DynPubSub: Architecture pair à pair pour les systèmes Pub/Sub orientés-sujets déployés à la périphérie des réseaux

par

Chamseddine BOUALLEGUE

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN TECHNOLOGIES DE L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 15 SEPTEMBRE 2021

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Julien Gascon-Samson, Directeur de mémoire
Département de génie logiciel et des TI, École de technologie supérieure

M. Aris Leivadeas, Président du jury
Département de génie logiciel et des TI, École de technologie supérieure

M. Kaiwen Zhang , Membre du jury
Département de génie logiciel et des TI, École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE "8 SEPTEMBRE 2021"

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à remercier toutes les personnes qui ont contribué de près ou de loin au succès de ma maîtrise.

Je désire d'abord remercier les membres de jury pour leur lecture attentive de mon mémoire, ainsi que d'avoir accepté de juger mon travail.

Je voudrais exprimer mes sincères remerciements à men encadrant M Julien Gascon-Samson pour ses conseils, sa disponibilité et sa qualité d'encadrement.

J'aimerais bien remercier MITACS, l'organisme qui m'a offert cette opportunité de poursuivre mes études supérieures au Canada dans un établissement de très haute qualité comme l'École de technologie supérieure ÉTS.

Je remercie toute ma famille et mes amis qui ont eu confiance en moi et m'ont encouragé durant les moments difficiles.

DynPubSub: Architecture pair à pair pour les systèmes Pub/Sub orientés-sujets déployés à la périphérie des réseaux

Chamseddine BOUALLEGUE

RÉSUMÉ

L'internet des objets (IoT) s'intègre de plus en plus dans notre vie quotidienne avec le nombre élevé des appareils connectés partout dans le monde. Cependant, les appareils IoT sont contraints en termes de ressources computationnelles et réseautiques, notamment la bande passante. Ainsi les applications IoT se distinguent par leurs criticités concernant le temps de réponse, ce que nous appelons la latence, ceci d'un simple système météorologique intelligent à une application de chirurgie à distance qui exige 1ms de latence. Pour faire face à ce défi, les chercheurs essaient toujours d'optimiser l'infrastructure et l'architecture réseau de leurs applications, ainsi que les protocoles de communications implémentés. Dans ce contexte, nous proposons une nouvelle architecture pair à pair pour les systèmes pub/sub basés sur les topics déployés à la périphérie des réseaux qui permet d'équilibrer la charge de trafic des noeuds IoT afin de respecter leurs contraintes de bande passante et minimiser la moyenne de latence d'échange de données. Notre solution est compatible avec le protocole de communication MQTT et permet de réduire la moyenne des latences jusqu'à 5% par rapport à un serveur central local et jusqu'à 60% par rapport à un serveur central cloud dans un déploiement local, ainsi elle permet de réduire la moyenne des latences jusqu'à 44% par rapport à un serveur central cloud dans un déploiement national et permet de réduire la moyenne des latences jusqu'à 18% par rapport à un serveur central cloud dans un déploiement mondial.

Mots-clés: Internet des objets, pub/sub, pair à pair, edge computing, MQTT

DynPubSub : Peer to Peer overlay for topic based pub/sub systems deployed at the edge

Chamseddine BOUALLEGUE

ABSTRACT

The Internet of Things (IoT) is becoming more and more integrated into our daily lives with the high number of connected devices around the world. However, IoT devices are constrained in terms of computational and networking resources, including bandwidth. Thus, IoT applications are distinguished by their criticality regarding response time, which we call latency, from a simple smart weather system to a remote surgery application that requires 1ms of latency. To face this challenge, researchers are always trying to optimize the network infrastructure and architecture of their applications, as well as the implemented communication protocols. In this context, we propose a new peer-to-peer architecture for topic-based pub/sub systems deployed at the edge that allows balancing the traffic load of IoT nodes to meet their bandwidth constraints and minimize the average data exchange latency. Our solution is compatible with the MQTT communication protocol and reduces average latency up to 5% compared to a local central server and up to 60% compared to a central cloud server in a local deployment, thus reducing average latency up to 44% compared to a central cloud server in a national deployment and reducing average latency up to 18% compared to a central cloud server in a worldwide deployment.

Keywords: Internet of things, pub/sub, peer-to-peer, edge computing, MQTT

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ALGORITHMES

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

6LoWPAN	IPv6 Low power Wireless Personal Area Networks
AMQP	Advanced Message Queuing Protocol
CoAP	Constrained Application Protocol
DDS	Data Distribution Service
GPIO	General Purpose Input/Output
IaaS	Infrastructure as a Service
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
JSON	JavaScript Object Notation
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
P2P	Peer to Peer
PaaS	Platform as a Service
Pub/Sub	Publish/Subscribe
SaaS	Software as a Service
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TI	Technologie de l'Information
TLS	Transport Layer Security
UDP	User Datagram Protocol
WAN	Wide Area Network

WiFi	Wireless Fidelity
WSN	Wireless sensor network
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

INTRODUCTION

0.1 Contexte et problématique

L'internet des objets est une appellation très connue de nos jours et peut avoir plusieurs définitions. Il s'agit d'interconnecter des milliers d'objets via internet afin d'offrir certains services qui s'intègrent de plus en plus dans notre vie quotidienne. En effet, avec la maison intelligente, la ville intelligente et les véhicules intelligents, notre vie devient plus sécurisée et confortable.

Afin d'assurer la connectivité entre les appareils IoT, il existe plusieurs protocoles de communications, chacun possède des avantages et des inconvénients et chacun est adaptée à certaines applications par exemple Coap (Constrained Application Protocol) (Kriddikorn Kaewwongsri, 2020) est dédié optimisé pour les périphériques contraints, XMPP (Extensible Messaging and Presence Protocol) (Nikolov, 2020) est dédié pour les services de messagerie instantanée notamment Google Talk et Jaber, DDS (Data Distribution Service) (Mukesh Kumar, 2019) est caractérisé par ses qualités de services et la sécurité qu'il offre, nous le trouvons généralement dans les applications critiques telles que le militaire et l'aéronautique. Nous ne pouvons pas parler des protocoles de communications sans mentionner le fameux MQTT (Message Queuing Telemetry Transport) adapté pour les applications IoT (messagerie, capteurs, monitoring, smart agriculture) grâce à sa légèreté et sa simplicité d'implémentation.

Les applications IoT se distinguent par leurs criticités en termes de temps de réponse. Pensons à l'exemple du réseau de capteurs sans fils qui n'est pas assez critique en termes de latence, ainsi à l'exemple de la chirurgie à distance, ou les véhicules autonomes qui exigent une latence alentour de 1ms. Cependant d'autres applications sont moins critiques en termes de latence tels que le réseau de capteurs sans fils. La latence est donc un facteur assez important lors de déploiement des applications IoT.

Les développeurs des applications IoT cherchent toujours à optimiser l'infrastructure réseau, ainsi que l'architecture de déploiement des noeuds IoT dans le but de minimiser la latence. Par exemple il peuvent choisir entre le déploiement dans l'infonuagique, ou bien le déploiement en périphérie (couches edge ou fog) selon les exigences. En effet si l'application IoT requiert des ressources physiques énormes à savoir processeur ou mémoire, les chercheurs ont tendance à prioriser l'infonuagique parce qu'il est riche en ressources computationnelles, cependant si l'application requiert une faible latence plus que les ressources, les chercheurs ont tendance à prioriser la périphérie du réseau *Edge Computing* et le *Fog Computing* puisqu'ils sont plus proches aux noeuds IoT.

Le choix de l'infrastructure réseau joue aussi un rôle important dans les phases de développement des applications IoT. En effet certaines applications fonctionnent avec une infrastructure client/serveur simple, par contre d'autres requièrent une infrastructure plus sophistiquée à savoir une infrastructure pair à pair. Ceci revient toujours aux exigences de l'application désirée, par exemple une infrastructure client/serveur ou centralisée minimise la maintenance du matériel, simplifie le traitement de et la recherche des données. Par contre une infrastructure pair à pair est plus tolérante aux pannes et plus performante puisque la charge ne sera pas centralisée sur un seul noeud. Ainsi elle réduit le volume des données à transmettre et le coût des communications comparées à une infrastructure centralisée.

Dans ce contexte, il est indispensable de trouver un compromis entre les architectures de déploiement, les infrastructures réseaux, ainsi que les protocoles de communications afin d'offrir une solution optimale qui respecte les contraintes des appareils IoT notamment la bande passante, avec pour objectif de réduire la latence des échanges de messages.

Notre solution DynPubSub vise à révéler les défis en liaison avec l'internet des objets, en optimisant le déploiement de l'infrastructure sur les dispositifs en périphérie (Edge). Globalement, DynPubSub propose une approche de partitionnement dynamique pour la couche de dissémination

(overlay) dans laquelle les sujets (topics) de l'infrastructure pub/sub sont dynamiquement (re)mappés sur les différents dispositifs de périphérie (Edge) en fonction de la charge actuelle du réseau et des capacités des différents nœuds de périphérie, avec la minimisation de la latence comme objective d'optimisation. En outre, nous fournissons une bibliothèque compatible avec la bibliothèque MQTT de Node.js (mqtt.js), ce qui permet aux développeurs de passer de manière transparente d'un déploiement centralisé de pub/sub à l'architecture DynPubSub de pair à pair sans avoir à modifier leur code.

0.2 Objectifs

L'objectif général de notre projet de recherche est de concevoir une nouvelle architecture pair à pair déployée en périphérie pour les systèmes pub/sub basé sur les *Topics*. En effet nous nous concentrons sur la dynamicité de l'architecture dont le but est de respecter les contraintes des appareils IoT à savoir la bande passante et réduire la latence moyenne pour l'échange des messages.

Nous avons identifié deux sous-objectifs à partir de notre objectif général.

1. concevoir une librairie, compatible avec une librairie existante (MQTT.js), qui gère les méthodes typiques du paradigme pub/sub orienté-sujet (publications, souscriptions, désouscriptions). Cette libraire va être partagée entre tous les noeuds IoT de notre architecture.
2. concevoir un module orchestrateur qui équilibre la charge entre les noeuds IoT suivant le volume de données échangées, et génère un plan optimal qui permet d'identifier quel noeud sera responsable de chacun des sujets (topics) dans le système afin de respecter la contrainte de bande passante et réduire la latence des échanges de messages.

Afin d'atteindre nos objectifs, nous avons commencé par explorer la librairie existante (MQTT.js) et nous avons étudié ses méthodes pour pouvoir l'adapter à notre libraire. Ensuite nous avons

identifié les paramètres à considérer tels que les publications, les souscriptions, les topics, la taille des messages dont notre orchestrateur aurait besoin afin de générer le plan optimal.

0.3 Méthodologie de recherche

Afin de réaliser notre projet de recherche, nous avons suivi une méthodologie de recherche comme le montre la figure 0.1 Nous avons réalisé une revue de littérature sur les notions de

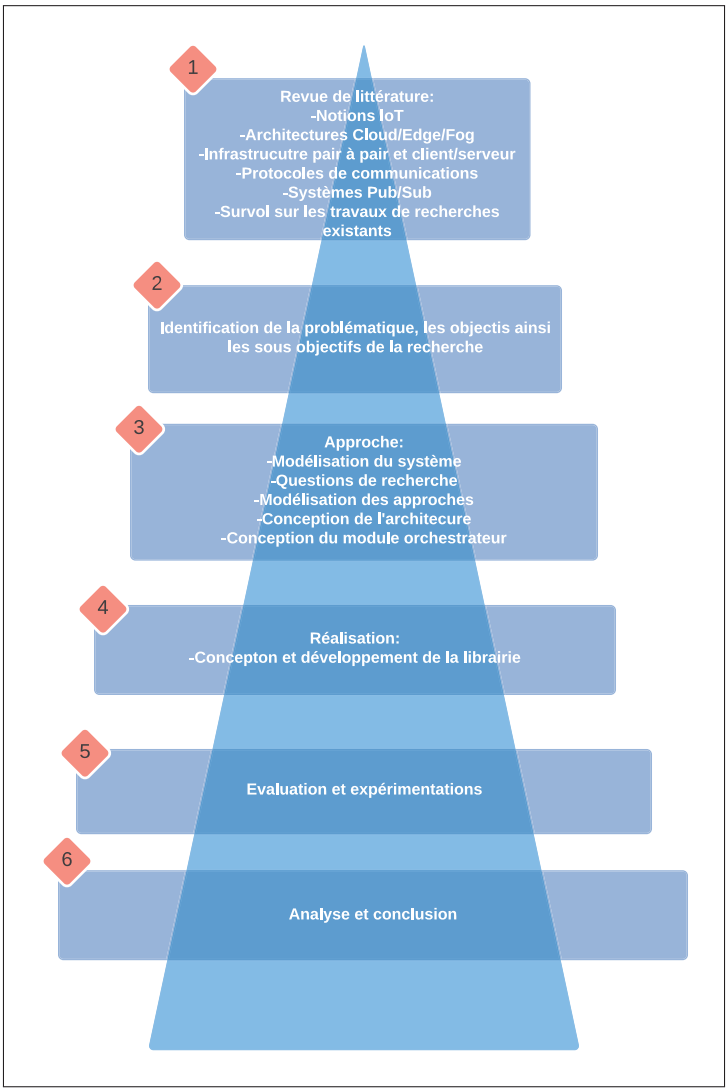

Figure 0.1 Méthodologie de recherche

l'internet des objets, les architectures de déploiement notamment l'infonuagique et la périphérie

des réseaux, ensuite les infrastructures réseaux pair à pair et client/serveur, les protocoles de communications et les systèmes pub/sub. Ainsi nous avons fait un survol sur les travaux de recherche dans ces domaines pour identifier les limites et les lacunes.

Ensuite nous avons défini la problématique et dégagé les objectifs à atteindre, en fonction des limites trouvées dans les travaux de recherches existants. Dans l'étape suivante, nous avons établi les questions de recherches à partir des objectifs, nous avons modélisé le système, ses approches et son architecture. Par la suite, nous avons exploré la libraire MQTT.js, et étudié les *plugins* et les modules qui nous aideront à l'implémentation de la solution notamment la conception de notre librairie et notre orchestrateur. Dans l'étape suivante, nous avons déterminé les expérimentations à réaliser pour répondre à nos questions de recherches et finalement nous avons consacré une étape pour l'analyse et une discussion sur les résultats obtenus.

0.4 Organisation du mémoire

Le mémoire est organisé de la manière suivante : Le chapitre 1 intitulé "revue de littérature" inclut les définitions et les notions de base de l'Iot, les architectures de déploiement à savoir *Cloud, Edge Computing, Fog Computing*, les infrastructures réseaux (pair et pair et client/serveur), les protocoles de communications les plus utilisés dans l'IoT, les modèles pub/sub ainsi un survol sur les travaux de recherches dans ces domaines. Le chapitre 2 intitulé "Approche" contient le modèle du système, les approches, le problème d'optimisation ainsi que l'architecture. Le chapitre 3 intitulé "Implémentation" contient les détails concernant la phase de réalisation notamment les outils de développement, et les algorithmes de notre librairie. Le chapitre 4 intitulé " Évaluation et résultats" présente les paramètres ainsi les résultats des expérimentations obtenus lors de l'évaluation de notre solution. Finalement, le chapitre 5 présente un résumé des résultats obtenus lors de l'évaluation de notre solution, ainsi les limites de l'étude.

CHAPITRE 1

REVUE DE LITTÉRATURE

Dans ce chapitre, nous présentons certains concepts-clés permettant de comprendre les sections suivantes de ce mémoire à savoir la définition de l'internet des objets (1.1), l'informatique en périphérie (1.2.3), les systèmes pair à pair (1.3.2), les systèmes pub/sub (1.4), ensuite nous présentons la relation entre l'internet des objets et ces systèmes et finalement nous faisons un survol sur les travaux de recherches existants qui sont liés à notre étude.

1.1 Internet des Objets

L'internet des objets (IoT) est une technologie qui permet d'interconnecter plusieurs objets à travers l'internet. Elle touche plusieurs domaines d'application (Patrick Eugster,Pascal Felber, 2017) à savoir l'agriculture, le transport, l'énergie, dont le nombre des appareils intelligents ne cesse de croître de jour en jour.

Le but principal de l'internet des objets est d'automatiser plusieurs tâches et rendre notre vie plus simple. Pensons par exemple de la domotique avec laquelle l'utilisateur peut surveiller et contrôler des capteurs et des actionneurs dans sa maison à distance, il peut contrôler des lumières, des thermostats et des systèmes de climatisation à travers une application web ou mobile. L'établissement de la communication entre les objets connectés se fait à l'aide des protocoles de communications bien étudiés qui répondent aux besoins et aux exigences de l'application demandée.

Dans ce contexte, l'internet des objets fait face à plusieurs défis tels que la sécurité, la gestion des données massives, la réponse en temps réel, et la bande passante.

1.2 Les architectures réseaux IoT

Dans les projets IoT, les développeurs ont besoin de choisir une architecture réseau qui répond aux besoins de l'application développée tels que la minimisation de la latence, l'optimisation

de la bande passante, et l'optimisation de la consommation des ressources physiques (CPU, mémoire, stockage). Dans ce contexte, les développeurs doivent choisir entre l'infonuagique (Cloud Computing), informatique de brouillard (Fog Computing) et l'informatique de périphérie (Edge Computing) qui sont décrits dans les sections qui suivent.

1.2.1 IoT et Cloud Computing

L'infonuagique (Cloud Computing) signifie la possibilité d'héberger sur internet des ressources, des services et des données selon le besoin des applications. L'infonuagique est caractérisée par :

1. service à la demande : offrir des ressources computationnelles (CPU, mémoire) à la demande.
2. accès via le réseau : les services sont disponibles via l'internet.
3. pooling de ressources : Servir plusieurs clients avec la même infrastructure et les mêmes ressources computationnelles.
4. élasticité rapide : automatiser l'allocation et la désallocation des ressources.
5. service mesuré : mesure, surveillance et suivi des ressources computationnelles utilisées.

On distingue entre troix niveaux de services à savoir logiciel en tant que service (SaaS), plateforme en tant que service (PaaS) et infrastructure en tant que service (IaaS). Il existe quatre modèles de déploiement pour l'infonuagique

1. nuage public : le public général est permis de l'utiliser (Amazon, Google, MS Azure).
2. nuage privé : une seule entreprise est permise de l'utiliser (Amazon Internal Cloud).
3. nuage communautaire : une communauté spécifique est permise de l'utiliser.
4. nuage hybride : une composition de deux ou plus modèles infonuagiques distinctes.

L'internet des objets est en étroite relation avec le l'infonuagique. En effet les applications Iot exigent une accessibilité et connectivité omniprésente sur divers objets/services hétérogènes et sur divers volumes d'utilisateurs (Abdur Rahim Biswas, 2014) y compris la mobilité à travers des API's standardisées. Ensuite L'IoT requiert un service d'orchestration et de gestion de milliards d'appareils connectés tout en offrant la possibilité de personnalisation de services offerts.

1.2.2 IoT et Fog Computing

L'informatique en brouillard (Fog Computing) étend l'infonuagique pour se rapprocher des objets qui produisent et agissent sur les données IoT. Une architecture informatique en brouillard est composée principalement de dispositifs appelés noeuds de brouillard (Fog) caractérisé par des capacités de calcul et de stockage qui peuvent être déployés n'importe où tel que l'exemple de la ville intelligente où les bâtiments et les véhicules sont interconnectés. L'informatique en brouillard (Fog Computing) est une couche (Mehreen Ahmed, 2020) entre les appareils IoT et l'infonuagiquee (cloud computing) comme le montre la figure ci-dessous (Figure 1.1)

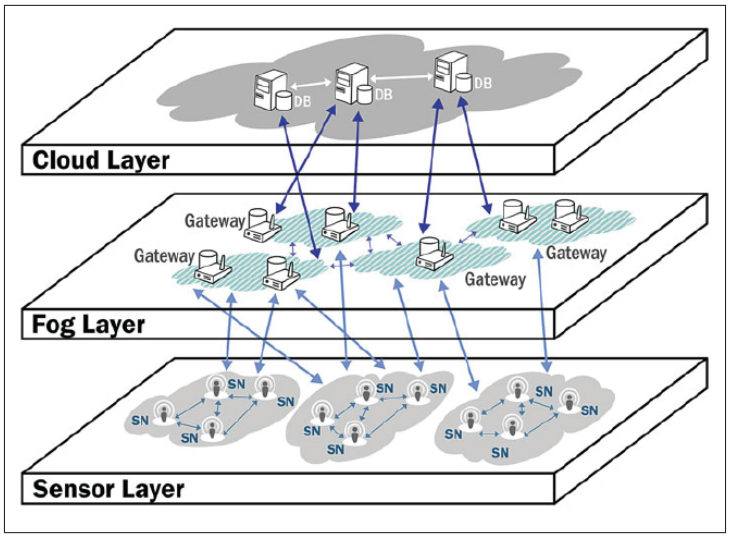

Figure 1.1 Architecture Fog
Tirée de Mehreen Ahmed (2020, p5)

L'informatique en brouillard (Fog Computing) offre plusieurs avantages pour les applications IoT. Par exemple nous pouvons citer :

1. faible latence : contrairement a l'infonuagique (cloud), l'informatique en brouillard (Fog Computing) minimise le délai de services offerts pour les applications IoT.
2. réduire le volume de données : l'introduction des noeuds fog a permis de réduire la charge sur le serveur central infonuagique (cloud).

3. amélioration du temps de réponse : contrairement au traitement centralisé, les données générées par les capteurs IoT vont être traitées à un rythme beaucoup plus élevé.
4. évolutivité : les données adjacentes à la source réduisent le niveau de traitement de données sur cloud.
5. conservation de la bande passante du réseau : l'informatique en brouillard (Fog Computing) réduit la communication entre les capteurs et le cloud d'où la réduction de la consommation de la bande passante.

1.2.3 IoT et Edge Computing

L'informatique en périphérie place le calcul et le stockage des données sur les appareils IoT eux-mêmes. Cette architecture permet de réduire les latences des services pour atteindre le temps réel pour des applications IoT critiques telles que la chirurgie à distance et les voitures autonomes. Elle permet ainsi de minimiser les besoins en bande passante en évitant de transmettre les données volumineuses vers des centres de données ou infonuagique à travers l'internet. Edge computing offre plusieurs avantages (Soumyalatha Naveen, 2019) pour les applications IoT telles que la réponse en temps réel, la réduction de la consommation de la bande passante, l'amélioration de la sécurité, la réduction des coûts, l'amélioration de l'efficacité et de l'analyse de données en temps réel. Le tableau (Soumyalatha Naveen, 2019) ci-dessous présente une comparaison entre les architectures cloud computing et edge computing.

1.3 Les infrastructures réseaux dans l'IoT

Parmi les étapes nécessaires de développement d'applications IoT, nous trouvons le choix de l'infrastructure réseau à implémenter selon les exigences de l'application en termes de latences et de disponibilité. Les deux infrastructures réseaux les plus utilisées sont client/serveur et pair à pair (peer to peer).

Tableau 1.1 Comparaison entre Edge Computing et Cloud Computing

Fonctionnalité	Edge Computing	Cloud Computing
Localisation	Noeuds distribués et proches de l'utilisateur	N'importe quel endroit éloigné
Temps de réponse	Faible	Variable-peut être élevé
Mobilité	Elevée	Faible
Traitement et prise de décision	Dans le dispositif local	cloud distant
Communication	Intéraction en temps réel	Contraintes en bande passante
Taille de stockage	Petite	Taille énorme
Sensibilisation au contexte	Oui	Non
Environnement opérationnel	Décidé par l'utilisateur	Contrôlé par l'opérateur Cloud
Hétérogénité des dispositifs	Hautement supporté	Support limité
Capacités computationnelles	Moyennes	Elevées
Coût de développement	Faible (propre appareils)	Elevé (réserver des ressources)
Applications	Applications critiques (temps réel)	Prend en charge la plupart des applications
Déploiement	Ne nécessite pas beaucoup de planification	Nécessite beaucoup de planification
Bande passante	Peut fonctionner sans beaucoup de connectivité réseau	Nécessite un bon réseau de connectivité

1.3.1 Infrastructure client/serveur et IoT

Dans une infrastructure client/serveur, les clients ou les stations de travail sont connectés à au moins un serveur central. Afin d'accéder aux ressources, les clients doivent passer par le serveur centralisé. Ce type d'infrastructure offre une vitesse d'accès rapide, car il est conçu pour gérer plusieurs clients et renforcer la sécurité de communication.

Il existe plusieurs travaux de recherche dans le domaine de l'internet des objets qui sont basés sur des infrastructures réseaux client/serveur.

Dans le travail de recherche (Brundha S.M,Lakshmi, 2017), les auteurs proposent une approche pour la domotique basée sur une infrastructure client/serveur qui permet d'ajuster la luminosité de la chambre selon la personne détectée. De plus ils ont introduit un système de sécurité qui permet de notifier les propriétaires de la maison sur la présence des intrus grâce à un système de reconnaissance faciale.

Coap (Constrained Application Protocol) est un protocole d'application très utilisé dans le domaine de l'internet des objets, il est basé sur l'infrastructure client/serveur et fonctionne à travers des requêtes HTTP (GET, POST, PUT, DELETE). Dans (Kriddikorn Kaewwongsri, 2020), les auteurs introduisent un système de surveillance météorologique basé sur l'architecture Coap qui permet de collecter des données en temps réel telles que la température, l'humidité, la vitesse et la direction du vent. Les données sont ensuite transmises vers un serveur central pour les stocker dans une base de données et les visualiser.

1.3.2 Infrastructure pair à pair et IoT

Dans une infrastructure pair à pair, tous les noeuds peuvent être à la fois clients et serveurs. Ils communiquent directement sans intermédiaire, et ils peuvent partager des ressources physiques ou logicielles. L'infrastructure pair à pair présente plusieurs avantages, à savoir l'optimisation de la bande passante, la réduction des coûts de maintenance, une meilleure tolérance aux pannes et la facilité d'extension et d'ajout de nouveaux équipements.
Il existe plusieurs travaux de recherche dans le domaine de l'internet des objets qui sont basés sur des infrastructures pair à pair.

L'article (David Tracey, 2019) analyse les architectures, les infrastructures réseaux et les approches qui sont les mieux adaptées aux applications IoT. Il relève ses défis principaux et propose une architecture Fog Computing basée sur une infrastructure pair à pair afin de garantir la scalabilité et la haute disponibilité.

Dans (Mahdi Ghorbani,Mohammad Reza Meybodi, 2019) les auteurs proposent une architecture pour gérer des systèmes IoT basés sur des réseaux cognitifs pair à pair. Cette architecture est

utilisée pour résoudre des problèmes très connus à savoir la découverte de ressources et la gestion de topologie. Les résultats montrent que l'architecture est capable de transférer une grande masse de données pour les systèmes distribués pair à pair pour l'IoT.

1.4 Les protocoles de communications IoT pub/sub

Publish-Subscribe est un moyen de publier des messages à partir des diffuseurs pour des abonnés qui s'intéressent à certaines catégories ou classes de messages. Il s'agit d'une forme de communication asynchrone, elle peut être utilisée dans des architectures évènementielles ou pour assurer le découplage des applications dans le but d'améliorer les performances, l'évolutivité et la fiabilité. Pub/Sub convient très bien dans un contexte IoT car il permet de mettre en relation de manière découplée les périphériques qui produisent des données des applications avec les périphériques qui consomment ces données. Parmi les protocoles de communications publish/subscribe les plus utilisés on peut citer, MQTT (Message Queuing Telemtry Transport), DDS (Data Distribution Service) et AMQP (Advanced Message Queuing Protocol).

1.4.1 Les principaux modèles de pub/sub

Il existe plusieurs modèles pub/sub, chacun possède des caractéristiques pour s'ajuster à un type bien précis d'application. Parmi les modèles les plus utilisés on peut trouver topic based pub/sub, content based pub/sub, type based pub/sub, attribute based pub/sub.

1.4.1.1 Modèle pub/sub basé sur les sujets (topic based pub/sub)

Il s'agit du premier modèle pub/sub, il est utilisé pour regrouper les pairs communicants et classifier le contenu des évènements. Ces pairs communicants (publishers et subscribers) peuvent publier des évènements et s'abonner à des sujets qui permettent de créer des groupes à l'aide des mots-clés (Patrick Eugster,Pascal Felber, 2003). Dans ce modèle les subscribers et les publishers doivent s'entendre sur les noms de canaux à exploiter.

