TABLE DES MATIERES

ABREVIATIONS

2G	Réseau Mobile De Deuxieme Generation
3G	Réseau Mobile De Troisieme Generation
3GPP	3rd Generation Mobile System
4G	Réseau Mobile De Quatrieme Generation
5G	Réseau Mobile De Cinquieme Generation
6LOWPAN	Ipv6 Low Power Wireless Personal Area Networks
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ARM	Advanced Risc Machine Limited
ARPANET	Advanced Research Projects Agency Network
BLE	Bluetooth Low Energy
BSON	Binary JSON
CEI/IEC	International Electrotechnical Commission
COAP	Constrained Application Protocol
CSS	Cascading Style Sheets
EJS	Embedded Javascript
EMQTT	Erlang Mqtt Broker
GNU	Gnu's Not Unix
GPIO	*General Purpose Input Output*
GSM	Global System For Mobile Communication
HDMI	High Definition Multimedia Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfert Protocol
I2C	Inter Integrated Circuit
IBM	International Business Machines Corporation
IDE	Integrated Develepment Environment
IEEE	Institute Of Electrical And Electronics Engineers
IETF	Internet Engineering Task Force
IOT	Internet Of Things
IP	Internet Protocol
IPV4	Internet Protocol Version 4
IPV6	Internet Protocol Version 6
ISM	Industrial, Scientific And Medical
ISO	International Standard Organisation
JAVA EE	Java Platform Enterprise Edition
JIT	Just In Time
JS	Javascript
JSON	Javascript Object Notation
JTC1	Joint Technical Committee
KM	Kilometre

LAN	Local Area Network
LLNS	Lawrence Livermore National Security
LORAWAN	Long Range Radio Wide Area Network
LPWA	Low Power Wide Area
LSB	Least Significant Bit
LTE	Long Term Evolution
LTE-M	Lte – Machine To Machine
LWT	Last Will Testament
M2M	Machine To Machine
MAC OS	Macintosh Os
MQTT	Message Queuing Telemetry Transport
MSB	Most Significant Bit
NFC	Near Field Communication
NOSQL	Not Only Sql
NPM	Node Package Manager
OASIS	Organization For The Advancement Of Structured Information Standards
OS	Operating System
PC	Personal Computer
PHP	Hypertext Preprocessor
PIR	Pyroelectric Infra Red
PPM	Pulse Position Modulation
PUB	Publish
PWM	Pulse Width Modulation
QOS	Quality Of Service
RAN	Radio Acess Network
REST	Representational State Transfer
RFID	Radio Frequency Identification
RX	Reception
SGBD	Système De Gestion De Base De Données
SMTP	Simple Mail Transfert Protocol
SPI	Service Provider Interface
SQL	Structured Query Language
SSH	Secure Shell
SSL/TLS	Transport Layer Security / Secure Sockets Layer
SUB	Subscribe
TCP	Tranfert Control Protocol
TX	Transmission
UIT	Union Internationale Des Télécommunications
UMTS	Universal Mobile Telecommunications System
UNB	Ultra Narrow Band
URL	Uniform Resource Locator
USB	Universal Serial Bus

USIM	Universal Subscriber Identity Module
UTF-8	Universal Character Set Transformation Format - 8 Bits
WAN	Wide Area Network
WBAN	Wireless Body Area Network
WI-FI	Wireless Fidelity
WPAN	Wireless Personal Area Network
WWW	World Wide Web
XDSL	Digital Subscriber Line
XML	'Extensible Markup Language
XMPP	Extensible Messaging And Presence Protocol

INTRODUCTION GENERALE

Le monde d'aujourd'hui se caractérise par l'avènement massif de nouvelles technologies de plus en plus fascinantes et révolutionnaire. Ce sont surtout les objets connectes. Dans notre quotidien nous vivons avec ces technologies sans même parfois le savoir ; nous faisons partie ainsi de cet environnement digital qui est en plein expansion.

Le terme Internet of Things (IoT) ou Internet des objets désigne l'extension d'Internet à des objets de notre quotidien. On parle des objets connectés que l'on connaît à travers les smartwatchs, les bracelets sportifs, la domotique ou encore dans le domaine de la e-santé.
L'IoT fait partie d'une évolution puisque cela permet d'avoir de plus en plus d'objets connectés qui interagissent entre eux et qui proposent des solutions au quotidien pour améliorer les conditions de vie. Mais il s'agit également d'une révolution, et ce dans tous les domaines d'application. D'ici 2020 on prévoit 40 à 50 milliards d'objets sur terre soit l'équivalent de cinq appareils par personne. Un nombre impressionnant quand on sait que de nombreux pays n'ont pas accès à internet. Les consommateurs y recherchent avant tout l'aspect pratique, facilitateur de la vie de tous les jours, tout en laissant place à l'émerveillement.

Ainsi la gestion de données échangées par tous ces objets connectées est une priorité absolue pour pouvoir en tirer vraiment profit. De nombreuses contraintes s'imposent tel que la gestion de l'autonomie des objets, les collectes de données et leur traitement en temps réel ainsi que le débit de transmission. Il faut donc trouver un protocole qui répond au mieux à ces attentes. C'est l'idée principale de notre travail qui s'intitule « Conception et création d'une plateforme IoT avec le protocole MQTT».

Pour bien aborder le thème de ce présent mémoire nous allons voir en premier chapitre l'environnement de l'internet des objets. Ensuite dans le deuxième chapitre nous étudierons de près le modèle client-serveur. Le troisième chapitre est dédié totalement au protocole de communication MQTT (Message Queuing Telemetry Transport) qui a un rôle principal tout au long de ce travail. Pour illustrer le tout nous allons voir en dernier chapitre l'application du concept de l'internet des objets muni du protocole MQTT dans le domaine de la domotique.

CHAPITRE 1
PRESENTATION DE L'INTERNET DES OBJETS

1.1 Introduction

Aux côtés de nos chers smartphones, ils sont les stars des nouvelles technologies : les objets connectés. Ces objets qui communiquent entre eux pour nous faciliter le quotidien connaissent un essor important depuis ces dernières années. Ils font partie d'un projet beaucoup plus vaste appelé Internet des Objets qui est l'avenir d'Internet aussi appelé « web 3.0 ». Il est donc nécessaire de comprendre les technologies utilisées dans le paradigme de l'Internet des Objets en fournissant d'une part des bases conceptuelles

1.2 Généralités

1.2.1 Définition

Selon l'Union internationale des télécommunications (UIT), l'Internet des objets ou IoT est une « infrastructure mondiale pour la société de l'information, qui permet de disposer de services évolués en interconnectant des objets (physiques ou virtuels) grâce aux technologies de l'information et de la communication interopérables existantes ou en évolution ». En réalité, la définition de ce qu'est l'internet des objets n'est pas figée. Elle recoupe des dimensions d'ordres conceptuel et technique.
D'un point de vue conceptuel, l'Internet des objets caractérise des objets physiques connectés ayant leur propre identité numérique et capables de communiquer les uns avec les autres. Ce réseau crée en quelque sorte une passerelle entre le monde physique et le monde virtuel.
D'un point de vue technique, l'IoT consiste en l'identification numérique directe et normalisée (adresse IP, protocoles smtp, http...) d'un objet physique grâce à un système de communication sans fil qui peut être une puce RFID, Bluetooth ou Wi-Fi.

Figure 1.01 : *L'IoT*

1.2.2 Concept de l'IoT

L'Internet des objets repose sur l'idée que tous les objets seront connectés un jour à Internet et seront donc capables d'émettre de l'information et éventuellement de recevoir des commandes. On parle aussi d' "ubiquitous computing", c'est à dire informatique omniprésente, ambiante, ou pervasive. Ce nouveau paradigme informatique est basé non plus sur le PC, mais sur des objets quotidiens intégrant des capteurs et des capacités de communication.

L'internet des objets propose de créer une continuité entre le monde réel et le monde numérique : il donne une existence aux objets physiques dans le monde numérique.

Sur le plan fonctionnel, l'Internet des objets désigne une informatique qui se fond dans notre quotidien pour nous simplifier la vie, nous faire gagner du temps, décharger notre cerveau de la mémorisation de données logistiques (itinéraires, agenda, etc.). Il permet de créer de nouveaux usages, comme, par exemple, des informations en temps réel sur la localisation de ses amis. Il permet aussi de faire des mesures exhaustives, là où on se contentait dans le passé d'un simple panel, par exemple avec la mesure du trafic automobile dans les rues de la capitale.

1.2.3 Importance de l'IoT

Pour pouvoir mesurer l'importance de l'IoT, il faut d'abord comprendre les différences entre l'Internet et le World Wide Web (ou Web), des termes souvent confondus. L'Internet est la couche physique, c'est-à-dire le réseau composé de commutateurs, de routeurs et d'autres équipements. Sa principale fonction est de transporter les informations d'un point A à un point B de façon rapide, fiable et sécurisée. Le Web, lui, est une couche applicative qui intervient sur l'Internet. Son rôle essentiel est de fournir une interface permettant d'exploiter les informations qui circulent sur l'Internet. [1]

1.2.3.1 Évolution du Web et de l'Internet

Le Web est passé par plusieurs phases distinctes :

Étape 1 : Le Web 1.0. : Le monde des "éditeurs"

Tout a commencé par une phase de recherche pendant laquelle le Web était appelé ARPANET (Advanced Research Projects Agency Network). Le Web était alors surtout utilisé par des universitaires à des fins de recherche. Au commencement du web, les contenus étaient réalisés par quelques éditeurs, qui les publiaient sur des sites web. On avait donc un nombre restreint d'émetteurs

d'informations, et un grand nombre de lecteurs : c'est le web 1.0., premier stade d'évolution d'Internet.

Étape 2. : Le Web 2.0. : Et l'utilisateur se mit à produire du contenu

Dans sa deuxième phase, le Web peut être qualifié de « brochure électronique ».

Nous assistions alors à une véritable prise d'assaut des noms de domaine. Toutes les entreprises ressentaient le besoin de partager des informations sur Internet pour faire connaître leurs produits et leurs services.

Avec l'arrivée des premiers réseaux sociaux, chaque individu s'est mis à générer lui-même des contenus. Quand quelqu'un publie un commentaire sur un article, créé un profil sur Facebook, publie une vidéo sur Youtube ou écrit un Tweet, il produit un contenu (texte, image, vidéo...) qui pourra être consulté par d'autres. C'est le web dit "social", souvent appelé web 2.0.

Étape 3. : Le web 3.0. : Les machines s'y mettent

Désormais, nous voyons émerger de nombreux objets physiques dits "connectés". Ces objets utilisent Internet pour envoyer des informations et en recevoir. Par exemple, des chaussures connectées envoient le nombre de foulées après chaque course à un système distant, cela permettra ainsi de consulter l'historique des courses sur un site web.

1.2.3.2 L'IoT, la première évolution de l'Internet

Contrairement au Web, l'Internet se développe et s'améliore continuellement, mais il n'a pas connu de transformation fondamentale. Sa fonction reste essentiellement la même que lors de la phase ARPANET. Par exemple, il existait au départ plusieurs protocoles de communication, notamment AppleTalk, Anneau à jeton (ou Token Ring) et IP. Actuellement, le protocole IP est devenu la norme.

Dans ce contexte, l'importance de l'IoT devient considérable, puisqu'il s'agit de la première véritable évolution de l'Internet. Celle-ci donnera lieu à des applications révolutionnaires capables de transformer profondément notre mode de vie, et notre façon d'apprendre, de travailler et de nous divertir. L'IoT a déjà doté l'Internet de capacités sensorielles (température, pression, vibration, luminosité, humidité, tension), ce qui nous permet d'anticiper plutôt que de simplement réagir.

En outre, l'Internet couvre maintenant des endroits jusqu'alors inaccessibles. Des patients ingèrent même des dispositifs connectés qui aident les médecins à diagnostiquer certaines pathologies et à

en déterminer les causes. Des capteurs extrêmement miniaturisés peuvent être placés sur des plantes, des animaux et des sites géologiques, et connectés à l'Internet. [2]

1.3 Les objets connectés

1.3.1 Définitions

On parle d'objets connectés pour définir des types d'objets dont la vocation première n'est pas d'être des périphériques informatiques ni des interfaces d'accès au web, mais auxquels l'ajout d'une connexion Internet a permis d'apporter une valeur supplémentaire en terme de fonctionnalité, d'information, d'interaction avec l'environnement ou d'usage.
On distinguera un objet connecté d'une interface d'accès au web, cette dernière pouvant en effet prendre la forme d'un objet du quotidien. Ainsi, de manière évidente, un smartphone n'est pas un objet connecté. La smartwatch (montre connectée), peut être une simple interface de visualisation ou d'accès au web. Elle peut aussi devenir un objet connecté à part entière à partir du moment où elle traite et génère elle-même des informations, comme par exemple une montre qui mesure le rythme cardiaque de l'utilisateur et envoie ses données à un serveur pour une exploitation ultérieure tout comme les capteurs d'activités.

1.3.2 Typologie

On peut distinguer plusieurs types d'objets dans notre sujet :

- Des **marqueurs passifs** (code barre, RFID, …) qui nécessitent le recours à un système de lecture
- Des **capteurs actifs spécialisés** (podomètre Fitbit, Mi-band) capables de transmettre leurs données directement vers Internet ou indirectement via un PC
- Des **objets "couteau suisse"**, généralement des Smartphones, capables d'exécuter plusieurs scénarios (suivi du sommeil, géolocalisation). Ces objets généralistes sont généralement moins précis dans leurs mesures que les capteurs spécialisés.

L'interaction avec le monde réel peut prendre plusieurs formes :

- Mesure de **données environnementales** : localisation, température, vent,…
- Mesure de **données comportementales** : mouvement de personnes, …

1.3.3 Fonctionnement des objets connectés

Les objets connectés sont reliés à Internet (on parle d'ailleurs d'Internet des objets, ou web 3.0) : ils peuvent donc communiquer avec d'autres systèmes pour obtenir ou fournir de l'information. Cela est rendu possible par la forte miniaturisation des composants électroniques, mais aussi par l'émergence de nouveaux réseaux de télécommunication de type M2M. Ils pourront par exemple :

- Collecter et stocker des informations en fonction de leur environnement : rythme cardiaque de l'utilisateur, hygrométrie d'une cave, etc.
- Déclencher une action en fonction des informations recueillies sur le web, comme par exemple l'arrosage d'une pelouse à la veille d'une forte journée de sécheresse.

1.3.4 Réseaux de télécommunication des objets

Si certains objets connectés, de par leur nature, peuvent être raccordés à un réseau wifi domestique ou professionnel (par exemple une balance corporelle connectée), de nombreux autres sont destinés à être utilisés partout en mobilité. Une des problématiques va donc être de trouver un moyen de faire communiquer ces objets sans sacrifier l'autonomie énergétique, et pour un coût raisonnable.

1.3.4.1 Communication GSM directe

La première solution consiste à utiliser les réseaux de données existants (3G, 4G, etc.). Cette solution est très peu adaptée dans la majorité des cas : elle nécessite une carte USIM, et utilise une technologie très consommatrice d'énergie. De plus, elle est relativement onéreuse.

1.3.4.2 Utilisation du Bluetooth

La seconde solution consiste à se connecter à un smartphone équipé de Bluetooth pour se servir de la connexion de ce dernier au réseau Internet. Dans ce cas, la consommation énergétique est moindre, mais la dépendance au smartphone peut être problématique.

1.3.5 Exemples d'objets connectés

1.3.5.1 Montre cardio

Un exemple d'objet connecté qui commence à inonder le marché et à devenir une tendance: la montre cardio que ce soit une montre connectée équipée d’un podomètre ou un capteur d’activités. Par exemple à chaque footing, les données de la course sont transférées à un serveur, permettant ainsi de visualiser le parcours ou mesurer la progression.

1.3.5.2 Réfrigérateur connecté

Avec le frigo connecté, plus de listes de courses ni d'oubli du beurre : la machine détecte les produits manquants et passe la commande pour vous pour effectuer le réapprovisionnement.

1.3.6 Faire soi-même des objets connectés

Pour transformer un objet du quotidien en objet connecté, il suffit de le raccorder à Internet et de le faire régir en fonction des données disponibles : météo, cours de bourse, action d'un utilisateur sur un smartphone... La réelle "intelligence" de l'objet est donc dans l'interface, et non dans l'objet lui-même. Par exemple, une simple ampoule, si elle est raccordée à une interface connectée à Internet (par exemple au niveau de la prise de courant), peut être commandée depuis un smartphone, et devient par conséquent une "ampoule" connectée. Pour faire soi-même un objet connecté, il est désormais possible de se procurer à des coûts très raisonnables des "kits", ou d'utiliser des nano ordinateurs comme le Raspberry Pi, couplé à des relais pour commander des appareils raccordés au 220V.

1.3.7 Les capteurs

Les capteurs sont de petits appareils disposant de capacités de mesures, voire d'actions, sur leur environnement. La température, le taux d'humidité, la luminosité ambiante, la détection de présences ou de mouvements via un accéléromètre, présence de gaz, de polluants ou encore la géolocalisation font partie des informations les plus couramment collectées sur ce type de matériel. Selon les capteurs, ou grâce à l'ajout de cartes additionnelles, la pression atmosphérique, le niveau de radiation ou la pression acoustique (événements sonores) peuvent aussi être quantifiés. Les spécifiés innovantes de ces capteurs résident dans leur taille et leur cout réduit, tout en étant dotés des capacités de traitements de l'information, et des possibilités de transmission sans fil [3] [4].

La nature de l'objet "intelligent" a ceci de particulier qu'il dispose de possibilités de calcul et de transmission des données. L'échange d'informations, voire l'interaction entre différents objets est envisageable, et ce même sans intervention des utilisateurs. L'objet va "capter", mesurer une caractéristique physique de son environnement, éventuellement lui appliquer un traitement informatisé, et fournir le résultat à d'autres (utilisateur, ordinateur, etc.).

Si les capteurs disposent d'éléments pour évaluer une caractéristique physique de leur environnement, certains d'entre eux peuvent également agir sur cet environnement : on parle alors d'effecteurs [5]. Ces effecteurs sont souvent plus puissants en termes de capacités mémoire, de traitement,

1.4 Le M2M

1.4.1 Définition

Le M2M (Machine-to-Machine) constitue un ensemble de technologies réseaux sans fil ou filaires rendant des systèmes communiquant et leur permettant de s'échanger automatiquement des informations, sans intervention humaine.

Il existe deux grandes familles de technologies, à savoir le sans-fil et les filaires. La plus connue des technologies sans-fil est celle du réseau Internet mobile de type 2G ,3G, 4G mais il en existe d'autres, basées sur des radiofréquences différentes allant de la plus courte portée avec le NFC, à la plus longue, avec le Wirless M-buss par exemple. Du côté des technologies M2M filaires, on trouve le courant porteur en ligne.

1.4.2 Différence entre M2M et IoT

Souvent désignés par leurs initiales la machine to machine (M2M) et l'Internet of Things (IoT) sont deux concepts très proches. Il existe néanmoins une différence essentielle à retenir. Nous vous l'expliquons dans les lignes qui suivent.

Le M2M et l'IoT sont deux concepts, deux phénomènes très importants et assez proches. Ce sont deux solutions qui proposent des accès à distances à des objets ou des capteurs. Le M2M se définit traditionnellement par un réseau de télécommunication point à point utilisant un module cellulaire ou WiFi intégré**,** pour connecter des machines ou des objets à un réseau. L'intervention humaine n'est pas nécessaires, les informations circulent d'un endroit à un autre et peuvent être relayées via un serveur vers un logiciel.

L'IoT repose quant à lui sur l'identification de chaque objet, souvent connecté les uns aux autres, afin qu'il puisse envoyer des données sur une plateforme Cloud ou à une application à plus large échelle.

1.5 Domaine d'applications

On peut distinguer deux grandes familles d'usage de l'Internet des objets :

1.5.1 Usages sans rétroaction

Dans ce cas, l'Internet des Objets est utilisé pour faire une sorte de monitoring :

- **consommation de l'objet** (énergie pour une voiture)
- **état de l'objet** (niveau d'huile pour une voiture)

- **mesure du contexte environnemental** (météo, trafic routier, foule dans les transports,…)
- **mesure des paramètres personnels d'un individu** ou Quantified Self (activité sportive,…)
- **activité d'une infrastructure** (Pass Navigo, réseau de télécommunication, réseau d'énergie, …)

Les utilisateurs des données peuvent être : l'usager final, un fournisseur, une administration publique.

Les données remontent des indicateurs permettant :

- des **tableaux de bords analytiques**, pour des fonctions décisionnelles (ex : augmenter le nombre de train sur le réseau Paris/Lyon)
- la **détection d'évènements anormaux** (ex : saturation des transports lors d'une manifestation)
- de **disposer d'informations sur un individu**, ses comportements et consommations, … (ex : mes performances sportives de la semaine)
- de **ramener les informations d'un individu à une population** à des fins de comparaison pour lui-même ou pour le fournisseur de service (ex : je consomme plus de gaz que les usagers vivant dans un appartement de même surface)

  de **proposer la "gamification",** c'est à dire une compétition entre l'individu et ses pairs afin d'optimiser l'usage du service (ex : faire baisser sa consommation électrique pour gagner des prix)

1.5.2 Usages avec rétroaction

Dans ce cas, l'Internet des Objets permet de :

- **piloter** les objets à distance, par exemple dans des applications domotiques (ex : allumer la lumière lorsque je rentre à la maison)
- **d'envoyer des notifications** au fournisseur (ex : provision de yaourts car les rayons du supermarché sont vides)
- **d'envoyer des notifications** aux usagers (ex : ralentir car l'autoroute est saturée)

1.6 Architectures des réseaux IoT

Pour bien comprendre le concept de l'IoT, il est nécessaire de connaitre son architecture réseau illustre par la figure 1.02 suivante :

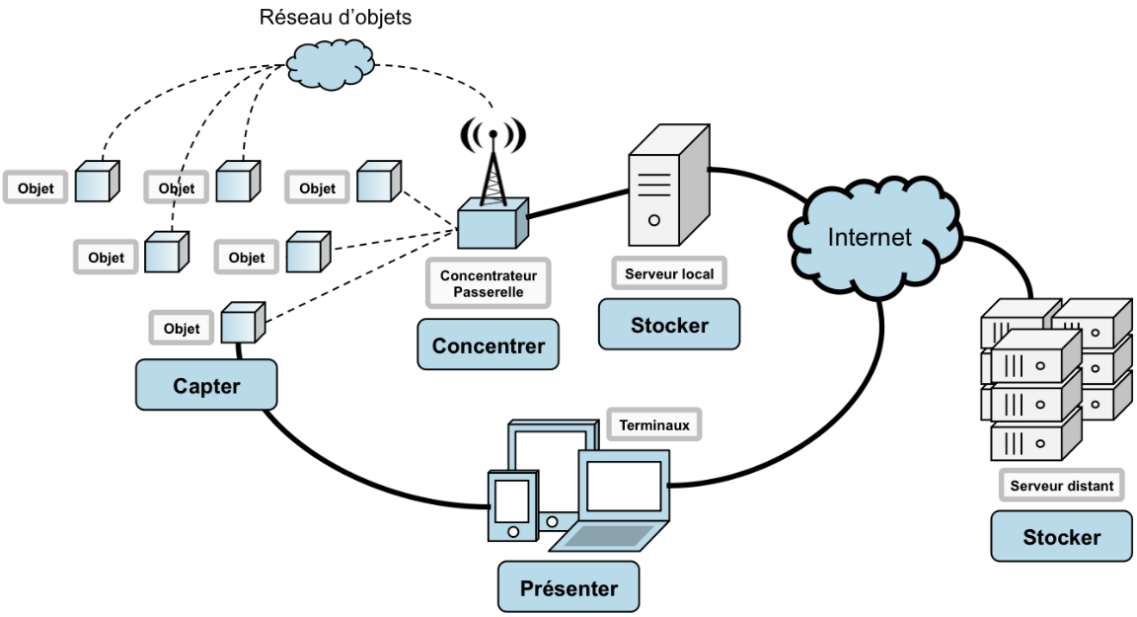

Figure 1.02 : *Architecture de l'internet des objets*

Précisons le rôle des différents processus présentés sur la figure 1.02 :

- **Capter** désigne l'action de transformer une grandeur physique analogique en un signal numérique.
- **Concentrer** permet d'interfacer un réseau spécialisé d'objet à un réseau IP standard (ex : WiFi) ou des dispositifs grand public.
- **Stocker** qualifie le fait d'agréger des données brutes, produites en temps réel, méta taguées, arrivant de façon non prédictible.
- Enfin, **présenter** indique la capacité de restituer les informations de façon compréhensible par l'Homme, tout en lui offrant un moyen d'agir et/ou d'interagir.

