet_Max2 = 4, Nb_Packet_Max3 =
                                    2, Minimal_QoS_CAP_Slots1= 4, Minimal_QoS_CAP_Slots2 = 3,
                                    Minimal_QoS_CAP_Slots3 = 2
20:                                 Environment_Changes ←  False &
                                    Update_Configuration_on_effectors // Execute functionality of
                                    MAPE-K
21:                           end if
22:                     else
23:                           if (Nb_QoS_Classes = 4) then // Analyze functionality of MAPE-K
```

```
24:                                    Retrieve_from_Knowledge_Base(Config_table[7]) // BO=SO= 2,
                                       Slot_Duration= 0.00384,  Slots(QoS_CAP_Initial1)= 6,
                                       Slots(QoS_CAP_Initial2) = 5, Slots(QoS_CAP_Initial3) = 3,
                                       Slots(QoS_CAP_Initial4) = 2, N1_Reference = 60, N2_Reference
                                       = 50, N3_Reference = 30,  N1_Reference = 20, Nb_Packet_Max1
                                       = 5 Nb_Packet_Max2 = 4, Nb_Packet_Max3 = 2,
                                       Nb_Packet_Max4 = 1, Minimal_QoS_CAP_Slots1= 3,
                                       Minimal_QoS_CAP_Slots2 = 3, Minimal_QoS_CAP_Slots3 = 2,
                                       Minimal_QoS_CAP_Slots4 = 2
25:                                    Environment_Changes ← False &
                                       Update_Configuration_on_effectors // Execute  functionality of
                                       MAPE-K
26:                              else
27:                                    if (Nb_QoS_Classes = 3) then // Analyze functionality of MAPE-
                                       K
28:                                           Retrieve_from_Knowledge_Base(Config_table[6]) //
                                              BO=SO= 2, Slot_Duration= 0.00384,
                                              Slots(QoS_CAP_Initial1)= 7, Slots(QoS_CAP_Initial2) =
                                              6, Slots(QoS_CAP_Initial3) = 3, N1_Reference = 70,
                                              N2_Reference = 60, N3_Reference = 30,
                                              Nb_Packet_Max1 = 6, Nb_Packet_Max2 = 5,
                                              Nb_Packet_Max3 = 2, Minimal_QoS_CAP_Slots1= 4,
                                              Minimal_QoS_CAP_Slots2 = 3,
                                              Minimal_QoS_CAP_Slots3 = 2
29:                                           Environment_Changes ← False &
                                              Update_Configuration_on_effectors // Execute
                                              functionality of MAPE-K
30:                                    else
31:                                           Retrieve_from_Knowledge_Base(Config_table[3]) //
                                              BO=SO= 2, Slot_Duration= 0.00384,
                                              Slots(QoS_CAP_Initial1)= 9, Slots(QoS_CAP_Initial2) =
                                              7, N1_Reference = 90, N2_Reference = 70,
                                              Nb_Packet_Max1 = 7, Nb_Packet_Max2 = 6,
                                              Minimal_QoS_CAP_Slots1= 5,
                                              Minimal_QoS_CAP_Slots2 = 4
32:                                           Environment_Changes ← False &
                                              Update_Configuration_on_effectors // Execute
                                              functionality of MAPE-K
33:                                    end if
34:                              end if
35:                        end if
36:                  end if
37:            end if
38:         end if
39:      end if
40:   end while
```

Titre : Gestion autonome de la qualité de service et de la sécurité dans un environnement Internet des objets

Mots clés : IoT, QoS, Sécurité, QBAIoT, IoT-MAAC, Gestion autonome