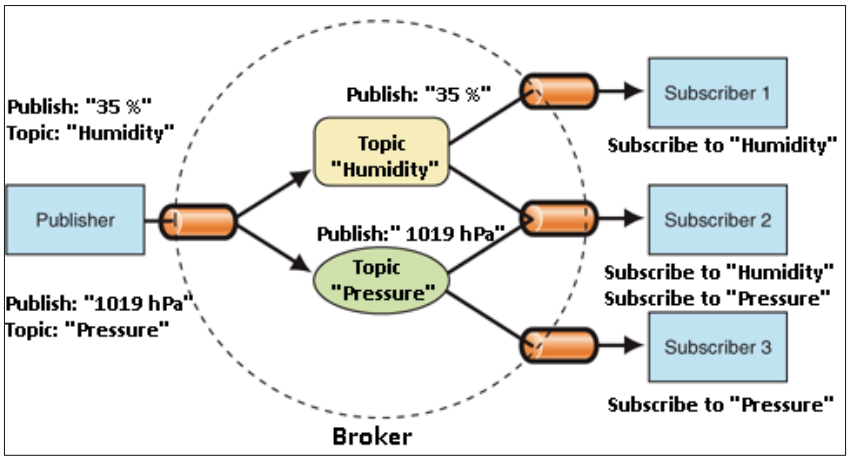

Figure 1.2 Scénario pub/sub basé sur les sujets (topics)

La figure 1.2 illustre un scénario de communication pub/sub basé sur les sujets (topics). En effet, nous avons deux clients qui sont souscrits au topic *humidity* (Souscripteur 1 et Souscripteur 2) et nous avons deux clients qui sont souscrits au topic *pressure* (Souscipteur 2 et Souscripteur 3), lorsque le client publieur (Publisher) publie des données sur les sujets (topics) *humidity* et *pressure*, chaque client souscripteur reçoit les données qui correspondent au topic auquel il est souscrit.

1.4.1.2 Modèle pub/sub basé sur le contenu (content based pub/sub)

Ce modèle permet d'améliorer le modèle topic based pub/sub, car ce dernier présente un schéma statique et des expressions limitées (Patrick Eugster, 2003). En effet le content based pub/sub introduit des souscriptions sur le contenu des évènements considérés, en d'autres termes les évènements ne sont pas classifiés selon des critères externes prédéfinis, mais plutôt sur les propriétés des évènements eux-mêmes.

La figure 1.3 illustre un scénario de communication pub/sub basé sur le contenu. En effet, nous avons un client souscrit à des messages dont le contenu est *<xml></xml>* (S1). Dès que le client publieur (P1) publie un message *<xml></xml>* le souscripteur (S1) le reçoit immédiatement.

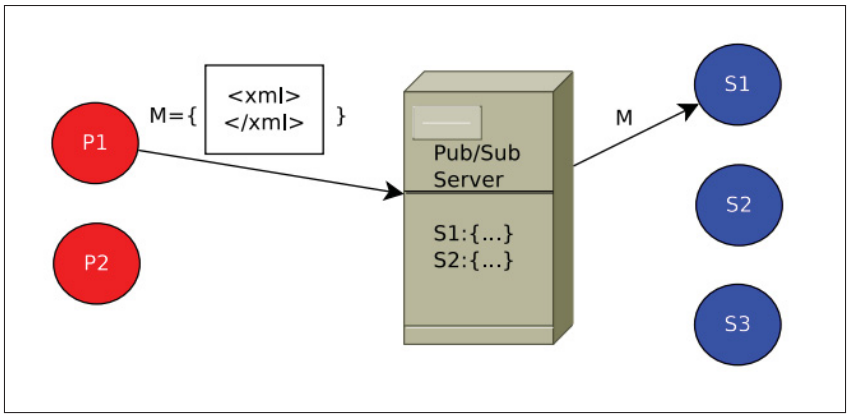

Figure 1.3 Scénario pub/sub basé sur le contenu
Tirée de Tarkoma (2012, p77)

1.4.1.3 Modèle pub/sub basé sur le type (type based pub/sub)

Dans ce modèle, le regroupement des événements est basé sur les points communs dans la structure et du contenu. Il remplace le modèle basé sur le nom (name-based) pour présenter un schéma qui permet de filtrer les événements selon leurs types .(Patrick Eugster, 2003). Il est important de noter que ce modèle peut mener à une description naturelle du filtrage basé sur le contenu à travers de membres de données publiques considéré comme le type pub/sub basé sur l'évenement, tout en offrant l'encapsulation de ces événements. Ceci peut être réalisé en déclarant uniquement des membres de données privés et en forçant leur accès par des méthodes publiques.(Patrick Eugster, 2003).

La figure 1.4 illustre un scénario de communication pub/sub basé sur le type. En effet, nous avons un client souscrit à des messages de type "A", donc il reçoit selon l'hiérarchie du service de notification les messages de type "A", "B", "C" et "D". Ainsi nous avons un client souscrit à des messages de type "B", donc il reçoit le message de type "C".

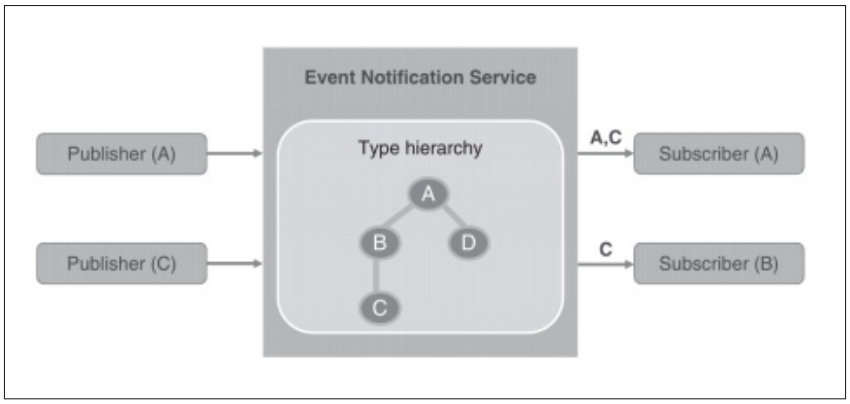

Figure 1.4 Scénario pub/sub basé sur le type
Tirée de Tarkoma (2012, p79)

1.4.2 MQTT : Message Queuing Telemtery Transport

MQTT est un protocole de messagerie léger basé sur le protocole TCP/IP. Prend en charge le paradigme publish/subscribe et est utilisé dans des environnements contraints tels que la domotique, le suivi de trafic et la messagerie instantanée.

L'objectif principal du protocole MQTT est de minimiser la bande passante du réseau et d'assurer la fiabilité de livraison de messages ce qui est très favorable pour les applications de type "machine-to-machine (M2M)" ou internet des objets dont la consommation de batterie est précieuse.

MQTT introduit plusieurs paramètres pour assurer une certaine qualité de service et une fonctionnalité testament. Ces paramètres sont Will, WillQoS, WillRetain, WillTopic et WillMessage. En effet, si le paramètre Will est à 1, la fonctionnalité "testament" du protocole sera activée et le serveur va sauvegarder un "dernier message" qui correspond à la session actuelle. Ensuite Le message va être transmis vers le topic correspondant indiqué dans WillTopic. Le paramètre WillQoS spécifie la qualité de service à utiliser, le paramètre WillMessage indique que le message testament va être stocké sur le serveur, le paramètre WillTopic introduit le topic à considérer lors l'envoi du message testament et finalement le paramètre WillRetain indique si

le serveur doit retenir le message testament après sa transmission ou pas, pour garantir une transmission fiable et efficace des messages.

MQTT fournit 3 niveaux de qualité de service (QoS 0, QoS 1 et QoS 2). En effet, QoS 0 (At most once) offre les mêmes garanties que le protocole TCP, il n'y a pas d'acquittement des messages publiés et le serveur ne stocke pas le message, il sera donc impossible de déterminer si le message a été reçu par les souscripteurs ou non. Avec QoS 1 (At least once), la confirmation de livraison est obligatoire, sinon le serveur réenvoie le message. Cette qualité de service garantit que le message sera livré au moins une fois comme son nom l'indique (At least once) et finalement avec QoS 2 (Exactly Once), le message sera livré exactement une seule fois ce qui nécessite un double mécanisme d'acquittement. L'inconvénient de cette qualité de service est le débit et la latence plus élevés.

1.4.3 DDS : Data Distribution Service

DDS prend en charge le paradigme pub/sub basé sur les sujets (topic based) et et sur le contenu (content based), il garantit un modèle de qualité de service très riche qui permet d'imposer diverses contraintes, telles que le temps maximal d'arrivée des données, le délai maximal, la persistance des données, les exigences en termes de livraison de message (Pardo-Castellote, 2003). DDS est utilisé dans des applications très strictes et sensibles telles que la défense, la bourse, l'énergie, les appareils médicaux, et le contrôle aérien.

1.4.4 AMQP : Advance Message Queuing Protocol

AMQP est un protocole ouvert pour la messagerie basé sur le paradigme publish/subscribe connu par son interopérabilité. Le but principal d'AMQP est de standardiser les échanges en se basant sur les fonctionnalités (orienté messages, file d'attente, routage, fiabilité et sécurité). En effet AMQP (Vinoski, 2006), résout plusieurs problèmes, il garantit une transmission de données fiables, et stocke les messages dans des files d'attente en se basant sur une communication

asynchrone. Le récepteur de message récupère le message dans la file d'attente, pour que l'émetteur puisse continuer à transmettre et éviter les périodes d'inactivités .

1.5 Évaluation des intergiciels (middlewares) pub/sub

Il existe plusieurs travaux qui ont pour but d'évaluer les intergiciels (middlewares) et les plateformes pub/sub, ces travaux permettent de choisir selon les exigences des applications IoT lequel des middlewares est le plus adéquat. Par exemple il y a des applications IoT qui demandent une faible latence donc il faut implémenter un middleware qui permet de garantir cette dernière, de même pour la bande passante, et la sécurité.

1.5.1 Leçons et défis liés à l'évaluation comparative des plates-formes IoT de type pub/sub.

Dans l'article (Ana Aguiar, 2019) les auteurs discutent de certains défis en lien avec les plateformes IoT publish/subscribe. Il présente l'emplacement des middlewares dans un système IoT d'un point de vue architectural et leurs rôles. Les auteurs ont ainsi présenté la différence entre les métriques qualitatives (les protocoles, l'interopérabilité des données, le contrôle d'accès, la sécurité ...) et quantitatives (les temps de publications et notifications, la bande passante, la charge IP, la charge TCP) qui doivent être prises en compte lors d'une implémentation d'un système IoT complet.

Pour garantir de bonnes performances, les auteurs ont mis l'accent sur le temps de publication qui peut être assez rapide si les publications sont parallélisées. Ils ont aussi mis l'accent sur la surcharge de données et le comportement des middlewares (Fiware, M2M, Ponte) en utilisant différents protocoles tels que Coap, Http, Mqtt. Ensuite les auteurs ont étudié les effets de l'organisation des données et de bande passante sur la performance. Finalement les auteurs résument les leçons acquises à en lien avec la performance d'un système publish/subscribe.

1.5.2 Outils et plates-formes pour l'évaluation comparative des intergiciels pub/sub

Le nombre d'appareils IoT augmente de jour en jour, ce qui va générer une très grande masse de données à échanger et analyser. Les intergiciels (middlewares) IoT jouent un rôle important pour la gestion de cette quantité de données énorme. Cependant il faut choisir le plus adéquat selon le type de l'application et ses exigences. Dans ce contexte, les auteurs de l'article "A Modular Tool for Benchmarking IoT Publish/Subscribe Middleware" (Zilhao, 2018) présentent un outil qui permet d'évaluer et comparer les intergiciels (middlewares) pub/sub IoT comme le montre la figure 1.5.

L'architecture de la solution est composée de certains blocs tels que le bloc Load qui permet de sauvegarder et charger les scénarios de benchmarking, le bloc Metrics qui contient les métriques à utiliser lors de l'évaluation telle que le temps de publication et le trafic généré, le bloc data qui contient les fonctions spécifiques du middleware, le bloc comm qui contient les protocoles de communications tels que MQTT, COAP, et les méthodes (POST/GET, PUT, DELETE), ainsi le bloc courtier (broker) qui permet de recevoir les messages issus des publishers et les livrer vers les subscribers. Les résultats obtenus permettent d'affirmer que la solution proposée a permis de comparer efficacement les intergiciels (middlewares "OM2M,Fiware et Ponte").

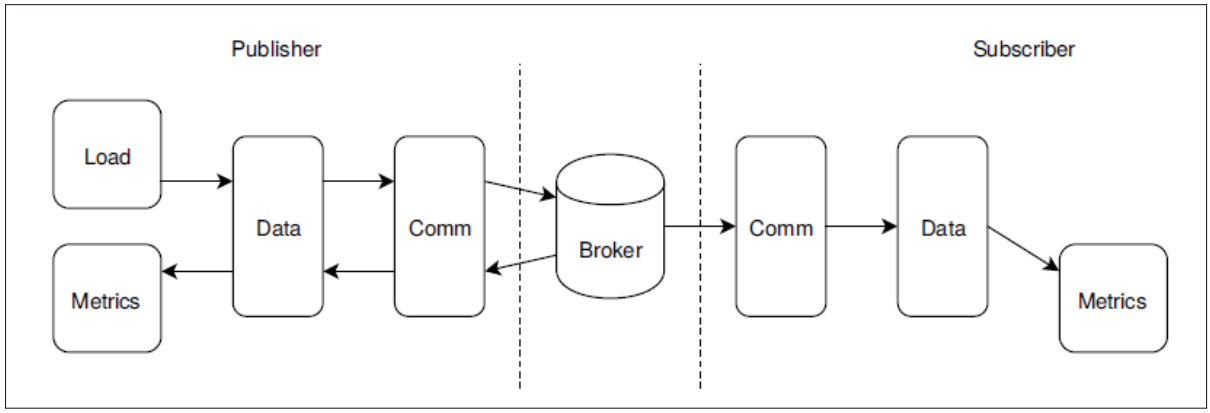

Figure 1.5 Architecture en bloc
Tirée de Zilhao (2018, p2)

Afin de garantir un déploiement efficace pour les appareils IoT, il faut intégrer un middleware performant dans les applications IoT. Dans ce contexte, les auteurs de l'article (Joao Cardoso,Carlos Pereira, 2017) proposent des dimensions quantitatives pour évaluer la performance des midllewares IoT, ces dimensions concernent :

1. le temps nécessaire entre l'envoi de la requête de publication et la réception de la réponse de publication.
2. le temps nécessaire entre l'envoi de la requête de publication et la notification des souscriptions.
3. le temps total entre l'envoi de la première publication et la réception de la dernière réponse de publication.
4. la taille des données rassemblées et la surcharge des entêtes.
5. la taille en octets de la charge utile du protocole de transport du paquet de publication.
6. la quantité totale de données utilisées pour publier une ressource.
7. le nombre utile d'octets envoyés dans une période, mesurée comme la taille des données sérialisées divisées par le temps de publication.

Ainsi les auteurs proposent des dimensions qualitatives qui sont liées à :

1. la prise en charge du modèle de communication souhaité (pub-sub et / ou demande-réponse).
2. les exigences d'application IoT.
3. la viabilité et les limitations possibles dans chaque scénario.
4. la disponibilité et clarté de la documentation, ainsi que les didacticiels disponibles et la qualité de l'assistance.

Les auteurs ont utilisé des données à grande échelle pour simuler le scénario des villes intelligentes, ils ont comparé les deux fameux middlewares FIWARE (une solution open-source qui introduit un ensemble universel de normes pour la gestion des données contextuelles dans le but de faciliter le développement de solutions intelligentes) et oneM2M (un projet de partenariat mondial dont le but est de créer une norme technique mondiale d'interopérabilité concernant l'architecture, les caractérisations API et les solutions de sécurité pour les technologies Machine-to-Machine).

Les auteurs ont identifié les lacunes lors des implémentations et ils ont aussi caractérisé les variations en termes de performance des deux middlewares (intergiciels).

1.6 Travaux IoT basés sur MQTT

MQTT (Message Queuing Telemetry Transport) est un protocole très utilisé dans le domaine de l'IoT à cause de sa légèreté, et sa simplicité d'intégration.

1.6.1 Dissémination des données lourdes sur des brokers MQTT hétérogènes

Dans l'article (Ryohei Banno,Jingyu Sun, 2017), les auteurs proposent une nouvelle architecture dans laquelle il existe une couche d'interconnexion des brokers MQTT distribués, qui permet aux plusieurs brokers MQTT de coopérer. La solution est divisée en deux algorithmes de coopération (Publication Flooding PF et Subscription Flooding SF), qui fournissent les fonctions spécifiques du MQTT telles que QoS et Retain.

Publication Flooding (PF) est une méthode qui permet de partager chaque message PUBLISH entre tous les noeuds à travers une couche d'interconnexion des courtiers MQTT distribués (ILDM), chaque nœud ILDM relaie un message SUBSCRIBE reçu d'un client local à son courtier local. Concernant un message PUBLISH, un nœud ILDM transfère également à ses nœuds ILDM voisins et à leur propre courtier local.

Subscription Flooding (SF) est une méthode qui permet de partager les informations de souscriptions entre les nœuds ILDM. Lorsqu'un nœud ILDM reçoit un message SUBSCRIBE, il informe les informations d'abonnement, par exemple, le nom du sujet et le niveau de QoS à ses nœuds ILDM voisins, ainsi il relaie le message à son courtier local. Cette opération est similaire à l'opération d'abonnement du protocole MQTT entre deux nœuds ILDM.
Afin d'évaluer la solution, les auteurs ont formulé une méthode de benchmarking qui peut être utilisée pour un seul ou plusieurs courtiers.

1.6.2 MQTT et réseaux de capteurs sans fils

Le réseau de capteurs sans fils est un grand domaine de recherche, présente une solution attirante dans différents domaines tels que le transport, l'environnement, l'automatisation de l'industrie, la surveillance environnementale, etc. La plupart de ces applications nécessitent un transfert de données dans des infrastructures réseau traditionnelles. Dans ce contexte, les auteurs de l'article (Urs Hunkeler,Hong Linh Truong, 2008) présentent MQTT-S, un protocole dédié pour les réseaux de capteurs sans fil basé sur le protocole topic-based MQTT. La solution est conçue de telle manière qu'elle peut être exécutée sur un capteur/actionneur qui fonctionne sur batterie et sur des contraintes de bande passante telles que les réseaux Zigbee. MQTT-S est implémenté et testé dans des réseaux de capteurs sans fil de IBM. MQTT-s offre trois messages de connexions (CONNECT), alors que MQTT offre juste une seule, les deux messages de connexions additionnels sont dédiés pour le transport des messages Will et WillTopic explicitement.

1.6.3 Intergiciel MQTT pour les applications déployées à la périphérie des réseaux

Le middleware publish/subscribe est une technologie populaire qui facilite la communication appareil à appareil dans des scénarios Internet des Objets à grande échelle. Dans l'article (Thomas Rausch,Stefan Nastic, 2018) les auteurs présentent EMMA (Distributed QOS Aware MQTT Middleware for Edge Computing Applications), qui propose un protocole de surveillance continuelle du réseau qui permet de détecter la proximité en se basant sur la latence du réseau. La solution propose aussi un mécanisme d'orchestration d'un réseau de passerelle client et brokers MQTT, et un schéma de reconfiguration réseau afin d'optimiser la qualité de service durant l'exécution en se basant sur la proximité des noeuds.

L'architecture de EMMA est basée sur quatre composants principaux, à savoir la passerelle, les brokers, le contrôleur, et le protocole de surveillance réseau. En effet les clients MQTT se connectent aux passerelles qui jouent le rôle d'un proxy inversé pour la connexion dynamique des clients et des brokers. Les brokers implémentent le serveur MQTT et l'approche de bridging

dynamique. Le contrôleur agit en tant que registre pour la surveillance et l'orchestration du système.

Les résultats obtenus montrent que EMMA offre une communication à faible latence pour les appareils à proximité, tout en assurant une dissémination de messages dispersée géographiquement au cout minimum. Le mécanisme de reconfiguration réseau permet aux clients la mobilité, l'approvisionnement dynamique des courtiers (brokers) et l'équilibrage de la charge.

1.7 Plateforme de messagerie sur Edge

La plupart des applications IoT offrent des services de messagerie dans le but d'échanger des données et des informations. Ces messages doivent être livrés dans un certain délai et avec une certaine fiabilité. Dans ce contexte, plusieurs travaux visent à implémenter des plateformes de messagerie sur le Edge Computing.

1.7.1 Intergiciel orienté sur les applications de messagerie déployé sur le périphérie de réseaux

Dans l'article (Rausch, 2017) les auteurs ont introduit un middleware pub/sub orienté message (MOM) et basé sur le protocole MQTT pour les applications Edge Computing pour satisfaire les exigences des applications IoT en termes de latence, temps de réponse tout en tenant compte de certains défis tels que la consolidation de la communication géographiquement dispersée, la mobilité des clients, l'allocation dynamique des ressources, et la confidentialité des données. Cette solution intègre des mécanismes de surveillance de la QoS, détection de proximité, bridging (Pont) dynamique, reconfiguration réseau, provisionnement élastique des ressources et renforcement des politiques de confidentialités.

1.7.2 Système de courtier de messagerie à travers des plates-formes hétérogènes

Dans l'article (Sherif Abdelwahab, 2016), les auteurs proposent "FogMQ", un courtier (broker) de messagerie basé sur le cloud qui surmonte les limites des systèmes broker traditionnels en

activant la découverte autonome, le déploiement automatique et la migration en ligne des brokers de message à travers des plateformes cloud hétérogènes. FogMQ offre un service performant de clonage des appareils qui est souscrit aux messages des appareils et qui facilite l'analyse de données à proximité du edge afin de garantir un faible délai de transmission. Le système proposé garantit la réduction du délai de bout en bout, la latence exigée pour les applications IoT, la découverte autonome et la migration hétérogène tout en exploitant des contrôleurs des plateformes cloud déjà existants.

1.8 Travaux basés sur le paradigme pub/sub

Comme mentionné précédemment Publish/subscribe est un mécanisme très utilisé dans les services d'échanges de messages, en effet il permet de diminuer le nombre des consommateurs de messages puisque seuls les abonnés peuvent recevoir les messages, par la suite ce mécanisme permet de réduire la charge sur le réseau ainsi que sur les émetteurs de messages. Les approches mentionnées dans cette section portent davantage sur le paradigme topic-based pub/sub en général et ne réfèrent pas à un protocole spécifique tel que MQTT.

1.8.1 Introduction aux systèmes pub/sub

Les paradigmes publish/subscribe gagnent de plus en plus d'attention dans les applications d'interaction distribuées à grande échelle. En effet dans un système publish/subscribe, les souscripteurs enregistrent leurs intérêts pour un certain évènement pour qu'ils soient notifiés dès que les publishers publient un événement correspondant. Dans l'article (Patrick Eugster, 2003), les auteurs exposent les points communs sous-jacents au découplage complet des entités communicantes dans le temps, l'espace et la synchronisation. Ils ont exploité ces trois dimensions de découplage pour mieux identifier les points communs et les divergences avec les paradigmes d'interaction traditionnels.

Les auteurs ont présenté les paradigmes de communications alternatifs tels que le passage des messages (message passing), les invocations à distance (Remote Procedure Call), les notifications,

ainsi la file d'attente des messages. Ensuite les auteurs ont introduit les types de communications publish/subscribe tels que topic-based, content-based, type-based, puis quelques difficultés d'implémentation en tenant compte des invocations, des architecutres et les types de messages, enfin ils ont présenté les qualités de service offertes par les systèmes publish/subscribe à savoir la persistence, les transactions, l'efficacité et la gestion des priorités.

Les nombreuses variantes sur le thème de la publication / souscription sont classées et synthétisées. En particulier, leurs avantages et inconvénients respectifs sont examinés à la fois en termes d'interfaces et d'implémentations. Le routage d'évènements distribué est considéré comme une technologie clé pour la dissémination évolutive des informations, son but principal étant de réduire les frais de réseau et de calcul par évènements de diffusion.
L'article (Roberto Baldoni,Leonardo Querzoni, 2005) présente une architecture publish/subscribe qui fait la décomposition entre la couche de routage basée sur les évènements, la couche infrastructure overlay, ainsi que la couche des protocoles réseau. L'article a introduit d'abord les éléments du système publish/subscribe et leurs rôles à savoir (publishers, subscribers...), puis les modèles de souscription (topic-based, content based, type, based), ensuite le modèle architectural dans lequel les différentes couches (protocoles réseau, infrastructure Overlay, routage d'évènement et le matching) existent. Finalement, les auteurs discutent des enjeux concernant le routage d'évènement et finalement ils définissent quelques systèmes pub/sub représentatifs.

1.8.2 Applications IoT basés sur les systèmes pub/sub

La solution proposée dans l'article (Cenk Gündoğan,Peter Kietzmann, 2018) consiste à développer un schéma publish/subscribe robuste pour les scénarios IoT qui vise les réseaux IoT dont les ressources des équipements sont contraintes. La solution permet de limiter les tables FIB (Forwarding Information base : Une table dynamique qui mappe les adresses MAC aux ports), supporte la mobilité, le partitionnement réseau, l'agrégation des données et la réactivité temps réel. L'évaluation de la solution a été réalisée dans des conditions réelles en variant le nombre des appareils contraints qui sont interconnectés via le protocole sans fil 802.15.4 LoWPANs.

Dans l'article (Sven Akkermans,Rafael Bachiller, 2016) les auteurs proposent un framework qui intègre un middleware pub-sub et la multidiffusion Ipv6 pour réduire les frais de communication. En effet la solution proposée relie les groupes d'abonnés dans la couche application avec les groupes de multidiffusion Ipv6 dans la couche réseau en se basant sur des protocoles de routage dédiés pour les réseaux à faible puissance et avec des pertes(RPL). La solution présente des résultats pertinents en termes de bande passante et de consommation énergétique lors de l'évaluation. Par exemple la consommation de la bande passante a été réduite jusqu'au 54% pour 10 abonnés et jusqu'au 66% pour 20 abonnés.

Les auteurs de l'article (Alex C. Olivieri,Gianluca Rizzo, 2015) proposent une approche publish/subscribe évolutive pour l'intégration de différents protocoles et technologies hétérogènes dans l'IoT. Les auteurs ont implémenté et évalué cette solution dans le scénario d'un bureau intelligent.
D'abord la solution proposée permet d'évoluer facilement avec l'augmentation de nombre des appareils IoT car elle est basée sur un framework résidant dans le Cloud, ensuite elle fournit une couche d'abstraction avec les détails des protocoles de communication qui permet aux entités de se concentrer sur les tâches de la couche application qu'ils doivent accomplir pour un scénario spécifique. Enfin elle offre un activateur unique pour une entité spécifique qui lui permet de communiquer avec les technologies qui sont déjà connectées grâce au framework.

Les systèmes de messageries pub/sub basés sur les topics sont très importants pour développer des applications IoT, cependant dans ces applications il faut prendre en compte la scalabilité et la sensibilisation à la localité afin de garantir un faible délai pour livrer les messages et une exploitation efficace des ressources réseau.
Dans ce contexte, les auteurs de l'article (Yuuichi Teranishi,Ryohei Banno, 2015) proposent une nouvelle méthode de superposition TBPS appelée "Skip Graph based TBPS with Locality Awareness(STLA)" qui étend les solutions de messagerie TBPS (Topic based pub/sub) basées sur l'algorithme "Skip Graph" pour ajouter la sensibilisation à la localité. Les résultats de simulation montrent que la solution permet d'atteindre la sensibilisation à la localité et réduit la latence

pour la dissémination des messages. De plus les auteurs ont effectué les expériences sur des centres de données réels pour affirmer la faisabilité de la solution en pratique.

Dans le travail de recherche (Saida, 2016), l'auteur propose un nouveau système de messagerie dynamique, stable, et efficace pour les réseaux véhiculaires appelés DYMES, qui offre des interactions en temps réel entre les voyageurs tout en respectant leurs anonymats et le dynamisme de leurs informations contextuelles.
Cette recherche est basée sur une stratégie de publish/subscribe appelée DPSCS (Dynamic Publish/Subscribe Clustering Strategy) qui permet à chaque voyageur de créer une communauté selon son propre intérêt . La propagation de la publication se fait de manière fiable et efficiente en choisissant les relais qui possèdent un lien stable avec le noeud publisher. Les résultats obtenus montrent que l'algorithme publish/subscribe (DPSCS) implémenté est très performant et a permis de créer des communautés de réseaux sociaux véhiculaires en millisecondes.