Deux autres processus n'apparaissent pas sur le schéma, car ils sont à la fois transverves et omniprésents :

Le **traitement des données** est un processus qui peut intervenir à tous les niveaux de la chaîne, depuis la capture de l'information jusqu'à sa restitution. Une stratégie pertinente, et commune quand on parle d'Internet des objets, consiste à stocker l'information dans sa forme intégrale. On collecte de manière exhaustive, « big data », sans préjuger des traitements qu'on fera subir aux données. Cette stratégie est possible aujourd'hui grâce à des architectures distribuées type NoSQL, capables d'emmagasiner de grandes quantités d'information tout en offrant la possibilité de réaliser des traitements complexes en leur sein (Map/Reduce par exemple).

La **transmission des données** est un processus qui intervient à tous les niveaux de la chaîne. Deux réseaux, supports des transmissions, cohabitent généralement :

- **Réseau local de concentration**. On utilise alors des technologies comme ZigBee, Z-wave, NFC ou Bluetooth LE.
- **Réseau WAN**, permettant d'interconnecter les réseaux spécialisés et de les interfacer avec des fermes de serveur. On utilise alors WiFi, les réseaux cellulaires (GSM, UMTS, LTE) ou encore les connexions physiques standard (Ethernet, fibre optique). Ces réseaux sont généralement connectés à Internet.

Les technologies de transmission utilisées dépendent essentiellement de l'application et du contexte. La transmission peut par exemple exploiter des differents protocoles applicatifs comme MQTT, WebSocket, http, … . Les canaux peuvent être bidirectionnels si l'application autorise une rétroaction. Dans certains cas, ces canaux devront transmettre les données en temps réel, dans d'autres cas, le temps ne sera pas un facteur déterminant.

Les **gateways** jouent le rôle d'intermédiaire pour connecter l'objet à internet et envoyer ses données au cloud. Un exemple de ces gateways sont les routeurs domestiques, les téléphones mobiles, le raspberryPi, l'Arduino. Ces gateways fournissent ce qui est nécessaire en termes de connectivité, de sécurité et de management des appareils. Les gateways traduisent aussi les protocoles propriétaires (exemple zigbee, BLE) au réseau Internet et certaines peuvent jouer le rôle d'agrégateurs de réseaux.

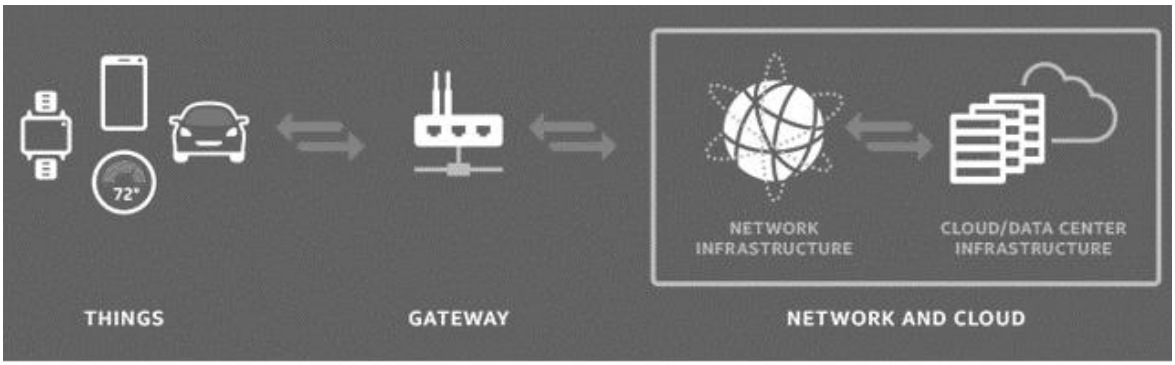

Figure 1.03 : *Architecture des réseaux IoT*

1.7 Les protocoles utilisés

D'autres protocoles de transfert pour les appareils à ressources limitées sont à l'étude. Comme le protocole pour applications contraintes (CoAP pour Constrained Application Protocol), qui utilise un modèle de communication de type requête/réponse. Ou le protocole de file de messages avancé (AMQP pour Advanced Message Queuing Protocol) qui, comme MQTT, s'appuie sur un modèle publication/abonnement.

1.7.1 Protocoles d'accès

Parmi les technologies de communication sans fil pour le M2M (machine-to-machine) et l'Internet des Objets, nous pouvons distinguer deux grands types de réseaux de communication radio : les réseaux courte portée et les réseaux moyenne et longue portée.

Le premier type concerne les technologies – bien connues du grand public – qui émettent de quelques centimètres à quelques centaines de mètres maximum : WiFi, Bluetooth, RFID, NFC, ZigBee, etc…

Le second concerne les technologies qui ont des portées plus importantes, de quelques centaines de mètres à plusieurs dizaines de kilomètres. Ce sont les réseaux cellulaires traditionnels (GSM, GPRS, LTE, etc..) que nous utilisons aussi pour nos communications mobiles.

Autre aspect sur lequel les réseaux se différencient : le débit. Quand une technologie comme le Wifi, propose des débits de l'ordre de la centaine de Mbits (voire plus) ; le ZigBee comme la RFID ne dépassent pas le Mbit.

1.7.1.1 Protocoles M2M / WPAN / WBAN

a. Bluetooth low energy

Le Bluetooth Low Energy est très largement utilisé dans le monde. Quasiment tous les smartphones sont équipés de cette technologie, fréquemment utilisée pour faire communiquer les wearables. Il a une portée de 60 mètres en terrain dégagé et consomme environ 20 fois moins d'énergie que le Wifi. La dernière version de cette technologie, le Bluetooth 5, est plus adaptée à l'IoT et dispose d'une portée deux fois supérieure à celle de son aînée.

Ce réseau de courte portée permet de transporter nettement moins d'infos que le Wifi : 1 mbps seulement, même si le nouveau standard permet de transférer quatre fois plus de données.

b. ZigBee

ZigBee est un protocole de haut niveau permettant la communication de petites radios, à consommation réduite, basée sur la norme IEEE 802.15.4 (Institute of Electrical and Electronics Engineers) pour les réseaux à dimension personnelle (Wireless Personal Area Networks : WPAN).
La spécification initiale de ZigBee propose un protocole lent dont le rayon d'action est relativement faible, mais dont la fiabilité est assez élevée, le prix de revient faible et la consommation considérablement réduite.
On retrouve donc ce protocole dans des « environnements embarqués » où la consommation est un critère de sélection. Ainsi, la domotique et les nombreux capteurs et télécommandes qu'elle implémente apprécient particulièrement ce protocole en plein essor et dont la configuration du réseau maillé se fait automatiquement en fonction de l'ajout ou de la suppression de nœuds. On retrouve aussi ZigBee dans les contrôles industriels, les applications médicales et les détecteurs de fumée et d'intrusion. Les nœuds sont conçus pour fonctionner plusieurs mois (jusqu'à dix ans pour les plus économes) en autonomie complète grâce à une simple pile de 1,5 V.
Le ZigBee permet de faire circuler plus de données que le Z-Wave (jusqu'à 250 kbps, contre 100 maximum). Il est également moins cher et plus facile à implémenter pour les fabricants d'objets connectés que le Z-Wave ou le Bluetooth (dans sa version 4, dite Low Energy comme dans sa version 5, qui vient d'être standardisée). Mais ce réseau n'a que 10 mètres de portée en moyenne soit 20 de moins que le Z-Wave et 50 de moins que le Bluetooth Low Energy.

c. Z-Wave

Le Z-Wave est un protocole de communication dédié à la domotique. Sans fil, il est facile à installer dans la maison. Il a à la base une portée de 30 mètres. C'est un réseau maillé, c'est-à-dire que chaque appareil connecté au système est émetteur de données mais peut aussi relayer celles qui sont émises par ses voisins. Cela permet d'élargir sa portée.
Un bémol toutefois : les clients qui créent dans leur logement un réseau Z-Wave doivent veiller à installer régulièrement dans l'espace des appareils reliés au secteur, faute de quoi ils risquent de créer des zones blanches. Ce sont les seuls qui restent en activité 100% du temps, pour éventuellement transmettre les données de leurs comparses. Ceux qui fonctionnent sur batterie sont la plupart du temps en sommeil pour ne pas consommer trop (le Z-Wave brûle tout de même deux fois plus d'énergie que le Bluetooth Low Energy). Ils ne relaient donc les données que lorsqu'ils

s'allument. Le Z-Wave n'est par ailleurs pas aussi universel que le Wifi. Tous les appareils de la maison ne pourront pas forcément communiquer avec un objet connecté via cette technologie.

d. RFID

Les protocoles de communication RFID désignent l'ensemble des caractéristiques qui permettent à une radio-étiquette de communiquer avec un lecteur. La RFID impose des exigences bien spécifiques tant d'un point de vue de la confidentialité et de la sécurité que des performances. Des solutions et améliorations sont proposées par la recherche et l'industrie, certaines font leur chemin jusqu'aux spécifications des normes.

1.7.1.2 Protocoles LAN

a. Ethernet

Ethernet est un protocole de réseau local à commutation de paquets. C'est une norme internationale : ISO/IEC 8802-3. Son rôle dans l'internet des objets est l'accès à Internet.

b. Wi-Fi

Le Wi-Fi, est un ensemble de protocoles de communication sans fil régis par les normes du groupe IEE 802.11 (ISO/CEI 8802-11). Un réseau Wi-Fi permet de relier par ondes radio plusieurs appareils informatiques (ordinateur, routeur, smartphone, décodeur Internet, etc.) au sein d'un réseau informatique afin de permettre la transmission de données entre eux. Le wifi a un très grand rôle dans l'internet des objets car presque les objets connectes sont munis de cette technologie pour communique et envoyer et recevoir ainsi des informations provenant d'un serveur.

1.7.1.3 Protocoles WAN

a. xDSL

Le terme DSL ou xDSL signifie (Digital Subscriber Line ou Ligne numérique d'abonné) et regroupe l'ensemble des technologies mises en place pour un transport numérique de l'information sur une simple ligne de raccordement téléphonique. [6] Les technologies xDSL sont divisées en deux grandes familles, celle utilisant une transmission symétrique et celle utilisant une transmission asymétrique. Dans une transmission symétrique le débit ascendant et descendant sont les mêmes par contre lors d'une transmission asymétrique, ces deux débits sont différents. [7]

b. Les réseaux cellulaires

Les différentes technologies 2G, 3G, 4G et bientôt le 5G offrent les possibilités à accéder à internet avec un débit de plus en plus évolutif. L'avantage principal des réseaux cellulaires c'est sa zone de couverture ainsi la limite de portée n'est plus un souci.

1.7.2 Protocoles Applicatifs

- IPv4 / IPv6 / 6LoWPAN

D'ici à 2020, 25 milliards d'appareils seront connectés à internet via l'Internet of Things (IoT). Pour en assurer la performance, nous avons besoin du nouveau protocole réseau IPv6, l'héritier de l'IPv4. Le protocole réseau classique IPv4 assure l'adressage de seulement 4,3 milliards d'appareils. Trop peu au vu de l'explosion de l'IoT : pas moins de 25 milliards d'objets seront connectés à internet d'ici 2020. L'IPv6 prend, quant à lui, en charge 340 sextillions (un milliard fois un milliard fois un milliard fois un milliard) d'adresses. Il s'avère donc crucial pour les fabricants d'appareils IoT. Le nouveau protocole garantit, en effet, que leurs appareils pourront continuer à fonctionner sur internet.

6LoWPAN est l'acronyme de IPv6 Low Power Wireless Personal Area Networks. C'est un réseau sans fil à faible puissance ou tous les nœuds ont leurs propres adresses IPV6, qui leurs permettent de se connecter directement à l'internet. 6LoWPAN est un mécanisme de compression et d'encapsulation d'entête permettant aux paquets IPv6 d'être envoyés via le protocole de communication IEEE-802.15.4.

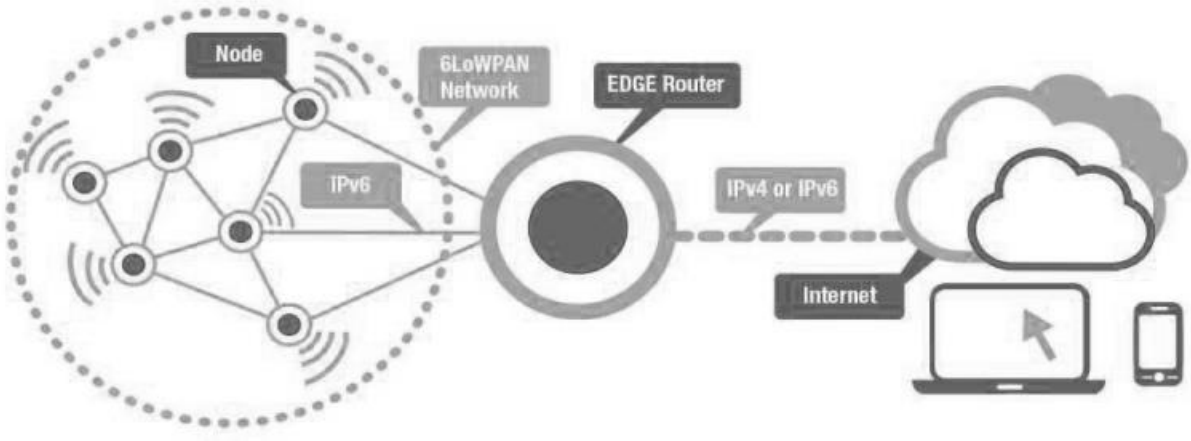

Figure 1.04 : *Réseau 6LoWPAN*

Noeuds	Routeur edge
- Tous les nœuds du réseau 6LoWPAN ont une adresse IPv6. - Les nœuds nécessitent une faible mémoire et une faible puissance de traitement - Dans une application domotique les nœuds du réseau peuvent être des capteurs de fumée, capteurs de présence…	- l'entête IPv6 comprimé nécessite un dispositif intermédiaire qui assure la conversion entre 6LoWPAN et l'entête IP standard. - le routeur edge peut être vu comme une passerelle simplifié de passage d'adresse IPv6 compressé en adresse IP internet standard (IPV4 ou IP)

Tableau 1.01: *Description d'un réseau 6LoWPAN*

- TCP/UDP : Ce sont des protocoles de communication en mode connectée et non connectée
- HTTP

Le protocole HTTP (HyperText Transfer Protocol) est le protocole le plus utilisé sur Internet. Le but du protocole HTTP est de permettre un transfert de fichiers (essentiellement au format HTML) localisés grâce à une chaîne de caractères appelée URL entre un navigateur (le client) et un serveur Web.

- CoAP

Bien que l'infrastructure du web soit disponible et utilisable par les appareils IoT, elle est trop lourde pour la majorité des applications de l'IoT. En juillet 2013, l'IETF a publié le Protocole d'application contrainte (CoAP, Constrained Application Protocol) pour utilisation avec une faible puissance et des nœuds et réseaux à pertes (limitées) (LLNs). CoAP, comme HTTP, est un protocole REST.

- MQTT

MQ Telemetry Transport (MQTT) est un protocole open source qui a été développé et optimisé pour les appareils limités et à faible bande passante, à latence élevée, ou pour les réseaux non fiables. Il s'agit d'un transport de messages de publication/d'abonnement qui est extrêmement léger et idéal pour le raccordement à des réseaux de petits appareils avec une bande passante minimale. MQTT utilise efficacement la bande passante, accueille toutes les données, et est en permanence informé de l'état de la session, car il utilise le protocole TCP. Il est destiné à minimiser les besoins en ressources de l'appareil tout en essayant d'assurer une fiabilité et un certain degré d'assurance de livraison avec des niveaux de service.

MQTT cible de grands réseaux de petits appareils qui doivent être surveillés ou contrôlés à partir d'un serveur back-end sur Internet. Il n'a pas été conçu pour le transfert d'appareil à appareil. Il n'a pas non plus été conçu pour « diffuser de manière multiple » des données vers de nombreux

récepteurs. MQTT est simple, offrant seulement quelques options de contrôle. Les applications utilisant MQTT sont généralement lentes dans le sens que la définition du « temps réel » dans ce cas est généralement mesurée en secondes.

- XMPP

XMPP (Extensible Messaging and Presence Protocol) est un bon exemple de technologie web existante qui trouve un nouvel emploi dans l'espace IoT.
XMPP prend ses racines dans les informations de messagerie instantanée et de présence, et s'est étendu aux appels vocaux et vidéo, à la collaboration, au middleware léger, à la syndication de contenu, et au routage généralisé de données XML. C'est un concurrent pour la gestion à grande échelle des produits blancs de consommation tels que les lave-linge, les sèche-linge, les réfrigérateurs, et ainsi de suite.
Les atouts de XMPP sont son adressage, sa sécurité et évolutivité. Cela le rend idéal pour les applications IoT grand-public.

1.8 Réseau Low Power Wide Area

Les réseaux LPWA pour Low Power Wide Area, comme le laisse deviner l'acronyme, sont des réseaux sans fils basse consommation, bas débit et longue portée, optimisés pour les équipements aux ressources limitées pour lesquels une autonomie de plusieurs années est requise. Ces réseaux conviennent particulièrement aux applications qui n'exigent pas un débit élevé.
Les LPWAN utilisent les bandes de fréquences à usage libre – sans licence – ISM (Industriel, Scientifique et Médical) disponibles mondialement, contrairement aux opérateurs mobiles qui utilisent des bandes sous licence pour lesquelles ils payent l'attribution. Il est à noter que l'utilisation des bandes ISM implique le partage des ressources avec les concurrents et avec les autres technologies (RFID, WiFi, Bluetooth, ZigBee, etc.). Toutefois, ce n'est pas la jungle non plus : ces bandes de fréquences sont régulées par des autorités organisatrices et il est tenu de respecter des règles d'utilisation. Compte tenu des faibles débits et de la faible occupation spectrale des signaux, il faut en moyenne, pour un réseau LPWA, 10 fois moins d'antennes pour couvrir la même surface qu'un réseau cellulaire traditionnel.

Les caractéristiques d'un réseau LPWA	
Caractéristique	**Valeur cible**
Portée	3 – 50 km selon l'environnement
Autonomie	entre 5 et 15 ans
Débit	de 0.1 à quelques centaines de kbits par seconde
Coût abonnement	quelques euros par an
Coût module	quelques euros

Tableau 1.02: *Caractéristiques d'un réseau LPWA*

1.8.1 Sigfox

Commençons les présentations avec Sigfox, une société pionnière sur le marché des LPWAN, arrivée en 2009 avec une technologie basée sur la transmission de signaux sur une bande ultra étroite (UNB pour Ultra Narrow Band) d'une centaine de hertz contre quelques centaines de kiloHertz pour les réseaux cellulaires traditionnels. Sigfox est un opérateur qui fournit sa technologie de connectivité bas débit au travers de son propre réseau cellulaire.

La technologie utilise les bandes ISM : autour de 868 MHz pour l'Europe et 915 MHz pour les USA.

Un capteur, équipé d'un module et d'un abonnement au réseau Sigfox, peut envoyer jusqu'à 140 messages par jour. Chaque message pouvant contenir jusqu'à 12 octets de données réelles de charge utile (l'identifiant du périphérique est transmis par le protocole), ce qui est largement suffisant pour la majorité des applications de monitoring. Avec une portée comprise entre 30 et 50 Km dans les zones rurales et entre 3 et 10 km dans les zones urbaines, Sigfox offre un débit de 100 bits par seconde

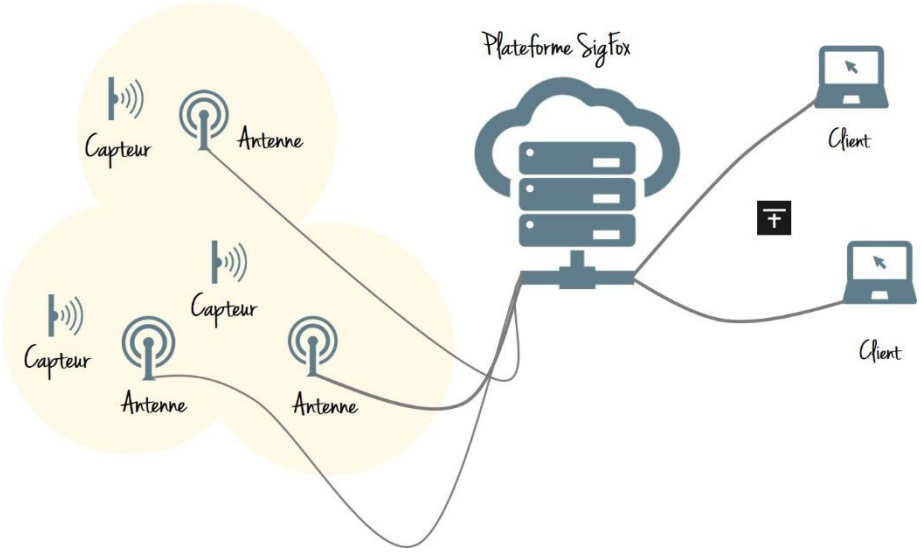

Figure 1.05 : *Schéma d'architecture d'un réseau SigFox*

L'intégration des données et la gestion des périphériques sont effectuées à partir de la plateforme web fournit par l'opérateur ou par ses partenaires. Il est à noter que dans les premières versions du protocole, celui-ci ne supportait que les communications unidirectionnelles : les capteurs transmettaient mais ne pouvaient pas recevoir de données.

Depuis, le protocole permet une communication bidirectionnelle mais extrêmement limitée : le capteur est disposé à recevoir quelques octets, pour son paramétrage par exemple, pendant un créneau très court qui suit sa phase d'émission.

1.8.2 LoRaWan

LoRaWan (Long Range Radio Wide Area Network) est un réseau LPWAN basé sur la technologie radio LoRa.LoRa utilise une technique d'étalement de spectre pour la transmission des signaux radio (chirp spread spectrum).

Sur un réseau LoRaWan, les données émises par les équipements sont centralisées par des gateways (des concentrateurs) qui transmettent les données à leur tour vers le serveur de gestion en ligne. La liaison entre les gateways et le serveur s'appuie sur des technologies très haut débit (Ethernet, Réseaux mobiles).