Résumé : De nos jours, les avancées technologiques font que l'Internet des Objets (IoT) est en train de s'imposer dans notre vie quotidienne afin d'en améliorer la qualité grâce à entre autres l'automatisation de certaines tâches. Un défi majeur dans le déploiement massif des applications et services IoT ainsi que leur utilisation dans différents domaines est l'amélioration du niveau de service correspondant. Ce niveau de service peut être caractérisé selon deux axes importants : la Qualité de Service (QoS) et la sécurité. De plus, ce niveau de service doit être géré d'une manière autonome au sein de l'IoT vu l'hétérogénéité et la taille des réseaux qui forment cet environnement rendant difficile, même impossible, leurs gestions d'une façon manuelle par les administrateurs. Dans le cadre de cette thèse, nous proposons un mécanisme de QoS, appelé QBAIoT (QoS Based Access for IoT environments) permettant d'assurer un traitement différencié des trafics existants dans l'environnement IoT afin de respecter les exigences de chacun des trafics selon différents paramètres de QoS (i.e., délai, gigue, taux de livraison de paquets, etc.). QBAIoT est ensuite amélioré afin d'être géré d'une manière autonome à travers deux fonctions importantes : l'auto-configuration et l'auto-optimisation. De plus, afin d'assurer la QoS au sein de l'environnement IoT, il faut optimiser la consommation énergétique des composants contraints en termes de ressources. Ainsi, nous proposons une adaptation de QBAIoT permettant de réduire sa consommation énergétique d'une manière autonome tout en respectant la précision des données remontées par les objets IoT. Notre contribution concernant le deuxième axe du niveau de service dans un environnement IoT, à savoir la sécurité, se traduit par un mécanisme permettant de contrôler l'accès des objets aux passerelles IoT. Nous appelons cette méthode de contrôle d'accès IoT-MAAC (IoT Multiple Attribute Access Control) car elle prend en compte différents paramètres spécifiques à l'environnement IoT (i.e., confiance des objets, identificateur de l'objet, empreinte digitale de l'objet, etc.). La prise de décision concernant le contrôle d'accès des objets IoT est gérée d'une façon autonome par les passerelles IoT et vise à respecter les exigences de cet environnement en termes de confiance.

Title : Autonomous management of Quality of Service and security in an Internet of Things environment

Keywords : IoT, QoS, Security, QBAIoT, IoT-MAAC, Autonomic Computing

Abstract : Nowadays, the Internet of Things (IoT) is becoming important in our daily lives thanks to technological advances. This paradigm aims to improve the quality of human life through automating several tasks. In this context, service level guarantee within IoT environments is a major challenge while considering a massive deployment of IoT applications and services as well as extending their usage to different domains. The IoT service level can be characterized in two parts: Quality of Service (QoS) and security. Moreover, this service level must be managed in an autonomic manner within the IoT environment given the heterogeneity and the size of its infrastructure making it difficult, even impossible, their management in a manual manner by the administrators. In this thesis, we propose a QoS based channel access control mechanism, called QBAIoT (QoS Based Access for IoT environments), to ensure a differentiated processing of existing traffics in the IoT environment. The differentiated processing allows satisfying the requirements of each traffic according to different QoS parameters (i.e., delay, jitter, packet delivery ratio, etc.). Then, QBAIoT is improved and upgraded to integrate self-management capabilities thanks to two important functions of the closed control loop: self-configuration and self-optimization. In addition, to offer a better QoS within the IoT environment, it is necessary to optimize the energy consumption of resources' constrained components. Thus, we propose an adaptation of QBAIoT allowing to reduce its energy consumption in an autonomic manner while respecting the data accuracy. Our contribution concerning the second part of service level guarantee within an IoT environment, which is security, consists is a mechanism enabling IoT objects access control to IoT gateways, called IoT-MAAC (IoT Multiple Attribute Access Control). This mechanism takes into account different parameters that are specific to IoT environments (i.e., IoT object trust, IoT object identifier, IoT object fingerprint, etc.). Finally, the decision making process regarding IoT object access control is autonomously managed by IoT gateways and aims to meet the requirements of IoT environment in terms of trust.

Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besancon