1.9 Modélisation de systèmes de pub/sub basés sur la théorie des graphes

La théorie des graphes est une discipline informatique et mathématique qui étudie des modèles sous forme de graphes et des dessins réseau qui interconnectent des objets. Dans ce contexte, plusieurs chercheurs ont implémenté cette théorie pour développer leurs solutions et modéliser leurs systèmes pub/sub. Dans l'article (Chen Chen,Hans-Arno Jacobsen, 2016), les auteurs ont proposé un algorithme MinAvg-TCO qui permet d'exploiter un minimum nombre de noeuds edges pour construire "topic-connected overlay (TCO)" afin de regrouper les noeuds qui s'intéressent au même topic et les connecter directement à un sub-overlay. La solution proposée offre un algorithme qui permet de joindre multiple TCOs dynamiquement pour faire face aux problèmes de surcharge dans les communications pub/sub.

La conception de l'overlay des systèmes pub/sub est l'une des plus importantes puisqu'elle affecte de façon directe la performance du système. Dans ce contexte, les auteurs de l'article (Vitenberg, 2015) se concentrent sur le problème MinAvg-TCO qui permet d'utiliser le nombre minimal d'appareils Edge pour construire un TCO (topic connected overlay) et organiser les noeuds qui

sont intéressés au même topic dans un sub-overlay directement connecté. Les algorithmes qui existent souffrent de quelques inconvénients tels que le cout élevé de l'exécution, centralisation des opérations, et exécution des opérations très basiques.
L'algorithme développé par les auteurs permet de faire face à ces problèmes de joindre plusieurs TCO (topic connected overlays) inspiré par les algorithmes de type divide-and conquer. Les résultats obtenus montrent que la solution proposée offre un équilibre entre l'efficacité du temps et le nombre d'appareils Edge, ainsi elle permet de gagner en termes de coût comparé au coût d'une solution naïve (algorithme glouton "Greedy").

1.10 Architectures pair à pair basées sur le paradigme pub/sub

Pair à pair est un modèle d'échange de messages ou de donnés dans lequel chaque élément du réseau peut être client ou serveur. Plusieurs recherches ont eu recours à ce modèle pour implémenter des mécanismes publish/subscribe.

1.10.1 Infrastructure pair à pair décentralisée pour les systèmes pub/sub

Dans l'article (M. Castro,P. Druschel, 2002) les auteurs présentent une infrastructure multicast décentralisée au niveau applicatif qui permet de gérer une grande échelle de noeuds connectés à l'aide d'un substrat de routage, pair à pair évolutif et auto-organisé (Pastry).
La dissémination des messages est basée sur la qualité de service "best-effort", mais la solution est ouverte pour implémenter des modèles de fiabilités plus forts facilement.

Les systèmes de notifications dans les réseaux sociaux sont basés sur le paradigme pub/sub, en effet les souscripteurs d'un certain contenu ou topic reçoive les notifications dès qu'elle est publiée. Dans ce contexte, le travail de recherche dans l'article (Nuno Apolonia,Stefanos Antaris, 2018) propose "SELECT", un système pub/sub pour les notifications des réseaux sociaux basés sur une architecture pair à pair. SELECT organise les pairs sur une topologie en anneau suivant un algorithme d'établissement de connexion pair à pair selon la disponibilité des utilisateurs, ceci permet de réduire le nombre de sauts pour la propagation des messages. L'algorithme

présenté est une heuristique pour un problème NP hard qui fait la correspondance des la charge des graphes sur une infrastructure pair à pair structurée.
Les résultats des expériences montrent que SELECT réduit le nombre de noeuds relais jusqu'à 89% par rapport aux systèmes de notification pub/sub, ainsi la communication entre deux pairs connectés est réduite par au moins 64% de sauts.

1.10.2 Architecture pair à pair pour les systèmes pub/sub

Le language et le modèle de données des systèmes publish/subscribe diffèrent d'une application à une autre, cependant, tel que mentionné précédemment, le modèle topic-based est le plus connu et le plus utilisé. Dans ce contexte, les auteurs de l'article (Vinay Setty,Maarten van Steen, 2012) présentent POLDERCAST, une architecture topic-based pub/sub et P2P qui vise à assurer une diffusion sans relais, rapide et robuste sur un overlay extensible avec un minimum coûts de maintenance. POLDERCAST réalise un équilibre délicat entre ces propriétés contradictoires.

Les auteurs ont commencé par comparer les caractéristiques du POLDERCAST avec d'autres travaux P2P pub/sub à savoir Scribe, Visits, et SpiderCast. Ensuite ils ont discuté des enjeux en lien avec la dissémination des évènements basée sur les anneaux, puis ils ont dédié une partie pour le mécanisme de gestion des overlay composés de trois modules à savoir RINGS, VICINITY,CYCLON.

RINGS est un module qui gère les liens d'anneau, il vise à découvrir un successeur et prédécesseur du nœud pour chaque thème de son abonnement, et l'adaptation aux nouveaux successeurs / prédécesseurs dans les réseaux dynamiques. VICINITY, il s'agit d'un module responsable du maintien des liens aléatoires induit par intérêt choisi entre des noeuds qui partagent un ou plusieurs sujets. Concernant CYCLON, il s'agit d'un module qui gère les liaisons aléatoires, il conserve l'ensemble des abonnés connectés dans une seule partition, et il constitue une source de liens sélectionnés uniformément aléatoire de l'ensemble du réseau.

Finalement les auteurs ont fait une étude sur la gestion des désabonnements et ils ont évalué la solution POLDERCAST expérimentalement en utilisant des cas réels tels que le tracement des

charges de Facebook, Twitter, ils ont utilisé Sribe (Castro et al., 2002) comme base de référence dans un certain nombre d'expériences, la robustesse avec le respect de désabonnement des nœuds est évaluée par des traces du superpeer Skype. Les résultats expérimentaux corroborent toutes les propriétés ci-dessus dont les paramètres peuvent atteindre 10 000 nœuds, 10 000 rubriques et 5 000 sujets par nœud.

1.11 Aspect innovateur en recherche

Notre solution DynPubSub possède un aspect innovateur comparé aux différents travaux de recherches étudiés. En effet, notre solution permet d'équilibrer la charge entre les différents noeuds IoT déployés à la périphérie de réseau grâce au rebalancement dynamique et à l'infrastructure décentralisée adoptée. De plus notre solution vise à réduire la moyenne des latences des échanges de données entre les noeuds (Edge) afin de répondre aux besoins des applications critiques en termes de temps de réponse tout en respectant la contrainte de bande passante. Également notre solution permet d'adapter une librairie existante pour simplifier la tâche aux développeurs de migrer à notre solution et d'exploiter l'infrastructure décentralisée au lien d'utiliser un seul courtier central.

CHAPITRE 2

APPROCHE

Dans ce chapitre, nous présentons l'approche adoptée pour notre travail de recherche. En effet nous présentons les différentes questions de recherches ciblées par notre étude, les étapes suivies pour y répondre. Par la suite, nous présentons le modèle de la solution proposée, les contributions, les fondements mathématiques, et un aperçu des algorithmes implémentés dans la solution.

2.1 Questions de recherches

L'objectif de notre étude est de fournir une architecture peer to peer dédiée aux systèmes publish/subscribe orientés-sujet (topic-based) nommée DynPubSub qui réduit la latence des échanges de messages tout en respectant les contraintes des systèmes IoT qui sont nombreuses telles que la consommation énergétique, l'usage processeur, et la mémoire. Dans notre cas, nous considérons la bande passante comme contrainte principale. Particulièrement, ce travail a pour but de répondre aux questions de recherches suivantes :

1. QR1 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement distribué d'un ensemble de noeuds situés dans un environnement à courte portée (local) tout en respectant la bande passante comme contrainte ?
2. QR2 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement distribué d'un ensemble de noeuds situés dans un environnement à l'échelle nationale tout en respectant la bande passante comme contrainte ?
3. QR3 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement distribué d'un ensemble de noeuds situés dans un environnement à l'échelle mondiale tout en respectant la bande passante comme contrainte ?
4. QR4 : Est-ce que DynPubSub offre la scalabilité de point de vue théorique ?

Afin de répondre aux trois premières questions de recherches, nous avons effectué des expérimentations dans des environnements qui correspondent à la portée en question c'est-à-dire à l'échelle locale, nationale et mondiale. Puis nous avons comparé les résultats des latences

obtenues par notre solution DynPubSub à des déploiements de référence ("baseline") à savoir un courtier (broker) local et un courtier (broker) infonuagique.
Pour répondre à la quatrième question, nous avons testé la performance de notre solution pour un grand nombre d'appareils, et nous avons vérifié que la solution répond à la scalabilité.

2.2 Modèle du système

Dans cette section, nous modélisons le système, présenter l'objectif d'optimisation et présenter l'approche adoptée pour notre solution.

2.2.1 Contraintes du système

Nous avons défini les contraintes à tenir en compte pour le bon fonctionnement de notre système qui sont : la bande passante, et la performance (mémoire, CPU), nous avons considéré ces contraintes parce que nous visons les appareils intelligents des systèmes IoT. Dans notre système, nous négligeons les contraintes liées aux processeurs et à la mémoire parce que nous allons travailler sur des Raspberry Pi 3 qui ont des ressources computationnelles suffisantes pour l'implémentation de notre solution. Par la suite avons décidé de nous focaliser sur la bande passante comme contrainte principale puisque dans un système IoT, une consommation excessive de cette dernière peut mener à un surcharge et une dégradation de performance des noeuds IoT.

2.2.2 Convention de nommage

Afin de clarifier les différentes notions utilisées dans notre système, nous avons opté pour une convention de nommage qui a été adaptée de (Julien Gascon-Samson,Franz-Philippe Garcia, 2015). En effet dans les sections qui suivent, la lettre P fait référence au noeud publisher qui publie des messages sur un certain sujet (topic), la lettre S fait référence au noeud subscriber qui peut s'abonner à un ou plusieurs sujets (topics) et la lettre T fait référence à un sujet (Topic). Un message (transmis d'un publieur au broker, et du broker à un souscripteur donné) est désigné par la lettre M, un client qui peut être publisher ou subscriber est dénoté par la lettre N. Dans le

cas où notre système contient plusieurs serveurs publish/subscribe, on va utiliser la lettre H pour désigner un serveur publish/subscribe hébergé sur un noeud IoT/edge.

2.2.3 Modélisation des capacités des périphériques

La modélisation des capacités des périphériques aide à fixer les objectifs de la solution à implémenter puisqu'ils sont les paramètres clés de cette dernière. En effet ils représentent les contraintes du système. Cette modélisation est définie comme suit :

1. Soit N un ensemble de noeuds Edge tel que $N = [N_1, N_2, ...N_n]$ avec N_i le nième noeud Edge ;
2. Dans notre système, on veut réduire au maximum possible la latence moyenne pour l'ensemble des messages échangés entre les noeuds publieurs et souscripteurs des noeuds publieurs et souscripteurs L_{N_i,N_j} avec i l'indice du noeud publieur et j l'indice du noeud souscripteur ;
3. On veut aussi éviter de dépasser les bandes passantes entrantes $b^{in}_{N_i}$ et sortantes $b^{out}_{N_i}$ des noeuds Edge qui vont héberger les courtiers(brokers) avec i l'indice d'un noeud de l'ensemble N ;

2.2.4 Modélisation des sujets

La modélisation des sujets (topics) permet de simplifier la gestion des échanges de messages dans notre système. Nous considérons dans notre système un ensemble de topics $T = [T_1, T_2, ...T_n]$ et pour chaque sujet (topic), nous avons un ensemble de souscripteurs ($T_i : S = [S_1, S_2, ...S_n]$), publieurs ($T_i : P = [P_1, P_2, ...P_n]$), un ensemble de courtiers (brokers) $B = [B_1, B_2, ...B_n]$ et un ensemble de messages $M = [M_1, M_2, ...M_n]$ entre les publieurs-courtiers-souscripteurs.
Dans notre solution, nous voulons savoir à quel noeud courtier chacun des sujets du système doit être assigné, afin d'offrir la plus faible moyenne de latence possible pour les messages échangés tout en respectant les contraintes de bande passante.

2.2.5 Liste des paramètres

Le tableau 2.1 résume les paramètres exploités pour l'implémentation de notre solution.

Tableau 2.1 Liste des paramètres

Notation	Description
$B = [B_1, B_2, ..B_n]$	Liste des brokers disponibles dans notre système
L_{N_i,B_j}	Latence entre un noeud N_i et un broker B_j
$b^{in}_{N_i}$	Bande passante entrante d'un noeud N_i
$b^{out}_{N_i}$	Bande passante sortante d'un noeud N_i
$T = [T_1, T_2, ...T_n]$	Liste de tous les topics du système
$T_i : [S_1, S_2, ...S_n]$	Liste de tous les clients souscripteurs sur un topic T_i
$T_i : [P_1, P_2, ...P_n]$	Liste de tous les publieurs sur un topic T_i
$P_i : [N_b(M)]$	Le nombre total des messages d'un publieur P_i
$P_i : [Size(M)]$	Taille totale des message d'un publieur P_i

2.3 Génération de solution optimale

Dans cette section, nous discutons la complexité de résoudre notre problématique afin de générer une solution optimale. En effet nous définissons la complexité NP, avec les problèmes NP-complet et NP-difficile dans le but d'identifier la nature de notre problématique.

2.3.1 NP Complexité

Un problème appartient à la classe NP (Impagliazzo, 2011) (non déterministe polynomial) s'il est résoluble par une machine de Turing en temps polynomial. Si le temps de résolution d'un problème P est limité par un polynôme, il est aussi NP. La démonstration de l'exactitude de la solution peut être réduite à une vérification P si un problème est connu comme étant NP et sa solution est connue. La solution des problèmes de NP nécessite une recherche avancée si P et NP ne sont pas équivalents.

2.3.2 Problème NP-complet

En théorie de la complexité de calcul, un problème NP-complet est un problème de décision qui vérifie les propriétés suivantes :

1. il est possible de vérifier une solution efficacement (en temps polynomial) ; la classe des problèmes vérifiant cette propriété est notée NP.
2. tous les problèmes de la classe NP se ramènent à celui-ci via une réduction polynomiale ; cela signifie que le problème est au moins aussi difficile que tous les autres problèmes de la classe NP.

2.3.3 Problème NP-difficile

Un problème est dit NP-difficile si un algorithme pour le résoudre peut être traduit en un pour résoudre tout autre problème NP. Il est beaucoup plus facile de montrer qu'un problème est NP que de montrer que c'est NP-difficile. Un problème à la fois NP et NP-difficile est appelé un problème NP-complet. NP-difficile, en théorie de la complexité de calcul, est la propriété définissant une classe de problèmes qui sont informellement « au moins aussi difficiles que les problèmes les plus difficiles dans NP ». Dans ce contexte, notre travail de recherche peut présenter plusieurs solutions potentielles à la problématique ce qui peut provoquer une explosion combinatoire, ainsi plusieurs travaux ont montré que l'aspect dynamique de la reconfiguration de la couche de dissémination des messages dans un système publish/subscribe distribué est NP-Hard. En effet Dynatops (Ye Zhao,Kyungbaek Kim, 2013) effectue des calculs afin de déterminer une topologie optimale qui minimise les coûts de reconfiguration et les latences des événements de notifications, le problème a été divisé en sous-problèmes pour proposer un algorithme efficace pour le résoudre. (Nuno Apolonia, 2018) propose un algorithme pour résoudre le problème NP-hard lié à la propagation des messages et qui mappe la charge de travail dans une infrastructure peer to peer dans le but de minimiser le nombre de sauts (hops). Dans (Chen Chen,Yoav Tock, 2015) les auteurs proposent des approches pour étudier des overlays basés sur la localisation pour des systèmes pub/sub et concevoir des algorithmes pour ce problème NP-hard pour garantir meilleures performances. On peut par la suite conclure que

nous voulons résoudre un problème NP-hard dans notre modèle, car on cherche à établir des reconfigurations overlay dans une infrastructure peer to peer. Donc nous allons proposer des algorithmes heuristiques pour notre modèle.

2.4 Approches architecturales

L'objectif principal de la solution est de concevoir une implémentation MQTT distribuée pouvant tirer profit des noeuds edge pour la dissémination des messages basée sur les sujets (topics). Puisque MQTT est un protocole léger et très adapté aux applications IoT, cette solution va permettre aux clients de se souscrire et de publier dans une infrastructure pair à pair déployée à la périphérie (Edge). Cette implémentation offre un modèle de partitionnement et d'équilibrage de charge qui permet de déterminer sur quel noeud edge chacun des sujets (topics) sera hébergé en prenant en considération les contraintes de l'IoT à savoir la bande passante entrante et sortante des noeuds de périphérie (Edge). L'objectif d'optimisation de notre solution est la latence moyenne de l'échange de message. Dans ce contexte, nous nous avons exploré et considéré différentes approches architecturales et nous avons retenu la troixième.

2.4.1 Première approche

Dans la première approche, chaque noeud possède un Broker local qui permet de gérer les topics qui lui sont alloués par l'orchestrateur (Load Balancer). Le client qui exploite une librairie préexistante du protocole MQTT va soit publier ou consommer des messages. L'orchestrateur reçoit dans un intervalle de temps "t" (une minute) l'ensemble des paramètres pertinents à l'exécution de l'algorithme de balancement de charge (les tailles des messages à échanger, le nombre de publications et souscriptions), il ajuste la configuration actuelle lorsque requis car notre système est dynamique, ainsi il décide quel noeud va héberger quel sujet (topic). Cette approche permet que les publications et souscriptions puissent être envoyées à différents courtiers (brokers) selon un "plan". La difficulté de cette solution est liée à la modification du code d'une librairie MQTT déjà existante, puisqu'il faudrait intégrer l'ensemble des algorithmes de distribution de charge à la librairie.

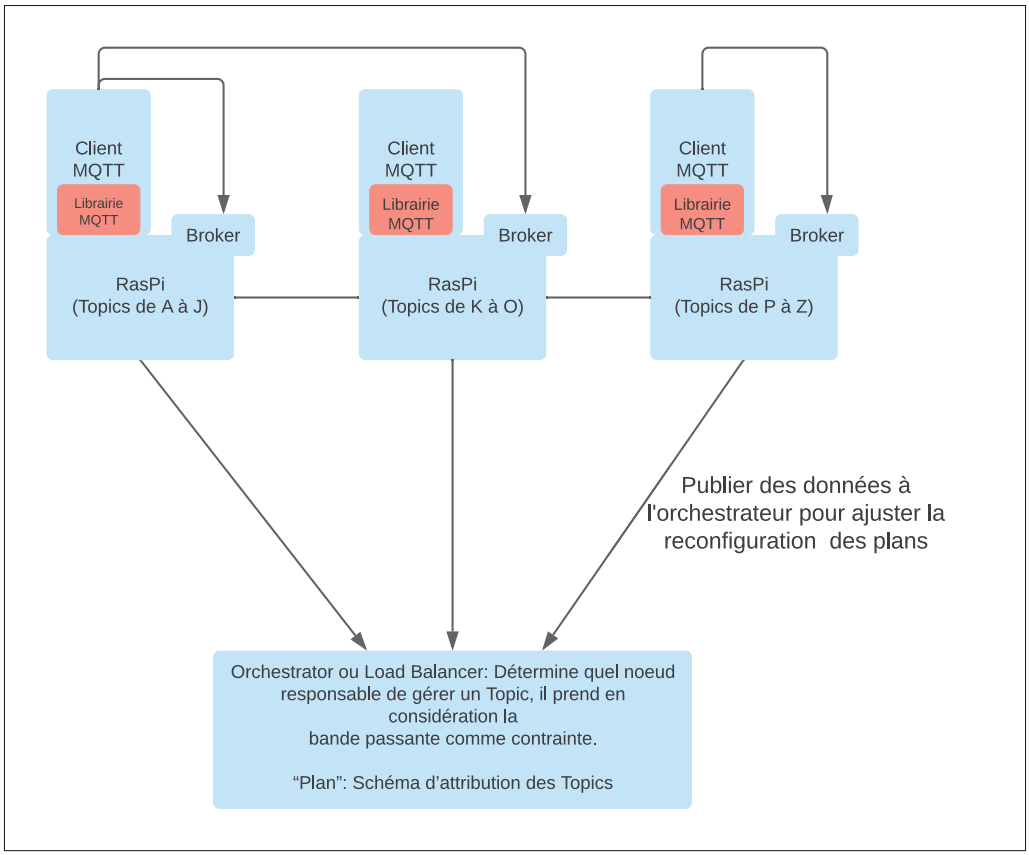

Figure 2.1 Première Approche

2.4.2 Deuxième approche

Dans la deuxième approche, nous intègrerons une librairie qui implémente le protocole MQTT et gère plusieurs brokers MQTT. En effet cette librairie reçoit les décisions de l’orchestrateur centrales sous la forme d’un plan selon les paramètres mentionnés ci-dessus à savoir la taille des messages, le nombre de publications et des souscriptions, les latences entre chaque noeud et tous les noeuds du système, pour qu’elle puisse par la suite rediriger les sujets (topics) vers les courtiers(brokers) responsables. La librairie doit être compatible avec le protocole MQTT et elle va créer une connexion MQTT à chaque courtier(broker). L’inconvénient de cette approche est que la librairie développée doit être capable d’accepter les requêtes publish/subscribe.

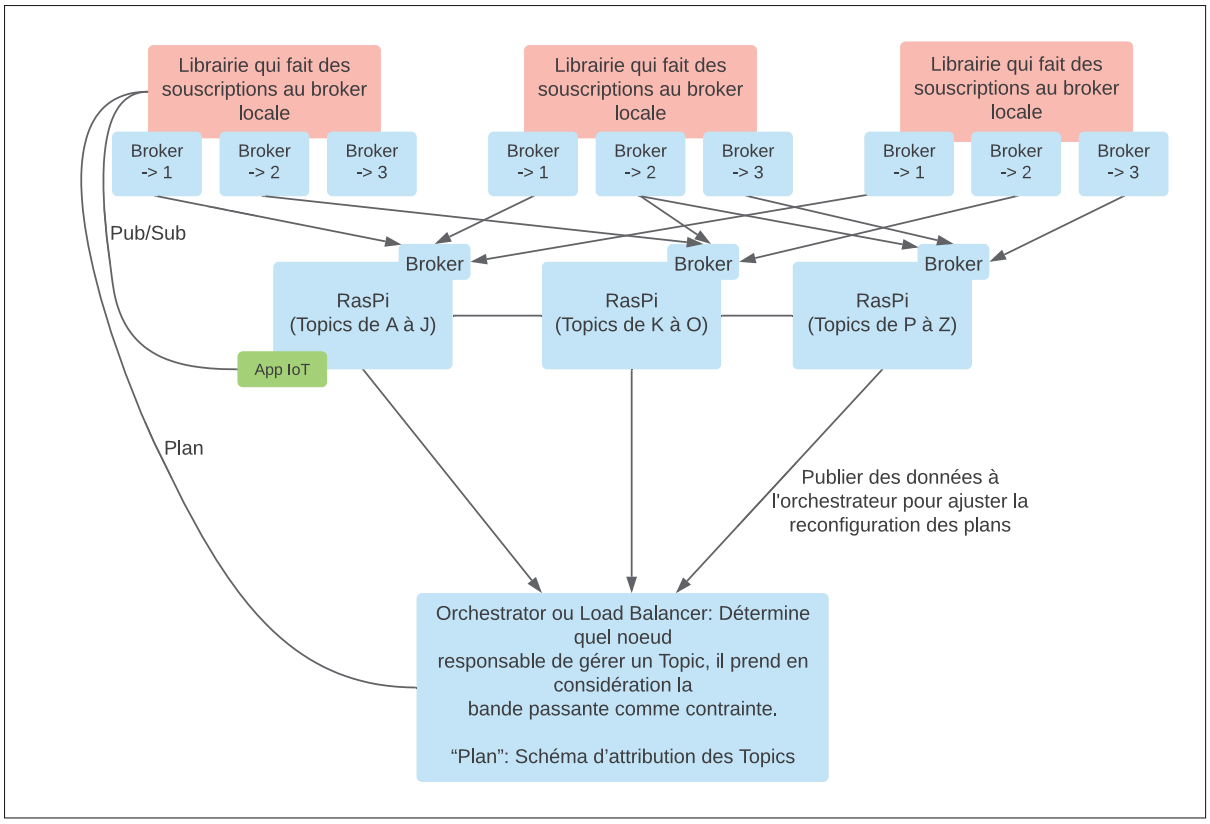

Figure 2.2 Deuxième Approche

2.4.3 Troisième approche

La troisième approche ressemble à la deuxième, cependant l'orchestrateur transmet ses décisions d'allocation des sujets (topics) à un courtier (broker) local qui à son tour transmet ces informations à la librairie responsable de rediriger les topics au brokers responsables. Cette librairie gère ainsi plusieurs brokers MQTT à la fois. L'application IoT établit une connexion MQTT standard aec le Broker local comme si elle communique avec un broker central. L'inconvénient de la deuxième approche est réglé dans cette approche puisque le Broker local est capable d'accepter les requêtes publish/subscribe, ainsi dans cette approche que nous n'avons pas besoin de modifier la librairie MQTT préexistante ce qui nous permet de garantir la possibilité d'utiliser d'autres implémentations MQTT (tant qu'elles respectent le standard MQTT) et la sécurité puisque nous pouvons filtrer les requêtes entrantes et sortante des neouds Edge.

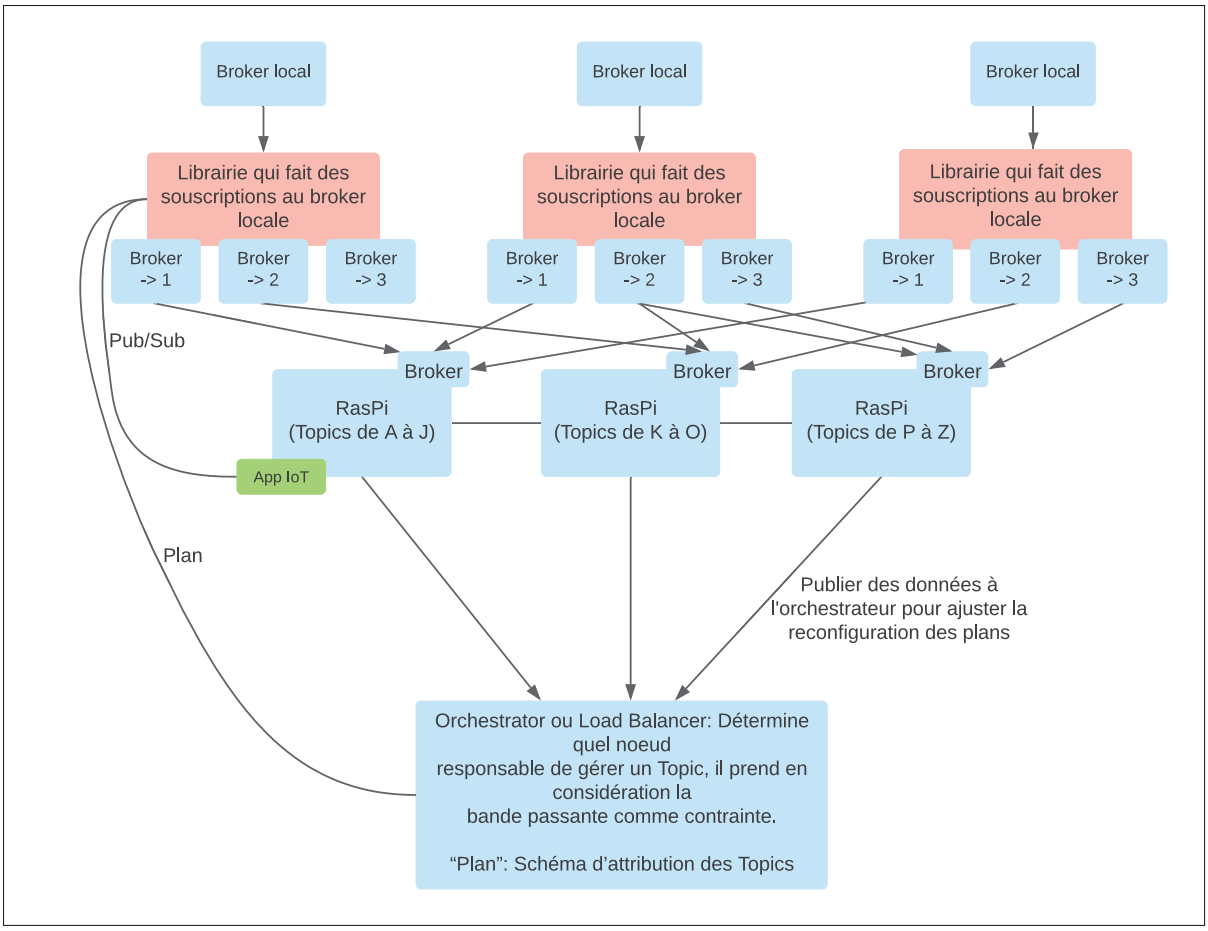

Figure 2.3 Troisième Approche

2.4.4 Aspect dynamique de la solution

Notre système doit être dynamique dans le sens où il peut s'adapter aux changements liés aux conditions actuelles de la communication entre les noeuds, par exemple si le nombre de messages à échanger et leurs tailles varient, ou bien les noeuds responsables de publications et de souscriptions changent, le système doit avoir la possibilité de s'auto-adapter et assigner automatiquement les noeuds qui doivent être responsables de disséminer les messages grâce à un système central qu'on va l'appeler "un orchestrateur". Cet aspect est plus intéressant pour les applications publish/subscribe et présente plusieurs avantages comparé à un aspect statique qui garde la même configuration réseau et peut mener à un surchage des noeuds Edge et une dégradation de leurs performances car il ne serait pas en mesure de s'adapter aux conditions qui

changent. Cependant, l'aspect dynamique amène à un ensemble de défis tels que la complexité de trouver la meilleure configuration, et le coût en ressources computationnelles et réseaux à prendre en considération lors des reconfigurations continuelles. Par la suite il faut également considérer le nombre et la fréquence des changements à la reconfiguration.