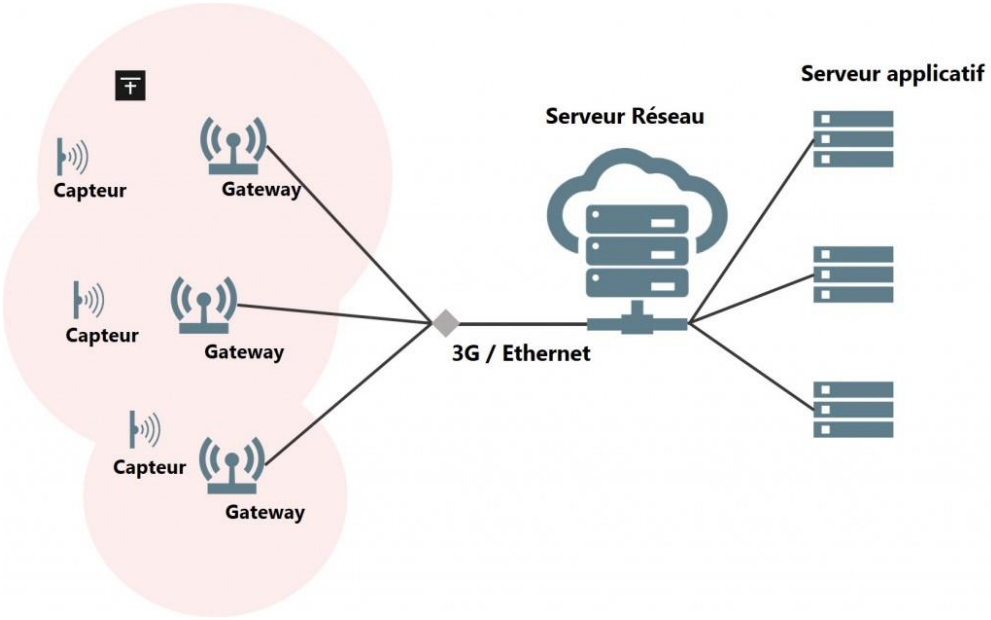

Figure 1.06 : *Schéma d'architecture d'un réseau LoRaWan*

Lora utilise également les bandes de fréquences ISM avec une portée comprise entre 15 et 20 km dans les zones rurales et entre 3 et 8 km dans les zones urbaines.

LoRaWan revendique un débit adaptatif compris entre 0,3 et 50 kbits par seconde et une communication bidirectionnelle moins limitée que son concurrent direct SigFox.

Pour cela, LoRaWan définit trois classes d'équipements :

- Classe A : cette classe concerne les équipements dont la consommation d'énergie doit être la plus faible possible. Les équipements de classe A permettent une communication bidirectionnelle par l'allocation de deux créneaux courts de réception après chaque émission.
- Classe B : en plus des créneaux de réception prévus par la classe A, les terminaux de la classe B planifient l'ouverture de créneaux supplémentaires à intervalles réguliers paramétrables. La planification de ces créneaux se réalise à la réception d'une balise de synchronisation émise par le réseau. Ce mode consomme davantage que le précédent en sollicitant davantage la liaison radio de l'équipement vers le réseau.
- Classe C : les équipements de classe C sont quasi-constamment « à l'écoute » d'une réception de données par le réseau. Les créneaux de réception ne se ferment que lorsque l'équipement est en phase d'émission. Ce mode est le plus consommateur, il n'est pas adapté aux équipements sur batterie ou pile.

1.8.3 LTE-M

LTE (Long Term Evolution), plus communément appelée « 4G », est une technologie très haut débit pour les communications mobiles. Cette technologie est définie par un organisme en charge de la spécification des technologies de réseaux mobiles (GSM, UMTS, LTE) nommé 3GPP. Au sein de cet organisme, le groupe de travail dédié aux réseaux d'accès radio (3GPP-RAN) travaille depuis peu (Q4 2014), sur une version M2M de la technologie LTE : la technologie LTE-M (LTE – Machine to Machine). Pour répondre aux besoins des applications M2M, la technologie LTE-M propose des débits (~ 1Mbits), une puissance d'émission (max 20dBm) et une occupation spectrale (1,4 MHz) beaucoup plus faibles que la technologie LTE.

Initialement spécifiée dans la version 12 (Rel.12) des spécifications 3GPP, une version 13 (Rel.13) est prévue pour le premier trimestre 2016. La Rel.13 prévoit notamment un mode « bande étroite » (NB-Narrow Band) avec une bande passante à 200 kHz, et un débit plus faible encore (~ 150kbits par seconde). La question de l'utilisation des bandes ISM en complément des bandes sous licence se pose également. Ces travaux menés par 3GPP, préparent l'arrivée de la prochaine génération de réseaux mobiles – la 5G – prévue en 2020 et pour laquelle l'IoT et notamment les LPWAN occuperont une place centrale. Vous l'avez compris, la technologie LTE-M est encore en cours de définition ; il n'y a pas encore matière à prototyper avec LTE-M.

1.9 Conclusion

La technologie est encore jeune, mais elle va prendre un tournant décisif ces prochaines années pour devenir complètement intégrée à nos modes de vie. Et avec le déploiement de la prochaine évolution des réseaux mobiles, la 5G, l'Internet des Objets aura tous les moyens de ses ambitions. Au final, l'Internet des Objets va changer la façon dont nous interagissons avec le monde qui nous entoure. Les objets du quotidien que nous considérions comme étant banals vont commencer à avoir un comportement autonome et à anticiper nos besoins.

CHAPITRE 2
MODELE CLIENT-SERVEUR

2.1 Introduction

Ces vingt dernières années ont vues une évolution majeure des systèmes d'information à savoir le passage d'une architecture centralisée à travers de grosse machines vers une architecture distribuées basée sur l'utilisation de serveurs et de postes clients grâce à l'utilisation des PC et des réseaux. Cette évolution a été possible essentiellement grâce à 2 facteurs qui sont la baisse des prix de l'informatique personnelle et le développement des réseaux.

2.2 Définitions

2.2.1 Client-serveur

Le modèle client-serveur se base sur le mode « message ». Le client envoie dans un premier message une requête d'exécution d'un traitement à un serveur. Le serveur effectue le travail et fournit dans un second message la réponse. Ce genre de communication par message de données en mode asynchrone est supporté par les protocoles de transport. [8]

Le modèle client-serveur se base sur l'appel de procédure distance.

- Le client (l'appelant) fait exécuter une procédure à distance par un site (le serveur).
- Le serveur (l'appelé) exécute la procédure.
- Le mécanisme de procédure distance se base sur la sémantique et la présentation syntaxique.

Le modèle client-serveur s'articule autour d'un réseau auquel sont connectés deux types d'ordinateurs le serveur et le client. Le client et le serveur communiquent via des protocoles.

Les applications et les données sont réparties entre le client et le serveur de manière à réduire les coûts. Le client-serveur représente un dialogue entre deux processus informatiques par l'intermédiaire d'un échange de messages. Le processus client sous-traite au processus serveur des services à réaliser. Les processus sont généralement exécutés sur des machines, des OS et des réseaux hétérogènes.

2.2.2 Serveur

Un serveur est initialement passif, il attend, il est à l'écoute, prêt à répondre aux requêtes envoyées par des clients. Dès qu'une requête lui parvient, il la traite et envoie une réponse.

Le processus serveur :

- offre un point d’entrée sur le réseau
- entre dans une boucle infinie d’attente de requêtes
- à la réception d’une requête, déclenche les processus
- associés à la requête, puis émet la réponse vers le client
- Deux types de serveurs :

➢ itératifs : ne gèrent qu’un seul client à la fois

➢ parallèles : fonctionnent en mode concurrent

Un serveur WEB est une machine sur laquelle le service HTTP est à l'écoute de requêtes http en provenance du réseau. L’application cliente d'un serveur HTTP est généralement un logiciel navigateur. Lorsqu'un utilisateur saisi une URL dans la barre d'adresse, il émet une requête à destination du service HTTP actif sur un serveur.

Par défaut, le service HTTP utilise le port 80, mais il est possible d'utiliser un autre port, soit pour des raisons de sécurité, soit parce qu'un serveur assure ce service pour différents sites web.

2.2.3 Client

On appelle logiciel client un programme qui utilise le service offert par un serveur. Une application cliente est moins complexe qu’une application serveur. [9]

• La plupart des applications clientes ne gèrent pas d’interactions avec plusieurs serveurs.

• La plupart des applications clientes sont traitées comme un processus conventionnel ; un serveur peut nécessiter des accès privilégiés.

Dans un réseau informatique, un **client** est le logiciel qui envoie des demandes à un serveur. Il peut s'agir d'un logiciel manipulé par une personne. L'ordinateur client est généralement un ordinateur personnel ordinaire, équipés de logiciels relatifs aux différents types de demandes qui vont être envoyées, comme un navigateur web, un logiciel client pour le World wide web.

Les caractéristiques d’un client sont les suivantes : il est d’abord actif (ou maître), il envoie des requêtes au serveur, il attend et reçoit les réponses du serveur.

2.2.4 Back-end

Le Back-End, c'est un peu comme la partie immergée de l'iceberg. Elle est invisible pour les visiteurs mais représente une grande partie du développement d'un projet web. Sans elle, le site web reste une coquille vide.

On peut décomposer le Back-End en trois parties essentielles:

- Un serveur (ou hébergement web)
- Une application (en l'occurrence le site web)
- Une base de données (ou l'on stocke les données de l'application)

Pour pouvoir conserver les mots de passe, les préférences qui ont été saisi grâce aux éléments de Front-End, il est nécessaire de les enregistrer dans une **base de données**. La base de données est comparable à un grand tableau avec des colonnes contenant par exemple «nom», «prénom», «mot de passe». Pour pouvoir conserver, traiter, modifier ces données et fournir des informations à jour sur un site internet (comme des actualités, des fiches produits, des images, des vidéos), le développeur Back-End va utiliser des **langages de programmation «dynamique»**.

Enfin, le développeur Back-End met également en place et configure le serveur qui accueillera le site lui-même.

Dans un environnement client-serveur, on considère que le client fait office de front-end alors que le serveur fait office de back-end.

2.2.5 Front-end

Lorsque l'on parle de «Front-End», il s'agit finalement des éléments du site que l'on voit à l'écran et avec lesquels on peut interagir. Ces éléments sont composés de HTML, CSS et de Javascript contrôlés par le navigateur web de l'utilisateur.

Les champs de compétence du Front-End peuvent être séparés en deux :

– Le **design**

– Le **développement** HTML, CSS, Javascript

2.3 Architecture Client-serveur

Une application fonctionne selon une architecture client-serveur quand : les machines « clientes » contactent une machine « serveur » afin que ce serveur leur fournisse un service (via l'exécution d'un programme) [10].

- Les clients envoient des requêtes
- Le serveur envoie des réponses

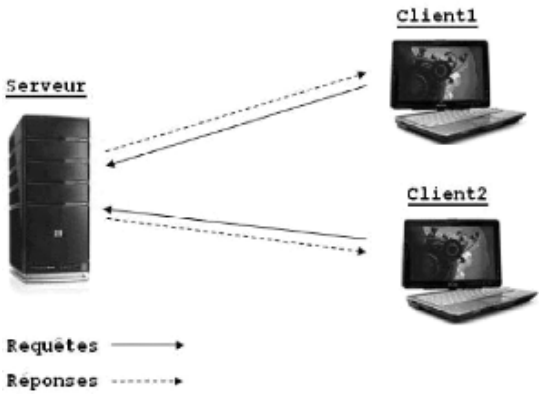

Figure 2.01 : *Architecture Client-Serveur*

2.3.1 Architecture Multi-Tier

Traditionnellement une application informatique est un programme exécutable sur une machine qui représente la logique de traitement des données manipulées par l'application.
Ces données peuvent être saisies interactivement via l'interface ou lues depuis un disque.
L'application émet un résultat sous forme de données qui sont, soit affichées, soit enregistrées sur un disque.

Figure 2.02 : *Schéma traitement données*

Dans cette figure 2.02, les traitements, les données d'entrées, les données de sortie sont sur une seule machine.
On peut alors séparer l'application en différentes parties :

- La couche interface homme machine
- La couche de traitement
- La couche de gestion des données.

Et toute application possède ces trois parties. On parle de couches, de niveaux ou de tier (de l'anglais tier : étage).

2.3.1.2 Architecture 2-Tier

Ce type d'application permet d'utiliser toute la puissance des ordinateurs présents sur le réseau et permet de fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données qui restent gérées de façon centralisée.

La gestion des données est prise en charge par un SGBD centralisé sur un serveur dédié. On interroge ce serveur à travers un langage de requête, le plus courant étant SQL.

Le dialogue entre le client et le serveur se résume donc à l'envoi de requêtes et aux données en réponse. [11]

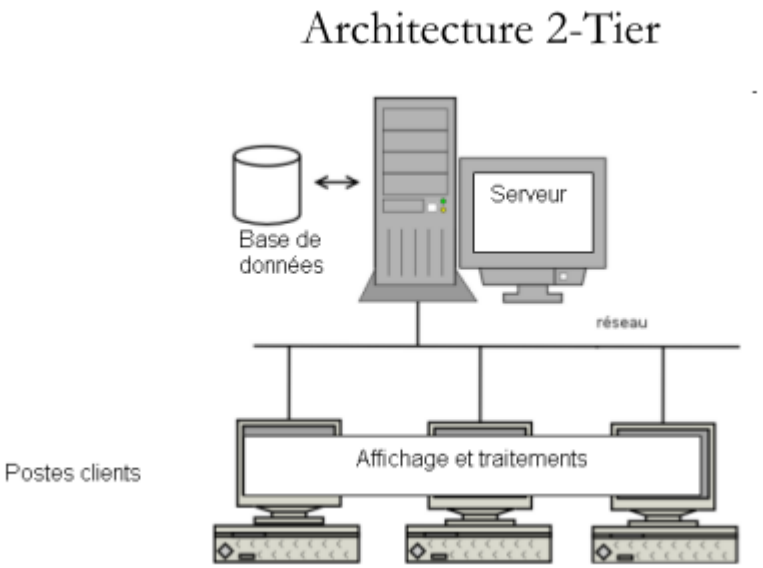

Figure 2.03 : *Architecture 2-tier*

2.3.1.3 Architecture 3-Tier

Les points importants d'une application 3-tiers :

- Le client ne sert qu'à requêter et à afficher les réponses du serveur.
- Le serveur lui s'occupe des calculs et même de requêter des serveurs additionnels.

Les avantages d'une architecture 3-tiers sont nombreux. Tout d'abord cette architecture étant plus divisé permet d'avoir du point de vue du développement, une spécialisation des développeurs selon le niveau de l'application (un développeur front-end, back-end…)

Et enfin, cette architecture offre une flexibilité beaucoup plus importante que l'architecture 2-tiers. En effet, la portabilité du tiers serveur permet d'envisager une allocation et ou modification dynamique au gré des besoins évolutifs au sein d'une entreprise.

Seul inconvénient selon moi de ce genre d'architecture est le coût. En effet d'après certaines études une architecture 3-tiers serait plus onéreuse.

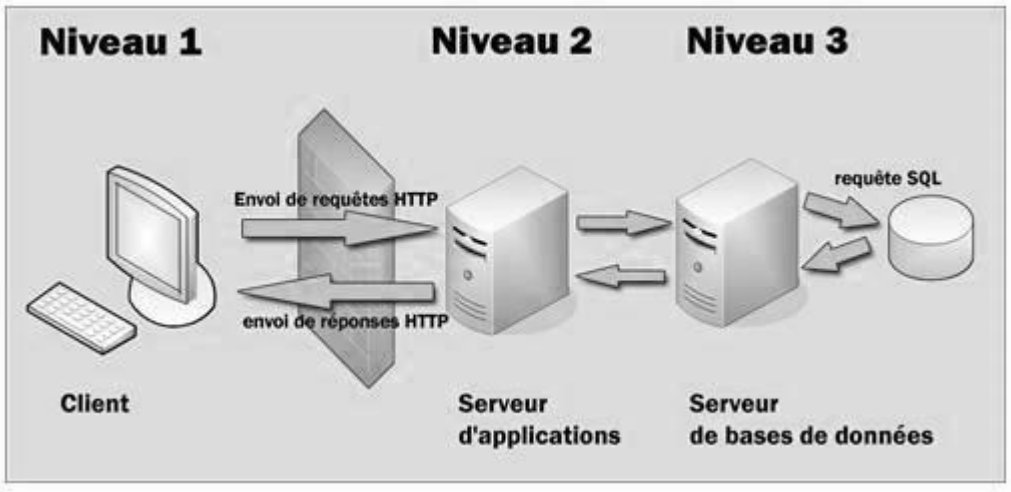

Figure 2.04 : *Architecture 3-tier*

La figure 2.04 nous montre l'architecture à 3 niveaux qui sont le client, le serveur d'application ainsi que le serveur de base de données.

2.3.2 Caractéristiques des systèmes client-serveur

Les éléments qui caractérisent une architecture client-serveur sont :

- Service

Le modèle client-serveur est une relation entre des processus qui tournent sur des machines séparées. Le serveur est un fournisseur de services. Le client est un consommateur de services.

- Partage de ressources

Un serveur traite plusieurs clients et contrôle leurs accès aux ressources

- Protocole asymétrique

Conséquence du partage de ressources, le protocole de communication est asymétrique le client déclenche le dialogue ; le serveur attend les requêtes des clients.

- Transparence de la localisation

L'architecture client-serveur doit masquer au client la localisation du serveur (que le service soit sur la même machine ou accessible par le réseau). Transparence par rapport aux systèmes d'exploitation

et aux plates-formes matérielles. Idéalement, le logiciel client serveur doit être indépendant de ces deux éléments

- Message

Les messages sont les moyens d'échanges entre client et serveur.

- Encapsulation des services

Un client demande un service. Le serveur décide de la façon de le rendre une mise à niveau du logiciel serveur doit être sans conséquence pour le client tant que l'interface message est identique.

- Evolution

Une architecture client-serveur doit pouvoir évoluer horizontalement (évolution du nombre de clients) et verticalement (évolution du nombre et des caractéristiques des serveurs).

2.4 Avantages et inconvénients

2.4.1 Avantages

Le modèle client/serveur est particulièrement recommandé pour des réseaux nécessitant un grand niveau de fiabilité, ses principaux atouts sont :

- **des ressources centralisées** : étant donné que le serveur est au centre du réseau, il peut gérer des ressources communes à tous les utilisateurs, comme par exemple une base de données centralisée, afin d'éviter les problèmes de redondance et de contradiction.
- **une meilleure sécurité** : car le nombre de points d'entrée permettant l'accès aux données est moins important.
- **une administration au niveau serveur** : les clients ayant peu d'importance dans ce modèle, ils ont moins besoin d'être administrés.
- **un réseau évolutif** : grâce à cette architecture il est possible de supprimer ou rajouter des clients sans perturber le fonctionnement du réseau et sans modification majeure. [12]

2.4.2 Inconvénients

Si trop de client veulent communiquer sur le serveur en même temps, ce dernier risque de ne pas supporter la charge.

Si le serveur n'est plus disponible, plus aucun des clients ne fonctionne. Les coûts de mise en place et de maintenance sont élevés. [13]

En aucun cas les clients ne peuvent communiquer entre eux, entraînant une asymétrie de l'information au profit des serveurs.

2.5 Node.js

2.5.1 Présentation de Node.js

Jusqu'ici, JavaScript avait toujours été utilisé du côté du client, c'est-à-dire du côté du visiteur qui navigue sur notre site. Le navigateur web du visiteur (Firefox, Chrome, Internet Explorer...) exécute le code JavaScript et effectue des actions sur la page web.

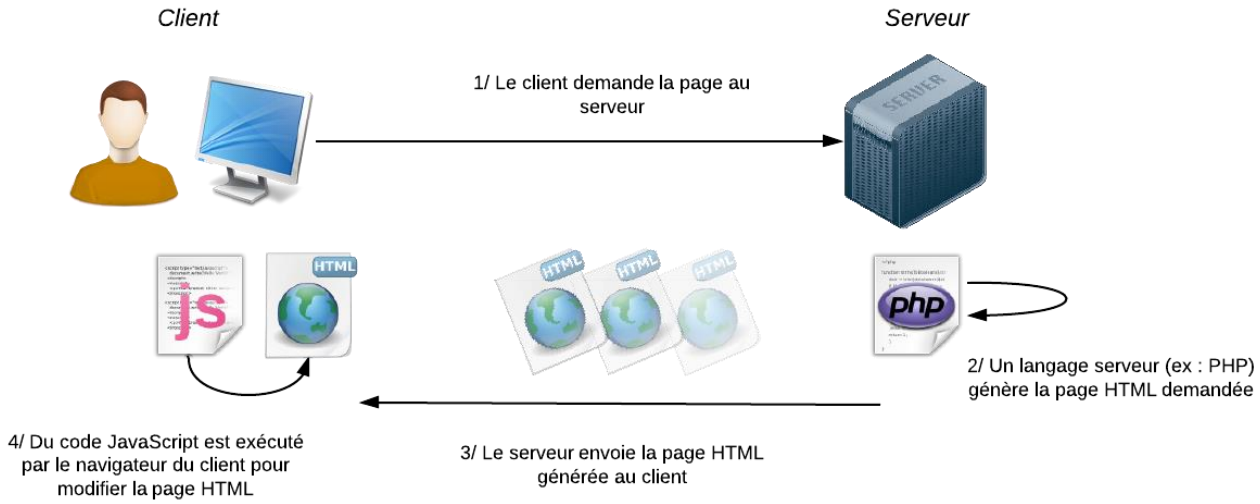

Figure 2.05 : *Le schéma classique : PHP sur le serveur, JavaScript chez le client*

Node.js offre un environnement côté serveur qui nous permet aussi d'utiliser le langage JavaScript pour générer des pages web. En gros, il vient en remplacement de langages serveur comme PHP, Java EE, etc.

Ainsi Node.js est une plateforme :

- permettant d'écrire des applications;
- basée sur le langage JavaScript (et l'interpréteur de Chrome);
- adaptée à la programmation de serveur Web;

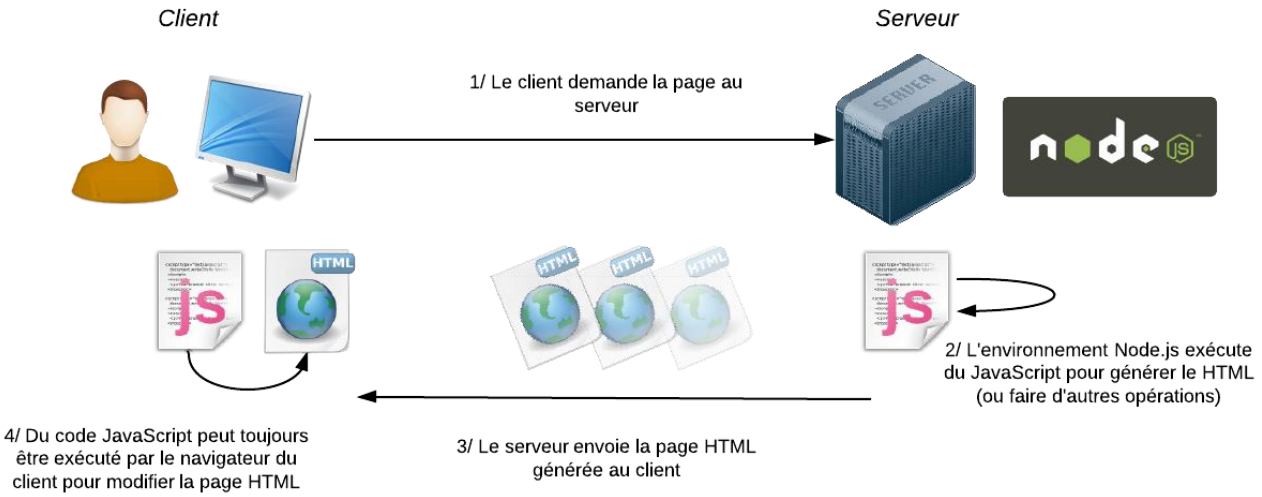

Figure 2.06 : *Avec Node.js, on peut aussi utiliser du JavaScript sur le serveur !*

La particularité de Node.js c'est qu'il utilise le JavaScript. JavaScript est un langage basé sur les évènements, donc Node.js est lui-même basé sur les évènements. Du coup, c'est toute la façon d'écrire des applications web qui change. Et c'est de là que Node.js tire toute sa puissance et sa rapidité.