2.4.5 Problème d'optimisation

Le problème d'optimisation est utilisé dans les domaines des mathématiques, économies et informatiques afin de trouver la solution la plus optimale parmi un ensemble de solutions possibles. Il existe deux types de problèmes d'optimisations selon la nature des variables à savoir des variables discrètes ou bien des variables continues. Dans un problème d'optimisation avec des variables discrètes, nous recherchons un entier ou un objet ; par contre, dans un problème d'optimisation avec des variables continues nous recherchons les problèmes multimodaux et les problèmes contraints. L'idée principale des problèmes d'optimisation est de trouver une solution qui permet de maximiser ou minimiser une quantité particulière tout en respectant des conditions auxiliaires.

Dans notre cas nous souhaitons minimiser le temps de réponse (la latence) nécessaire pour la dissémination des messages en fonction de certaines variables à savoir le nombre de publications, le nombre de souscriptions, le nombre de sujets (topics), la liste des courtiers (brokers), la bande passante entrante ainsi la bande passante sortante.

1. Les variables sont : Le nombre de courtiers (brokers), le nombre des noeuds Edge, le nombre de sujets (topics), le nombre de publications et souscriptions ;
2. La quantité qu'on veut minimiser est le temps de réponse de dissémination des messages (latence) ;
3. La formule de la latence à minimiser a été simplifiée pour des raisons de lisibilité, elle est présentée en fonction des variables ;

$$L_{AVG} = \sum_{i=1,j=1}^{Nodes} \sum_{k=1}^{Brokers} [L_{N_i,B_k} + L_{N_j,B_k}] / \sum_{t=1}^{Topics} \sum_{p=1}^{nPub} \sum_{s=1}^{nSub} (T_t \times pubs_p \times subs_s) \qquad (2.1)$$

avec L_{AVG} est la latence moyenne qu'on veut minimiser,$Nodes$, le nombre de noeuds Edge, $Topics$ la liste des topics, $nPub$ nombre de publications, $nSub$ nombre de souscripteurs et $Brokers$ le nombre de courtiers (brokers) ;

4. Tout en considérant la contrainte de bande passante entrant avec la formule suivante :

$$b^{in}_{AVG} = \sum_{i=1}^{Nodes} b^{in}_{N_i}/Nodes < softConstraint \times hardConstraint \tag{2.2}$$

avec b^{in}_{AVG} est la moyenne de bande passante entrante pour tous les noeuds Edge, $b^{in}_{N_i}$ est la bande passante entrante pour un Noeud N_i, $Nodes$ le nombre de noeuds, $softConstraint$ la contrainte qui correspond au pourcentage du $hardConstraint$ (la valeur seuil à ne pas dépasser) ;

5. La contrainte de bande passante sortante est présentée par la formule suivante :

$$b^{out}_{AVG} = \sum_{i=1}^{Nodes} b^{out}_{N_i}/Nodes < softConstraint \times hardConstraint \tag{2.3}$$

avec b^{out}_{AVG} est la moyenne de bande passante sortante pour tous les noeuds Edge, $b^{out}_{N_i}$ est la bande passante entrante pour un Noeud N_i,$Nodes$ le nombre de noeuds, $softConstraint$ la contrainte qui correspond au pourcentage du $hardConstraint$ à ne pas dépasser ;

2.5 Architecture

Notre architecture (Figure 2.4) est composée de trois couches :

1. Couche orchestrateur : Il s'agit d'un noeud qui héberge l'orchestrateur, calcule et génère le plan optimal (discuté dans la section "Concept de modélisation") pour l'échange des messages ;
2. Couche intermédiaire : Constitue la couche d'infrastructure de dissémination des messages MQTT - elle est déployée sur les noeuds Edge ;

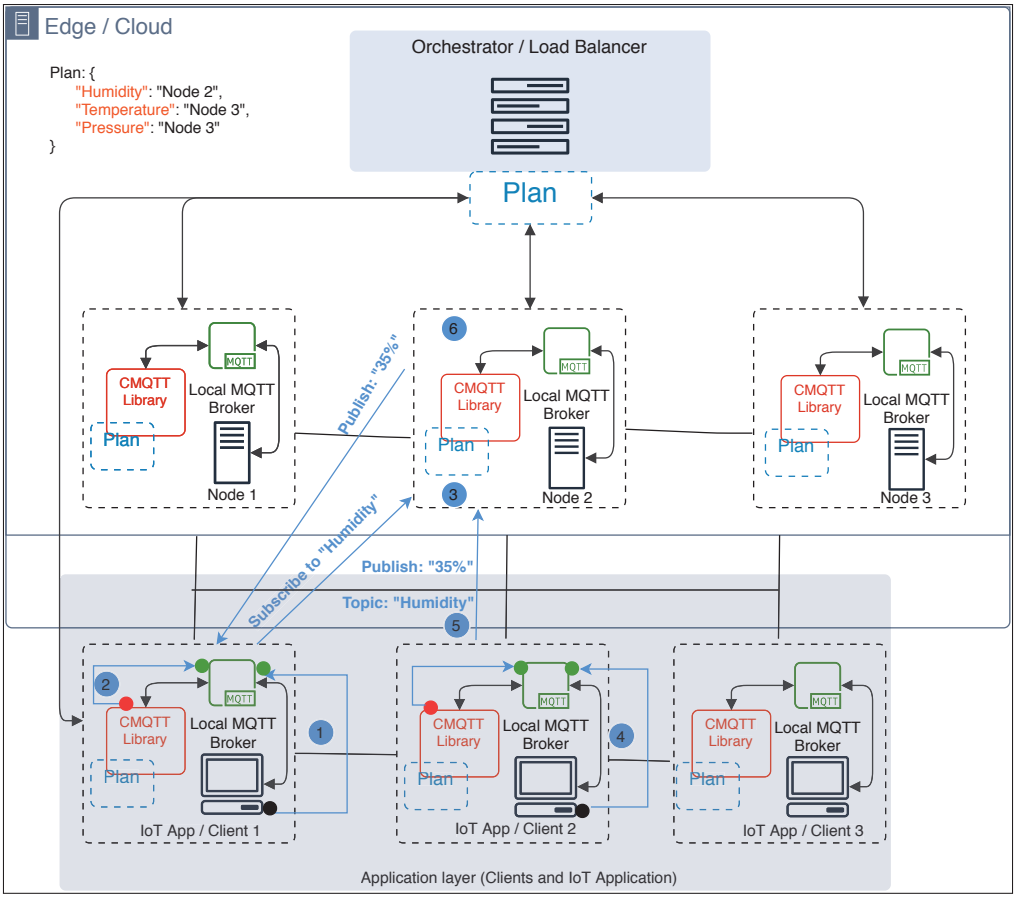

Figure 2.4 Architecture

3. Couche applicative : Une couche qui contient les noeuds qui hébergent des applications IoT (publications/souscriptions), chaque noeud expose son courtier (broker) local qui fait partie de la couche intermédiaire, cette couche possède la librairie CMQTT et une copie du plan ;

Les couches applicative et intermédiaire incluent les mêmes périphériques, mais elles sont découplées d'un point de vue de l'architecture logicielle. Les noeuds edge qui ont des capacités de traitement et de bande passante élevées peuvent être des courtiers (brokers) et des clients à la fois (par exemple dans un réseau de capteurs sans-fil les noeuds de capteur peuvent être des Raspberry Pi), cependant ces noeuds Edge peuvent être juste des clients (par exemple un microcontrôleur avec des ressources limitées connecté avec des capteurs).

2.5.1 Concept de modélisation

Dans cette section, nous définissons le concept de modélisation de la solution et plus précisément la notion du plan utilisé lors de l'implémentation de la solution, ainsi que les plus importants algorithmes implémentés pour l'orchestrateur.

Le plan est un objet JSON composé du nom de sujet (topic) et de l'adresse ip du broker qui l'héberge, pour chacun des sujets (topics) qui sont gérés dans notre système, comme le montre la figure de l'architecture (Figure 2.4). Le plan est dynamique et change à chaque rebalancement par l'orchestrateur afin de respecter la contrainte de bande passante et réduire la moyenne des latences d'échanges de messages. En effet, dans notre solution nous avons défini un paramètre appelé *number of migrations* qui sert à bien calibrer le compromis entre adapter le système afin de réduire la latence (tout en respectant les contraintes de bande passante), et éviter trop de changements de configuration simultanés pouvant impacter le système et qui sert à indiquer le nombre de topics qu'il faut les permuter et les allouer à d'autres *brokers*. Ceci nous a permis de tester plusieurs plans afin de trouver le plus optimal qui réduit la latence et respecte la contrainte de bande passante afin de le publier par la suite à tous les noeuds IoT pour mettre à jour leurs plans actuels et effectuent des publications et des souscriptions suivant le nouveau plan.

2.6 Algorithmes de l'orchestrateur

Comme mentionné, nous faisons face à un problème d'assignation des sujets (topics) à un ensemble de courtiers (brokers) qui est un problème NP-difficile lié à l'algorithme de sélection d'une configuration (plan) optimale en raison du nombre élevé de paramètres à gérer. Nous proposons une approche heuristique constituée d'un ensemble d'algorithmes qui permet d'approximer la solution optimale. Dans ce contexte, nous présentons dans les sections suivantes les principaux sous-algorithmes de l'orchestrateur.

2.7 Rebalancement dynamique

Comme mentionnée, notre solution garantit le rebalancement dynamique entre les différents noeuds déployés à la périphérie du réseau. En effet, à chaque génération d'un nouveau plan, l'orchestrateur vérifie si le trafic généré respecte la contrainte de bande passante des noeuds en calculant la bande passante estimée. Sinon, l'orchestrateur réalise une nouvelle itération. Le plan optimal généré est celui qui réduit la moyenne des latences d'échanges de messages.

2.7.1 Algorithme des bandes passantes estimées

Dans cette section, nous présentons l'algorithme qui permet de mesurer les bandes passantes entrantes et sortantes estimées. En effet, selon le trafic des messages échangés, cet algorithme permet d'estimer la bande passante entrante et sortante consommée par chaque noeud edge en fonction du plan des sujets (topics) courant.

Tableau 2.2 Liste des paramètres de l'algorithme 2.1

Paramètre	Description	Entrée/Sortie
$ipclient$	Adresse ip du client	Entrée
$currentPlan$	Le plan courant	Entrée
$listTopics$	Liste des topics exploités	Entrée
$estimatedIncomingBandwidth$	Bande passante entrante estimée	Sortie
$estimatedOutgoingBandwidth$	Bande passante sortante estimée	Sortie

Comme le montre l'algorithme 2.1 et le tableau 2.2, nous avons plusieurs paramètres en entrée à savoir l'adresse ip du client, le plan actuel et la liste des sujets (topics), et nous voulons estimer la bande passante entrante et la bande passante sortante pour une configuration donnée, afin de pouvoir essayer plusieurs configurations. Pour ce faire, nous parcourons la liste des topics et nous vérifions si le client est un courtier (broker) actif c'est à dire qu'il est responsable de gérer l'un des sujets (topics), si oui nous calculons le trafic entrant en excluant le trafic causé par le client en question et nous calculons le trafic sortant en multipliant la totalité du trafic lié au topic

Algorithme 2.1 Algorithme des bandes passantes estimées

```
1  Algorithme : Algorithme des bandes passantes estimées
   Input : ipClient, currentPlan,listTopics
   Output : estimatedIncomingBandwidth, estimatedOutgoingBandwidth

2  incomingTraffic ← 0 ;
3  outgoingTraffic ← 0 ;
4  for i ∈ {0,...,listTopics.length} do
5      if (currentPlan[listTopics[i]]===ipClient) then
6          totPubPerTopic ← getTotalPubTopic(listTopics[i],ipClient);
7          incomingTraffic ← incomingTraffic +
             totPubPerTopic.totMessagesSize;
8          totPubPerTopicIncludeClient ←
             getTotalPubTopicIncludeClient(listTopics[i]);
9          outgoingTraffic ← outgoingTraffic
             +((totPubPerTopicIncludeClient.totMessagesSize)
             *getTotalSubscribers(listTopics[i],ipClient)) ;
10     end if
11 end for
12 estimatedIncomingBandwidth ← incomingTraffic/1000 ;
13 estimatedOutgoingBandwidth ← outgoingTraffic/1000 ;
```

en question par la totalité des souscripteurs. Ensuite, nous divisons le trafic entrant et le trafic sortant par 1000 pour l'unité Ko car la taille des messages est en octets.

2.7.2 Algorithme de vérification de contrainte de bande passante

Dans cette section, nous présentons l'algorithme qui permet de vérifier que tous les noeuds IoT respectent la contrainte de bande passante. Cet algorithme est très intéressant parce qu'il parcourt tous les courtiers (brokers) et s'assure s'ils respectent la contrainte de bande passante selon une valeur seuil bien définie.

Comme le montrent l'algorithme 2.2 et le tableau 2.3, nous avons plusieurs paramètres en entrée à savoir *softConstraint*, le plan actuel, la liste des *brokers*, la bande passante entrante maximale, la bande passante sortante maximale et nous avons une valeur booléenne comme sortie qui

Tableau 2.3 Liste des paramètres de l'algorithme 2.2

Paramètre	Description	Entrée/Sortie
$softConstraint$	Contrainte soft en pourcentage	Entrée
$currentPlan$	Le plan courant	Entrée
$listBrokers$	Liste des courtiers (brokers)	Entrée
$maxIncomingBandwidth$	Bande passante entrante maximale	Entrée
$maxOutgoingBandwidth$	Bande passante sortante maximale	Entrée
$isExceeded(boolean)$	Variable de vérification	Sortie

Algorithme 2.2 Algorithme des bandes passantes estimées

```
1 Algorithme : Algorithme des bandes passantes estimées
  Input : softConstraint, currentPlan, listBrokers,
          maxIncomingBandwidth,maxOutgoingBandwidth
  Output : isExceeded

2 isExceeded ← false ;
3 foreach broker ∈ listBrokers do
4 |   {estimatedIncomingBandwidth, estimatedOutgoingBandwidth} ←
  |    getEstimatedBandwidth(broker.ipAddress,currentPlan) ;
5 |   if (estimatedIncomingBandwidth>maxIncomingBandwidth ∗ softConstraint
  |    ||estimatedOutgoingBandwidth>maxOutgoingBandwidth ∗ softConstraint
  |    ) then
6 |   |   isExceeded ← true ;
7 |   end if
8 end foreach
```

indique s'il existe un noeud IoT qui a dépassé la contrainte de bande passante. En effet nous parcourons la liste des brokers, ensuite nous mesurons la bande passante entrante et sortante correspondante, et nous vérifions par la suite si les mesures estimées ont dépassé les valeurs de bandes passantes maximales multipliées par la contrainte soft (*softConstraint*) qui représente la pourcentage de la valeur maximale de bande passante entrante ou sortante à ne pas dépasser.

2.7.3 Algorithme de permutation de topics

Dans cette section, nous présentons l'algorithme qui permet de permuter les topics entre les courtiers (brokers) qui sert par la suite à générer le plan optimal qui respecte la contrainte de bande passante et minimise la moyenne des latences. Cet algorithme est considéré comme l'algorithme principal pour notre solution, car il permet d'équilibrer la charge entre les noeuds edge.

Tableau 2.4 Liste des paramètres de l'algorithme 2.3

Paramètre	Description	Entrée/Sortie
$numberMigrations$	Nombre de migrations de topics	Entrée
$currentPlan$	Le plan courant	Entrée
$listBrokers$	Liste des courtiers (brokers)	Entrée
$trials$	Nombre d'essais pour trouver un nouveau plan	Entrée
$listTopics$	Liste des topics exploités	Entrée
$softConstraint$	Contrainte soft en pourcentage	Entrée
$optimalPlan$	Plan optimal	Sortie

Algorithme 2.3 Algorithme de permutation des topics

```
1  Algorithme : Algorithme de permutation des topics
   Input : numberMigrations, currentPlan, listBrokers,trials, listTopics, softConstraint
   Output : optimalPlan

2  for i ∈ {0,...,trials} do
3      for j ∈ {0,...,numberMigrations} do
4          randomTopic ← getRandomTopic(listTopics);
5          currentNewPlan ← permuteTopic(randomTopic,listBrokers,currentPlan);
6      end for
7      isExceeded ← checkExceededBandwidth(softConstraint,currentNewPlan);
8      if (isExceeded===false) then
9          candidatePlans.push(currentNewPlan) ;
10     end if
11 end for
12 optimalPlan ← getMinLatenctyPlan(candidatePlans)
```

Comme le montre l'algorithme 2.3 et le tableau 2.4, nous avons plusieurs paramètres en entrée à savoir le nombre de migrations de topics, le plan actuel, la liste de *brokers*, le nombre d'essais, la liste de topics, et la *soft Constraint*. Nous voulons déterminer le plan optimal qui permet de respecter les contraintes de bande passante et réduire le minimum possible la moyenne des latences. En effet selon le nombre d'essais et le nombre de permutations de sujets (topics), nous choisissons un topic aléatoirement et nous l'allouons à un courtier (broker) aléatoirement, ensuite nous vérifions si le nouveau plan respecte la contrainte de bande passante et nous l'insérons dans un tableau pour les plans candidats, enfin nous choisissions parmi les plans candidats celui qui minimise la latence moyenne des échanges de messages de noeuds IoT. Le nombre d'essais est un paramètre qui nous a permis de déterminer combien de fois l'orchestrateur effectuera des reconfigurations pour trouver le plan optimal, le nombre de permutations et un paramètre qui nous a permis de préciser combien de sujets (topics) à allouer aux différents courtiers (brokers).

CHAPITRE 3

IMPLÉMENTATION

Dans ce chapitre, nous présentons les différentes phases d'implémentation de notre solution. Nous avons décidé de consacrer tout un chapitre pour l'implémentation parce qu'il s'agit d'un morceau important du travail réalisé. En effet, nous présentons la librairie `cmqtt` qui sera compatible avec le protocole MQTT et la librairie mqtt.js, puisqu'il s'agit d'un requis de notre projet. Par la suite présentons les modules nécessaires à son implémentation et ses sous-algorithmes.

3.1 Outils de développement et modules

Dans cette section, nous présentons les différents outils et modules pour l'implémentation de notre solution.

3.1.1 Outils de développement

Pour la phase de la réalisation, nous avons eu un choix entre différents langages de programmation et plateformes logicielles à savoir NodeJs, Java, Python, cependant nous avons choisi NodeJs pour le développement de notre solution.

3.1.1.1 NodeJs

NodeJs est un environnement serveur *open source* multiplateforme pour l'exécution JavaScript, développé en 2009 par Ryan Dahl en C, C++ et JavaScript. Il est basé sur les événements et fonctionne sur le moteur V8 (Lakshmi Prasanna Chitra, 2017). L'exécution du code NodeJs se fait sur une seul *thread* donc toutes les requêtes reçues sont traitées par ce *thread*.
NodeJs est très adapté pour les applications asynchrones basées sur un paradigme événementiel qui permet d'optimiser le temps d'exécution.

NodeJs possède une API pour les opérations I/O, permet d'implémenter des serveurs HTTP et des opérations réseaux et possède un riche écosystème de modules tierces-parties. Ces modules peuvent être installés en utilisant le gestionnaire de paquets NPM et peuvent être étendus avec des paquets supplémentaires.

3.1.1.2 Comparaison NodeJs et Python

Tableau 3.1 Comparaison NodeJs et Python

NodeJs	**Python**
Node.js est le mieux adapté à la programmation asynchrone.	Python n'est pas la meilleure option pour la programmation asynchrone grâce à son modèle évenementiel à un seul thread.
Node.js utilise le langage JavaScript, donc ses bases restent simples à apprendre pour les développeurs qui bien souvent ont déjà une expérience en développement web.	Le plus grand avantage de l'utilisation de Python est que les développeurs doivent écrire moins de lignes de code.
Node.js est une plateforme idéale disponible actuellement pour traiter les applications web en temps réel.	Ce n'est pas une plateforme idéale pour traiter les applications web en temps réel.
La meilleure solution pour les activités nécessitant beaucoup de mémoire.	Déconseillé pour les activités nécessitant beaucoup de mémoire.
Node.js supporte le callback. Sa programmation est basée sur les événements et les rappels, ce qui rend son traitement plus rapide.	Il prend en charge les générateurs, ce qui le rend beaucoup plus simple.

Comme le montre le tableau 3.1, NodeJs est une meilleure solution pour les applications événementielles, le temps d'exécution des applications est plus rapide grâce au mode asynchrone supporté. De plus NodeJs est très riche en termes de disponibilités de librairies et modules. Pour ces raisons nous avons choisi NodeJs pour l'implémentation de notre solution et avons décidé d'offrir une compatibilité avec le module mqtt.js, très utilisé parmi les développeurs JavaScript.

3.1.2 Librairies NodeJs

Comme nous l'avons mentionné, l'une des principales raisons d'utiliser NodeJs pour l'implémentation de la solution était la disponibilité énorme des modules et librairies. Ainsi comme

nous l'avons mentionné, nous voulons que notre librairie CMQTT soit compatible avec la libraire mqtt.js de NodeJs. Dans cette section, nous présentons les principales librairies utilisées.

3.1.2.1 csv-writer

csv-writer (ryukn) est une librairie NodeJs *open source* installable à l'aide du gestionnaire des packets NPM qui convertit les objets et les tableaux JSON en un chaine de caractères de type CSV. Cette librairie nous a permis générer des fichiers csv à partir de nos applications. Nous avons généré les dataset pour les simulations de la moyenne de la latence pour les différents déploiements (local, à l'échelle nationale et à l'échelle mondiale). Ainsi nous l'avons également utilisée pour générer les résultats d'évaluation des bandes passantes, des latences et de la performance de la solution.

3.1.2.2 geo-ip-lite

geo-ip-lite (bluesmoon) est une librairie NodeJs *open source* installable à l'aide du gestionnaire des packets NPM qui retourne des détails géographiques tels que le pays, la région, la ville, le fuseau horaire, la latitude et la longitude à partir d'une adresse ip. Cette librairie nous a permis d'extraire les datasets pour les mesures de latences voulues selon la région et le pays. En effet, nous introduisons l'adresse ip du noeud IoT et nous vérifions si la mesure de la latence correspond au pays et/ou région en question à partir du dataset.

3.1.3 Outils logiciels Linux

Pendant les étapes d'implémentations, nous avons eu besoin de modules et librairies externes qui ont été inclus dans le code NodeJs. Parmi ces modules nous pouvons citer *vnstat* et *iperf*.

3.1.3.1 Vnstat

Vnstat est un outil léger installable sur les systèmes d'exploitation Linux. Il s'agit d'un moniteur de trafic réseau pour une ou plusieurs interfaces réseau sélectionnées. Vnstat nous a permis

d'obtenir la bande passante entrante et sortante actuelle d'un noeud IoT à partir d'une ligne de commande. Parmi les caractéristiques de Vnstat :

1. rapide et simple à installer.
2. les données peuvent être conservées lors du redémarrage du système.
3. peut surveiller plusieurs interfaces réseau en même temps.
4. faible utilisation du processeur.
5. résume les données selon la date (horaire, quotidien, mensuel, annuel, jours).
6. le format de la sortie *output* peut être choisi (JSON, text, png, console).

3.1.3.2 Iperf

Iperf est un outil léger de mesure active de la bande passante maximale pour une interface réseau donnée. Cet outil prend en réglage plusieurs paramètres à savoir les protocoles TCP/UDP, la version de l'adresse ipv4/ipv6. Iperf nous a permis de tester la capacité maximale de bande passante de nos différents noeuds IoT. Parmi les caractéristiques de iPerf :

1. multiplateforme (Windows, Linux, macOS X, Android...).
2. rapide et léger.
3. le client et le serveur acceptent plusieurs connexions simultanément.
4. le serveur utilisé est dynamique.
5. transmission bidirectionnelle.

3.2 MQTT Brokers

Afin d'assurer l'échange de messages, notre solution est basée sur le protocole de messagerie MQTT, notre solution est construite sur la couche de courtiers (brokers) tels que décrits à la section 2.4.3, et ne modifie pas le logiciel courtier (broker) utilisé, ce qui présente un avantage puisque le choix des courtiers (brokers) est donc interchangeable, il serait également possible théoriquement d'utiliser différents brokers sur différents noeuds. Dans ce contexte, nous avons eu l'opportunité de choisir entre plusieurs *brokers MQTT*. Parmi les plus utilisés nous pouvons citer *Mosquitto, Mosca, HiveMQ, RabbitMQT*.

Dans cette section, nous allons présenter les différents *brokers* et comparer entre eux et justifier notre choix.

3.2.1 Mosquitto

Mosquitto est un *broker MQTT open source* léger, multiplateforme et très populaire développé par Eclipse (Kitae Hwang,Jae Moon Lee, 2019). Il peut être déployé comme sur des serveurs puissants comme sur les appareils à faibles ressources physiques (CPU, mémoire). Mosquitto gère les sockets TCP des clients connectés et garde une liste de souscriptions des abonnés connectés ainsi que le *Topic* que chaque abonné attend. Mosquitto offre plusieurs avantages :

1. **disponibilité** : Mosquitto est portable et disponible pour téléchargement pour plusieurs plateformes.
2. **test et implémentation** : Mosquitto offre la possibilité de créer sa propre instance en quelques minutes, mais il offre aussi un broker public pour le test à l'adresse test.mosquitto.org.
3. **communauté** : il existe une large communauté pour Mosquitto à laquelle nous pouvons signaler des bugs et soumettre des modifications.

3.2.2 Mosca

Mosca est un *broker MQTT open source*, multitransports compatibles avec d'autres *brokers* tels que Mosquitto, RabbitMQ, ZeroMQ. Mosca peut être utilisé en mode standalone ou embarqué dans une application NodeJs. Mosca offre les avantages suivants :

1. échange rapide des données.
2. supporte les qualités de services QoS0 et QoS1.
3. peut être utilisé dans n'importe quelle application NodeJs.
4. options de stockage variées pour la qualité de service QoS1.

3.2.3 HiveMQ

HiveMQ est un *broker MQTT open source* et une plateforme de messagerie basée sur le client, développé pour assurer une connectivité efficace et fiable entre les appareils IoT. Il utilise le protocole MQTT pour un transfert bidirectionnel et instantané des données. HiveMQ offre plusieurs avantages :

1. création d'applications IoT fiables et évolutives, essentielles pour l'entreprise.
2. livraison rapide des données pour répondre aux attentes des utilisateurs finaux en matière de produits IoT réactifs.
3. faible coût d'exploitation grâce à la consommation optimale des ressources physiques et réseaux.
4. intégration des données IoT dans les systèmes industriels.