Avec Node.js, vous pouvez créer des applications rapides comme : un serveur de Chat, un système d'upload très rapide et de façon générale n'importe quelle application qui doit répondre à de nombreuses requêtes rapidement et efficacement, en temps réel.

2.5.2 Rapidité de Node.js

Si Node.js est rapide, cela tient principalement à deux choses : le moteur V8 et son fonctionnement non bloquant.

2.5.2.1 Le moteur V8

Node.js utilise le moteur d'exécution ultrarapide V8 de Google Chrome. Ce moteur V8 avait fait beaucoup parler de lui à la sortie de Google Chrome, car c'est un outil open source créé par Google qui analyse et exécute du code JavaScript très rapidement.

Jusqu'à la sortie de Chrome, la plupart des navigateurs lisaient le code JavaScript de façon peu efficace : le code était lu et interprété au fur et à mesure. Le navigateur mettait beaucoup de temps à lire le JavaScript et à le transformer en code machine compréhensible pour le processeur.

Le moteur V8 de Google Chrome, qui est réutilisé ici par Node.js, fonctionne complètement différent. Très optimisé, il fait ce qu'on appelle de la compilation JIT (Just In Time). Il transforme

le code JavaScript très rapidement en code machine et l'optimise même grâce à des procédés complexes : *code inlining* et *copy elision*.

2.5.2.2 Modèle bloquant vs modèle non bloquant

Imaginez un programme dont le rôle est de télécharger un fichier puis de l'afficher. Voici comment on écrirait le code dans un **modèle bloquant** :

```
Télécharger un fichier
Afficher le fichier
Faire autre chose
```

Les actions sont effectuées dans l'ordre. Il faut lire les lignes de haut en bas :

1. Le programme va télécharger un fichier sur Internet
2. Le programme affiche le fichier à l'utilisateur
3. Puis ensuite le programme peut faire d'autres choses (effectuer d'autres actions).

Maintenant, on peut écrire le même code sur un **modèle non bloquant** :

```
Télécharger un fichier
    Dès que c'est terminé, afficher le fichier
Faire autre chose
```

Le programme n'exécute plus les lignes dans l'ordre où elles sont écrites. Il fait ceci :

1. Le programme lance le téléchargement d'un fichier sur Internet
2. Le programme fait d'autres choses (le programme suit son cours)
3. Dès que le téléchargement est terminé, le programme effectue les actions qu'on lui avait demandées : il affiche le fichier

Schématiquement, l'exécution du programme peut donc se représenter comme ça :

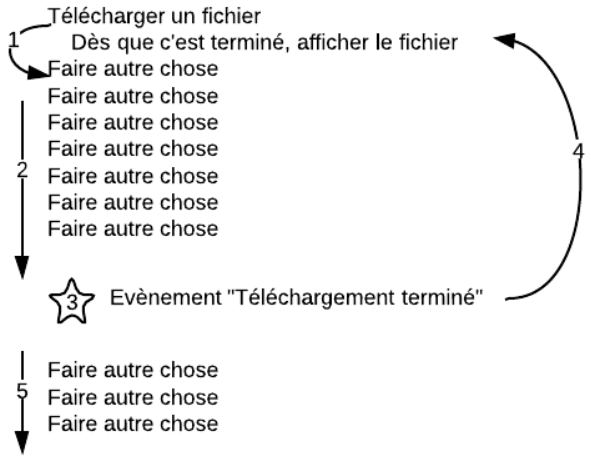

Figure 2.07 : *Le modèle non bloquant en programmation*

C'est justement comme ça que fonctionne Node.js. Dès que l'évènement "Fichier téléchargé" apparaît, une fonction appelée *fonction de callback* est appelée et effectue des actions (ici, la fonction de callback affiche le fichier).

2.5.2.3 Le modèle non bloquant avec Node.js

Comme JavaScript est un langage conçu autour de la notion d'évènement, Node.js a pu mettre en place une architecture de code entièrement non bloquante.

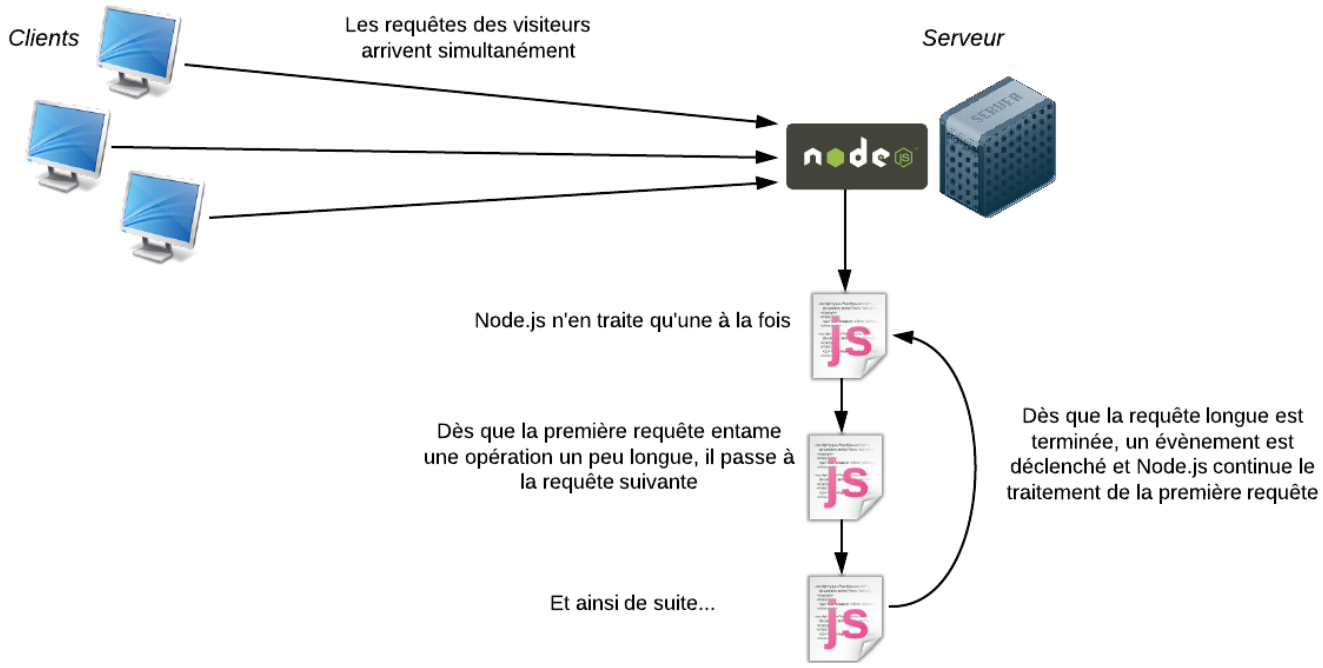

Figure 2.08 : *Illustration du modèle non bloquant avec Node.js*

Avec du vrai code cela. Voici un exemple de programme Node.js qui télécharge un fichier sur Internet et affiche "Fichier téléchargé !" quand il a terminé :

```
request('http://www.site.com/fichier.zip',function(error, response, body) {
    console.log("Fichier téléchargé !");
});
console.log("Je fais d'autres choses en attendant...");
```

La requête de téléchargement est lancée en premier. Ensuite, le programme fait d'autres choses (ici, il affiche un message dans la console, mais il pourrait faire n'importe quoi d'autre). Dès que le téléchargement est terminé, le programme va à la ligne 2 et affiche "Fichier téléchargé !".

On a là une fonction de callback. En JavaScript on peut tout à fait envoyer une fonction en paramètre d'une autre fonction. Cela signifie ici : "Exécute cette fonction quand le téléchargement est terminé". Ici, la fonction n'a pas de nom. On dit que c'est une fonction anonyme. Mais on pourrait décomposer ce code comme ceci, le résultat serait identique :

```
var callback = function (error, response, body) {
    console.log("Fichier téléchargé !");
});
request('http://www.site.com/fichier.zip', callback);
console.log("Je fais d'autres choses en attendant...");
```

La fonction de callback est enregistrée dans une variable. Comme toutes les fonctions, elle n'est pas exécutée tant qu'on ne l'a pas appelée.
Ensuite, on envoie cette fonction de callback en paramètre de la fonction request() pour dire : "Dès que la requête de téléchargement est terminée, appelle cette fonction de callback".

Supposons qu'on demande le téléchargement de 2 fichiers à Node.js :

```
var callback = function (error, response, body) {
    console.log("Fichier téléchargé !");
});
request('http://www.site.com/fichier.zip', callback);
request('http://www.site.com/autrefichier.zip', callback);
```

Si le modèle avait été bloquant, le programme aurait :

1. Lancé le téléchargement du fichier 1, et attendu qu'il se termine...
2. ... puis lancé le téléchargement du fichier 2, et attendu qu'il se termine.

Or, avec Node.js, les deux téléchargements sont lancés *en même temps*. Le programme n'attend pas la fin du premier téléchargement pour passer à l'instruction suivante.
Du coup, le téléchargement des 2 fichiers au total va beaucoup plus vite puisque le programme fait les 2 à la fois :

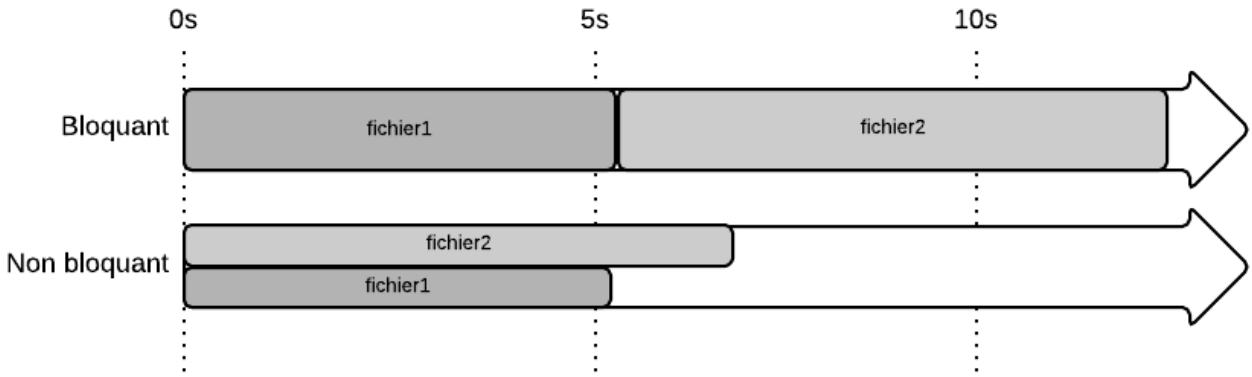

Figure 2.09 : *En modèle non bloquant (comme Node.js), les 2 fichiers sont téléchargés en même temps et l'ensemble finit plus vite*

Dans les applications web, il est courant d'avoir des opérations longues et bloquantes comme :

- Les appels aux bases de données
- Les appels à des services web (ex : l'API de Twitter)

Node.js nous évite de perdre du temps en nous permettant de faire d'autres choses en attendant que les actions longues soient terminées.

2.6 Websocket

WebSocket est une fonctionnalité supportée par l'ensemble des navigateurs récents. Elle permet un **échange bilatéral synchrone** entre le client et le serveur.
Habituellement, sur le Web, la communication est asynchrone. Le Web a toujours été conçu comme cela : le client demande et le serveur répond.

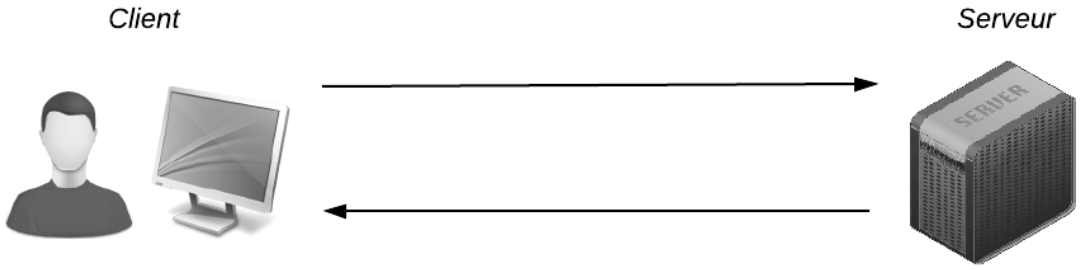

Figure 2.10 : *Communication asynchrone : le client demande, le serveur répond.*

C'était suffisant aux débuts du Web, mais c'est devenu trop limitant ces derniers temps. On a besoin d'une communication plus réactive et immédiate. Dans la figure 2.10 par exemple, le serveur ne peut pas décider de lui-même d'envoyer quelque chose au client (par exemple pour l'avertir "il y a un nouveau message !"). Il faut que le client recharge la page ou fasse une action pour solliciter le serveur, car celui-ci n'a pas le droit de s'adresser au client tout seul.
Websocket est une nouveauté du Web qui permet de laisser une sorte de "tuyau" de communication ouvert entre le client et le serveur. Le navigateur et le serveur restent connectés entre eux et peuvent s'échanger des messages dans un sens comme dans l'autre dans ce tuyau. Désormais, le serveur peut donc lui-même décider d'envoyer un message au client comme un grand.

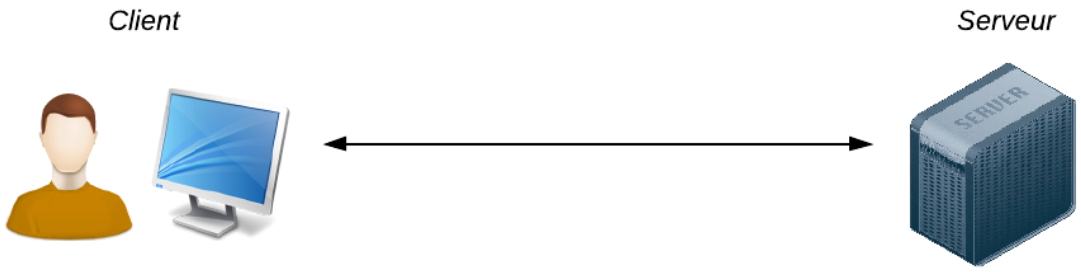

Figure 2.11 : *Communication synchrone avec websocket: un tuyau de communication reste ouvert entre client et serveur*

Il existe plusieurs modules de Node.js qui nous permettent d'utiliser les webSockets très facilement comme socket.io, ws et websocket. Et, comme tous les navigateurs ne gèrent pas webSocket, par exemple socket.io est capable d'utiliser d'autres techniques de communication synchrones si elles sont gérées par le navigateur du client.

2.7 La base de données MongoDB

MongoDB est une base de données "orientée document". Totalement open-source, cette dernière est développée en C++.

Figure 2.12 : *Logo MongoDB*

Contrairement à MySQL par exemple, MongoDB va stocker des données sous forme de JSON(Javascript Object Notation).

MongoDB est un système de base de données dans la mouvance NoSQL. Il est orienté documents. Son nom vient de *Humongous* qui veut dire énorme ou immense. L'objectif est donc de pouvoir gérer de grandes quantités de données. Le moteur de base de données facilite l'extension (on parle de *scaling*) si bien que l'on pourra supporter l'accroissement de la quantité de données par l'ajout de machines.

Dans un système de bases de données relationnelles, les informations sont stockées par ligne dans des tables. Ces tables sont mises en relation en utilisant des clés primaires et étrangères. Dans MongoDb, l'information est modélisée sur un document au format JSON.

2.7.1 NoSQL

NoSQL est une catégorie de base de données très différente de celle des bases de données SQL. Le terme NoSQL est souvent utilisé pour désigner des systèmes de gestion de données « différents de SQL » ou une approche de gestion des données qui n'intègre « pas seulement SQL ». La catégorie NoSQL englobe un certain nombre de technologies, notamment des bases de données de documents, des magasins de clés/valeurs, des magasins de familles de colonnes et des bases de données de graphiques, qui sont les plus courantes parmi les applications de jeux, de réseaux sociaux et IoT.

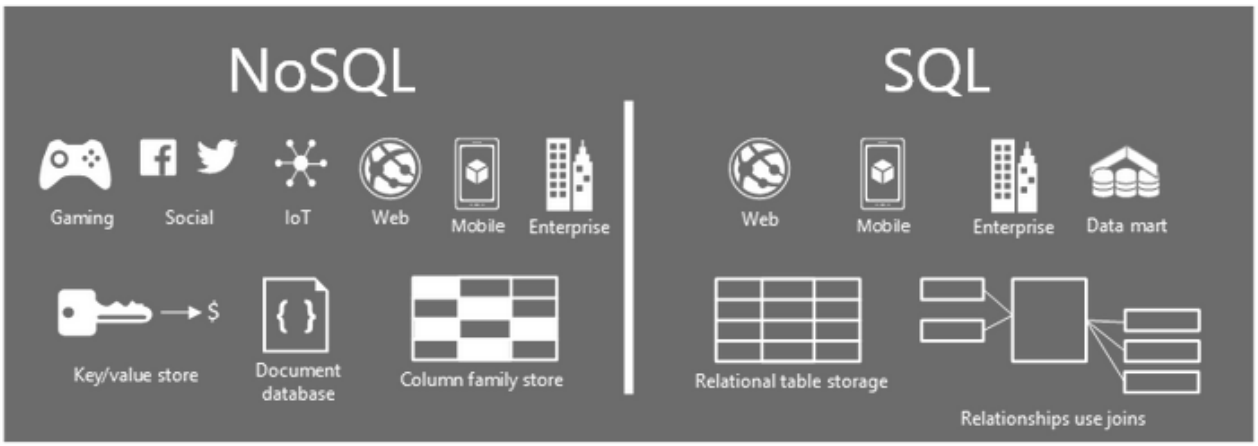

Figure 2.13 : *Comparaison entre NoSQL et SQL*

2.7.2 MongoDB

MongoDB fait partie d'une catégorie de systèmes de gestion de bases de données qu'on désigne sous le terme NoSQL. MongoDB est un SGBD NoSQL document-based.

Les SGBD NoSQL document-based: ils stockent les données sous la forme de documents au format variable mais structuré (généralement JSON ou XML); c'est à cette catégorie qu'appartient MongoDB.

MongoDB se présente sous la forme d'un serveur; un daemon/service système.

Par défaut, pour y accéder, on se connectera en TCP/IP sur le port TCP 27017. Comme la plupart des SGBD, il supporte la gestion d'utilisateurs/mots de passe pour s'identifier.

MongoDB stocke les données non pas sous la forme de tables à la structure statique, mais sous la forme de documents rédigés dans un langage proche de JSON: BSON (Binary jSON).

● En interne, les données sont stockées sous un format binaire; mais elles sont évidemment la plupart du temps lues, écrites et manipulées sous un format textuel (c'est le SGBD qui compile le format texte).

● BSON est en réalité un subset de JSON qui inclut quelques fonctionnalités additionnelles (notamment des types et fonctions non disponibles en JSON).

2.8 Conclusion

Le modèle client /serveur est la base de tous les services réseaux informatique, c'est pour cela que nous nous sommes intéressé par l'étude de ce modèle. Nous avons parlé également du serveur web Node.js qui tire toute sa puissance du modèle non bloquant. La base de données MongoDB qui est une base de données orientée document est un outils de stockage qui permettra l’exploitation de grands volumes de données facilement.

CHAPITRE 3
LE PROTOCOLE MQTT

3.1 Introduction

Créé en 1999 par le Dr Andy Stanford-Clark d'IBM (International Business Machines Corporation), et Arlen Nipper d'Arcom (maintenant Eurotech) MQTT est le protocole qui joue un rôle important dans l'Internet des objets. D'où l'intérêt de bien le connaître.
En août 2010, IBM a publié le protocole MQTT V3.1 spécification, qui a été utilisé dans le développement de futures normes. Eclipse, et plus précisément le projet Eclipse Paho, a été le premier à déplacer MQTT sur son chemin de normalisation. En octobre 2014, MQTT est devenu une norme OASIS,
MQTT, une norme de base sur l'Internet des objets (IoT) développée par le consortium OASIS, a été approuvée pour publication par l'Organisation internationale de normalisation (ISO) et la Commission électrotechnique internationale (IEC) en janvier 2016. La version 3.1.1 du MQTT a été votée par l'entremise du Comité technique mixte sur les technologies de l'information (JTC1) de l'ISO et de la IEC et a reçu la désignation «ISO / IEC 20922».

MQTT est un protocole de transport de messagerie de publication / abonnement Client Server. Il est léger, simple et conçu de façon à être facile à mettre en œuvre. Ces caractéristiques le rendent idéal pour une utilisation dans de nombreuses situations, y compris des environnements contraints tels que pour la communication dans les contextes Machine to Machine (M2M) et Internet of Things (IoT) où une petite empreinte de code est requise et / ou la bande passante du réseau est à la prime.

3.2 Le protocole MQTT

3.2.1 Présentation

MQTT permet concrètement aux appareils d'envoyer des informations sur un sujet donné à un serveur qui fonctionne comme un courtier de messages appelé broker. Le broker pousse ces informations vers les clients qui se sont précédemment abonnés. Pour l'utilisateur, un sujet

ressemble à un chemin hiérarchique. [14] Les clients peuvent s'abonner à un niveau spécifique de la hiérarchie d'un sujet ou à plusieurs niveaux s'ils utilisent un caractère générique.

3.2.2 Le modèle de publisher / subscriber

Le modèle publish / subscribe (pub / sub) est une alternative au modèle client-serveur traditionnel, où un client communique directement avec un terminal.

Avec ce pattern, chaque application n'a pas besoin de connaître les autres applications avec lesquelles elle communique. Les messages ne sont pas envoyés d'un émetteur à un récepteur, mais sont échangés sur des files d'attente (les « topics ») :

Une application réceptrice peut s'abonner (« subscribe ») à un topic pour recevoir les messages associés, sans forcément connaître les applications qui vont émettre dessus.

De la même manière, une application émettrice va envoyer (« publish ») un message dans un topic sans connaître les applications qui se sont abonnées. Les abonnés, s'il y en a, sont alors notifiés et reçoivent le message.