3.2.4 RabbitMQ

RabbitMQ est un *Broker open source* léger, très populaire et facile à implémenter localement ou au cloud. À part MQTT il supporte plusieurs autres protocoles de messagerie. RabbitMQ est multiplateforme qui peut être déployé selon une configuration distribuée pour réponde aux exigences de scalabilité et de haute disponibilité. RabbitMQ offre les avantages suivants :

1. supporte la messagerie asynchrone.
2. disponible pour tous les langages de programmation (Java, PHP, .Net, Python, JavaScript ...).
3. offre plusieurs outils et *plugins* pour les systèmes d'entreprises.
4. offre la possibilité de surveillance et gestion grâce à des Api-HTTP.

3.2.5 Comparaison entre les brokers MQTT

Dans cette section, nous présentons une comparaison entre les différents *brokers* mentionnés ci-dessus comme le montre le tableau 3.2

Tableau 3.2 Comparaison entre les courtiers (brokers) MQTT les plus utilisés
Tiré de Jaidip Kotak (2019, p3)

Broker	**Détails**	**Fonctionnalités**
Mosquitto	1. Supporte le protocole MQTT 3.1 et 3.1.1 ;	Supporte tous les QoS et les *Topics* dynamiques
Mosca	1. Supporte MQTT 3.1 et 3.1.1 ; 2. *Broker* MQTT pour les applications NodeJs ; *standalone* et embarquées.	Supporte Qos0 et Qos1, il est utilisation dans les applications NoodeJs.
HiveMQ	1. Supporte MQTT 3.1,3.1.1,3.5 ; 2. Basé sur *SDK* Java ; 3. Extensions pré-construits ;	Supporte tous les Qos, il est automatique et scalable
RabbitMQ	1. Supporte MQTT 3.1.1 ; 2. Peut être déployé avec bosh,docker, et Puppet ; 3. Compatible avec différents langages de programmation ;	Support tous les QoSn peut être déployé en tant ; que cluster pour un haut débit et une haute disponibilité ;

La figure 3.1 présente une comparaison entre les *brokers* mentionnés ci-dessus (Mosquitto, Mosca, HiveMQ, RabbitMQ) en termes de débit offert pour l'échange de messages. La figure montre que Mosquitto est le broker le plus performant suivi par Mosca, HiveMQ et RabbitMQ.

Dans ce contexte, nous avons choisi d'implémenter notre solution en utilisant Mosquitto en tant que *broker* MQTT grâce à sa haute disponibilité, se performance, sa simplicité d'implémentation et de large communauté disponible.

3.3 La librairie CMQTT

Dans cette section, nous présentons la librairie CMQTT qui permet d'assurer la connectivité optimale entre les noeuds, sa relation avec l'orchestrateur, ainsi ses différents sous-algorithmes. En effet, comme nous l'avons mentionné cette librairie est similaire à la librairie NodeJs préexistante "MQTT.js", elle présente les mêmes méthodes à savoir connect, publish, subscribe,

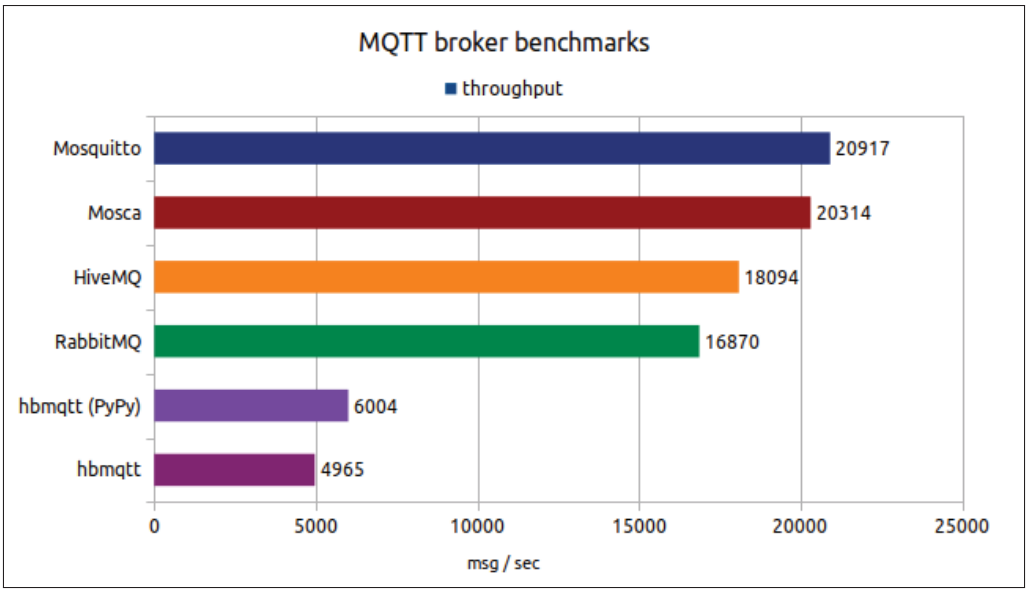

Figure 3.1 Comparaison entre les *Brokers*
Tirée de Mutsch (2020, p65)

unsubscribe, cependant elle permet de gérer plusieurs courtiers (brokers) et plusieurs noeuds Edge.

3.3.1 Algorithme de connexion et d'initialisation

Dans cette section, nous présentons l'algorithme qui permet de gérer les connexions des noeuds IoT et leurs initialisations pour communiquer avec l'orchestrateur.

Tableau 3.3 Liste des paramètres de l'algorithme 3.1

Paramètre	Description	Entrée/Sortie
$clientIp$	Adresse ip du client	Entrée
$listConnections$	Liste des connexions disponibles	Entrée
$eventEmitter$	Emetteur d'évennement	Entrée
$ipOrchestrator$	Adresse ip de l'orchestrateur	Entrée
$message$	Le message à publier	Entrée
$conection$	La connexion établie	Sortie

Algorithme 3.1 Algorithme de connexion et d'initialisation

Algorithme : Algorithme de connexion et d'initialisation
Input : clientIp, listConnections, eventEmitter,ipOrchestrator,message
Output : connection

if *(ipClient in listConnections===false)* **then**
    $localBroker \leftarrow$ mqttConnect($clientIp$);
    $orchestratorBroker \leftarrow$ mqttConnect($ipOrchestrator$);
    $listConnections[ipClient] \leftarrow localBroker$;
    ForwardMessages($localBroker, eventEmitter, message$)
end if
$connection \leftarrow listConnections[ipClient]$;

Comme le montrent l'algorithme 3.1 et le tableau 3.3, nous avons plusieurs paramètres en entrée tels que l'adresse ip du client, la liste de connexions, l'émetteur d'événements, l'adresse ip de l'orchestrateur ainsi que le message à transmettre. Nous avons comme sortie la connexion qui correspond à l'adresse ip du client.

À partir de cet algorithme, nous voulons vérifier si un client existe déjà dans notre liste de connexions selon son adresse ip, s'il n'existe pas nous allons créer la connexion à son *broker locale* et au *broker* de l'orchestrateur. Ensuite l'algorithme reste en écoute s'il y a une publication pour transmettre le message grâce à l'émetteur d'événements offert par NodeJs.

3.3.2 Algorithme d'envoi de données agrégées

Dans cette section, nous présentons l'algorithme qui nous permet d'envoyer les données agrégées qui correspondent aux différents noeuds IoT à l'orchestrateur pour qu'il puisse générer le plan optimal à partir de ces données.

Tableau 3.4 Liste des paramètres de l'algorithme 3.2

Paramètre	Description	Entrée/Sortie
$clientIp$	Adresse ip du client	Entrée
$orchestratorIp$	Adresse ip de l'orchestrateur	Entrée

Algorithme 3.2 Algorithme d'envoi de données agrégées

1 **Algorithme :** Algorithme d'envoi de données agrégées
Input : clientIp, orchestratorIp

2 $clientBandwidth \leftarrow$ getRealBandwidth() $cpuConsumption \leftarrow$ getCpuConsumption()
$orchestratorConnexion \leftarrow$ mqttConnect(orchestratorIp)
sendAggregatedData($clientIp$, $orchestratorConnexion$, $clientBandwidth$, $cpuConsumption$)

Dans cet algorithme 3.2 et dans le tableau 3.4, nous avons deux paramètres à savoir l'adresse ip du client et l'adresse ip de l'orchestrateur. Cet algorithme nous permet d'envoyer sur un intervalle de temps (chaque seconde) les données agrégées des noeuds IoT afin que l'orchestrateur exécute l'algorithme de génération du plan optimal.

3.3.3 Algorithme de publication

Dans cette section, nous présentons l'algorithme qui nous a permis publier les messages à l'aide de MQTT suite à l'appel de la fonction *publish*.

Tableau 3.5 Liste des paramètres de l'algorithme 3.3

Paramètre	Description	Entrée/Sortie
$topic$	Le topic auquel le client va publier	Entrée
$message$	Le message à publier	Entrée
$plan$	Le plan actuel	Entrée

Dans cet algorithme 3.3 et dans le tableau 3.5, nous avons trois paramètres à savoir le topic, le contenu de message et le plan. Cet algorithme nous permet de transférer le message à publier vers les souscripteurs de la même manière offerte par la librairie existante mqtt.js. En effet, nous vérifions d'abord s'il existe déjà un plan, ensuite nous identifions le *broker* responsable de gérer le topic passé en paramètre, ensuite nous établissions la connexion avec le *broker* responsable, si elle n'existe pas déjà - sinon elle est réutilisée et finalement nous transférons le message en question vers le topic demandé.

Algorithme 3.3 Algorithme de publication

```
1 Algorithme : Algorithme de publication
  Input : topic, message, plan
2 if (plan) then
3 |   ipBroker ← plan[topic]
4 |   clientConnexion ← getConnection(ipBroker)
5 |   publish(clientConnexion,topic,message)
6 end if
```

3.3.4 Algorithme de souscription

Dans cette section, nous présentons l'algorithme qui nous a permis de nous souscrire à un topic à l'aide de MQTT suite à l'appel de la fonction *subscribe*.

Tableau 3.6 Liste des paramètres de l'algorithme 3.4

Paramètre	Description	Entrée/Sortie
$topic$	Le topic auquel le client va se souscrire	Entrée
$plan$	Le plan actuel	Entrée

Algorithme 3.4 Algorithme de souscription

```
1 Algorithme : Algorithme de souscription
  Input : topic, plan
2 if (plan) then
3 |   ipBroker ← plan[topic]
4 |   clientConnexion ← getConnection(ipBroker)
5 |   subscribe(clientConnexion,topic)
6 end if
```

Dans cet algorithme 3.4 et dans le tableau 3.6, nous avons deux paramètres à savoir le topic et le plan. Cet algorithme nous permet de nous souscrire à un topic de la même manière offerte par la librairie existante mqtt.js. En effet nous vérifions d'abord s'il existe déjà un plan, ensuite

nous identifions le *broker* responsable de gérer le sujet (topic) passé en paramètre, ensuite nous établissions la connexion avec le *broker* responsable, si elle n'existe pas déjà - sinon elle est réutilisée et finalement nous effectuons la souscription au topic en question. Afin d'éviter la redondance, nous n'allons pas consacrer une section pour l'algorithme de désouscription parce qu'il est très similaire à l'algorithme de souscription.

3.4 Relation entre orchestrateur et CMQTT

Dans cette section, nous présentons la relation entre l'orchestrateur et la librairie CMQTT en détaillant les méthodes qu'ils offrent. Comme le montre la figure 3.2, un orchestrateur gère une

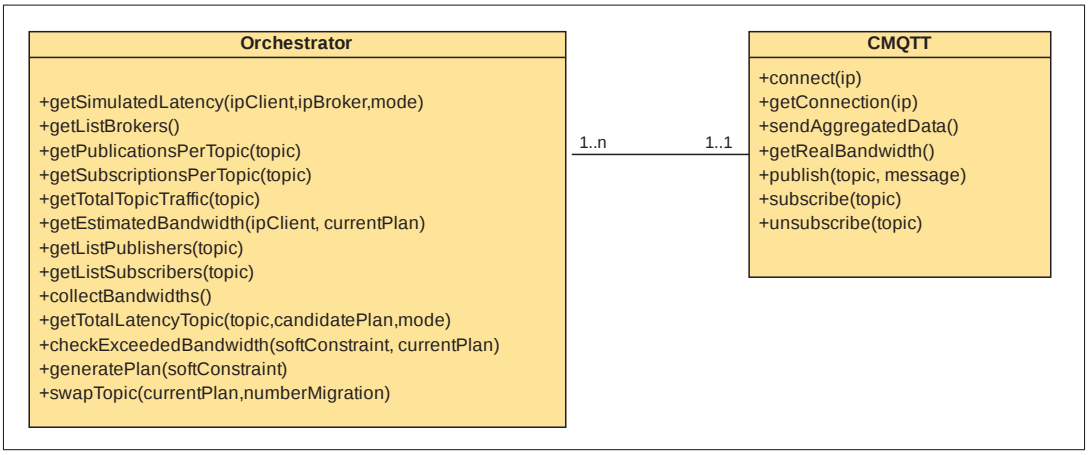

Figure 3.2 Diagramme de classes

ou plusieurs librairies CMQTT car ces dernières sont déployées sur tous les noeuds IoT. Par contre, une instance de la librairie CMQTT communique avec un seul orchestrateur centralisé qui génère les plans. En effet, l'orchestrateur est composé de plusieurs méthodes qui permettent de retourner la liste des publications, la liste de souscriptions, collecter les bandes passantes réelles et estimées, collecter les latences de communications, vérifier s'il existe un dépassement de contrainte de bande passante et générer le plan optimal. De l'autre côté, la librairie CMQTT offre aussi plusieurs méthodes qui permettent de connecter les noeuds IoT, transmettre leurs données agrégées, publier des messages sur un topic bien déterminé, se souscrire et désouscrire à un topic.

3.5 Similitude MQTT.js/CMQTT

Comme nous l'avons mentionné, un des objectifs principaux de notre solution est de garantir un similarité entre la libraire existante mqtt.js et notre librairie CMQTT, afin qu'un développeur qui maîtrise déjà la librairie mqtt.js puisse travailler aisément avec notre libraire. Il suffit simplement d'inclure la librairie (CMQTT), inclure l'adresse ip de l'orchestrateur au lieu d'un seul broker centralisé et appeler la méthode "connect", tout le reste du code demeure compatible.

Extrait 3.1 Publish/Subscribe CMQTT

```
1  var cmqtt=require("../cmqtt");
2  //imports the library
3  var client=cmqtt.connect("mqtt://192.168.1.15:1883");
4  //connects to the orchestrator
5  client.on('connect',function(){});
6  //uses connect callback
7  function publishHumidity(){
8      let hum=35;
9      client.publish("Humidity",hum.toString()+'%');
10 }
11 client.on('connect',function(){
12     client.subscribe("Humidity");
13     //subscribes to topic "Humidity"
14 });
15 client.on('message',function(topic,message){
16     console.log(message.toString());
17 });
```

Extrait 3.2 Publis/Subscribe MQTT.js

```
1  var cmqtt=require("mqtt");
2  //imports the library
3  var client=mqtt.connect("mqtt://192.168.1.15:1883");
4  //connects to the centralized broker
```

```
client.on('connect',function(){});
//uses connect callback
function publishHumidity(){
    let hum=35;
    client.publish("Humidity",hum.toString()+'%');
}
client.on('connect',function(){
    client.subscribe("Humidity");
    //subscribes to topic "Humidity"
});
client.on('message',function(topic,message){
    console.log(message.toString());
});
```

Comme le montrent les deux extraits ci-dessus, la librairie CMQTT est très similaire à la librairie MQTT.js, la seule différence est l'adresse du courtier (broker) et de l'orchestrateur, tout le reste est le même. En effet, cette exemple de code permet d'établir la connexion avec l'orchestrateur, puis connecter le client qui va publier des messages sur le topic "humidity". Ensuite, connecter le client qui va se désouscrire au sujet (topic) "humidity " pour recevoir et afficher les messages.

CHAPITRE 4

ÉVALUATION ET RÉSULTATS

4.1 Introduction

Dans ce chapitre nous allons présenter les résultats obtenus lors de l'évaluation de notre solution. En effet nous allons proposer une analyse sur la consommation de la bande passante pour s'assurer que notre système respecte vraiment les contraintes, ainsi nous allons proposer une évaluation en termes de latence d'échange de messages offerte sur trois déploiements différents à savoir déploiement local, à l'échelle nationale et à l'échelle mondiale. Finalement nous allons vérifier la mise à l'échelle de la solution proposée et son support d'échange de grand volume de données en analysant ses performances en termes d'usage processeur, consommation de mémoire vive et temps écoulé pour l'exécution de l'algorithme.

4.2 Environnement de l'évaluation

L'évaluation de la solution a été effectuée sur un cluster ThunderX CN8890 (96 Cpus @1.9 GHZ , 48 Cores d'architecture Arm), 128 Go de RAM avec Ubuntu 18.04.5 LTS 64 bits comme système d'exploitation, et sur des Raspberry Pi 3 connectés via Wi-Fi dont le processeur est Arm Cortex-A53 (4 Cpus @1.2GHz, 4 Cores) 1 Go Ram avec Ubuntu 18.04.5 LTS.

4.3 Paramètres de base de l'évaluation

Dans cette section, nous présentons les paramètres de bases de toutes les analyses (analyse, bande passante, analyse de latence, analyse de performance de solution).
Comme le montre le tableau 4.1 nous avons fixé la valeur du `hardConstraint` pour la bande passante consommée qui représente la capacité maximale de bande passante à 500 Ko/s pour suivre sa variation au cours du temps et ne pas se limiter à une faible valeur, ensuite nous avons fixé la valeur du `softConstraint` qui représente un pourcentage de la capacité maximale de bande passante à 70% pour être un peu proche du `hardConstraint`, il est

Tableau 4.1 Liste des paramètres de base

Paramètre	Description	Valeur
softConstraint	Contrainte soft en pourcentage	70%
hardConstraint	La valeur maximale (contrainte hard)	500 Ko/s
Taille des messages	Taille des messages à publier	500 Octets

nécessaire de limiter le `softConstraint` afin d'éviter de s'assurer à ne pas dépasser le `hardConstraint`. Finalement nous avons fixé la taille de messages à 500 octets pour tester notre solution et l'évaluer en fonction d'une charge de données importante. Les évaluations de latence et bande passante ont le temps comme axe X (abscisse), ceci est pour représenter le dynamisme dans le système, puisque notre orchestrateur raffine continuellement la configuration actuelle.

4.4 Analyse de la bande passante

Dans cette section, nous allons présenter les résultats obtenus lors de l'évaluation de la moyenne de la bande passante consommée sur chaque noeud (Raspberry Pi3) selon un déploiement local. Il n'est pas nécessaire de refaire la même analyse sur le déploiement à l'échelle nationale et mondiale parce que cela sera redondant étant donné que la distance n'a pas un impact sur la bande passante, mais plutôt sur la latence. L'objectif principal de cette analyse est de démontrer que la bande passante entrante et sortante consommée sur chaque noeud ne dépasse jamais la valeur seuil appelée `softConstraint`, ceci est important pour garantir que notre solution ne dépasse jamais la valeur `hardConstraint`, si l'orchestrateur détecte qu'un plan dont la moyenne des bandes passantes des noeuds Edge a dépassé cette valeur (`softConstraint`), il refait la permutation des sujets (topics) et ajuste la reconfiguration pour offrir un autre plan qui respecte la contrainte.

Nous avons effectué l'expérimentation sur 30 noeuds edge (Raspberry Pi) (`brokers`) avec 100 publications chaque seconde et 200 souscriptions sur 50 topics (topic0,topic1,.....topic49) dont la taille des messages est 500 octets comme le montre la figure 4.1 et le tableau 4.2. Nous avons déterminé une valeur maximale 500 ko/s comme `hardConstraint` dont 70% de cette

Tableau 4.2 Liste des paramètres d'analyse de la bande passante

Paramètre	Description	Valeur
Topics	Nombre de topics	50
Brokers	Nombre de courtiers (brokers)	30
Publications	Nombre de publications	100
Souscriptions	Nombre de souscriptions	200

valeur soit 350Ko/s est défini comme `softConstraint` pour les bandes passantes entrantes et sortantes.

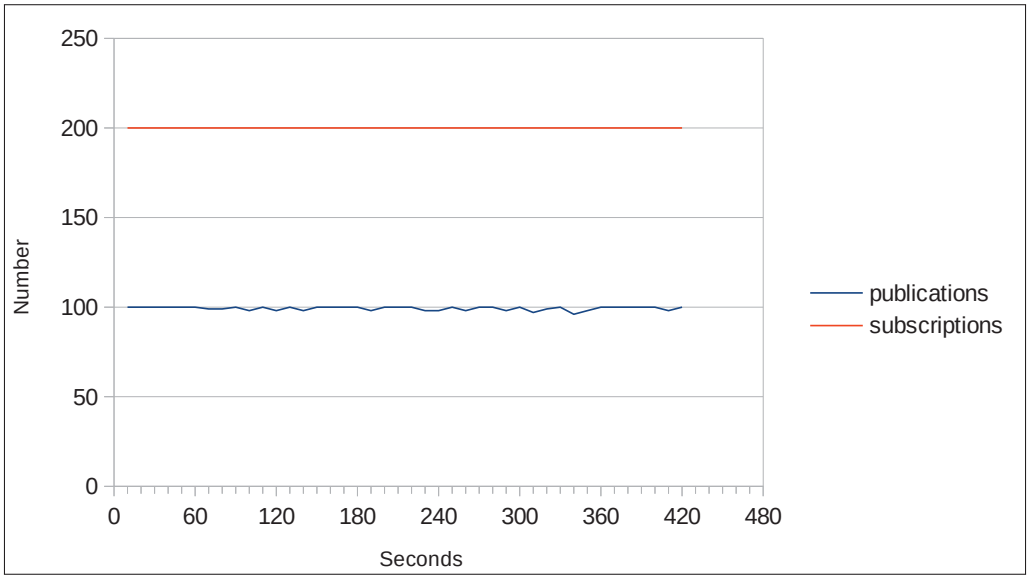

Figure 4.1 Nombre de Publications/Souscriptions

La figure 4.1 représente le nombre de publications et souscriptions effectuées sur 30 noeuds edge sur 50 sujets (topics) aléatoirement tout au long de l'évaluation de la consommation de la bande passante.

4.4.1 Analyse de la bande passante entrante

Dans cette section, nous présentons les résultats obtenus lors de l'évaluation de la consommation de la bande passante entrante des noeuds edge. En effet, nous voulons vérifier que notre solution

DynPubSub a réussi à équilibrer la charge entre les noeuds edge, afin que leurs consommations en bande passante entrante ne dépassent pas la valeur seuil `softConstraint`.

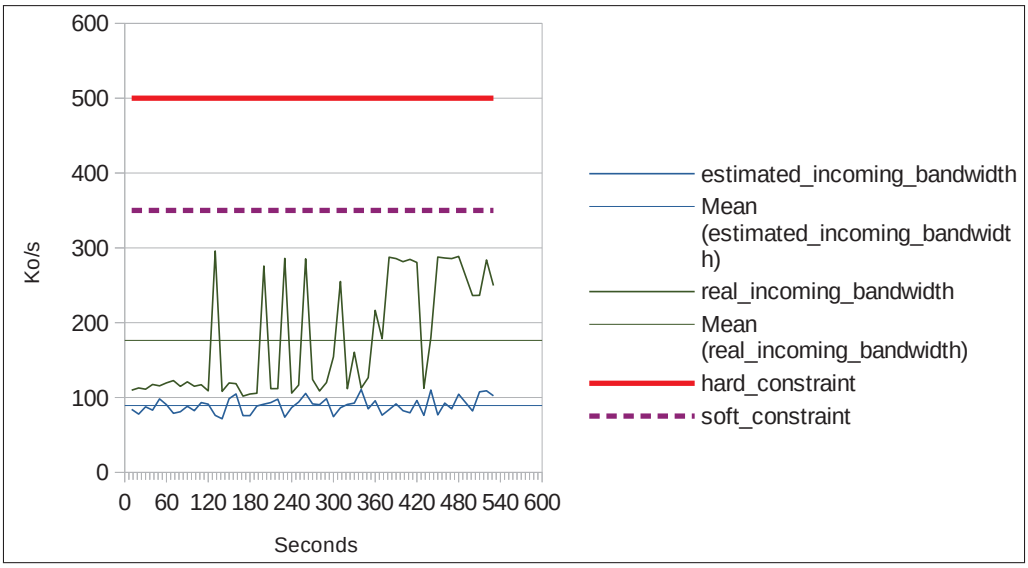

Figure 4.2 Evaluation de la bande passante entrante pour la moyenne des brokers

La figure 4.2 montre la consommation moyenne de la bande passante entrante des différents noeuds (30 Raspberry Pi's). En effet, elle représente la bande passante entrante moyenne estimée prédite par l'algorithme de génération de plan (`estimated_incoming`) et a bande passante entrante moyenne réelle (`real_incoming`) mesurée expérimentalement avec du vrai traffic généré. Ainsi les valeurs (`hard constraint`) et (`soft constraint`), les valeurs moyennes ont été représentées pour montrer le rapprochement des valeurs de bandes passantes réelles comparées aux estimées. Nous pouvons remarquer que la contrainte (`softConstraint`) a été bien respectée, toutes les valeurs moyennes de consommations de bande passante par tous les noeuds n'ont pas dépassé 350Ko/s ce qui montre que l'orchestrateur a bien réussi à équilibrer la charge de données entre les différents noeuds et notre solution DynPubSub nous a permis d'atteindre notre objectif concernant la bande passante entrante. La différence entre la moyenne des bandes passantes estimées et réelles est due aux entêtes rajoutées par les paquets du trafic réel, ainsi aux conditions réseau réelles.

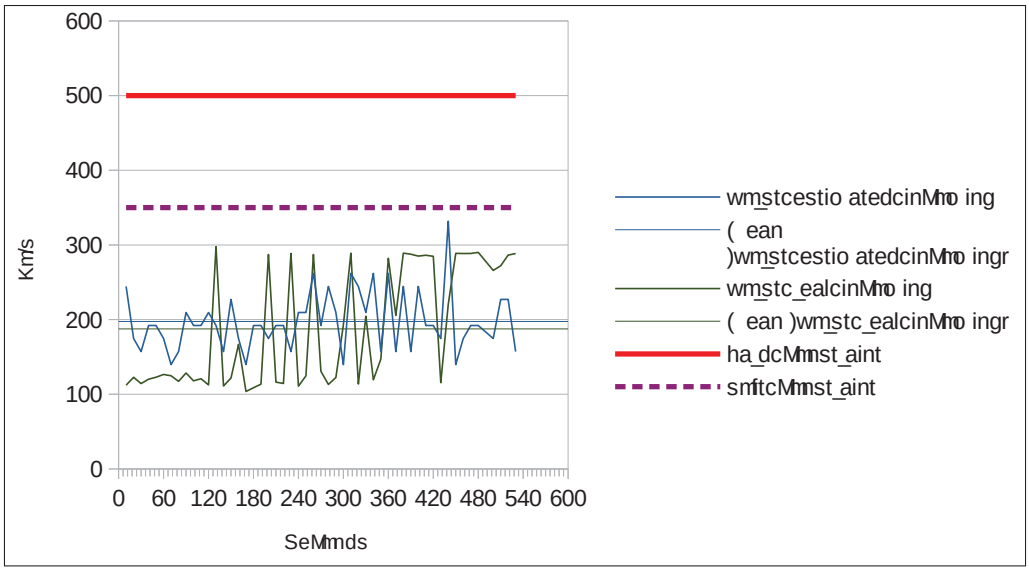

Figure 4.3 Évaluation de la bande passante entrante (pire cas)

Afin de nous assurer que tous les noeuds respectent la contrainte de la bande passante, nous présentons les consommations en bande passante entrante les plus élevées dans chaque noeud (pire cas), comme le montre la figure 4.3. En effet, elle illustre les consommations de bande passante entrante les plus élevées estimées (`worst_estimated_incoming`) et réelles (`worst_-real_incoming`), nous pouvons aussi remarquer que mêmes les mesures de consommations de bande passante entrante les plus élevées ne dépassent pas la (`softConstraint`) (350Ko/s), ce qui rassure que l'orchestrateur a bien réussi à équilibrer la charge entre les différents noeuds IoT (`brokers`).