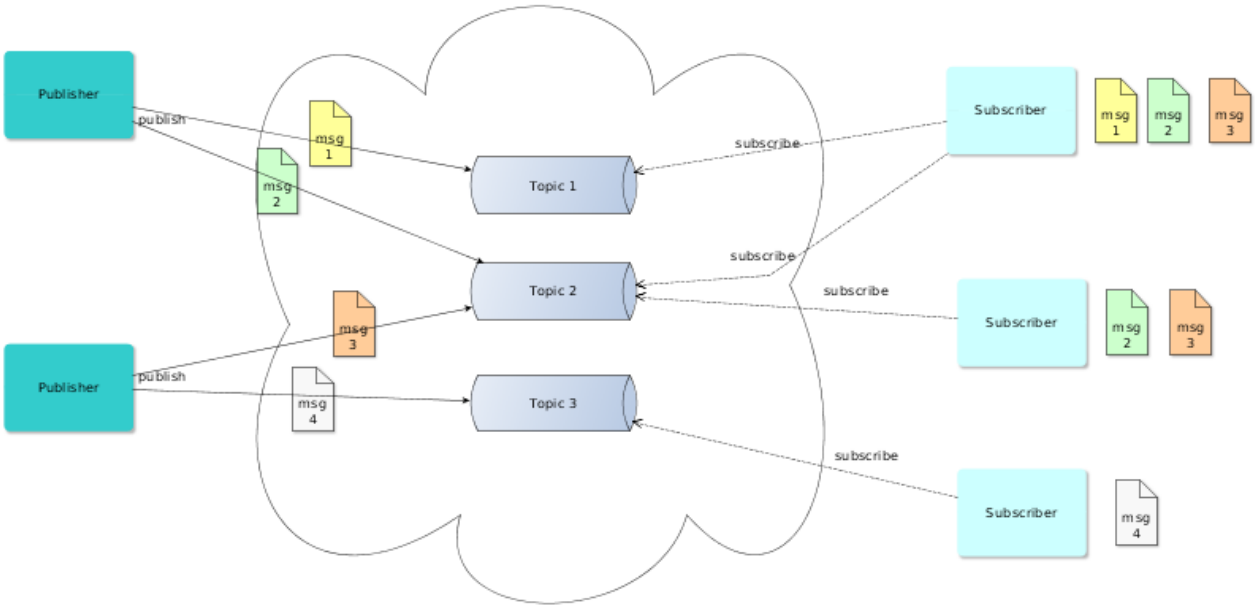

Figure 3.01 : *Modèle publisher/subscriber*

Le modele pub / sub est le découplage du publisher (ou celui qui publie le message sur un toipic) et le récepteur, qui peut être différencié en plusieurs dimensions:

- Découplage de l'espace: le publisher et le subscriber n'ont pas besoin de se connaître (par adresse IP et port par exemple)
- Découplage de temps: le publisher et le subscriber n'ont pas besoin de s'exécuter en même temps.
- Découplage de synchronisation: les opérations sur les deux composants ne sont pas arrêtées pendant la publication ou la réception

Donc, ce qui est intéressant, c'est comment le broker filtre tous les messages, de sorte que chaque subscriber reçoit uniquement les messages qui l'intéressent.

Le filtrage par sujet est basé sur un sujet ou « topic », qui fait partie de chaque message. Le client récepteur s'abonne sur les sujets auxquels il est intéressé à travers le broker et à partir de là il reçoit tous les messages publiés sur le sujet donné. Les « Topics » sont en général des chaînes de caractères avec une structure hiérarchique que nous détaillerons un peu plus tard.

3.2.3 Les spécifications

La spécification identifie un certain nombre de fonctions distinctives du protocole :

- Il s'agit d'un protocole de type publication/abonnement.
- Outre la distribution de type "un à plusieurs" des messages, la fonction de publication/abonnement découple les applications. Ces deux fonctions sont importantes dans les applications qui ont beaucoup de clients.
- Il ne dépend aucunement du contenu des messages.
- Il fonctionne sur TCP/IP, qui fournit la connectivité réseau de base.
- Il propose trois qualités de service pour la distribution des messages que nous verrons plus tard.
- Il dispose de la fonction « Dernières volontés et testament » qui notifie les abonnés de la déconnexion anormale d'un client du serveur MQTT.

3.2.4 Les avantages du protocole

MQTT est ouvert, simple, léger et facile à mettre en œuvre.
Il est idéal pour répondre aux besoins suivants :

- Particulièrement adapté pour utiliser une très faible bande passante,
- Idéal pour l’utilisation sur les réseaux sans fils,

- Faible consommateur en énergie,
- Très rapide, il permet un temps de réponse supérieur aux autres standards du web actuel,
- Permet une forte fiabilité si nécessaire,
- Nécessite peu de ressources processeurs et de mémoires.

3.3 Les composants du protocole

3.3.1 Client

En parlant d'un client, cela signifie presque toujours un client MQTT. Cela inclut le publisher ou les subscribers, les deux étiqueter un client MQTT qui ne font que publier ou s'abonner (En général, un client MQTT peut être à la fois publisher et subscriber en même temps). Un client MQTT est un périphérique à partir d'un micro-contrôleur jusqu'à un serveur à part entière, qui a une bibliothèque MQTT en cours d'exécution et se connecte à un broker MQTT sur tout type de réseau. Il peut s'agir d'un périphérique réellement limité et restreint en ressources, connecté à un réseau sans fil et doté d'une bibliothèque au minimum ou d'un ordinateur typique exécutant un client MQTT graphique à des fins de test, essentiellement tout périphérique possédant une pile TCP / IP et parle MQTT au-dessus. La mise en œuvre client du protocole MQTT est très simple et vraiment réduit à l'essence. C'est un aspect, pourquoi MQTT est idéalement adapté pour les petits appareils.

3.3.2 Broker

Il s'agit du composant central du middleware, dont le rôle est de router les messages entre les applications et les clients. La contrepartie d'un client MQTT est le broker MQTT. En fonction de la mise en œuvre concrète, un broker peut gérer jusqu'à des milliers de clients simultanément connectés MQTT. Le broker est principalement responsable de recevoir tous les messages, de les filtrer, de décider qui s'y intéresse et d'envoyer le message à tous les clients subscribers. Il contient également la session de tous les clients persistants, y compris les abonnements et les messages manqués. Une autre responsabilité du broker est l'authentification et l'autorisation des clients. Et la plupart du temps, un broker est également extensible, ce qui permet d'intégrer facilement l'authentification, l'autorisation et l'intégration personnalisées dans les systèmes back-end. Particulièrement l'intégration est un aspect important, parce que souvent le broker est le composant, qui est directement exposé sur l'Internet et gère beaucoup de clients et passe ensuite des messages le long de l'analyse en aval et des systèmes de traitement. Dans l'ensemble, le broker est le hub central et

que chaque message doit passer. Par conséquent, il est important, qu'il est hautement évolutif, intégrable dans les systèmes back-end, facile à surveiller et bien sûr résistant aux défaillances.
Voici des differents brokers MQTT :

- ActiveMQ qui permet d'ajouter MQTT à un serveur Web Apache (Développé par la fondation Apache)
- JoramMQ pour l'intégration de MQTT en Java
- Mosquitto, le broker open-source le plus utilisé dans les projets DIY soutenu par la fondation eclipse.org
- RabbitMQ, un projet open source disponible également avec un support commercial
- EMQTT, un projet développé en Erlang/OTP disponible pour Windows, Mac Os X et Linux conçu pour recevoir de très nombreuses connexions (jusqu'à 1 million par serveur). Il est possible de créer un cluster (réseau de serveur) pour accroitre le nombres de connexions simultanées.
- ActiveMQ et JoramMQ sont des brokers assez spécifiques. RabbitMQ est plus orienté entreprise avec son offre commerciale.

3.3.3 Topic

Un sujet est une chaîne UTF-8, utilisée par le broker pour filtrer les messages de chaque client connecté. Un sujet se compose d'un ou plusieurs niveaux thématiques. Chaque niveau de sujet est séparé par une barre oblique (séparateur de niveau de sujet).

Figure 3.02 : *Exemple de topic MQTT*

Par rapport à une file d'attente de messages, un sujet est très léger. Il n'est pas nécessaire pour un client de créer le sujet souhaité avant de le publier ou de s'y abonner, car un broker accepte chaque sujet valide sans initialisation préalable.

3.3.4 Connexion MQTT

Le protocole MQTT est basé sur TCP / IP et le client et le broker doivent disposer d'une pile TCP / IP.

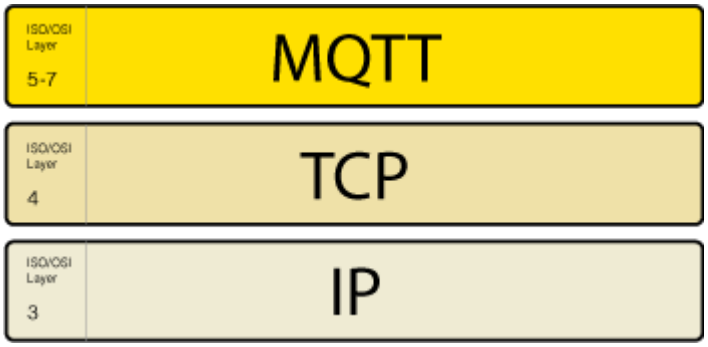

Figure 3.03 : *Le protocole MQTT dans le modèle OSI*

La connexion MQTT elle-même est toujours entre un client et le broker, aucun client n'est connecté directement à un autre client. La connexion est initiée par un client envoyant un message CONNECT au broker. La réponse du broker avec un CONNACK et un code d'état. Une fois la connexion établie, le broker le gardera ouvert tant que le client n'enverra pas une commande de déconnexion ou qu'il perdra la connexion.

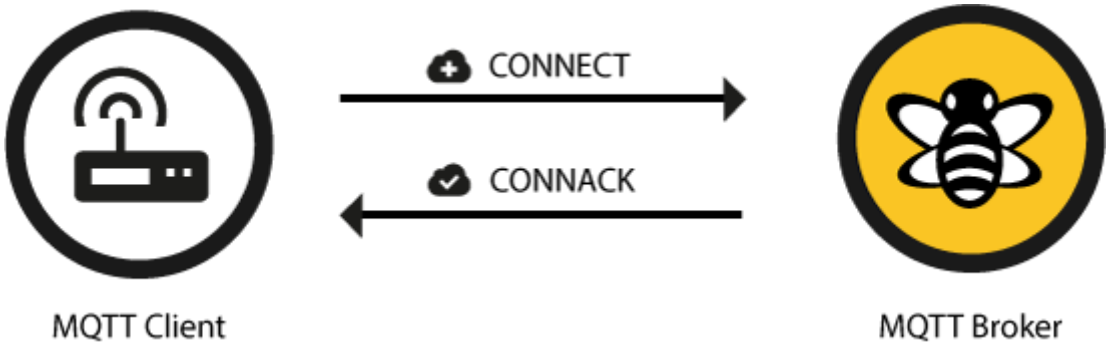

Figure 3.04 : *Connexion MQTT*

3.4 Fonctionnement de MQTT

MQTT est un service de publication/abonnement TCP/IP simple et extrêmement léger. Il fonctionne sur le principe client/serveur. [15]

Le serveur, nommé *broker*, va collecter des informations que les *publishers* (les Objet communicants) vont lui transmettre. Certaines informations collectées par le *broker* seront renvoyées à certains *publishers* ayant préalablement fait la demande au broker.

Le principe d'échange est très proche de celui de Twitter. Les messages sont envoyés par les *publishers* sur un canal appelé *topic*. Ces messages peuvent être lus par les *subscribers* (abonnés). Les *topics* (ou canaux d'informations) peuvent avoir une hiérarchie qui permet de sélectionner finement les informations que l'on désire.

Les messages envoyés par les objets communications peuvent être de toutes sortes mais ne peuvent excéder une taille de 256 Mo.

Exemple

Le *topic* '/home/salon/temperature' communiquera les températures du salon si un objet connecté s'y abonne (la sonde de température présente dans le salon publiera régulièrement la température relevée sur ce *topic*).

Si un *publisher* s'abonne au *topic* '/home/salon/#' il recevra toutes les données du salon (on peut imaginer : luminosité, humidité, température…).

S'il s'abonne au *topic* '/home/#', il collectera toutes les données des sondes de la maison.

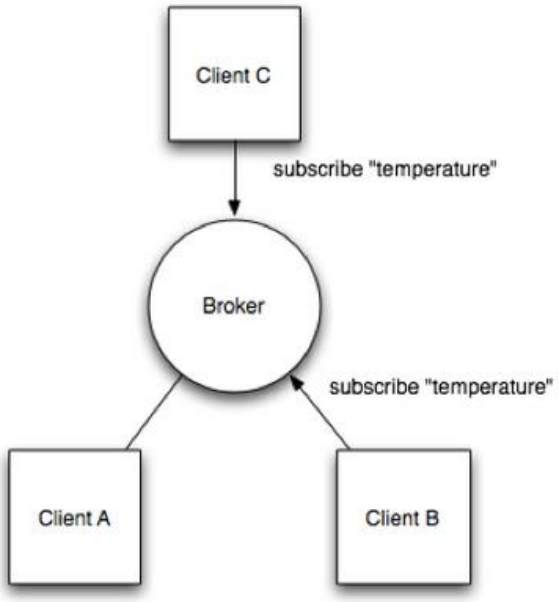

Figure 3.05 : *Architecture autour du protocole MQTT*

Les trois clients établissent une connexion TCP avec le broker. Les clients B et C souscrivent au topic temperature.

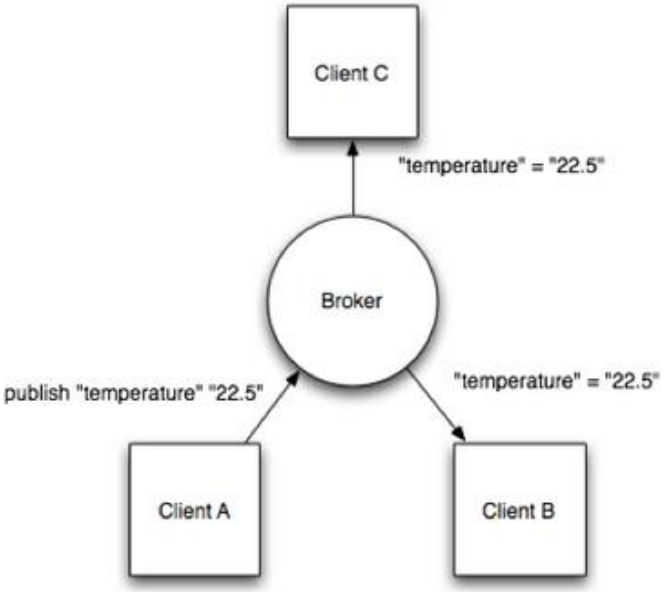

Figure 3.06 : *Exemple concret d'un flux MQTT*

Le Client A publie sur le topic temperature une valeur de 22,5°. Le broker propage le message à tous les clients ayant préalablement souscrit au topic Temperature.

Une session MQTT est divisée en quatre étapes : connexion, authentification, communication et terminaison.

3.4.1 Connexion

Un client commence par créer une connexion TCP/IP vers le broker en utilisant soit un port standard, soit un port personnalisé défini par les opérateurs du broker. Lors de la connexion, le serveur peut continuer une ancienne session s'il reconnaît une identité client précédemment utilisée.

3.4.2 Authentification

Les ports standards sont les suivants : 1883 pour la communication non chiffrée et 8883 pour la communication cryptée utilisant SSL/TLS (Transport Layer Security / Secure Sockets Layer). Pendant l'établissement de liaison (ou handshake) SSL/TLS initial, le client valide le certificat du serveur afin d'authentifier le serveur. Lors de cet échange, le client peut également fournir au broker un certificat client que le broker pourra ensuite utiliser pour authentifier le client. Bien que cela ne soit pas spécifié dans la norme MQTT, les brokers prennent habituellement en charge l'authentification des clients avec leurs certificats SSL/TLS.

Le protocole MQTT étant avant tout destiné aux appareils disposant de ressources limitées, SSL/TLS n'est pas toujours disponible et dans certains cas, il n'est pas souhaité. Le client s'authentifie alors en envoyant un nom d'utilisateur et un mot de passe en clair au serveur lors de la séquence de paquets CONNECT/CONNACK.

3.4.3 Communication

Au cours de la phase de communication, un client peut effectuer les opérations suivantes : publication, abonnement, désabonnement ou ping.
Les abonnements à des sujets sont réalisés au moyen d'une paire de paquets SUBSCRIBE/SUBACK. Le désabonnement est réalisé de manière similaire à l'aide d'une paire de paquets UNSUBSCRIBE/UNSUBACK.
Les chaînes décrivant un sujet forment une arborescence en utilisant la barre oblique (/) comme caractère de séparation. Un client peut s'abonner à des branches entières de l'arborescence d'un sujet (ou se désabonner) à l'aide des deux caractères génériques suivants : le signe plus (+), qui correspond à un seul niveau, et le dièse (#) pour plusieurs niveaux. Un caractère spécial, le dollar ($), permet d'exclure un sujet de tout abonnement utilisant un caractère générique à la racine. En règle générale, le dollar sert à transporter des messages système ou serveur spécifiques.

Keep Alive

Jetons un coup d'oeil sur les messages de keep alive en détail. Il y a deux messages impliqués dans la fonctionnalité de maintien en vie : PINGREQ/PINGRESP.
Un client peut effectuer une quatrième opération au cours de la phase de communication : il peut envoyer une commande ping vers le serveur du broker en utilisant une séquence de paquets PINGREQ/PINGRESP. Grossièrement traduit, cela signifie « ES-TU VIVANT/OUI JE SUIS VIVANT ». Cette opération n'a pas d'autre fonction que de maintenir une connexion active et de s'assurer que la connexion TCP n'a pas été coupée par une passerelle ou un routeur.

a. PINGREQ

Le PINGREQ est envoyé par le client et indique au courtier que le client est encore en vie, même s'il n'a pas envoyé d'autres paquets (PUBLIER, S'ABONNER, etc.). Le client peut envoyer un PINGREQ à tout moment pour s'assurer que la connexion réseau est toujours en vie. Le paquet PINGREQ n'a pas de charge utile.

b. PINGRESP

Lors de la réception d'un PINGREQ, le BROKER doit répondre avec un paquet PINGRESP pour indiquer sa disponibilité au client. Comme pour le PINGREQ, le paquet ne contient aucune charge utile.

3.4.4 Terminaison

Lorsqu'un éditeur ou un abonné souhaite mettre fin à une session MQTT, il envoie un message DISCONNECT au broker, puis met fin à la connexion. On parle d'arrêt progressif, car il permet au client de se reconnecter facilement en fournissant son identité client et de reprendre la session là où il l'avait laissée.

En cas de déconnexion brutale ne laissant pas le temps à l'éditeur d'envoyer un message DISCONNECT, le broker peut envoyer aux abonnés un message de l'éditeur que le broker avait précédemment mis en cache. Le message, appelé *dernières volontés et testament*, indique aux abonnés les mesures à prendre en cas d'arrêt inopiné de l'éditeur.

3.5 Format du paquet de contrôle MQTT

3.5.1 Structure d'un paquet de contrôle MQTT

Le protocole MQTT fonctionne en échangeant une série de paquets de contrôle MQTT d'une manière définie. Cette section décrit le format de ces paquets.

Un paquet de contrôle MQTT se compose de trois parties au maximum, toujours dans l'ordre suivant, comme illustré dans la figure 3.07.

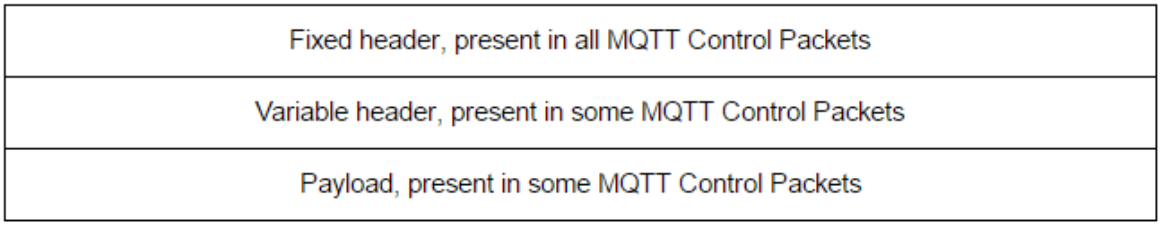

Figure 3.07 : *Structure d'un paquet de contrôle MQTT.*

3.5.2 En-tête fixe

Chaque paquet de contrôle MQTT contient un en-tête fixe. La figure 3.08 illustre le format d'en-tête fixe.

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2…	Remaining Length							

Figure 3.08 : *Le format d'en-tête fixe*

Voyons maintenant un a un ces composants.

3.5.2.1 MQTT Control Packet type

Représentées comme une valeur non signée à 4 bits, les valeurs sont répertoriées dans le tableau 3.01.

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment
PUBREC	5	Client to Server or Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment

PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Tableau 3.01: *Types de paquets de contrôle.*

3.5.2.2 Flags

Les bits restants [3-0] de l'octet 1 de l'en-tête fixe contiennent des indicateurs spécifiques à chaque type de paquet de contrôle MQTT tel qu'indiqué dans le tableau 3.02 ci-dessous. Lorsqu'un bit de drapeau est marqué comme «Reserved» dans le tableau 3.02, il est réservé pour une utilisation future et doit être réglé à la valeur indiquée dans ce tableau. Si des drapeaux non valides sont reçus, le récepteur doit fermer la connexion réseau.

Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP	QoS	QoS	RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0
UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0

Tableau 3.02: *Flag Bits*

a. DUP

Position: byte 1, bit 3.

Si l'indicateur DUP est mis à 0, cela signifie que c'est la première fois que le client ou le serveur a tenté d'envoyer ce paquet MQTT PUBLISH. Si l'indicateur DUP est défini sur 1, cela signifie qu'il peut s'agir d'une nouvelle tentative d'envoi du paquet.

Le drapeau DUP doit être mis à 1 par le Client ou le Serveur quand il tente de remettre un paquet PUBLIER. Le drapeau DUP doit être mis à 0 pour tous les messages QoS 0.

b. QOS

Position: byte 1, bits 2-1.

Ce champ indique le niveau d'assurance pour la livraison d'un message d'application. Les niveaux de qualité de service sont énumérés dans le tableau 3.03, ci-dessous.

QoS value	Bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

Tableau 3.03: *Définitions QoS*

Un paquet PUBLIER ne doit pas avoir les deux bits QoS mis à 1. Si un serveur ou un client reçoit un paquet PUBLIER qui a à la fois QoS bits mis à 1 il doit fermer la connexion réseau.

c. Retain

Si le RETAIN est mis à 1, dans un paquet PUBLISH envoyé par un client à un serveur, le serveur doit stocker le message d'application et sa QoS, afin qu'il puisse être livré aux futurs abonnés dont les abonnements correspondent à son nom de rubrique ou topic. Lorsqu'un nouvel abonnement est établi, le dernier message conservé sur chaque nom de topic correspondant doit être envoyé à l'abonné. Si le serveur reçoit un message QoS 0 avec le RETAIN Flag positionné sur 1, il doit éliminer tout message précédemment retenu pour cette rubrique. Il devrait stocker le nouveau message QoS 0 comme nouveau message conservé pour cette rubrique, mais peut choisir de le jeter à tout moment - si cela se produit, il n'y aura pas de message conservé pour cette rubrique.

3.5.2.3 Remaining Length

Position: commence par le byte 2.

La longueur restante ou remaining Length est codée à l'aide d'un schéma de codage de longueur variable qui utilise un seul octet pour des valeurs allant jusqu'à 127. Les valeurs plus importantes sont traitées comme suit. Les sept bits les moins significatifs ou LSB de chaque octet codent les données, et le bit le plus significatif est utilisé pour indiquer qu'il y a des octets suivants dans la représentation. Ainsi, chaque octet code 128 valeurs et un "bit de continuation". Le nombre maximal d'octets dans le champ Longueur restante est quatre.

3.5.3 En-tête variable

Certains types de paquets de contrôle MQTT contiennent un composant d'en-tête variable. Il se trouve entre l'en-tête fixe et la charge utile ou payload. Le contenu de l'en-tête variable varie selon le type de paquet. Le champ Identificateur de paquet ou « Packet Identifier » de l'en-tête variable est commun dans plusieurs types de paquets.

Packet Identifier

L'élément d'en-tête variable pour plusieurs types de paquets de contrôle inclut un champ Identificateur de paquet de 2 octets. Ces paquets de contrôle sont publiés (où QoS> 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

Les paquets de contrôle qui nécessitent un identificateur de paquets sont répertoriés dans le tableau 3.04.

Control Packet	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)
PUBACK	YES
PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES
SUBACK	YES

UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO

Tableau 3.04: *Paquets de contrôle qui contiennent un identificateur de paquet.*

3.5.4 Payload

Le tableau 3.05 listent les paquets de contrôle qui nécessitent une charge utile ou payload.

Control Packet	**Payload**
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	None
PINGREQ	None
PINGRESP	None
DISCONNECT	None

Tableau 3.05: *Les paquets de contrôle qui contiennent une charge utile*

3.6 QoS

Les trois niveaux de qualité de service déterminent la façon dont le protocole MQTT gère le contenu. Bien que les niveaux plus élevés soient plus fiables, ils sont également plus gourmands en termes de latence et de bande passante. Les clients abonnés peuvent spécifier le niveau de QoS maximal qu'ils souhaitent recevoir.

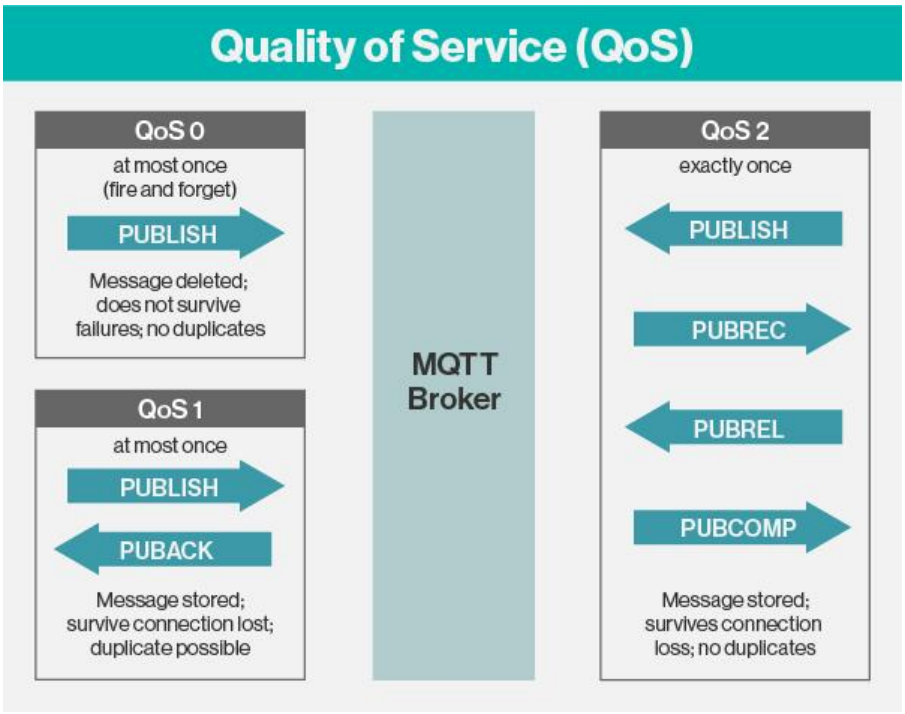

Figure 3.09 : *Qualité de service MQTT*

3.6.1 QoS 0

Le niveau de qualité de service le plus simple est le service non confirmé (Unacknowledged Service). Ce niveau utilise une séquence de paquets PUBLISH : l'éditeur envoie un message une seule fois au broker et ce dernier transmet ce message une seule fois aux abonnés. Aucun mécanisme ne garantit la réception du message et le broker ne l'enregistre pas non plus. Ce niveau de qualité de service est également appelé « QoS niveau 0 » ou QoS0, ou encore « At most once » (au plus une fois). Cette qualité de service est par exemple suffisante pour la communication des données d'un détecteur d'ambiance. La perte d'un relevé individuel est sans importance s'il est rapidement suivi d'un autre.

3.6.2 QoS 1

Le deuxième niveau de service est le service confirmé (Acknowledged Service). Ce niveau utilise une séquence de paquets PUBLISH/PUBACK (Publish Acknowledge) entre l'éditeur et son broker, ainsi qu'entre le broker et les abonnés.

Un paquet de confirmation vérifie que le contenu a été reçu et un mécanisme de renvoi du contenu d'origine est déclenché si l'accusé de réception n'est pas reçu en temps voulu. Cela signifie que l'abonné peut recevoir plusieurs copies du même message. Ce niveau de qualité de service est

également appelé « QoS niveau 1 » ou QoS1, ou encore « At least once » (au moins une fois). Les messages arrivent de façon certaine, mais ils peuvent arriver en double.

3.6.3 QoS 2

Le troisième niveau de QoS est le service garanti (Assured Service). Ce niveau délivre le message avec deux paires de paquets. La première est appelée PUBLISH/PUBREC et la seconde, PUBREL/PUBCOMP. Les deux paires s'assurent que quel que soit le nombre de tentatives, le message ne sera délivré qu'une seule fois. Ce niveau de qualité de service est également appelé « QoS niveau 2 » ou QoS2, ou encore « Exactly once » (exactement une fois). Les messages arrivent de façon certaine, une seule fois. Cette qualité de service est par exemple nécessaire aux systèmes de facturation. Les doublons ou les messages perdus peuvent causer des problèmes et générer des demandes de paiement erronées.

3.7 Last will and testament

Le MQTT est souvent utilisé dans des scénarios où les réseaux non fiables sont très fréquents. Par conséquent, il est supposé que certains clients se déconnecteront de temps en temps, car ils ont perdu la connexion, la batterie est vide ou tout autre cas imaginable. Par conséquent, il serait bon de savoir, si un client connecté a déconnecté gracieusement (ce qui signifie avec un message MQTT DISCONNECT) ou non, afin de prendre des mesures appropriées. La fonction dernière volonté et testament permet aux clients de faire exactement cela.
La fonctionnalité Last Will and Testament (LWT) est utilisée dans MQTT pour informer les autres clients d'un client déconnecté indûment. Chaque client peut spécifier son dernier message (un message MQTT normal avec un sujet, un retained flag, une QoS et un payload) lors de la connexion à un broker. Le broker stockera le message jusqu'à ce qu'il détecte que le client a déconnecté gracieusement. Si le client se déconnecte brusquement, le broker envoie le message à tous les clients abonnés sur la rubrique, qui a été spécifié dans le dernier message. Le message LWT mémorisé sera ignoré si un client se déconnecte gracieusement en envoyant un message DISCONNECT.
Selon la spécification MQTT 3.1.1, le broker distribuera le LWT (Last Will Testament) d'un client dans les cas suivants:

- Une erreur d'E / S ou de réseau est détectée par le serveur.
- Le client ne parvient pas à communiquer à l'intérieur de la durée Keep Alive.
- Le client ferme la connexion réseau sans envoyer un paquet DISCONNECT en premier.

- Le serveur ferme la connexion réseau en raison d'une erreur de protocole.

3.8 Wildcard

Lorsqu'un client s'abonne à une rubrique, il peut utiliser la rubrique exacte pour laquelle le message a été publié ou peut s'abonner à d'autres rubriques à la fois à l'aide de caractères génériques. Un caractère générique peut uniquement être utilisé lors de l'abonnement à des sujets et n'est pas autorisé lors de la publication d'un message. Dans la suite, nous allons examiner les deux types différents un par un: joker de niveau unique et multi niveau.

3.8.1 Niveau unique: +

Comme le nom l'indique déjà, un joker de niveau unique est un substitut pour un niveau de sujet. Le symbole + représente un joker de niveau unique dans la rubrique.

Figure 3.10 : *Topic avec « single level wildcard »*

Tout sujet correspond à une rubrique incluant le caractère générique à un seul niveau s'il contient une chaîne arbitraire au lieu du caractère générique. Par exemple, un abonnement à myhome / groundfloor / + / temperature correspond ou ne correspond pas aux sujets suivants:

Figure 3.11 : *Schéma représentant le résultat du topic précèdent*

3.8.2 Niveau multi: #

Alors que le wildcard niveau unique ne couvre qu'un seul niveau de sujet, le joker de niveau multi couvre un nombre arbitraire de niveaux de sujet. Pour déterminer les thèmes correspondants, il est

nécessaire que le caractère générique à niveaux multiples soit toujours le dernier caractère de la rubrique et qu'il soit précédé d'une barre oblique.

Figure 3.12 : *Topic avec « multi-level wildcard avec le resultat»*

Un client subscriber à une rubrique avec un caractère générique à plusieurs niveaux reçoit tous les messages, qui commencent par le motif avant le caractère générique, peu importe la longueur ou la profondeur des rubriques. Si vous spécifiez uniquement le caractère générique multiniveau en tant que sujet (#), cela signifie que vous recevrez tous les messages envoyés sur le broker MQTT. Si vous prévoyez un débit élevé, il s'agit d'un anti-modèle, consultez les meilleures pratiques ci-dessous.

3.9 Type de message MQTT

Le tableau 3.06 suivant nous montre la liste des messages MQTT avec leur description respective.

Message MQTT	Description
CONNECT	Requête d'un client qui veut se connecter au serveur
CONNACK	Accusé de la connexion
PUBLISH	Publication d'un message
PUBACK	Accusé de la publication
PUBREC	Publication reçue
PUBEREL	Libération du canal
PUBCOMP	Publication terminée
SUBSCRIBE	Requête d'un client qui veut souscrire à un topic
SUBACK	Accusé de la souscription
UNSUBSRIBE	Requête de désabonnement
UNSUBACK	Accusé du désabonnement
PINGREQ	Requête PING
PINGRESP	La réponse de la requête PING
DISCONNECT	Déconnexion d'un client

Tableau 3.06: *Tableau illustrant les différentes types de messages MQTT*

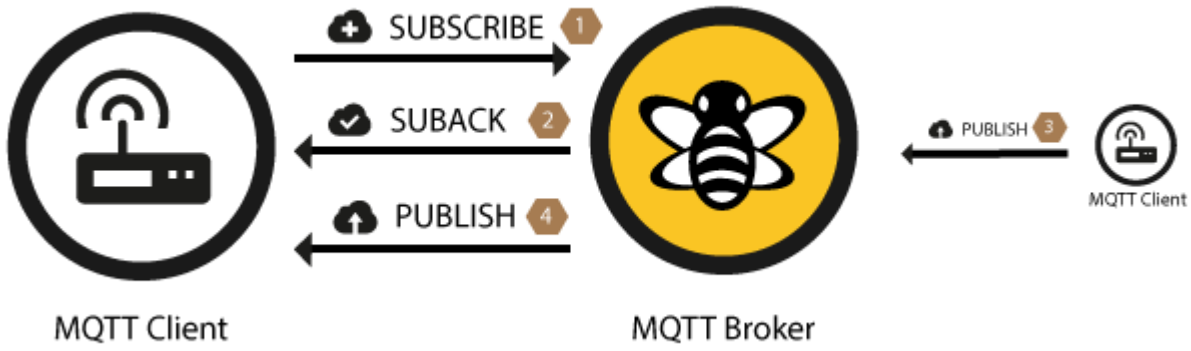

Figure 3.13 : *Exemple de flux de message MQTT*

Après qu'un client a correctement envoyé le message SUBSCRIBE et a reçu le message SUBACK, il recevra chaque message publié correspondant à la rubrique ou topic de l'abonnement.

3.10 Défis liés à l'utilisation de MQTT pour l'Internet des objets

Dans la mesure où MQTT n'intègre pas de mécanismes de sécurité, ce protocole est traditionnellement utilisé dans des réseaux sécurisés à des fins applicatives spécifiques. La structure des sujets MQTT peut facilement devenir une immense arborescence. Or, il n'existe pas de méthode simple permettant de la diviser en domaines logiques plus petits qui pourraient être fédérés. Cela rend difficile la création d'un réseau MQTT évolutif à grande échelle, car dès lors que l'arborescence des sujets grandit, la complexité augmente.

MQTT présente un autre inconvénient : son manque d'interopérabilité. Etant donné que la charge utile des messages est binaire, sans aucune information quant à leur codage, des problèmes peuvent survenir, notamment dans des architectures ouvertes où des applications de différents fabricants sont censées fonctionner de manière transparente les unes avec les autres.

Comme nous l'avons vu plus haut, le protocole MQTT intègre des fonctionnalités d'authentification minimales. Nom d'utilisateur et mot de passe sont envoyés en clair et toute utilisation sécurisée de MQTT exige de mettre en œuvre SSL/TLS, qui malheureusement n'est pas un protocole léger.

Authentifier des clients avec des certificats client n'est pas un processus des plus simples et MQTT ne permet pas, hormis par des moyens propriétaires hors bande, de déterminer qui est propriétaire d'un sujet et qui peut y publier des informations. Il est donc très facile d'injecter des messages malveillants, volontairement ou non, dans le réseau. En outre, il n'y a aucun moyen pour le récepteur

de savoir qui a envoyé le message d'origine, à moins que cette information ne soit contenue dans le message.

La couche de sécurité propriétaire qui doit être ajoutée à MQTT augmente donc l'empreinte logicielle et complique la mise en œuvre du protocole.

En dépit de ces considérations, de nombreux experts estiment que MQTT jouera un rôle important dans l'Internet des objets, en facilitant des opérations telles que le suivi des stocks, la télématique automobile, la surveillance des ressources et les réseaux du corps médical.

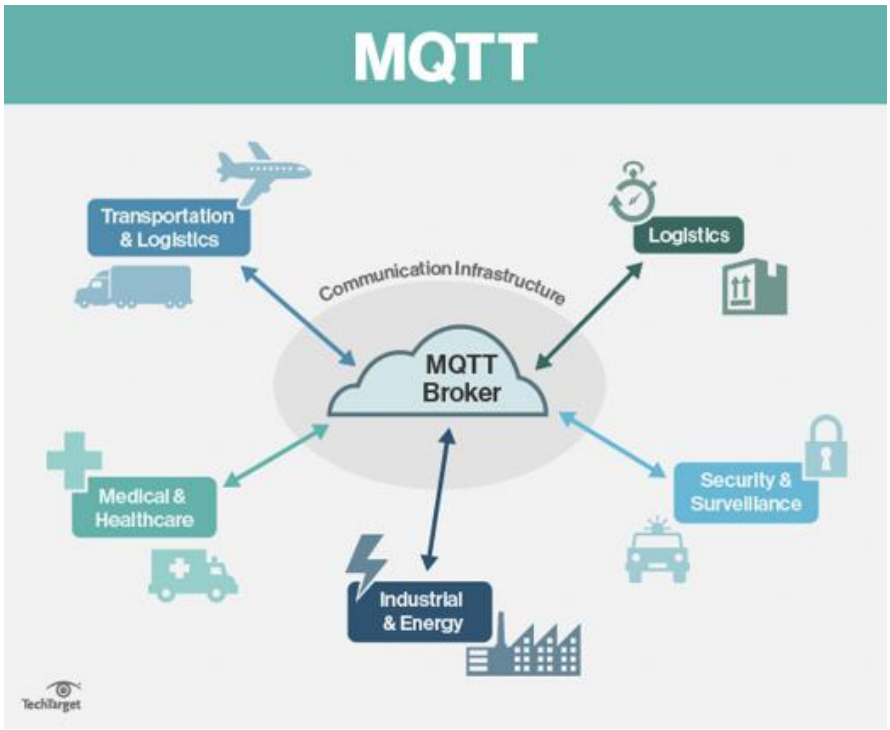

Figure 3.14 : *Schéma représentant un réseau d'objet avec le protocole MQTT*

3.11 Conclusion

Nous avons pu voir dans cette chapitre le protocole MQTT qui s'appuie surtout sur le modèle publish/subsribe ou encore publication/souscription. Ce protocole de communication joue un grand rôle dans un réseau IoT grâce à ses différents avantages dont le principal est le fait d'être léger ainsi que sa faible consommation. Le protocole s'améliore en permanence et prend désormais en charge WebSockets, un autre protocole qui permet une communication bidirectionnelle en temps réel entre les clients et les brokers.

CHAPITRE 4
APPLICATION DANS LE DOMAINE DE LA DOMOTIQUE

4.1 Introduction

Les objets connectés ont pour vocation première d'accompagner les hommes dans leur vie quotidienne ainsi que dans la facilitation des tâches qu'ils doivent accomplir. Grâce à la domotique, ensemble des technologies de l'électronique de l'information et des télécommunications utilisées dans les domiciles, il est désormais possible d'équiper l'intégralité de sa maison d'objets connectés centralisés et automatisés pour la rendre plus ergonomique. Par exemple, des systèmes qui permettent de gérer les équipements de la maison à distance, grâce à une application web. La "maison connectée" est l'innovation grand public du moment. Cette réalisation permet de relier tous les objets de la maison à un serveur. Ainsi, ces objets intéressent de plus en plus le monde puisqu'ils permettent de faciliter la vie du foyer au quotidien.

4.2 Présentation du projet

La réalisation consiste à mettre en place le protocole de communication MQTT dans un réseau d'objets de la domotique.

L'idée principale de ce projet est de contrôler et de surveiller les différentes solutions domotiques qui sont implémentées avec une carte Arduino à partir d'une application Web fonctionnant au-dessus de Node.JS actionné par un Raspberry Pi. En réalité, le Pi peut être remplacé pour un autre ordinateur capable d'exécuter Node.JS, mais l'un des objectifs du projet est de le rendre le plus ordonné possible, et le Pi fournit également certaines fonctionnalités qui pourraient être utilisées à l'avenir pour étendre le projet. L'accent sera mis sur l'interaction entre les cartes Arduino et l'application Web via le protocole MQTT.

Objectifs du projet :

Les principaux objectifs de notre projet sont :

La création d'un circuit électronique qui représente les différents scenarios dans le monde réel :

- Commande lumière via web
- Les températures
- Le détecteur de mouvement
- La supervision du courant

La création d'une application web qui est responsive capable de controller la carte Arduino via le protocole MQTT.

La création d'un serveur web capable d'envoyer les informations depuis la carte arduino vers notre plateforme.

Stockage des informations vers la base de données mongoDB.

4.3 Les principaux HARDWARE

Fondamentalement, le projet comporte trois composantes principales, dont la plus importante est le Raspberry Pi. Tous les principaux programmes et applications sont en cours d'exécution sur cette dernière. Ensuite, connectés par USB, nous pouvons trouver la carte Arduino. L'appareil est alimenté et communique avec le Pi via le Shield Ethernet. Le Pi utilise une source d'alimentation de 5V fourni par son port mini-USB.

Nous connectons le Raspberry Pi à un routeur D-link utilisant son port Ethernet pour l'ajouter à notre LAN. Maintenant, nous pouvons y accéder par "SSH" pour gérer le projet (par exemple, build Arduino sketches, déboguer l'application, installer des dépendances, etc.). Après cela, nous pouvons ouvrir l'application Web par tout type de dispositif. Le routeur a des capacités WiFi, ainsi nous pouvons y accéder via des terminaux sans fil en utilisant un téléphone ou une tablette et il s'affichera normalement quelle que soit la taille de l'écran.

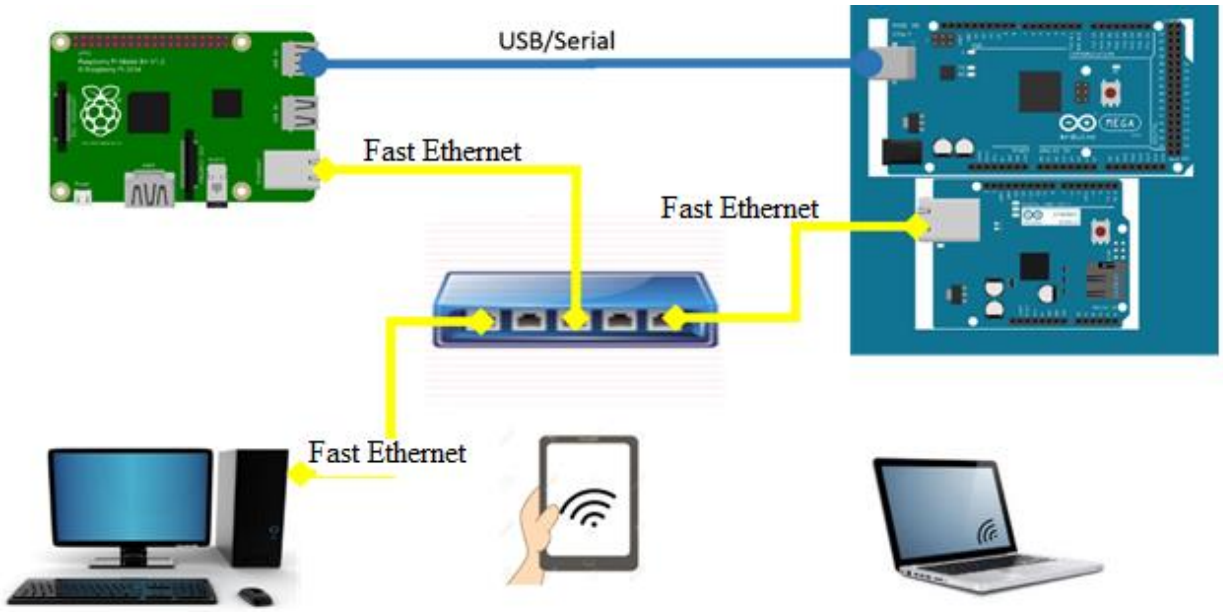

Figure 4.01 : *Architecture réseau de connexion entre le Raspberry Pi et la carte Arduino*

4.3.1.1 Le Raspberry Pi

Le Raspberry Pi est une nano-ordinateur monocarte à processeur ARM qui a la taille d'une carte de crédit. Il est destiné à encourager l'apprentissage de la programmation informatique; il permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.
Les Raspberry Pi requièrent au minimum pour fonctionner :

- Un support de stockage mémoire compatible (carte SD ou micro SD selon le modèle) contenant un système d'exploitation adapté qui est le « Raspbian ». La majorité des cartes mémoires du marché sont reconnues par l'appareil, une carte mémoire de classe 6 minimum est néanmoins recommandée.
- Une source d'alimentation électrique adaptée. Soit via un système de piles ou de batteries, soit via un transformateur électrique délivrant la bonne tension et l'intensité minimum préconisées. Le Raspberry Pi est compatible avec de nombreuses alimentations, micro-USB étant par exemple une norme pour les téléphones mobiles récents en Europe.

Le trio, écran, clavier et souris n'est pas indispensable pour faire fonctionner le Raspberry Pi. En effet, sur certains systèmes d'exploitation (Raspbian par exemple) le protocole SSH est activé et configuré par défaut. Ainsi, le Raspberry Pi peut être contrôlé à distance par le réseau dès le premier démarrage.
Le raspberry pi possède, en plus des connectiques classiques USB, HDMI, etc... un connecteur GPIO. GPIO signifie en anglais *General Purpose Input Output* et pourrait être traduit en francais par entrées/sorties numériques.
Ces entrées/sorties permettent d'étendre les fonctionnalités du raspberry pi en lui donnant la possibilité d'agir sur des leds ou des afficheurs LCD par exemple, lire l'état d'un interrupteur, d'un capteur, etc...
Ce connecteur GPIO dispose de différents types de connexion :

- des broches utilisables en entrée ou sortie numérique tout ou rien
- des broches pour une interface I2C (permettant de se connecter sur du matériel en utilisant uniquement 2 broches/pins de contrôle.)
- une interface SPI pour les périphériques SPI,
- les broches Rx et Tx pour la communication avec les périphériques séries.

- de broches pouvant être utilisé en PWM ("Pulse Width Modulation") permettant le contrôle de puissance ou PPM ("Pulse Position Modulation") permettant de contrôler des servo moteurs par exemple.