4.4.2 Analyse de la bande passante sortante

Dans cette section, nous présentons les résultats obtenus lors de l'évaluation de la consommation de la bande passante sortante des noeuds edge. En effet, nous voulons vérifier que notre solution DynPubSub a réussi à équilibrer la charge entre les noeuds edge, afin que leurs consommations en bande passante sortante ne dépassent pas une certaine valeur seuil. Nous avons suivi la même méthodologie adoptée pour l'analyse de la bande passante entrante.

Dans la figure 4.4 nous présentons les résultats obtenus lors de l'évaluation de la bande passante sortante moyenne consommée par les noeuds IoT (Raspberry Pi3), elle représente la consommation moyenne de la bande passante sortante estimée (`estimated_outgoing`) et réelle (`real_outgoing`). La valeur moyenne de la bande passante sortante consommée par les noeuds Edge est aussi faible comparée à celle de la bande passante entrante parce que nous avons plus de souscriptions que de publications et nous gérons plusieurs sujets (topics). Les pics dans la courbe de la moyenne de bande passante sortante réelle s'expliquent par le phénomène du chargement par lot (file d'attente). La moyenne de la consommation de bande passante sortante réelle et estimée n'a pas dépassé la valeur *soft constraint* 350Ko/s ce qui prouve que l'orchestrateur a bien géré aussi la consommation de bande passante sortante comme la bande passante entrante.

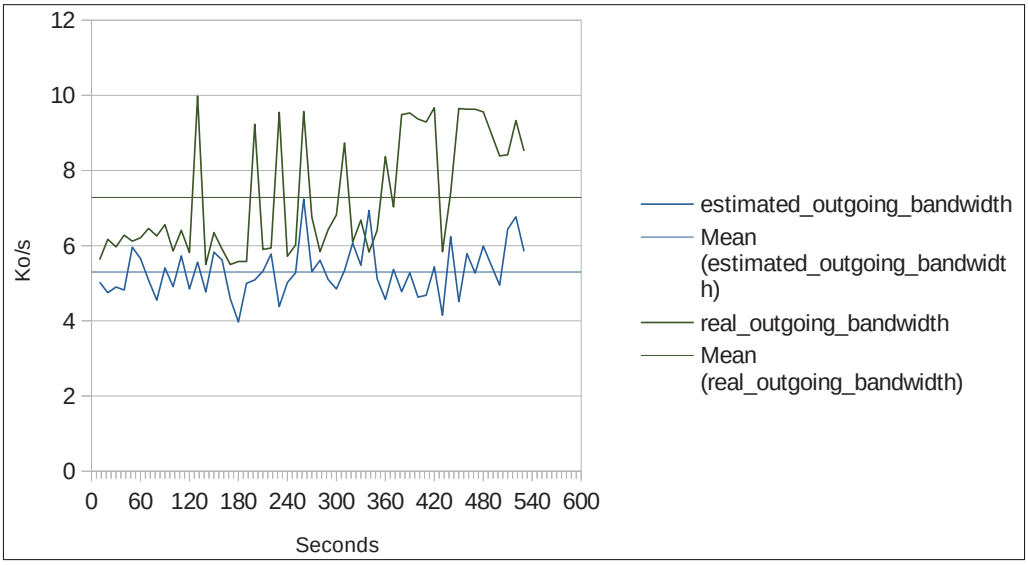

Figure 4.4 Evaluation de la bande passante sortante pour la moyenne des brokers

Afin de s'assurer que toutes les consommations de bande passante sortante par tous les noeuds sont au-dessous de la valeur seuil (`softConstraint`) nous présentons dans la figure 4.5 les mesures de consommations des bandes passantes sortantes les plus élevées qui correspondent à tous les noeuds (30 Raspberry Pi's), cette figure illustre les consommations de bande passante les plus élevées sortantes estimées (`worst_estimated_outgoing`) et réelles (`worst_-`

`real_outgoing`).

Nous pouvons remarquer que toutes les mesures n'ont pas dépassé (`softConstraint`) 350 Ko/s, ce qui permet de conclure que l'orchestrateur a réussi d'équilibrer la charge entre les noeuds et respecter la contrainte de bande passante sortante.

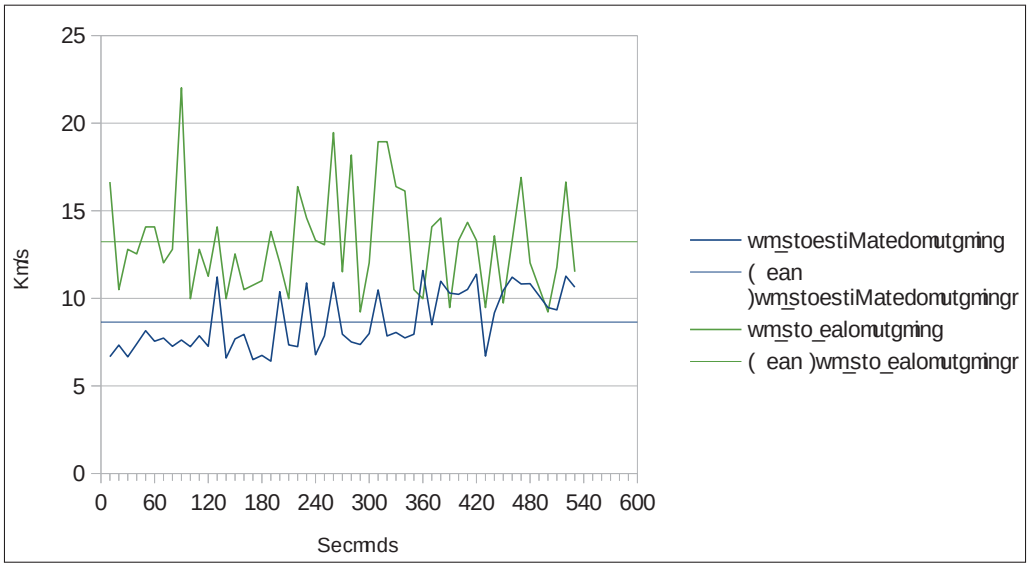

Figure 4.5 Evaluation de la bande passante sortante (pire Cas)

4.5 Analyse de la latence

Afin d'analyser la latence, nous avons eu recours à une méthodologie qui permet d'évaluer les performances de la solution sur trois déploiements différents à savoir (local, national et mondial), ceci pour s'assurer que notre solution minimise la moyenne des latences d'échange de messages tout en respectant la contrainte de bande passante quelle que soit la dispersion géographique des noeuds edge. Dans ce contexte, nous avons utilisé deux types de datasets. Le premier "dataset" concerne le déploiement local qui consiste à un ensemble de mesures de latences entre les noeuds mesurées dans trois pièces différentes, afin de modéliser un déploiement local dans un même environnement (plus de détails sur la méthodologie à la section 4.5.1), le deuxième dataset "King dataset" (Krishna Gummadi, 2002) concerne les déploiements à l'échelle nationale et mondiale. Comme le montre le tableau 4.3, nous avons fixé le nombre de sujets(topics) à 50, le nombre de

courtiers (brokers) à 30, le nombre de publications à 100 et le nombre de souscriptions à 200 afin d'analyser la latence.

Tableau 4.3 Liste des paramètres d'analyse de la latence

Paramètre	Description	Valeur
Topics	Nombre de topics	50
Brokers	Nombre de courtiers (brokers)	30
Publications	Nombre de publications	100
Souscriptions	Nombre de souscriptions	200

4.5.1 Évaluation locale

Dans cette section, nous évaluons notre solution en termes de latence offerte pour l'échange des messages dans un déploiement local. En effet nous comparons entre la moyenne des latences prédites par l'orchestrateur selon le plan généré par DynPubSub, les mesures réelles c'est-à-dire la moyenne des latences offertes et mesurées par la librairie "cmqtt", la moyenne des latences prédites selon le plan généré qui alloue tous les topics à un serveur situé localement (plus de détails à la section 4.5.1.1) et la moyenne des latences offerte selon le plan qui alloue l'ensemble des topics à un serveur cloud unique centralisé, qui permet d'émuler un déploiement infonuagique dans un centre de données Amazon.

4.5.1.1 Méthodologie de simulation

Afin de réaliser la simulation, nous avons opté à émuler les latences qui caractérisent la bande passante et les latences typiques obtenues par une connexion sans fil (Wifi) de raspberry Pis placées à différents emplacements autour d'un point d'accès, ce qui nous a permis de générer un tableau de latences caractéristiques. En effet nous avons placé les noeuds aléatoirement dans les trois pièces séparées, ensuite nous avons alloué des latences entre chaque paire de clients ($client_i$ et $client_j$) que l'on a appelé $Lmatrix[i, j]$ comme le montre l'algorithme 4.1. La figure 4.6 illustre la méthode adoptée afin de collecter les données de latences et créer la base de données pour le déploiement local. En effet, nous avons modélisé un scénario selon lequel les noeuds IoT

Algorithme 4.1 Algorithme de génération tableau de latences (déploiement local)

Algorithme : Algorithme de génération tableau de latences (déploiement local)
Input : List of clients $Clients = \{\textbf{client}_1, ..., \textbf{client}_J\}$, latencies tables from 3 rooms $\mathcal{R}ooms = \{\textbf{Room}_1, \textbf{Room}_2, \textbf{Room}_3\}$
Output : Latencies matrix $\mathcal{L}matrix$

for $i \in \{1, \ldots, numberClients\}$ **do**
  for $j \in \{1, \ldots, numberClients\}$ **do**
    $room_i \leftarrow$ checkRoom($client_i$);
    $room_j \leftarrow$ checkRoom($client_j$);
    $Lmatrix[i, j] \leftarrow$ randomLatencyBetweenRooms($room_i$,$room_j$)/2
  end for
end for

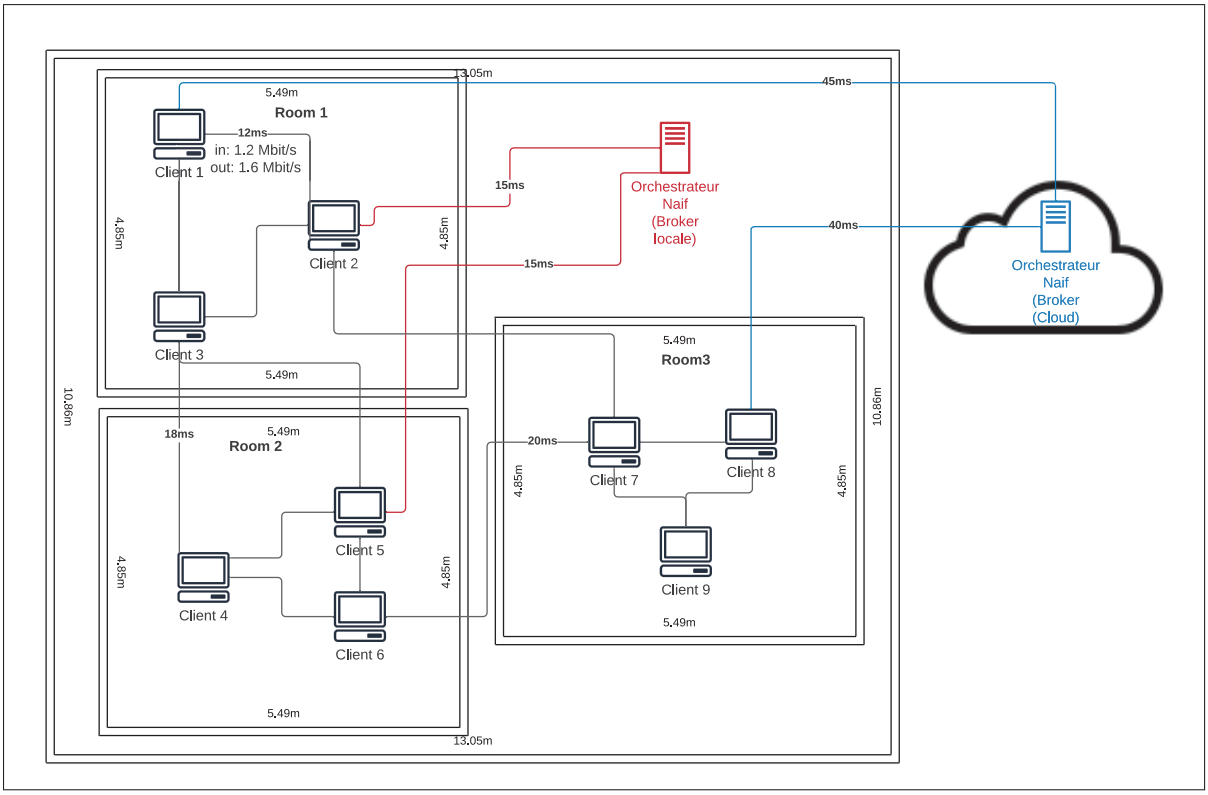

Figure 4.6 Méthodologie de simulation (déploiement à l'échelle locale)

sont déployés dans un bâtiment ou une infrastructure industrielle dans lequel il y a un serveur centralisé (serveur local) et un accès à serveur cloud (très proche géographiquement). Pour le

faire nous avons fixé un noeud (ordinateur) dans une pièce et nous avons mesuré des latences entre cet ordinateur et un noeud (Raspberry Pi 3) en nous déplaçant dans trois pièces.

4.5.1.2 Résultats de l'évaluation de latence locale

Dans cette section, nous présentons les résultats obtenus lors de la simulation de notre solution suivant un déploiement local, tout en considérant les paramètres mentionnés au tableau (4.3)

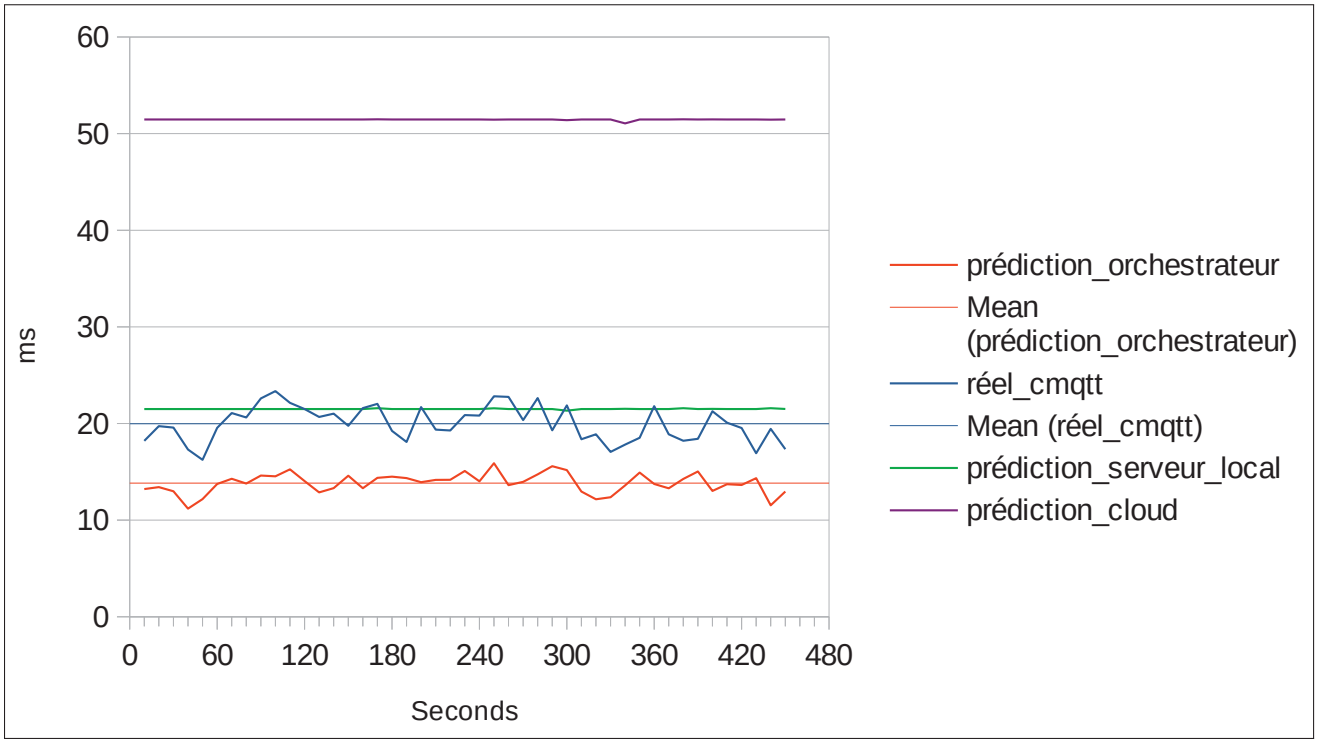

Figure 4.7 Evaluation de la latence (déploiement local)

La figure 4.7 montre les résultats de simulation de la latence pour un déploiement local. Elle représente la moyenne des latences prédite par l'orchestrateur, la moyenne des latences réelles mesurées lors de l'échange de messages, la moyenne de latences obtenues avec la configuration serveur local centralisé, et la moyenne des latences avec la configuration serveur cloud. Nous pouvons remarquer que la moyenne des latences offerte selon le plan généré par l'orchestrateur est la plus faible comparée à la méthode naïve et cloud. Les résultats mesurés expérimentalement (librairie CMQTT) sont proches de ceux prédits par l'orchestrateur, la différence s'explique par

des délais dans l'émulation des latences selon la méthodologie décrite ci-haut.
L'objectif est atteint, grâce au déploiement distribué des brokers déployées sur les noeuds edge réalisé par DynPubSub ainsi qu'à sa stratégie de rebalancement dynamique, nous avons pu obtenir une moyenne de latence 5% plus faible que celle offerte par la configuration serveur local centralisé et 60% plus faible que la configuration avec un serveur infonuagique centralisé. Un autre avantage offert par DynPubSub par rapport au serveur local est que les périphériques edge déployés permettent d'augmenter la fiabilité en cas de panne d'un serveur centralisé.

4.5.2 Jeu de données "King"

King est un jeu de données qui contient des mesures de latences entre des noeuds dans différentes régions géographiques. L'outil King (Krishna Gummadi, 2002) qui a permis d'obtenir ce jeu de données est basé sur des requêtes DNS récursives. En effet la latence entre deux noms de serveurs peut être mesurée à l'aide des requêtes DNS récursives et la latence entre les terminaux peut être approximée à la latence entre leurs noms de serveurs comme le montre la figure 4.8. Toutes les mesures obtenues sont des mesures directes et non pas des extrapolations hors lignes.

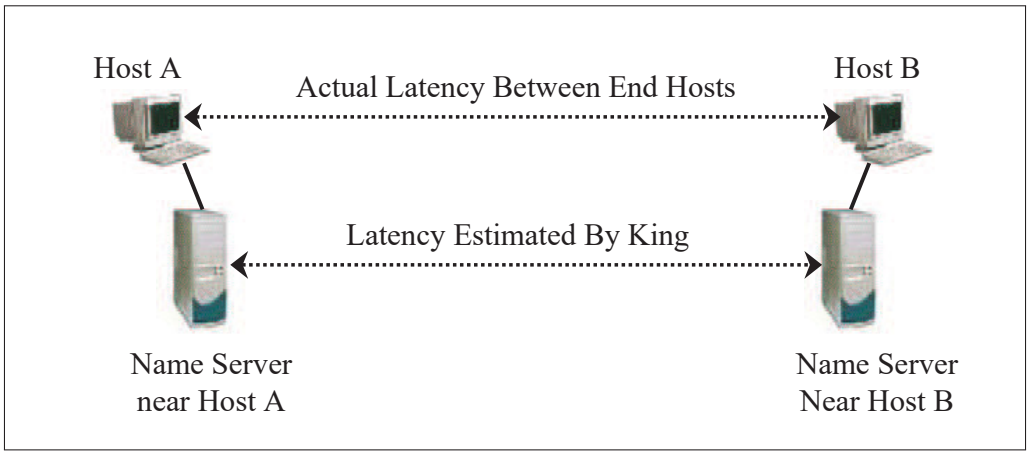

Figure 4.8 Estimation de la latence avec le jeu de données (kingdataset)
Tirée de Krishna Gummadi (2002, p2)

Nous avons utilisé le jeu de données (kingdataset) afin d'émuler les latences pour nos simulations et évaluations, car il est impossible de faire le tour du monde pour placer les noeuds IoT dans de

différentes zones géographiques puisque la base de données King offre des latences entre des noeuds selon leurs adresses ip, il est possible de déterminer la localisation approximative des différents noeuds en utilisant une basse de données de géolocalisation d'adresses IP.

4.5.3 Algorithme des latences simulées

Dans cette section, nous présentons l'algorithme qui permet de retourner la latence entre un noeud et un broker selon le mode à évaluer (local ou infonuagique). Cet algorithme nous a permis d'émuler partiellement les latences entre les noeuds Edge puisque nous ne pouvons pas placer réellement ces derniers dans plusieurs zones géographiques dispersées pour les déploiements à l'échelle nationale et mondiale.

Tableau 4.4 Liste des paramètres de l'algorithme 4.2

Paramètre	Description	Entrée/Sortie
$ipBroker$	Adresse ip du courtier (broker)	Entrée
$ipClient$	Adresse ip du client	Entrée
$mode$	Le mode (infonuagique ou local)	Entrée
$cloudLatenciesDataset$	Dataset des latences infonuagiques	Entrée
$localLatenciesDataset$	Dataset des latences locales	Entrée
$clientsArray$	Listes des clients avec leurs indices	Entrée
$Latency$	La latence en question	Sortie

Algorithme 4.2 Algorithme d'obtention de latences simulées

```
1 Algorithme : Algorithme d'obtention de latences simulées
  Input : ipBroker, ipClient, mode, cloudLatenciesDataset,localLatenciesDataset,
          clientsArray
  Output : Latency

2 if (mode===local) then
3 |   fileLine ← localLatenciesDataset[clientsArray[ipBroker]];
4 |   Latency ← fileLine.split(',')[clientsArray[ipClient]];
5 else
6 |   Latency ← cloudLatenciesDataset[clientsArray[ipClient]];
7 end if
```

Comme le montre l'algorithme 4.2 et le tableau 4.4, nous avons plusieurs paramètres en entrée à savoir l'adresse ip du *broker*, l'adresse ip du client, le mode à évaluer (cloud ou locale), le jeu de données (dataset) des latences locales, le jeu de données (dataset) des latences cloud et un tableau qui contient les adresses ip des clients avec leurs identifiants. La sortie est la latence simulée entre le *broker* et le client.

En effet, comme première étape, nous vérifions le mode de l'évaluation s'il s'agit du mode local. Ensuite nous parcourons le jeu de données des latences locales et retenons la ligne qui correspond à l'identifiant du *broker* ensuite nous découpons cette ligne en tranches avec la méthode split(',') puisqu'il s'agit d'un fichier CSV et nous retournons la latence comme la tranche dont l'indice (*index*) correspond à l'identifiant du client. Sinon, si le mode de l'évaluation est cloud, nous retournons la latence comme la ligne à partir du dataset cloud dont l'indice correspond à l'identifiant du client. Les datasets préparées sont dédiées pour différents types de déploiement (local, à l'échelle nationale ou à l'échelle mondiale) et différents modes (local ou infonuagique). En effet pour un déploiement local nous considérons le dataset crée à partir des simulations de latences entre des noeuds déployés dans un environnement local à l'aide des (ping) , sinon s'il s'agit d'un déploiement à l'échelle nationale ou modiale nous avons choisi de travailler avec un jeu de données préexistant nommé King dataset (Krishna Gummadi, 2002) qui définit la latence entre des neouds localisés partout dans le monde, plus de détails concernant la méthodologie de simulations des latences se trouvent dans la section 4.5.

4.5.4 Évaluation de la latence (échelle nationale)

Dans cette section, nous évaluons notre solution en termes de latence offerte pour l'échange des messages dans un déploiement à l'échelle nationale (ensemble de clients IoT MQTT déployés dans une zone géographique (USA)). En effet nous comparons entre la moyenne des latences fournies selon le plan généré par l'orchestrateur, la moyenne des latences obtenues par la librairie "cmqtt", et la moyenne des latences fournies selon le plan généré par un serveur cloud unique centralisé. Contrairement au déploiement local, pour le déploiement à l'échelle nationale nous ne considérons pas de configuration visant à émuler un déploiement sur un serveur local centralisé

puisque cela ne serait pas utile étant donné que les clients sont répartis partout dans la zone géographique des USA et par la suite ça revient à une configuration d'un serveur infonuagique.

4.5.4.1 Méthodologie de simulation

Afin de réaliser la simulation nous avons opté à émuler les latences. Nous avons généré un tableau de latences aléatoirement pour les clients qui se trouvent dans différents états des États-Unis (US) et un autre tableau de latences entre les noeuds clients et un ensembe de noeuds localisés en Virginie (USA) servant à émuler la localisation du centre infonuagique Amazon US-East comme le montre l'algorithme 4.3. Le choix de USA comme pays pour émuler les latences pour le déploiement à l'échelle nationale est dû à la grade disponibilité des mesures de latence liées au USA dans le Kingdataset, ainsi nous avons choisi l'Etat Virginie pour le considérer pour la configuration infonuagique centralisé parce qu'il existe un centre de données d'Amazon à Virginie. L'émulation est nécessaire puisque pour des raisons pratiques, il n'était pas possible de déployer les noeuds physiquement aux différents endroits et récolter les données de latence, contrairement au scénario local.

Algorithme 4.3 Algorithme de génération de tableau de latences (déploiement à l'échelle nationale)

Algorithme : Algorithme de génération de tableau de latences (déploiement à l'échelle nationale)
Input : kingDataset, List of clients $Clients = \{\textbf{client}_1, ..., \textbf{client}_J\}$
Output : Latencies matrix for orchestrator $\mathcal{L}matrix$,Latencies table for cloud $\mathcal{L}cloud$

usLatencies←filterKingDataSet(kingDataset);
virginieLatencies←filterUsLatencies(usLatencies);
for $i \in \{1, \ldots, numberClients\}$ **do**
    for $j \in \{1, \ldots, numberClients\}$ **do**
        $state_i \leftarrow$ checkState($client_i$);
        $state_j \leftarrow$ checkState($client_j$);
        $Lmatrix[i, j] \leftarrow$ randomLatencyBetweenStates($state_i$,$state_j$)/2;
        $Lcloud[i] \leftarrow$ randomLatencyStatesCloud($state_i$,virginieLatencis)/2
    end for
end for

La figure 4.9 présente un exemple de déploiement à l'échelle nationale, en effet nous avons placé les noeuds IoT aléatoirement dans les États-Unis et de façon dispersée pour des résultats pertinents : nous avons choisi des noeuds qui sont proches du cloud qui est situé à Virginie, d'autres, moyennement éloignés, et d'autres très éloignés. Nous avons adopté cette méthodologie car nous ne pouvons pas collecter nous-mêmes des données expérimentales de latence puisque nous n'avons pas la possibilité de déployer nos noeuds edge réellement. Considérant que le jeu de données King est utilisé dans un grand nombre d'articles, nous jugeons qu'il s'agit d'une solution acceptable sur le plan méthodologique.