4.3.1.2 La carte Arduino

L'Arduino est une famille de cartes électroniques à micro-contrôleur open-source née en Italie en 2005. Ces cartes basées sur une interface entrée/sortie simple et sur un environnement de développement proche du langage C. Arduino peut être utilisé pour construire des objets interactifs indépendants (prototypage rapide), ou bien peut être connecté à un ordinateur pour communiquer et superviser en utilisant des logiciels de programmation (flash, labview, etc).
Il existe un nombre un nombre très impressionnant de cartes dites "shield" qui s'adaptent directement sur la carte Arduino et qui permettent d'ajouter rapidement des fonctions à notre projet.

La carte Arduino Mega 2560 est basée sur un ATMega2560 cadencé à 16 MHz. Elle dispose de 54 E/S dont 14 PWM, 16 analogiques et 4 UARTs. Elle est idéale pour des applications exigeant des caractéristiques plus complètes que la Uno. Des connecteurs situés sur les bords extérieurs du circuit imprimé permettent d'enficher une série de modules complémentaires.
Elle peut se programmer avec le logiciel Arduino.

4.4 Les circuits électriques

4.4.1.1 Le shield Ethernet

Le module Ethernet Arduino permet à une carte Arduino de se connecter à internet. Ce module est basé sur le circuit intégré Wiznet W5100. Le Wiznet W5100 fournit une pile réseau (IP) capable à la fois de TCP et UDP. Il supporte jusqu'à quatre connexions simultanées. Il suffit d'utiliser la librairie Ethernet pour écrire des programmes qui se connectent à internet en utilisant ce module. Le module ethernet se connecte à une carte Arduino grâce à ses longues broches qui dépassent du circuit imprimé. Ainsi le brochage de la carte Arduino n'est pas modifié et permet d'enficher un autre module par-dessus et laisse l'accès aux broches de la carte Arduino.

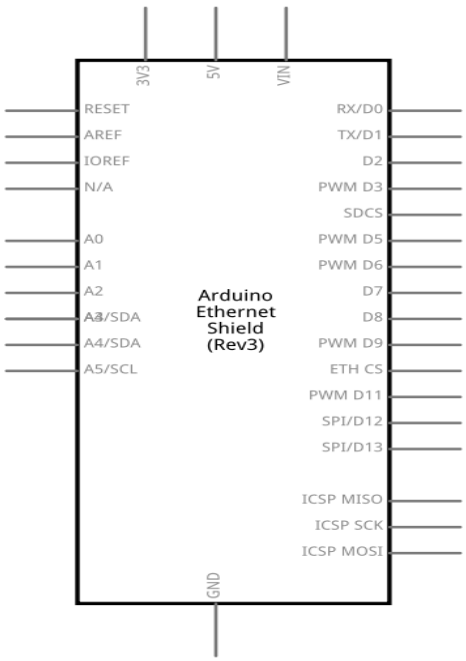

Figure 4.02 : *Schéma d'un module Ethernet arduino*

4.4.1.2 Le capteur de température

Le capteur de température LM35 est un capteur analogique de température fabriqué par Texas Instruments. Il est extrêmement populaire en électronique, car précis, peu couteux, très simple d'utilisation et d'une fiabilité à toute épreuve.

Le capteur de température LM35 est capable de mesurer des températures allant de -55°C à +150°C dans sa version la plus précise et avec le montage adéquat, de quoi mesurer n'importe quelle température. [17]

La sortie analogique du capteur est proportionnelle à la température. Il suffit de mesurer la tension en sortie du capteur pour en déduire la température. Chaque degré Celsius correspond à une tension de +10mV.

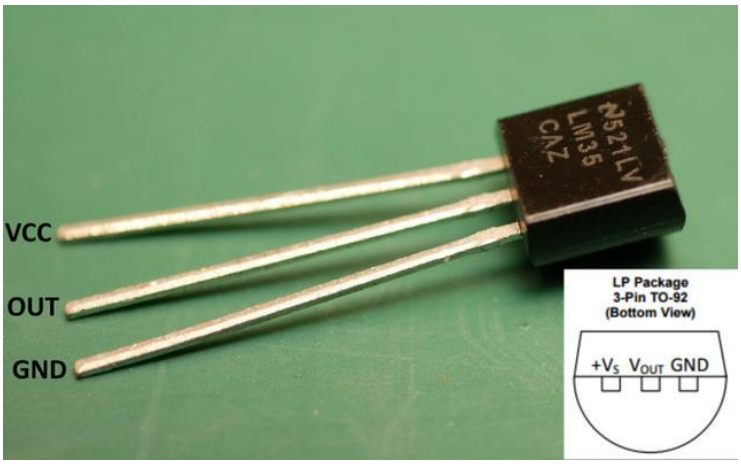

Figure 4.03 : *Brochage du capteur LM35*

4.4.1.3 Le capteur de luminosité

Ce module capteur de lumière à photo-résistance est prêt à l'emploi. Il est ajustable avec un potentiomètre de réglage, à dévisser pour réduire le seuil.
La résistance décroît avec la lumière ce qui augmente le courant qui circule. La relation n'est pas linéaire, on peut fixer des seuils avec Arduino ou faire un étalonnage.

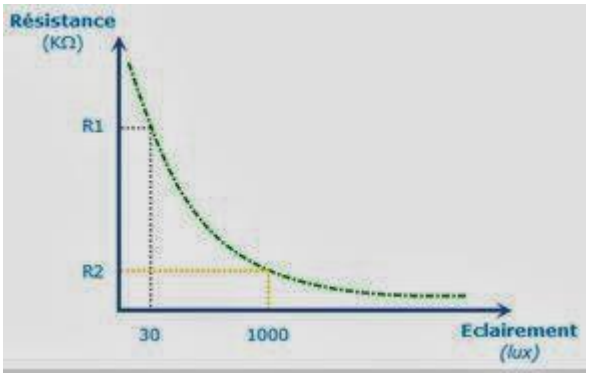

Figure 4.04 : *Courbe caractéristique du module*

4.4.1.4 Le détecteur de mouvement

Un capteur infrarouge permet de détecter un mouvement dans son champ de vision en se basant sur l'infrarouge. On parle aussi de capteur pyroélectrique ou PIR. Le PIR sont capable de détecter une variation des ondes infrarouges, ce qui génère un courant électrique. Dans le cas de notre capteur, il est en fait divisé en deux parties différentes reliées ensemble afin de détecter une variation lors

qu’une des moitiés capte plus qu’une autre. On a ainsi un relevé d’une différence, et non plus d’une valeur simple.

Lors d’un mouvement, la variation des deux moitiés va varier, et on va donc capter cette variation positive.

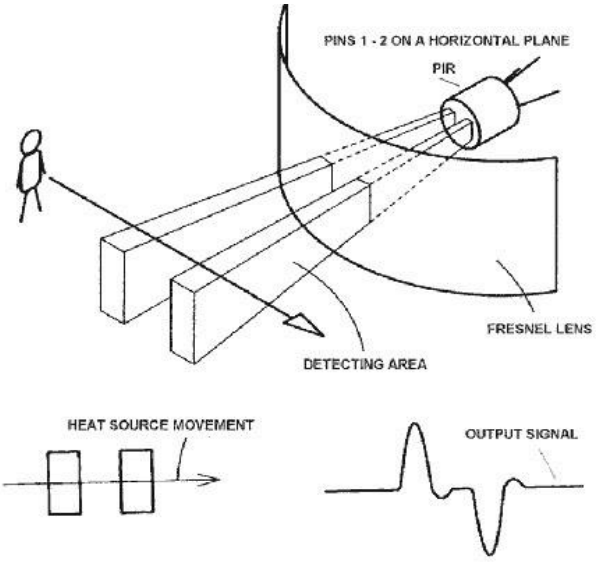

Figure 4.05 : *Schéma du principe de fonctionnement*

4.4.1.5 Les relais

La tension sur les pins I/O de l’Arduino est de 5V, un peu limite quand on sait que la tension de l’appareillage électrique de la maison est de 220V. Un relais est un composant qu’on pourrait comparer à une vanne : il peut soit laisser passer le courant, soit ne pas le laisser passer. Si nous appliquons un signal de 5V, celui-ci laissera passer le courant. Le courant qu’on contrôle peut avoir une tension de continue allant jusqu’à 30V ou une tension alternative de 250V max. En pratique, cela signifie qu’on peut contrôler du courant secteur.

4.4.1.6 Le capteur de courant ACS712

Basé sur l'ACS712 d'Allegro, ce senseur se branche en série avec la charge sur un circuit alternatif et permet de mesurer le courant qui traverse le senseur. Le senseur utiliser le champ magnétique généré par le courant (et donc l'effet hall) pour mesurer le courant qui le traverse. Le senseur produit en sortie une tension continue proportionnelle au courant à raison de 0.100V/A (100 mV par ampère). On peut donc lire cette tension sur une entrée analogique de l Arduino.

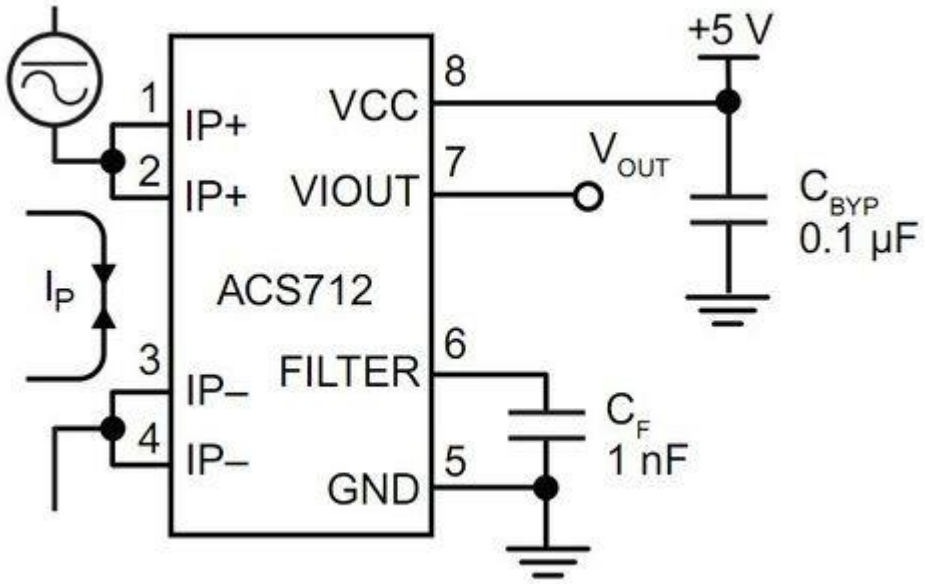

Figure 4.06 : *Schéma d'un ACS 712*

4.5 Softwares

4.5.1 Node,js et les fichiers Java Script

Presque tout le code de ce projet a été écrit sur JavaScript sauf pour les codes Arduino et les styles et la structure des applications Web (écrits sur CSS et HTML). Pour sa rapidité ainsi que pour tous les bénéfiques qu'elle offre, Node.js a été utilisées pour :

- Développer notre Serveur Web.
- L'administration des bases de données.
- La gestion des événements.
- Administrer des connexions MQTT
- Gérer la communication en temps réels avec les clients

4.5.1.1 MQTT

Le module MQTT qu'offre node.js permet la gestion et l'administration des connexions MQTT ainsi que l'échange des messages.

```
var mqtt = require('mqtt')
var client  = mqtt.connect('mqtt://test.mosquitto.org')

client.on('connect', function () {
  client.subscribe('presence')
  client.publish('presence', 'Hello mqtt')
})

client.on('message', function (topic, message) {
  // message is Buffer
  console.log(message.toString())
  client.end()
})
```

Figure 4.07 : *Exemple de code avec mqtt*

4.5.1.2 Mangoose

Mongoose est une librairie NodeJS qui permet l'utilisation de la base de données MongoDB. MongoDB est un système de gestion de base de données orientée documents. Son principal intérêt est qu'il utilise le format JSON, sans schéma prédéterminé, nous pouvons donc réécrire le document à volonté facilement.

4.5.1.3 Express.js

Le Framework « Express » fournit des outils pour la création d'applications web par exemple. Il permet de gérer facilement les routes, afin de faciliter la navigation sur le site. Il nous permet de coder en quelques lignes un site web efficace sans être trop compliqué.
Exemple de code Express:

```
var express = require('express');

var app = express();

app.get('/', function(req, res) {
    res.setHeader('Content-Type', 'text/plain');
    res.end('Vous êtes à l\'accueil');
});

app.listen(8080);
```

Figure 4.08 : *Exemple de code express*

Ci-dessus, un exemple très simple de l'utilisation du Framework « Express ». Il va créer une page web d'accueil, et l'application va écouter le port 8080. Il faut installer Express, en faisant un « npm install express » pour pouvoir l'utiliser de cette façon.

4.5.1.4 Passeport.js

La plupart des serveurs ont besoin d'authentification (déterminer l'identité de l'utilisateur), que ce soit via une bonne vieille saisie login + mot de passe, ou par des API comme s'authentifier directement via son compte facebook, twitter, google+ en en-têtes de requête http. Le module de référence (une collection de modules, en fait) pour ça, c'est Passport.
La mise en place peut paraître un peu obscure, mais ça permet, plutôt facilement, de mettre en place toute une gamme de stratégies d'authentification pour notre plateforme.

4.5.1.5 EJS

Pour notre application, nous avons opté pour l'utilisation du template EJS. Ce dernier ne nécessite pas l'apprentissage d'un nouveau langage car il peut être écrit en langage HTML.

4.5.2 Arduino Sketches

C'est l'IDE qui nous permet d'écrire des codes pour piloter la carte arduino ainsi que le shield ethernet. Le langage de programmation utilisé est proche du C++.

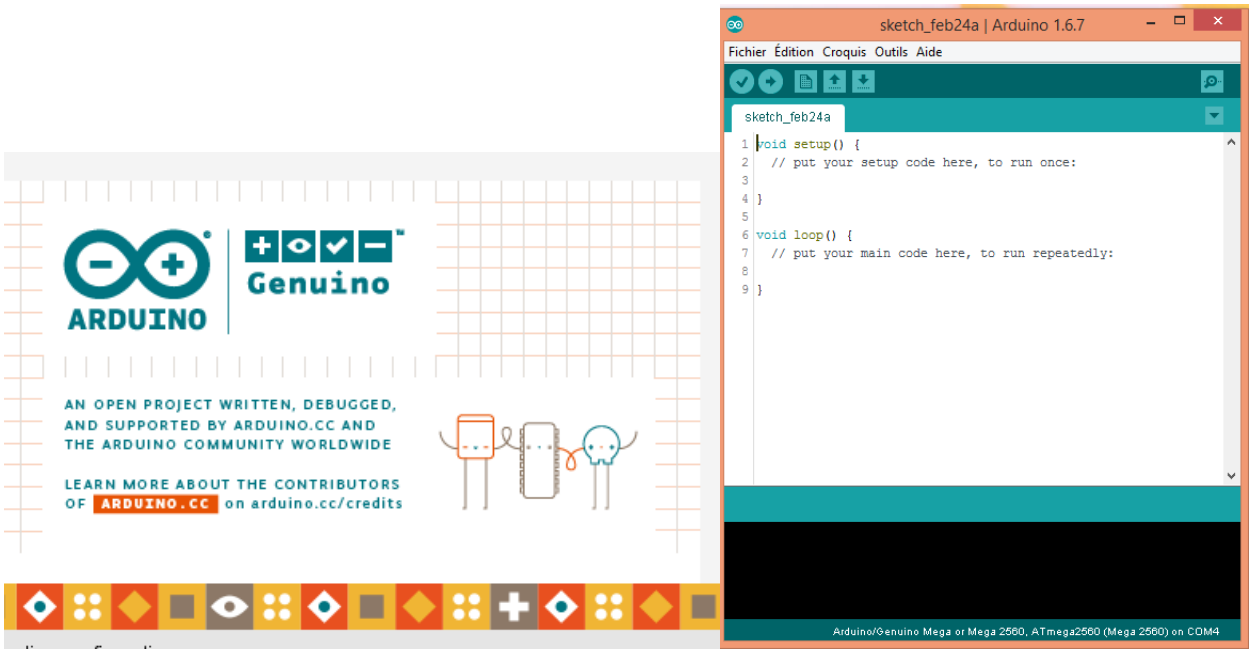

Figure 4.09 : *Interface de notre IDE*

Cet IDE nous a permis de rendre possible la connexion MQTT entre les capteurs et le broker MQTT grâce à une librairie spécialisée.

4.6 Socket.io

Socket.io est l'une des bibliothèques les plus prisées par ceux qui développent avec Node.js parce qu'elle permet de faire très simplement de la communication synchrone dans l'application, c'est-à-dire de la communication en temps réel.

Les possibilités qu'offre socket.io sont en réalité immenses et vont bien au-delà du Chat : tout ce qui nécessite une communication immédiate entre les visiteurs d'un site peut en bénéficier. Cela peut être par exemple une brique de base pour mettre en place un jeu où on voit en direct les personnages évoluer dans le navigateur ou encore dans le cas de notre projet le suivi en temps réel de la température ou encore de la consommation . Tout cela sans la nécessité de rafraichir la page.

C'est une bibliothèque qui nous simplifie beaucoup les choses. Socket.io se base sur plusieurs techniques différentes qui permettent la communication en temps réel. La plus connue d'entre elles, et la plus récente, est WebSocket.

C'est une nouveauté récente apparue plus ou moins en même temps que HTML5, mais ce n'est pas du HTML : c'est une API JavaScript.

Nous allons voir à présent comment émettre et recevoir des messages avec socket.io

4.6.1 Connexion d'un client

Quand on utilise socket.io, on doit toujours s'occuper de deux fichiers en même temps :

- Le fichier serveur (ex : app.js) : c'est lui qui centralise et gère les connexions des différents clients connectés au site.
- Le fichier client (ex : index.html) : c'est lui qui se connecte au serveur et qui affiche les résultats dans le navigateur.

4.6.2 Envoi et réception de messages

Quand le client est connecté, on peut échanger des messages entre le client et le serveur. Il y a 2 cas de figure :

- Le serveur veut envoyer un message au client

Lorsqu'on détecte une connexion, on émet un message au client avec socket.emit(). La fonction prend 2 paramètres :

- Le type de message qu'on veut transmettre.
- Le contenu du message. Là où on peut transmettre ce que l'on veut.

- Le client veut envoyer un message au serveur

4.6.3 Communication avec plusieurs clients

Dans la pratique, plusieurs clients sont connectés à l'application Node.js. Quand on a plusieurs clients, il faut être capable :

- D'envoyer des messages à tout le monde d'un seul coup. On appelle ça les broadcasts.
- De se souvenir d'informations sur chaque client (comme son pseudo par exemple). On a besoin pour ça de variables de session.

Prenons un cas :

1. Le client A envoie un message au serveur
2. Le serveur l'analyse
3. Il décide de broadcaster ce message pour l'envoyer aux autres clients connectés : B et C

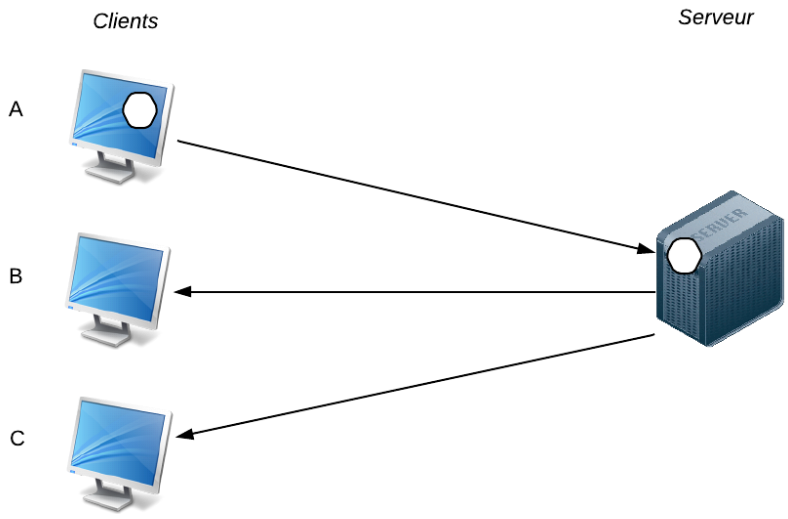

Figure 4.10 : *Broadcast : le serveur envoie un message à tous les autres clients connectés*

4.7 Détails techniques

4.7.1 Description autour du protocole MQTT

Les clients

La partie client s'agit des différents capteurs connectent à l'arduino ainsi que les terminaux notamment les ordinateurs, smartphones et tablettes

Le broker

Eclipse Mosquitto est un « broker » open source qui implémente les versions 3.1 et 3.1.1 du protocole MQTT.

Les topics

Chaque capteur et objets publient et souscrivent tous à un topic qui leur est tous propre.

PIECES	CAPTEURS OU OBJETS	TOPIC	PIN ARDUINO
CHAMBRE	état lampe	lum/chambre	
	température	temperature/chambre	A0
	ampoule	cmd/chambre	2
SALON	état lampe	lum/salon	
	température	temperature/salon	
	ventilateur	cmd/ventilo	5
	ampoule	cmd/salon	4
CUISINE	état lampe	lum/cuisine	3
	ampoule	cmd/cuisine	
CONSOMMATION	ACS 712	consommation/piece	A8
EXTERIEURE	capteur luminosité	ext/lumiere	A2
	détecteur de mouvement	mov/stat	6

Tableau 4.01: *Listes des topics dans le projet*

4.7.2 Fonctionnement MQTT au niveau arduino

Le logiciel Arduino doit prendre en charge quatre tâches principales pour cet exemple:

• Rassembler périodiquement les relevés des capteurs de lumière ainsi que la température.

• Publier des lectures de capteurs via MQTT

• Écouter les commandes via MQTT

Un client MQTT est créé dans la fonction « setup ».

Dans la fonction « loop() »:

- Le client MQTT se connecte s'il n'est pas encore connecté
- Il publie les données sur le topics bien définis
- Il se met en écoute s'il y a des messages MQTT destines pour lui.

4.7.3 Flux de données

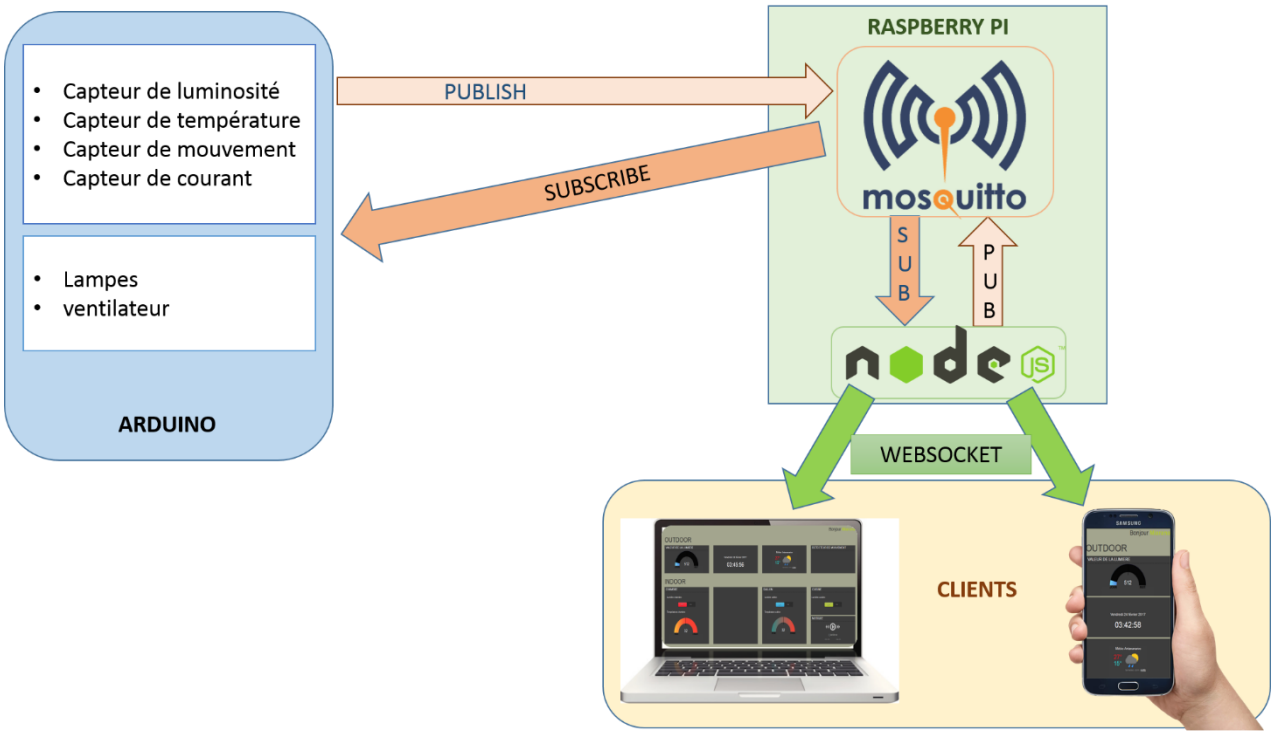

Figure 4.11 : *Flux de données échangé lors de la communication*

La figure 4.11 nous montre le flux de messages échanges lors de la réalisation.