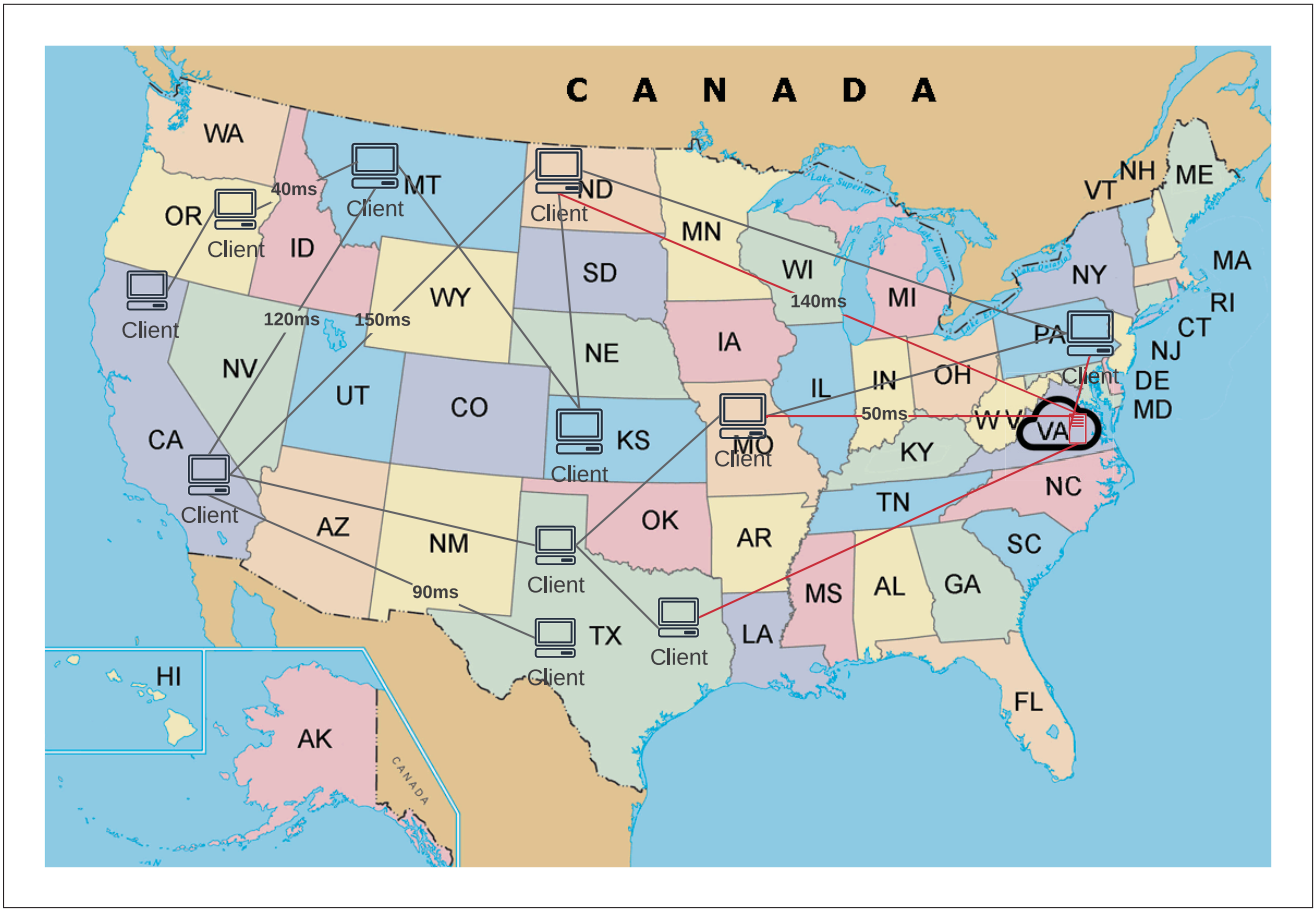

Figure 4.9 Méthodologie de simulation (déploiement à l'échelle nationale)

4.5.4.2 Résultats de l'évaluation de latence à l'échelle nationale

Dans cette section, nous présentons les résultats obtenus lors de la simulation de notre solution suivant un déploiement à l'échelle nationale.

La figure 4.12 montre les résultats obtenus, elle représente la moyenne des latences obtenues grâce à l'orchestrateur, la moyenne des latences réelles obtenues à l'aide de la librairie "cmqtt" et la moyenne des latences obtenues à l'aide d'un serveur cloud. Nous pouvons remarquer que la moyenne des latences offertes selon le plan généré par l'orchestrateur est la plus faible comparée à la méthode cloud. Les résultats latence mesurés expérimentalement (par la librairie cmqtt) obtenus à l'aide de l'émulation partielle des latences avec les valeurs dérivées du jeu de données King selon l'emplacement des noeuds et qui sont très similaires aux résultats de l'orchestrateur. Nous pouvons conclure à partir de la figure 4.12 que notre objectif est atteint aussi dans le déploiement à l'échelle nationale, puisque le modèle distribué offert par DynPubSub nous a permis d'obtenir une moyenne de latences largement plus faible de 44% que celles obtenues grâce à un serveur cloud centralisé.

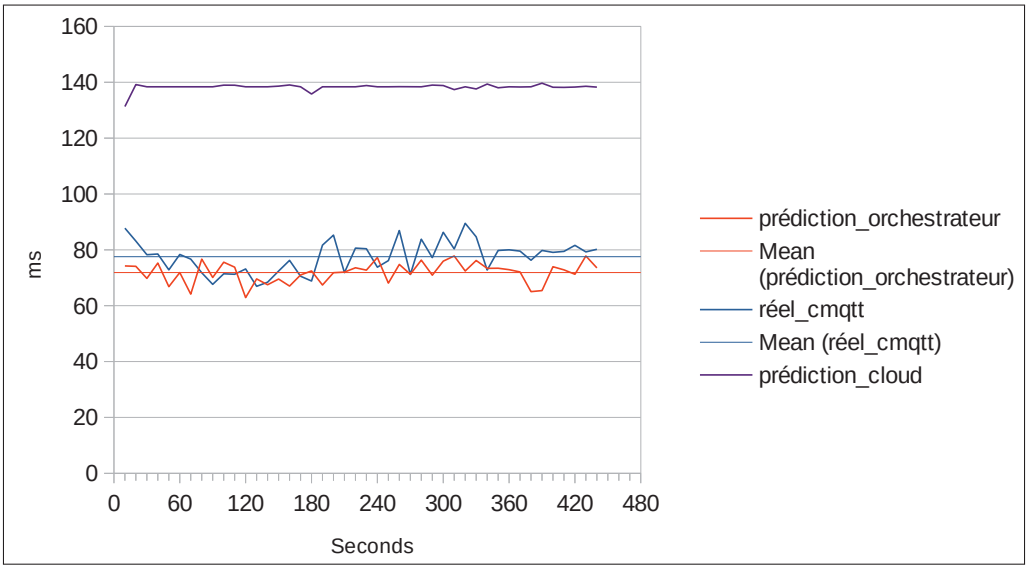

Figure 4.10 Evaluation de la latence (déploiement à l'échelle nationale)

4.5.5 Évaluation de la latence (échelle mondiale)

Dans cette section, nous allons évaluer notre solution en termes de latence offerte pour l'échange des messages dans un déploiement à l'échelle mondiale (clients IoT MQTT répartis dans une zone géographique couvrant plusieurs pays et continents), en effet nous allons comparer entre la moyenne des latences offerte selon le plan généré par l'orchestrateur, la moyenne des latences réelles obtenue à l'aide de la librairie cmqtt et la moyenne des latences offerte selon le plan généré par un serveur cloud unique centralisé. Comme dans le déploiement à l'échelle nationale nous n'allons pas utiliser la configuration d'un serveur unique (qui gère tous les topics) parce que ça revient à la même pour la configuration infonuagique, étant donné que les clients sont dispersés géographiquement dans plusieurs pays.

4.5.5.1 Méthodologie de simulation

De manière similaire à l'évaluation à l'échelle nationale, nous émulons les latences à l'aide du jeu de données King, nous avons généré un tableau de latences aléatoirement pour des clients qui se trouvent dans différents pays dans le monde et un autre tableau de latences entre les différents noeuds, et un sous-ensemble de noeuds situés en Virginie, afin d'émuler des valeurs de latence avec le service infonuagique Amazon US-East comme le montre l'algorithme 4.4.
La figure 4.11 présente un exemple de déploiement à l'échelle mondiale. En effet nous avons placé les noeuds IoT aléatoirement partout dans le monde et de façon dispersée afin d'obtenir des résultats pertinents, il y'a des neouds qui sont proches du cloud qui est situé à Virginie, d'autres moyennement, éloignés, et d'autres loin.

4.5.5.2 Résultats de l'évaluation de latence pour un déploiement à l'échelle mondiale

Dans cette section, nous présentons les résultats obtenus lors de la simulation de notre solution suivant un déploiement à l'échelle mondiale.
La figure 4.12 montre les résultats de simulation de la latence pour un déploiement à l'échelle mondiale. Elle représente la moyenne des latences obtenue grâce à l'orchestrateur, la moyenne

Algorithme 4.4 Algorithme de génération de tableau de latences (déploiement à l'échelle mondiale)

Algorithme : Algorithme de génération de tableau de latences (déploiement à l'échelle mondiale)

Input : kingDataset, List of clients $Clients = \{\textbf{client}_1, ..., \textbf{client}_J\}$

Output : Latencies matrix for orchestrator $\mathcal{L}matrix$,Latencies table for cloud $\mathcal{L}cloud$

virginieLatencies←filterVirginieLatencies(kingDataset);
for $i \in \{1, \ldots, numberClients\}$ **do**
  for $j \in \{1, \ldots, numberClients\}$ **do**
    $country_i \leftarrow$ checkCountry($client_i$);
    $country_j \leftarrow$ checkCountry($client_j$);
    $Lmatrix[i, j] \leftarrow$ randomLatencyBetweenCountries($country_i$,$country_j$)/2;
    $Lcloud[i] \leftarrow$ randomLatencyCountriesCloud($country_i$,virginieLatencis)/2
  end for
end for

Figure 4.11 Méthodologie de simulation (déploiement à l'échelle mondiale)

des latences réelles obtenue à l'aide de la librairie cmqtt et la moyenne des latences obtenue en émulant un serveur infonuagique situé en Virginie (US). Nous pouvons remarquer que la moyenne des latences offertes selon le plan généré par l'orchestrateur est la plus faible comparé à la méthode cloud. Les résultats de la librairie cmqtt sont obtenus grâce à l'émulation partielle des latences avec les valeurs dérivées du jeu de données King selon l'emplacement des noeuds et sont très similaires aux résultats de l'orchestrateur. Comme pour un déploiement local et à l'échelle nationale, nous avons aussi atteint notre objectif pour un déploiement à l'échelle mondiale puisque le système de rebalancement dynamique DynPubSub nous a permis d'obtenir une moyenne de latence 18% plus faible comparée au cloud centralisé.

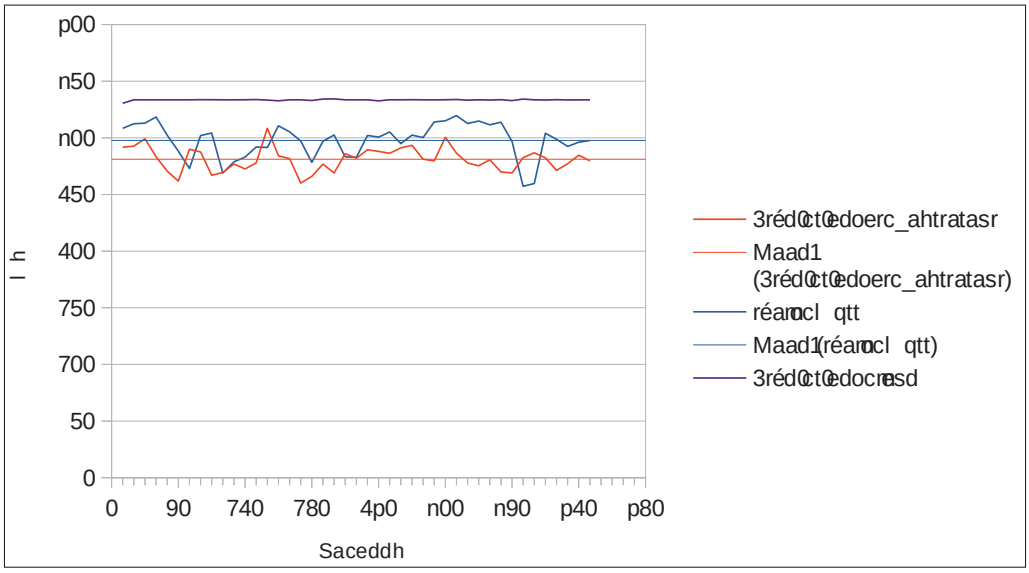

Figure 4.12 Evaluation de la latence (déploiement à l'échelle mondiale)

4.6 Analyse de la performance

Dans cette section, nous analysons les performances de notre solution en termes de consommation de ressources (processeur et mémoire) et en termes de temps écoulé pour l'exécution de reconfiguration DynPubSub, afin de s'assurer de sa scalabilité et sa possibilité de gérer un nombre important de publieurs, souscripteurs et de sujets. Pour le faire, nous avons varié

plusieurs paramètres à savoir le nombre de topics, le nombre de publications et le nombre de souscriptions.

4.6.1 Évaluation avec variation de topics

Dans cette section, nous présentons l'effet de la variation du nombre de topics sur notre système tout en fixant le nombre de publications et souscriptions.

Tableau 4.5 Paramètres de simulation avec variation du nombre de topics

Paramètre	Description	Valeur
Topics	Nombre de topics	10,20,50,100,200,300,500,1000
Publications	Nombre de publications	30
Souscriptions	Nombre de souscriptions	60
Brokers	Nombre de courtiers (brokers)	30 (Raspberry Pi's)

Comme le résume le tableau 4.5, nous avons fixé le nombre de publications à 20, le nombre de souscriptions à 50, le nombre de brokers à 30 et nous avons varié le nombre de topics (10,20,50,100,200,300,500,1000).

4.6.1.1 Analyse de l'utilisation CPU

Dans cette section, nous présentons les résultats de performance liés à l'utilisation du processeur, nous évaluons l'effet de variation du nombre de topics sur l'utilisation CPU.
La figure 4.13 montre l'usage du processeur lors de l'exécution de l'algorithme. Nous pouvons remarquer que lorsqu'on augmente le nombre de topics l'usage processeur augmente linéairement jusqu'à atteindre 18% pour 1000 topics, cependant l'utilisation du processeur demeure faible comparé à un nombre de topics très élevé à savoir 1000 topics.

4.6.1.2 Analyse de la mémoire

Dans cette section, nous présentons les résultats de performance liés à la consommation mémoire. Nous évaluons l'effet de variation du nombre de topics sur la consommation de la mémoire vive.

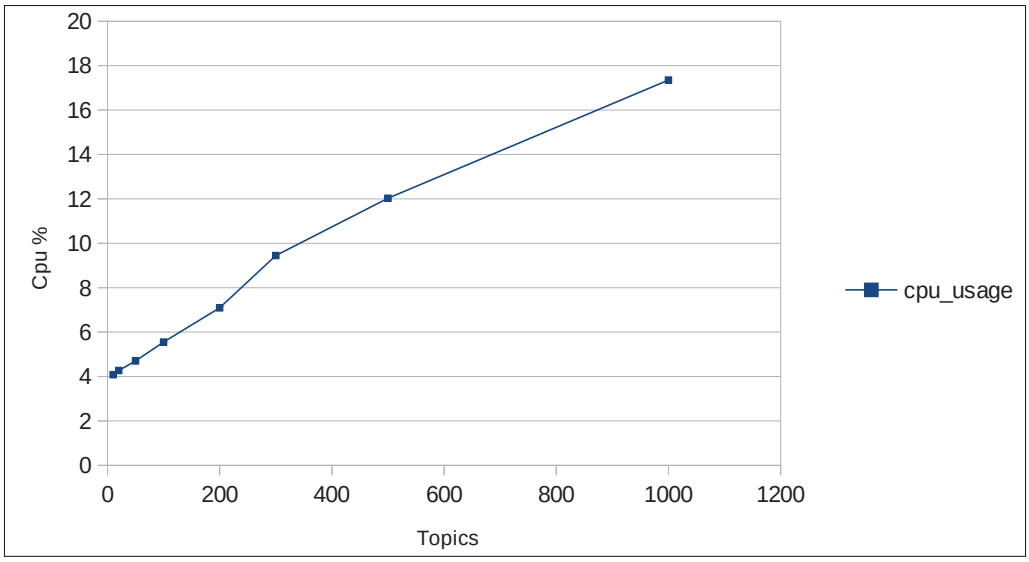

Figure 4.13 Impact de la variation du nombre de sujets (topics) sur l'usage processeur

Comme le montre la figure 4.14, lorsqu'on augmente le nombre de topics la consommation de la mémoire vive augmente aussi jusqu'atteindre 26 Mb pour 1000 sujets (topics). La consommation de la mémoire n'est pas élevée comparé au nombre de topics gérés, ainsi elle peut varier en raison du garbage collector de la machine virtuelle JS. Nous pouvons conclure que notre solution est raisonnable en termes de consommation de la mémoire vive. Nous pouvons conclure que DynPubSub performe bien en termes de consommation mémoire vive.

4.6.1.3 Analyse du temps d'exécution

Dans cette section, nous présentons le temps écoulé pour l'exécution de l'algorithme et l'effet de variation du nombre de topics sur ce dernier. Comme le montre la figure 4.15, lorsqu'on augmente le nombre de topics, le temps écoulé pour l'exécution de la l'algorithme augmente linéairement jusqu'à 3500ms pour 1000 topics, ce qui est attendu, ce nombre reste raisonnable comparé au nombre de topics gérés, ce qui prouve que notre solution DynPubSub est performante en termes d'optimisation pour le temps d'exécution de l'algorithme.

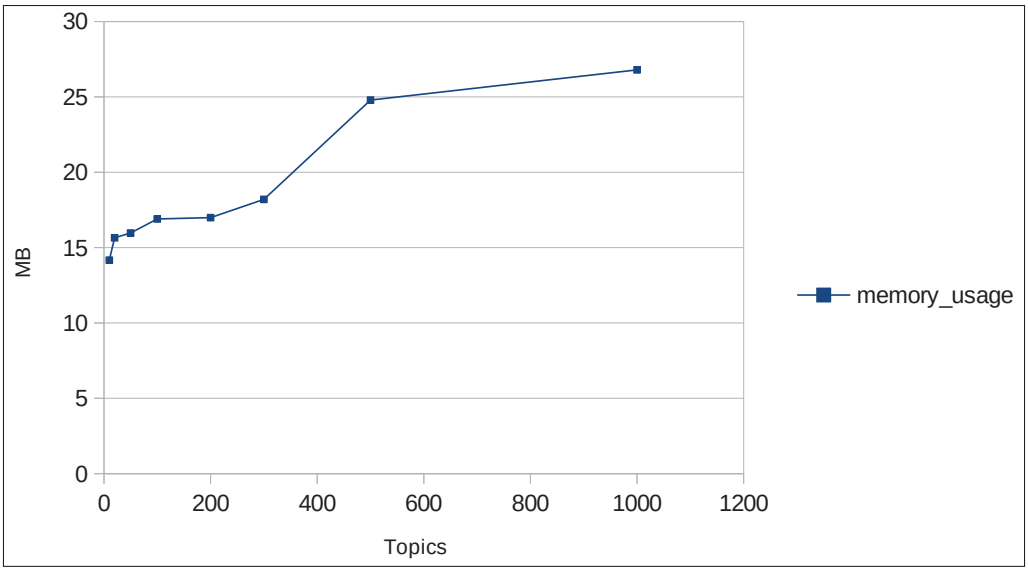

Figure 4.14 Impact de la variation du nombre de sujets (topics) sur la consommation mémoire

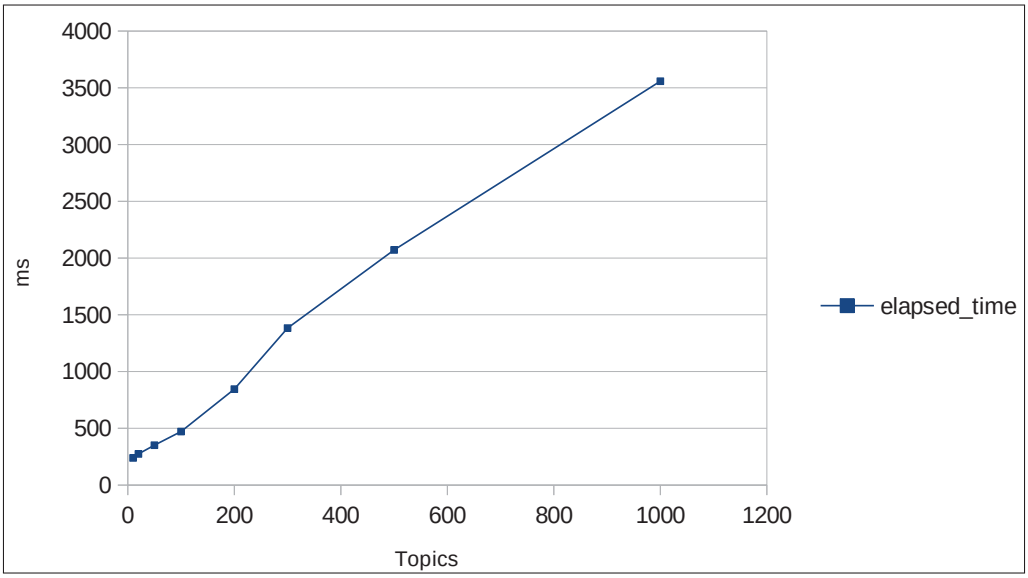

Figure 4.15 Impact de la variation du nombre de sujets (topics) sur le temps d'exécution

4.6.2 Variation du nombre de publications

Dans cette section, nous présentons l'effet de variation du nombre de publications des messages échangés sur notre système tout en fixant le nombre de topics et le nombre de souscriptions. Comme le résume le tableau 4.6, nous avons fixé le nombre de topics à 50, le nombre de

Tableau 4.6 Paramètres de simulation avec variation du nombre de publications

Paramètre	Description	Valeur
Topics	Nombre de topics	50
Publications	Nombre de publications	50,100,200,400,600,1000
Souscriptions	Nombre de souscriptions	60
Brokers	Nombre de courtiers (brokers)	30 (Raspberry Pi's)

brokers à 30, le nombre de souscriptions à 60 et nous avons varié le nombre de publications (50,100,200,400,600,1000).

4.6.2.1 Analyse de l'utilisation CPU

Dans cette section, nous évaluons la performance de notre solution en termes d'utilisation du processeur et nous analysons cette performance lorsqu'on varie le nombre de publications.

Comme le montre la figure 4.16, lorsque le nombre de publications augmente, le taux d'usage de processeur augmente linéairement jusqu'au 70% pour 1000 publications, la consommation n'est pas trop élevée comparé au nombre de publications. L'effet d'augmentation de nombre de publications est plus important que l'effet d'augmentation de nombre de topics parce que le nombre de publications a un impact important sur le coût de reconfiguration et sur le volume du trafic à gérer.

Notre solution reste performante et s'adapte bien malgré l'augmentation du nombre de publications. De plus l'implémentation de la solution a été réalisée en utilisant NodeJs, or, ce dernier nous offre l'exploitation d'un seul *thread* du processeur donc 70% d'un seul *thread* pour 1000

publications sont de bons résultats, par la suite nous pourrons supporter de plus grand nombre en concevant une application multi-thread.

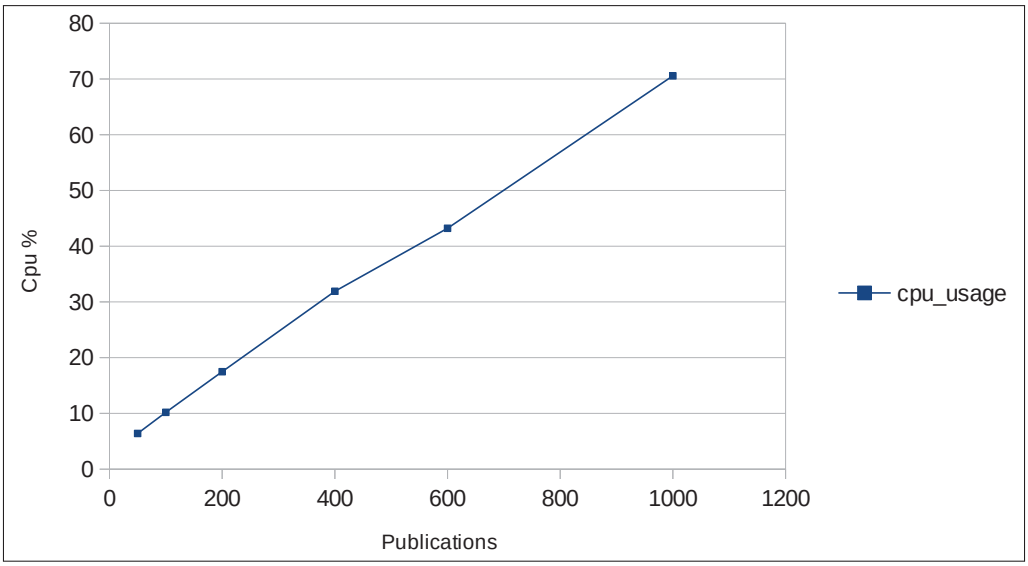

Figure 4.16 Impact de la variation du nombre de publications sur l'usage processeur

4.6.2.2 Analyse de la mémoire

Dans cette section, nous analysons l'effet de variation de nombre de publications sur la consommation de la mémoire vive.

Comme le montre la figure 4.17, lorsqu'on augmente le nombre de publications, la consommation de la mémoire vive augmente jusqu'au 21 MB pour 1000 publications, cette consommation reste faible comparée au nombre de publications, ce qui prouve que notre solution DynPubSub est raisonnable en termes de la consommation de la mémoire malgré l'augmentation du nombre de publications.

4.6.2.3 Analyse du temps d'exécution

Dans cette section, nous analysons la performance de notre solution en termes de temps écoulé pour d'exécution de l'algorithme.

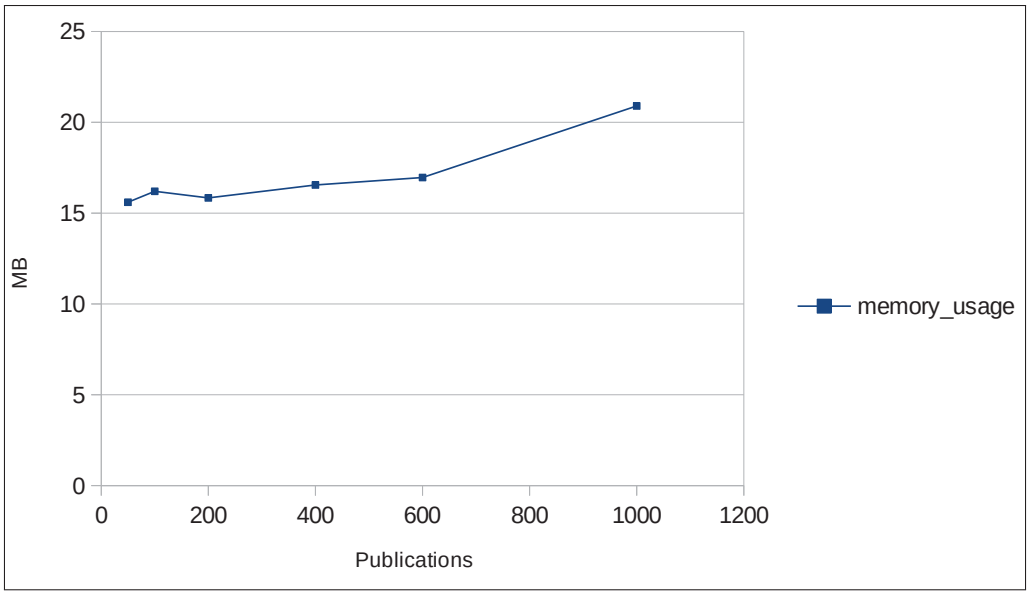

Figure 4.17 Impact de la variation du nombre de publications sur la consommation mémoire

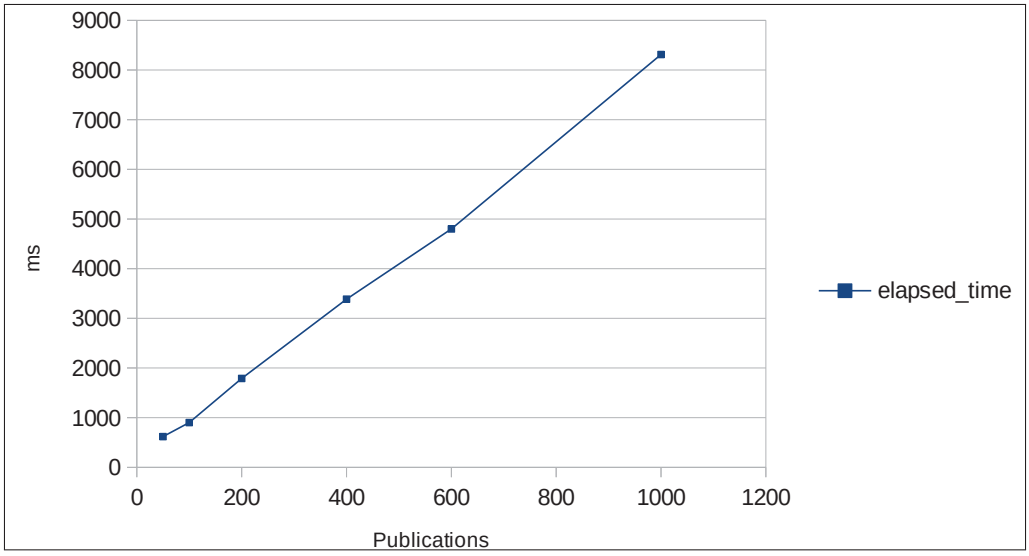

Figure 4.18 Impact de la variation du nombre de publications sur le temps d'exécution

Comme le montre la figure 4.18, lorsque le nombre de publications augmente le temps écoulé pour l'exécution de l'algorithme augmente linéairement jusqu'à 8500 ms pour 1000 publications, le temps écoulé demeure faible comparé au nombre de publications gérées. Ces résultats montrent

que notre solution DynPubSub est performante en termes d'optimisation du temps d'exécution de l'algorithme.