Etape 1 : Les différents capteurs publient sur des « topic » propre à chacun les valeurs qu'ils obtiennent.

Etape 2 : en contrepartie il faut bien qu'il y a des clients qui souscrivent à ces « topics » afin de les traiter. Dans notre cas c'est le serveur Node.js qui gère la récupération et le traitement de messages provenant de ces « publisher ». En terme technique notre serveur web est un « subscriber ».

Etape 3 : Apres traitement des messages MQTT, le serveur web va afficher en temps réel les informations nécessaires sur la plateforme web.

Ces 3 étapes concernent la communication Arduino-Client. Voyons maintenant l'échange du cote inverse.

Etape 1' : Via l'interface web le client peut commander les lampes de différentes pièces. Ces commandes sont envoyées via le websocket vers le serveur web

Etape 2' : Le serveur récupère le message et publie ces messages sur des « topics » spécifiques vers les objets connectes a la carte arduino

Etape 3’ : ces objets souscrivent au « topics » provenant des subscriber et ils obtiennent ainsi les commandes et les exécutent.

Notons que tout échange de message MQTT transite sur le broker dans notre cas mosquitto. C’est ce serveur qui coordonne le flux de données MQTT, tout doit passer par lui.

4.7.4 Partie réseau de communication

Un serveur web est installé sur la Raspberry Pi. Nous avons également créé un point d’accès Wi-Fi auquel le client peut se connecter avec n’importe quel smartphone, tablette ou ordinateur. Ainsi, le client peut accéder directement au service fourni par la plateforme.

La carte arduino est connectée via un câble réseau grâce au shield ethernet au raspberry pi à l’aide d’un switch. En conséquence autre ordinateur qui ne possède pas de wifi peut se joindre à notre réseau via une connexion par câble RJ45.

4.8 Présentation plateforme

Partie authentification

Dans la partie authentification, seul l’utilisateur ayant un nom d’utilisateur et mot de passe valide peut accéder à l’interface principale. Pour pouvoir enregistrer ces utilisateurs, seul l’administrateur de la plateforme peut accéder à la page « signup » en se rendant à une adresse URL bien définie qui est normalement inconnu du grand publique.

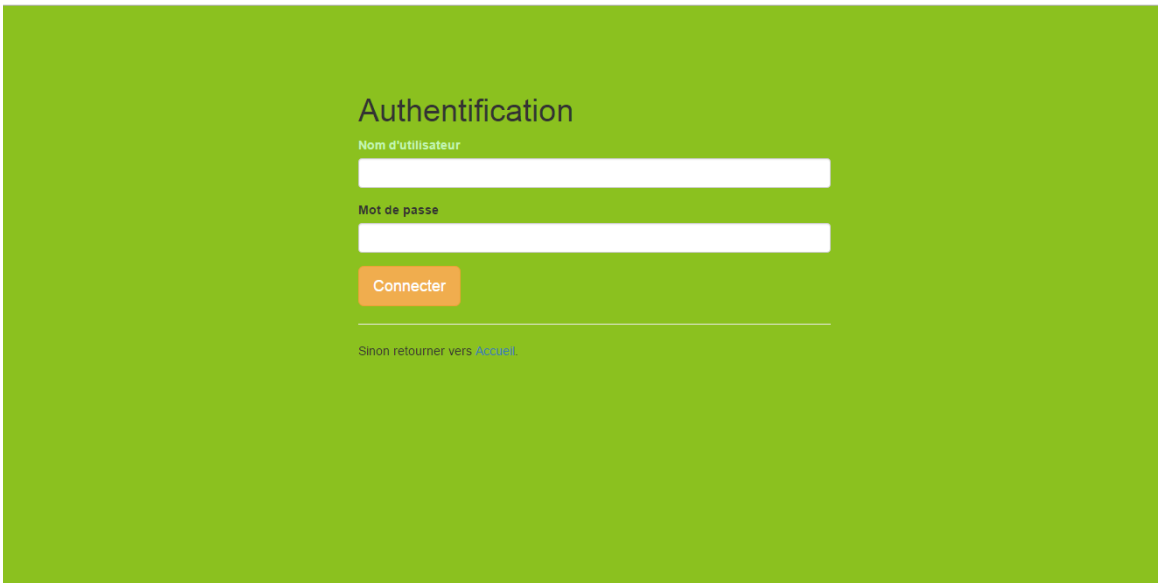

Figure 4.12 : *Page d’authentification*

Une fois authentifiée l'utilisateur sera redirigé vers l'interface principale de la plateforme illustree par les figures 4.13 et 4.14 ci-dessous.

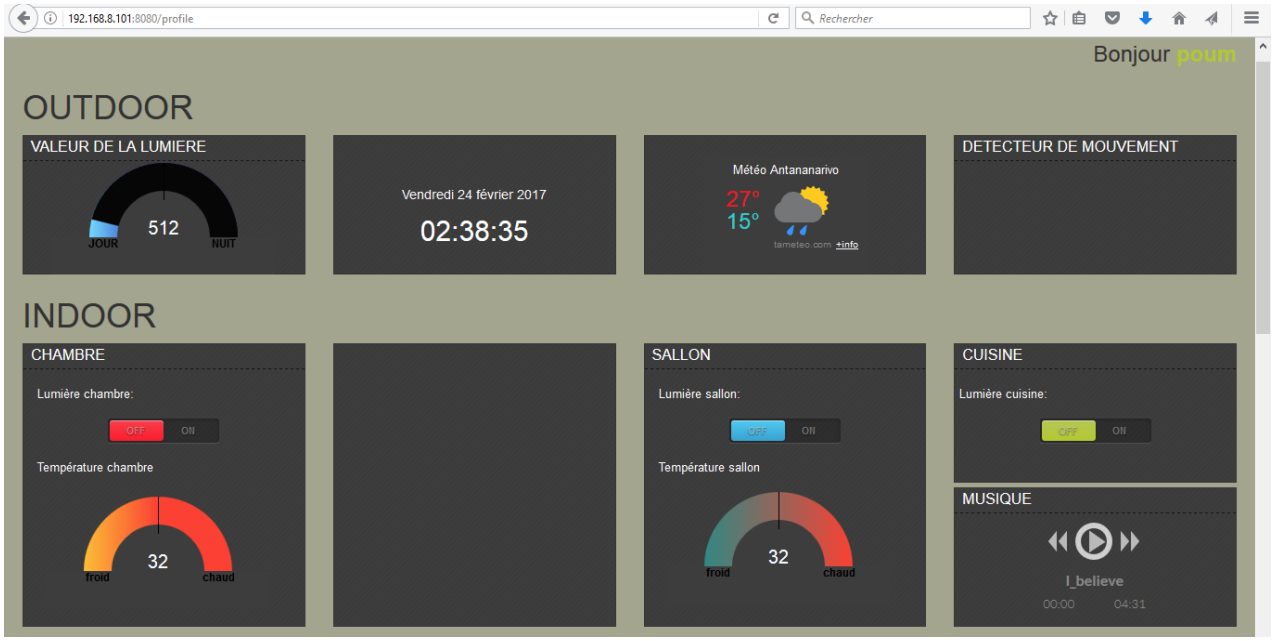

Figure 4.13 : *L'interface web de la plateforme*

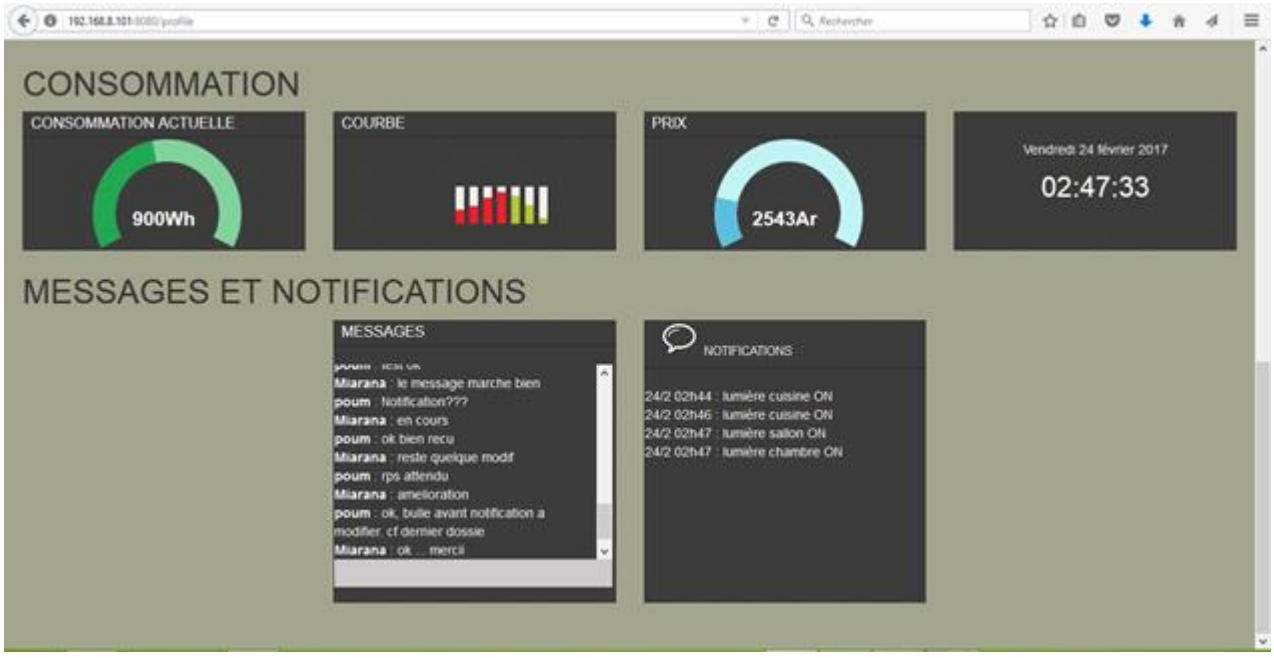

Figure 4.14 : *Partie interface web de la plateforme*

Présentons maintenant un à un les fonctionnalités que présentent cette page web.

Outdoor

Dans cette partie nous verrons la luminosité extérieure ainsi que le détecteur de mouvement s'il y a une personne ou non. On y trouve également la date et l'heure ainsi que la météo du jour grâce à l'API fournit par «tameteo.com ».

Indoor

Cette partie est divisée en 3 pièces : la chambre, la cuisine et le salon. Dans chaque pièce on pourra commander la lampe. Dans la partie chambre et salon on peut surveiller l'évolution la température en temps réel la salle grâce à des jauges circulaires. Particulièrement dans le salon le ventilateur se mettra en marche automatiquement au-delà d'une valeur seuil de température.
Dans cette partie également un playlist de musique sera mis à disposition pour tout client qui se connecte à la plateforme.

Consommation

Cette rubrique permet de faire un monitoring en temps réel de la consommation électrique de la maison. Ainsi le prix de cette consommation sera générer automatiquement.

Notifications et messages

La partie message permet d'échanger des messages en temps réels entre les utilisateurs connectes. Et la partie notification permet de journaliser chaque évènement qui se passe au niveau de la plateforme. Par exemple elle notifie quand le lumière de la cuisine est allumée ou éteint en donnant la date et l'heure précise du changement d'état.

4.9 Conclusion

Nous avons bien pu mettre en place notre plateforme IoT au sein de la domotique. Pour ce faire nous avons bâti notre serveur web avec Node.js et mis en valeur le protocole MQTT pour transmettre les informations depuis les capteurs vers le serveur et réciproquement des données provenant de la page web vers les objets. Comme matériel nous avons la carte Arduino, le raspberry pi et des terminaux comme l'ordinateur portable, smartphone.

CONCLUSION GENERALE

Ce présent mémoire nous a permis de se familiariser avec les nouvelles technologies que ce soit du hardware tel que la carte Arduino qui est en plein essor actuellement pour pouvoir piloter des différents capteurs ou encore des objets mais aussi le Raspberry pi qui est un micro-ordinateur très pratique pour implémenter et programmer les systèmes embarqués comme dans notre cas. Et en ce qui concerne la partie software, on s'est focalisé sur l'administration du système Raspbian Jessie, le Node.js, les languages de programmations C/C++, javascript et HTML.

Au cours de ce travail nous avons vu une large panoplie de solutions possible pour la réalisation d'un projet IoT mais en considérant les différentes contraintes telle que la bande passante ou encore le souci d'alimentation ; il faut donc savoir choisir le protocole ainsi que la technologie qui correspond au mieux à ces attentes. Dans notre cas nous avons choisi le protocole MQTT développé par IBM qui est encore très récent. Ce protocole est spécialement dédié pour l'IoT basé sur le principe de souscription et publication (pub/sub).

L'avenir de l'IoT repose sur les nouvelles technologies récentes ou encore en cours d'étude comme la 5G. Face à l'essor de l'IoT, du M2M et des services respectueux de l'environnement, il devient essentiel de dépasser la technologie 4G. La 5G permettra d'assurer la continuité et la qualité de l'expérience de l'utilisateur n'importe où n'importe quand. Le réseau ainsi couvert de façon plus large et plus homogène. L'amélioration du projet est l'implémentation d'une intelligence artificielle au niveau de la plateforme.

Pour conclure, l'Internet des objets est un système complexe intelligent qui entraîne des bouleversements économiques et sociétaux majeurs. Sa caractéristique d'interconnectivité, étant à la fois sa plus grande force et sa plus grande faiblesse, pose la problématique de la sécurité et du maintien de la confidentialité des données qu'elle génère. Un défi de taille pour l'industrie informatique de demain.

ANNEXES

ANNEXE 1 : NODE.JS

A1.1 Code de création du modèle utilisateur avec passeport.js et la base de données mongoDB.

```
var mongoose = require('mongoose');
var bcrypt   = require('bcrypt-nodejs');
var userSchema = mongoose.Schema({
    local            : {
        username        : String,
        password     : String,
    },
    facebook         : {
        id           : String,
        token        : String,
        email        : String,
        name         : String
    },
    twitter          : {
        id           : String,
        token        : String,
        displayName  : String,
        username     : String
    },
    google           : {
        id           : String,
        token        : String,
        email        : String,
        name         : String
    }
});
userSchema.methods.generateHash = function(password) {
    return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);
};
userSchema.methods.validpswd= function(password) {
    return bcrypt.compareSync(password, this.local.password);
};
module.exports = mongoose.model('User', userSchema);
```

A1.1 Code de définition des vues

Les lignes de codes suivantes représentent la définition des vues avec Node.js

```
module.exports = function(app, passport) {
    var path =  require('path'); // to use simple expes
   app.get('/', function(req, res) {
      res.render('index.ejs'); // load the index.ejs file
   });
      app.get('/login', function(req, res) {

      // render the page and pass in any flash data if it exists
      res.render('login.ejs', { message: req.flash('loginMessage') });
   });

   app.post('/login', passport.authenticate('local-login', {
      successRedirect : '/profile',
      failureRedirect : '/login',
      failureFlash : true
   }));

   app.get('/signup', function(req, res) {
      res.render('signup.ejs', { message: req.flash('signupMessage') });
   });

   app.get('/profile', isLoggedIn, function(req, res) {
      res.render('index_express.ejs', {
         user : req.user // get the user out of session and pass to template
      });
   });
{
      req.logout();
      res.redirect('/');
   });
};

// route middleware to make sure
function isLoggedIn(req, res, next) {

   // if user is authenticated in the session, carry on
   if (req.isAuthenticated())
      return next();

   // if they aren't redirect them to the home page
   res.redirect('/');
}
```

A1.2 Code gestion des connexions MQTT et du websocket

```
var express  = require('express');
var app      = express();
var http = require('http').createServer(app)  ,
require('socket.io').listen(http);
var mqtt = require('mqtt'); var client = mqtt.connect('192.168.1.26:1883');
var port     = process.env.PORT || 8080;
```

```
var mongoose = require('mongoose');
var passport = require('passport');
var flash    = require('connect-flash');
var temp1 = '', temp3 = '';
var clientTempC = mqtt.connect('192.168.1.26:1883');
clientTempC.on('connect', function() {
    clientTempC.subs('temperature/chambre', function() {
            clientTempC.on('message', function(topic, temperature1, packet) {
            io.sockets.emit('tempC', temperature1);

        });
    });
});
```

ANNEXE 2 : EXTRAIT DU CODE SOURCE

Code Arduino pour la connexion MQTT

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

// Update these with values suitable for your network.
byte mac[]    = {  0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
IPAddress ip(172, 16, 0, 100);
IPAddress server(172, 16, 0, 2);

void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i=0;i<length;i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

EthernetClient ethClient;
PubSubClient client(ethClient);

void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Attempt to connect
    if (client.connect("arduinoClient")) {
      Serial.println("connected");
      client.publish("outTopic","hello world");
      client.subscribe("inTopic");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
```

```
      delay(5000);
    }
  }
}

void setup()
{
  Serial.begin(57600);

  client.setServer(server, 1883);
  client.setCallback(callback);

  Ethernet.begin(mac, ip);
   delay(1500);
}
void loop()
{
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

BIBLIOGRAPHIE

[1] D. Evans, « *L'Internet des objets Comment l'évolution actuelle d'Internet transforme-t-elle le monde? »,* Cisco IBSG © Avril 2011.

[2] C. Trout « *Researchers Debut One-Cubic-Millimeter Computer, Want to Stick It in Your Eye* », Endadget, 26 février 2011

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, *« A survey on sensor networks »*, Communications Magazine, IEEE, 2002.

[4] C.F García-Hernandez, P.H.I.Gonzalez, Joaquín García-Hernandez, and J.A.P.Díaz, « *Wireless sensor networks and applications : a survey, IJCSNSInternational* », Journal of Computer Science and Network Security 7 ,2007

[5] I.F. Akyildiz and I.H. Kasimoglu, « *Wireless sensor and actor networks : research challenges* », Ad hoc networks 2, 2004

[6] M. Butty, « ADSL *Connaissances de base* », Swisscom SA Network Training, septembre 2000.

[7] A. Delley, M. Francioli et P. Zbinden, « *Technologies d'accès aux réseaux »*, Ecole d'ingénieurs et d'architectes de Fribourg, 1999, suppport de cours.

[8] A. WEI, « *Technologie des applications client-serveur RSX 102* «, CNAM Paris, mars 2012.

[9] A. Tanenbaum, P. Education, « *Réseaux* », 4ième édition, ISBN 2-7440-7001-7.

[10] M. Simatic , « *Module CSC4508/M2* »TELECOM SudParis, publication, Avril 2012.

[11]P. Arlaud – J. Dupire, « «*Architecture N-Tier* », CNAM 2009 – 2010.

[12] Y. Prié « *Architecture client-serveur* », UFR Informatique Université Claude Bernard Lyon 2004-2005 – Master SIB M1 – UE 3 / Bloc 4 – Cours 3

[13] C. Bulfone « *Le modèle client/serveur* » – Master IC2A/DCISS].

[14] La Rédaction, « *Internet des Objets : bien comprendre MQTT* », Le MagIT, juillet 2016.

[15]Sébastien Bonnet « *MQTT : Un protocole dédié pour l'IoT* », http://www.digitaldimension.solutions/blog/avis-d-experts/2015/02/mqtt-un-protocole-dedie-pour-liot/ 27 février 2015.

[16] A. Stanford-Clark and Hong Linh Truong, « *MQTT For Sensor Networks (MQTT-SN) Protocol Specification* », Version 1.2, November 14, 2013].

[17] Texas Instruments, « *LM35 Precision Centigrade Temperature Sensors* », SNIS159G – AUGUST 1999 –REVISED AUGUST 2016.

RENSEIGNEMENTS

Nom : RAMAROSAONA

Prénoms : Miarana Jehiela

Adresse : Lot IVS 63 Antanimena

jehielamia.ramarosaona@gmail.com

+261 34 06 369 06

Titre du mémoire :

« CONCEPTION ET CREATION D'UNE PLATEFORME IoT AVEC LE PROTOCOLE MQTT »

Nombre de Pages : 85

Nombre de Tableaux : 9

Nombre de figures : 46

Directeur de mémoire :

RAJAONARISON Roméo

+261 34 64 761 10

rajaonarisonromeo11***@gmail.com***

RESUME

L'Internet des objets (ou IoT) se traduit à l'heure actuelle par l'accroissement du nombre d'objets connectés, c'est-à-dire d'appareils possédant une identité propre et des capacités de calcul et de communication de plus en plus sophistiquées: montres, appareils ménagers, etc. Ces objets embarquent un nombre grandissant de capteurs et d'actionneurs leur permettant de mesurer l'environnement et d'agir sur celui-ci, faisant ainsi le lien entre le monde physique et le monde virtuel. Spécifiquement, l'Internet des objets pose plusieurs problèmes, notamment du fait de sa très grande échelle, de sa nature dynamique et de l'hétérogénéité des données et des systèmes qui le composent (appareils puissants/peu puissants, fixes/mobiles, batteries/alimentations continues, etc.). Ces caractéristiques nécessitent des outils et des méthodes idoines pour la réalisation d'applications capables d'extraire des informations utiles depuis les nombreuses sources de données disponibles et d'interagir aussi bien avec l'environnement, au moyen des actionneurs, qu'avec les utilisateurs, au moyen d'interfaces dédiées. Ce qui nous a conduit à mettre en place une plateforme IoT basée sur la domotique en utilisant le protocole MQTT qui est particulièrement léger.

Mots clés IOT, MQTT, M2M, Domotique, node.js

ABSCTRACT

The Internet of Things (IoT) is currently characterized by an ever-growing number of networked Things, i.e., devices which have their own identity together with advanced computation and networking capabilities: smartphones, smart watches, smart home appliances, etc. In addition, these Things are being equipped with more and more sensors and actuators that enable them to sense and act on their environment, enabling the physical world to be linked with the virtual world. Specifically, the IoT raises many challenges related to its very large scale and high dynamicity, as well as the great heterogeneity of the data and systems involved (e.g., powerful versus resource-constrained devices, mobile versus fixed devices, continuously-powered versus battery-powered devices, etc.). These challenges require new systems and techniques for developing applications that are able to collect data from the numerous data sources of the IoT and interact both with the environment using the actuators, and with the users using dedicated GUIs. This led us to set up an IoT platform based on home automation using the MQTT protocol, which is particularly lightweight.