4.6.3 Variation du nombre de souscriptions

Dans cette section, nous analysons l'effet de variation du nombre de souscriptions sur l'usage processeur, la consommation de la mémoire vive et le temps écoulé pour l'exécution de l'algorithme tout en fixant le nombre de topics et le nombre de publications.

Tableau 4.7 Paramètres de simulationavec variation du nombre de souscriptions

Paramètre	Description	Valeur
Topics	Nombre de topics	50
Publications	Nombre de publications	50
Souscriptions	Nombre de souscriptions	50,100,200,400,600,1000
Brokers	Nombre de courtiers (brokers)	30 (Raspberry Pi's)

Comme le montre le tableau 4.7, nous avons fixé le nombre de topics à 50, les publications à 50, les courtiers (brokers) à 30 et nous avons varié le nombre de souscriptions à (50,100,200,400,600,1000).

4.6.3.1 Analyse de l'utilisation CPU

Dans cette section, nous analysons la performance de la solution en termes d'usage processeur lorsqu'on varie le nombre de souscriptions. Comme le montre la figure 4.19, lorsque le nombre de souscriptions augmente l'usage processeur augmente jusqu'à 46% pour 1000 souscriptions. Cet usage reste faible comparé au nombre de souscriptions gérées. Nous pouvons remarquer que l'augmentation du nombre de publications a un effet plus important que l'augmentation du nombre de souscriptions sur l'utilisation du processeur et c'est attendu, car pour un nombre de souscriptions donné les publications vont être multipliées par ce nombre.
Nous pouvons conclure que notre solution DynPubSub performe bien face à l'augmentation du nombre de souscriptions.

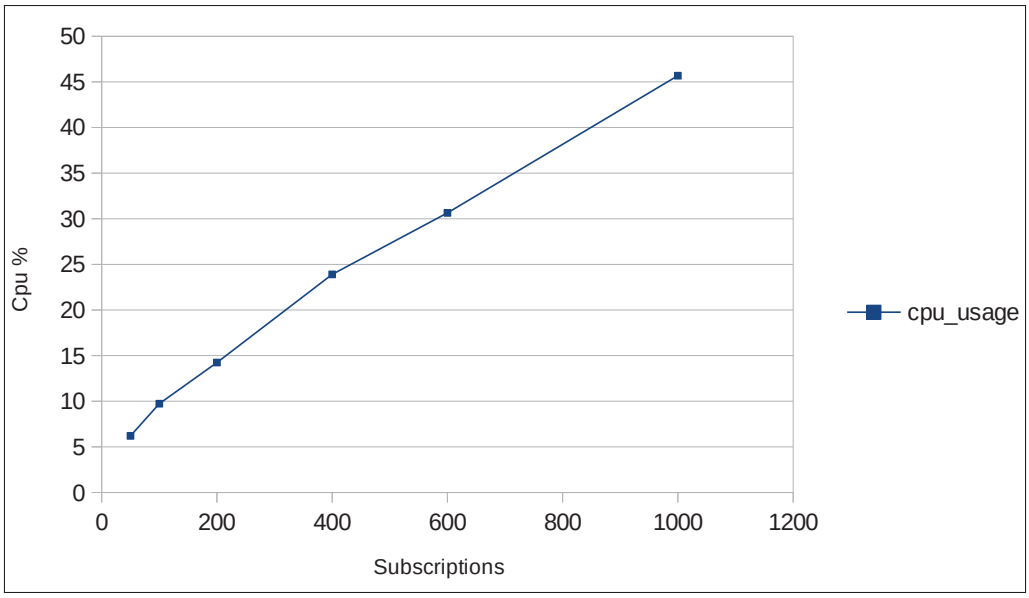

Figure 4.19 Impact de la variation du nombre de souscriptions sur l'usage processeur

4.6.3.2 Analyse de la mémoire

Dans cette section, nous analysons la performance de la solution en termes de la consommation de la mémoire vive lorsqu'on varie le nombre de souscriptions.

Comme le montre la figure 4.20, lorsque le nombre de souscriptions augmente la consommation mémoire vive augmente jusqu'au 16.7 MB pour 1000 souscriptions. Cette consommation reste faible comparé au nombre de souscriptions gérées.

Nous pouvons conclure que notre solution DynPubSub performe bien en termes de consommation mémoire vive face à l'augmentation du nombre de souscriptions.

4.6.3.3 Analyse du temps d'exécution

Dans cette section, nous analysons l'effet de variation du nombre de souscriptions sur le temps écoulé pour l'exécution de l'algorithme.

Comme le montre la figure 4.21, lorsque le nombre de souscriptions augmente le temps écoulé pour l'exécution de l'algorithme augmente linéairement jusqu'au 5000 ms pour 1000

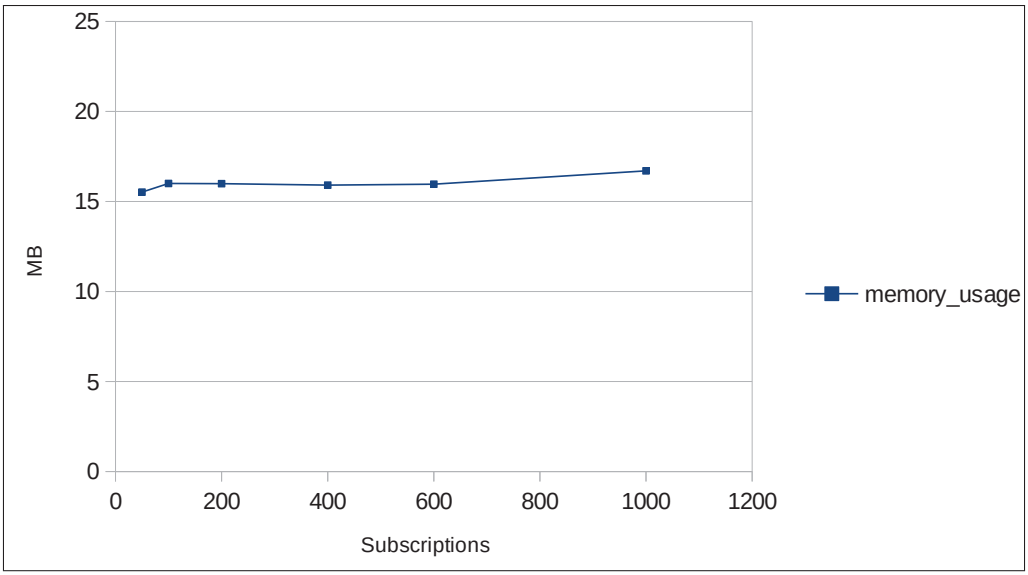

Figure 4.20 Impact de la variation du nombre de souscriptions sur la consommation mémoire

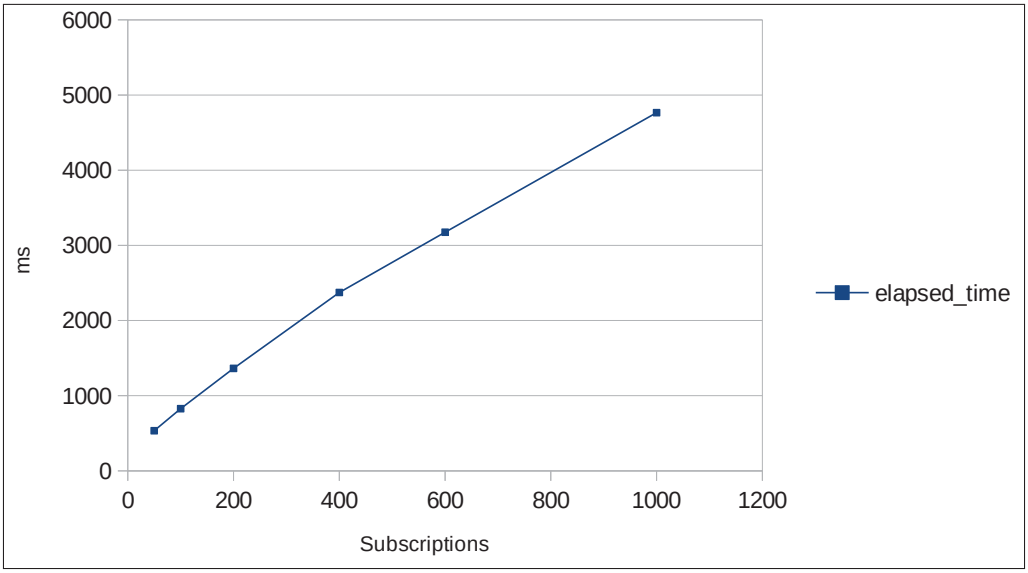

Figure 4.21 Impact de la variation du nombre de souscriptions sur le temps d'exécution

souscriptions. Cette valeur reste faible comparée au nombre de souscriptions gérées.

Nous pouvons conclure que notre solution optimise le temps d'exécution malgré le nombre de souscriptions élevé.

4.6.4 Variation du nombre de publications et souscriptions

Dans cette section, nous présentons l'effet de variation du nombre de publications et souscriptions des messages échangés en même temps tout en fixant le nombre de topics.

Tableau 4.8 Paramètres de simulation avec variation du nombre de publications et souscriptions

Paramètre	Description	Valeur
Topics	Nombre de topics	50
Publications	Nombre de publications	50,100,200,300,400
Souscriptions	Nombre de souscriptions	50,100,200,300,400
Brokers	Nombre de courtiers (brokers)	30 (Raspberry Pi's)

Comme le résume le tableau 4.8, nous avons fixé le nombre de topics à 50, le nombre de brokers à 30, et nous avons varié le nombre de publications ensemble (50/50, 100/100, 200/200, 300/300, 400/400).

4.6.4.1 Analyse de l'utilisation CPU

Dans cette section, nous analysons les résultats de performance en termes d'usage processeur lorsqu'on varie le nombre de publications et souscriptions ensemble.
Comme le montre la figure 4.22, lorsque le nombre de publications et souscriptions augmente l'usage processeur augmente jusqu'au 98% pour 400 publications et 400 souscriptions. Nous voulons bien rappeler ici qu'il s'agit d'un seul *thread* utilisé à cause du NodeJs. Nous pouvons conclure par la suite que notre solution DynPubSub performe bien en termes de l'utilisation CPU malgré le nombre très élevé de publications et souscriptions. Le ralentissement de la croissance s'explique car le % CPU (un seul coeur) ne peut pas dépasser 100%.

4.6.4.2 Analyse de la mémoire

Dans cette section, nous analysons les résultats de performance en termes de la consommation de la mémoire vive lorsqu'on varie le nombre de publications et souscriptions ensemble.

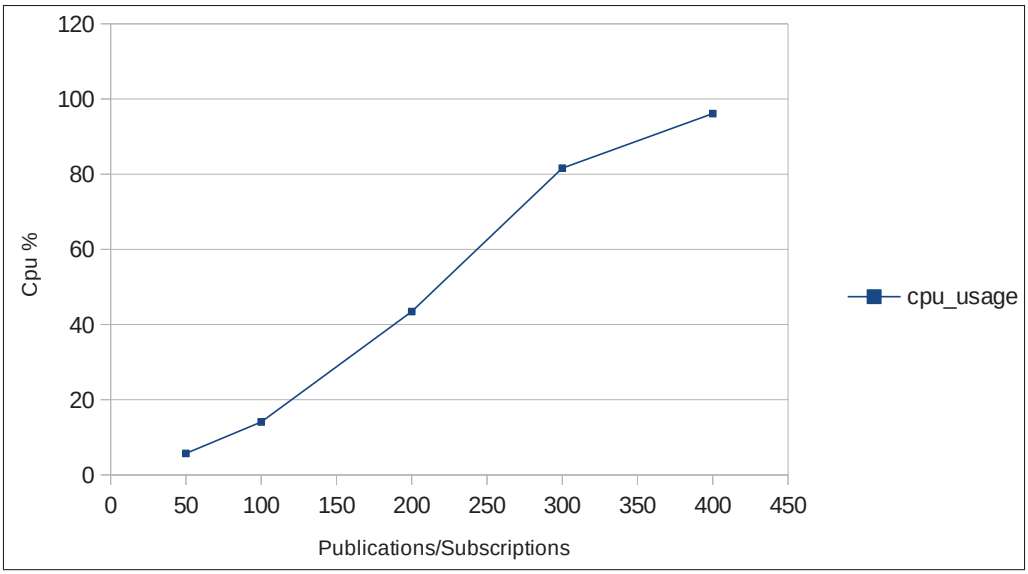

Figure 4.22 Impact de la variation du nombre de publications/souscriptions sur l'usage processeur

Comme le montre la figure 4.23, lorsque le nombre de publications et souscriptions augmente la consommation mémoire vive augmente jusqu'au 22.5 MB pour 400 publications et 400 souscriptions. Cette consommation reste faible malgré le nombre très élevé de publications et souscriptions, ce qui nous permet de conclure que notre solution DynPubSub garantit une consommation de mémoire raisonnable malgré le nombre élevé de publications et souscriptions.

4.6.4.3 Analyse du temps écoulé

Dans cette section, nous analysons les résultats de performance en termes de temps écoulé pour l'exécution de l'algorithme lorsqu'on varie le nombre de publications et souscriptions ensemble. Comme le montre la figure 4.24, lorsque le nombre de publications et souscriptions augmente, le temps écoulé pour l'exécution de l'algorithme augmente jusqu'au 15800 ms pour 400 publications et 400 souscriptions. La courbe possède l'allure de croissance quadratique (n^2) puisqu'on varie deux paramètres à la fois.

Les résultats sont satisfaisants comparé au nombre de publications, souscriptions, topics et courtiers (brokers) gérés. Donc on peut s'assurer que notre solution DynPubSub garantit une

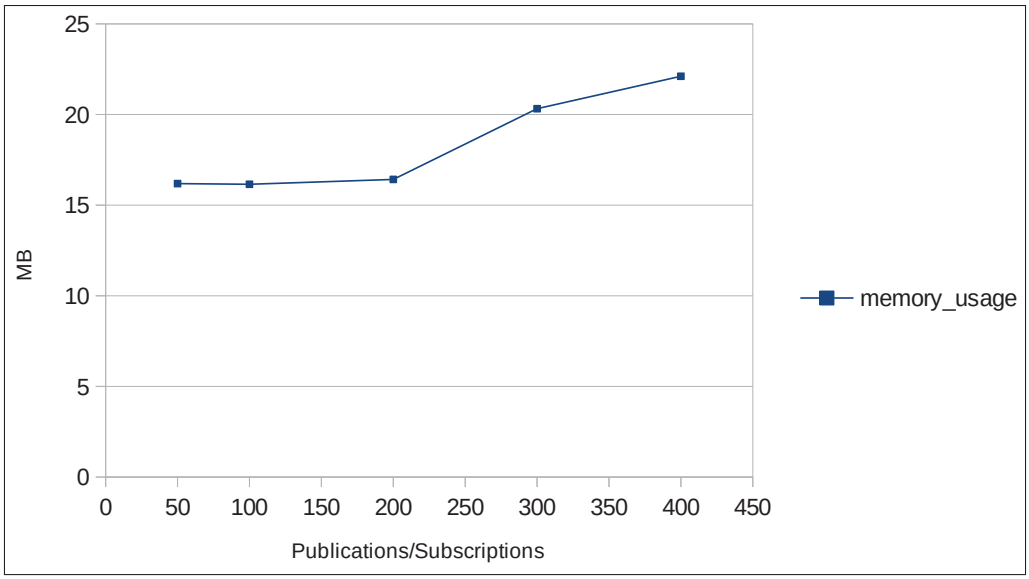

Figure 4.23 Impact de la variation du nombre de publications/souscriptions sur la consommation mémoire

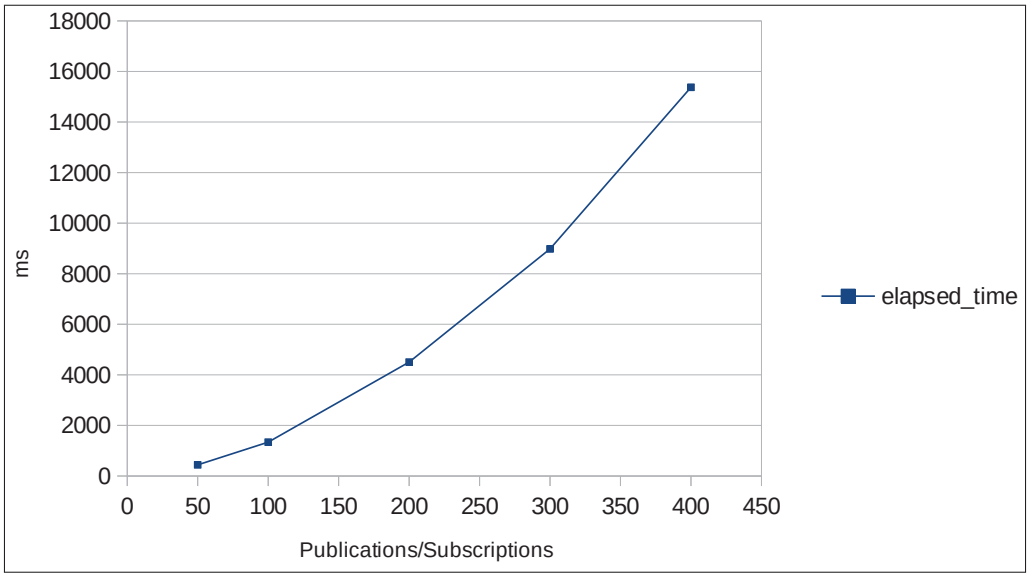

Figure 4.24 Impact de la variation du nombre de publications/souscriptions sur le temps d'exécution

consommation de ressources physique(mémoire, CPU) raisonnable malgré l'augmentation des paramètres à savoir le nombre de publications, souscriptions, topics, brokers.

CHAPITRE 5

SYNTHÈSE DES RÉSULTATS ET LIMITES DE L'ÉTUDE

Dans ce chapitre, nous résumons les résultats obtenus lors de l'évaluation de notre solution en fonctions des questions de recherches posées dans le chapitre "Approche". Nous discutons également des différentes limitations de notre travail de recherche.

5.1 Synthèse des résultats

Nous commençons d'abord par rappeler les questions de recherches :

1. QR1 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement à l'échelle locale tout en respectant la bande passante comme contrainte ?
2. QR2 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement à l'échelle nationale tout en respectant la bande passante comme contrainte ?
3. QR3 : Est-ce que DynPubSub réduit les latences dans un contexte de déploiement à l'échelle mondiale tout en respectant la bande passante comme contrainte ?
4. QR4 : Est-ce que DynPubSub offre la mise à l'échelle de point de vue théorique ?

Afin de répondre à la deuxième partie de 3 premières questions de recherches (QR1, QR2, QR3) nous avons évalué notre solution et nous avons vérifié qu'elle respecte la contrainte de bande passante pour le déploiement local, en effet les différentes mesures de bandes passantes moyennes et les plus élevées des différents noeuds IoT ne dépassent pas la *soft constraint* définie, ce qui prouve que notre orchestrateur a réussi à équilibrer la charge entre les noeuds et respecter la contrainte de bande passante. Ces résultats reviennent aux mêmes pour les déploiements à l'échelle nationale et mondiale parce que la bande passante n'est pas influencée par la distance contrairement à la latence.

Afin de répondre à la première partie de 3 premières questions de recherches nous avons évalué notre solution en termes de moyenne de latence offerte pour l'échange des messages suivant trois déploiements à savoir local, à l'échelle nationale et mondiale. Concernant le déploiement local

nous avons comparé la moyenne de latence obtenue grâce à l'orchestrateur avec la moyenne des latences réelles, la moyenne des latences obtenue avec un orchestrateur naïf (qui gère tous les topics de manière centralisée" et la moyenne des latences obtenue avec un serveur cloud. Les résultats montrent que la plus faible moyenne des latences est obtenue grâce à notre solution DynPubSub. Concernant les déploiements à l'échelle nationale et mondiale nous avons comparé la moyenne des latences obtenue grâce à l'orchestrateur avec la moyenne des latences réelles et la moyenne des latences obtenue grâce à un serveur Cloud émulé en émulant des latences entre les clients et le centre de données infonuagique US-East (Virginia) d'Amazon. Les résultats obtenus montrent que notre solution DynPubSub offre la moyenne des latences la plus faible. La moyenne des latences réelles (c'est-à-dire mesurées expérimentalement avec du trafic réseau généré entre les noeuds) sont très similaires à celle obtenue grâce à notre solution DynPubSub ce qui prouve l'efficacité de notre orchestrateur.

Afin de réponde à la troisième question de recherche, nous avons évalué performance de notre solution DynPubSub en termes d'utilisation du processeur, consommation de mémoire vive et temps d'exécution en variant plusieurs paramètres à savoir le nombre de topics, le nombre de publications, le nombre de souscriptions. Les résultats obtenus montrent que même avec un grand nombre de topics, de publications et souscriptions notre solution DynPubSub optimise la consommation des ressources computationnelles et le temps d'exécution et prouve la mise à l'échelle.

5.2 Limite de l'étude

Dans cette section, nous présentons les principales limites de notre étude qui peuvent avoir un effet sur la validité de nos résultats

5.2.1 Validité externe des résultats de l'étude

La validité externe correspond à la capacité de généraliser les résultats obtenus dans le cadre du travail de recherche. La validité externe de notre étude peut être limitée par :

1. notre étude est limitée aux 30 Noeuds IoT(Raspberry Pi). Ce nombre semble satisfaisant, mais ne représente pas par exemple un réseau de capteurs dont le nombre de noeuds peut être supérieur à 100.
2. nous avons choisi MQTT en tant que protocole pub/sub pour l'implémenter notre solution, cependant il des systèmes IoT qui ne supportent pas ce protocole, cela peut aussi affecter la validité interne de notre étude.
3. utiliser juste un seul type (Raspberry Pi) de noeud IoT peut être aussi une limite de notre étude.

5.2.2 Validité interne des résultats de l'étude

La validité permet d'évaluer la fiabilité et pertinence des résultats d'un travail de recherche. Pour assurer la validité interne de notre étude, nous avons effectué les expérimentations sur le même réseau et les mêmes conditions. Cependant la validité interne de notre étude peut être limitée par :

1. nous ne pouvons pas placer des noeuds IoT partout dans des régions géographiques dispersées ou partout dans le monde, donc nous avons utilisé un jeu de données "KingDataset" pour émuler les latences.
2. afin de simuler les latences des noeuds IoT nous avons utilisé la fonction "timeout" du JavaScript qui rajoute quelques ms à la valeur demandée, par la suite nous aimerons regarder pour d'autres méthodes plus précises pour émuler des conditions à savoir Netem.
3. nous n'avons pas fait une analyse spécifiquement pour évaluer la dynamicité de l'orchestrateur, par la suite nous aimerons bien essayer de varier beaucoup les conditions et voir comment l'orchestrateur va réagir.

CONCLUSION ET RECOMMANDATIONS

L'objectif général de notre étude est de concevoir une nouvelle architecture pair à pair pour les systèmes pub/sub orientés-sujets déployés à la périphérie des réseaux qui permet de respecter la contrainte des noeuds IoT notamment la bande passante et qui minimise la moyenne de latence des échanges de messages. Nous avons conçu une librairie compatible à une librairie existante pour le protocole de communication MQTT (MQTT.js), afin que les développeurs n'aient pas à modifier leur code et nous avons conçu un module orchestrateur qui permet d'équilibrer la charge entre les noeuds IoT afin d'atteindre notre objectif.

Les expérimentations réalisées nous ont permis de valider notre objectif, en effet nous avons vérifié que la moyenne ainsi que les valeurs maximales de bande passante des différents noeuds IoT ne dépassent pas une certaine *soft constraint* bien définie, ainsi les résultats prouvent que nous avons pu réduire la moyenne de latences d'échanges de messages suivant trois déploiements (local, à l'échelle nationale, à l'échelle mondiale). En effet, notre solution DynPubSub nous a permis de réduire la moyenne des latences jusqu'à 5% par rapport à un serveur central local et jusqu'à 60% par rapport à un serveur central cloud dans un déploiement local, ainsi elle nous a permis de réduire la moyenne des latences jusqu'à 44% par rapport à un serveur central cloud dans un déploiement national et nous a permis de réduire la moyenne des latences jusqu'à 18% par rapport à un serveur central cloud dans un déploiement mondial.

Les résultats obtenus montrent la mise à échelle de notre solution, en effet avec la variation des paramètres de simulations à savoir le nombre de topics, le nombre de publications, le nombre de souscriptions DynPubSub optimise la consommation de ressources computationnelles et le temps d'exécution.

En termes de perspectives et travaux futurs, nous proposons d'inclure autres contraintes des noeuds IoT à part la bande passante telles que la consommation d'énergie, l'utilisation du

processeur et la consommation mémoire, ainsi qu'introduire l'utilisation du processeur lors d'équilibrage de charge et la génération du plan réalisée par l'orchestrateur.

BIBLIOGRAPHIE

Abdur Rahim Biswas, R. G. (2014). IoT and Cloud Convergence : Opportunities and Challenges. *2014 IEEE World Forum on Internet of Things (WF-IoT)*.

Alex C. Olivieri,Gianluca Rizzo, F. M. (2015). A Publish-Subscribe Approach to IoT Integration : The Smart Office Use Case. *2015 IEEE 29th International Conference on Advanced Information Networking and Applications Workshops*.

Ana Aguiar, R. M. (2019). Lessons learned and challenges on benchmarking publish-subscribe IoT platforms. *CPS-IoTBench '19 : Proceedings of the 2nd Workshop on Benchmarking Cyber-Physical Systems and Internet of Things*.

bluesmoon. csv-writer. Repéré à https://www.npmjs.com/package/geoip-lite.

Brundha S.M,Lakshmi, S. S. (2017). Home Automation in Client-Server Approach with User Notification along with Efficient Security Alerting system. *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*.

Cenk Gündoğan,Peter Kietzmann, T. C. S. M. W. (2018). HoPP : Robust and Resilient Publish-Subscribe for an Information-Centric Internet of Things. *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*.

Chen Chen,Yoav Tock, H.-A. J. R. V. (2015). Weighted Overlay Design for Topic-based Publish/Subscribe Systems on Geo-Distributed Data Centers. *2015 IEEE 35th International Conference on Distributed Computing Systems*.

Chen Chen,Hans-Arno Jacobsen, R. V. (2016). Algorithms Based on Divide and Conquer for Topic-Based Publish/Subscribe Overlay Design. *IEEE/ACM Transactions on Networking*.

David Tracey, C. S. (2019). How to see through the Fog ? Using Peer to Peer (P2P) for the Internet of Things. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*.

Impagliazzo, R. (2011). Relativized Separations of Worst-Case and Average-Case Complexities for NP.

Jaidip Kotak,Anal Shah, A. S. P. R. (2019). A Comparative Analysis on Security of MQTT Brokers. *2nd Smart Cities Symposium (SCS 2019)*.

Joao Cardoso,Carlos Pereira, A. A. R. M. (2017). Benchmarking Pub/Sub IoT middleware platforms for smart services. *Journal of Reliable Intelligent Environments*.

Julien Gascon-Samson,Franz-Philippe Garcia, B. K. J. K. (2015). Dynamoth : A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud.

Kitae Hwang,Jae Moon Lee, I. H. J. D.-H. L. (2019). Modification of Mosquitto Broker for Delivery of Urgent MQTT Message. *2019 IEEE Eurasia Conference on IOT, Communication and Engineering*.

Kriddikorn Kaewwongsri, K. S. (2020). Design and Implement of a Weather Monitoring Station using CoAP on NB-IoT Network. *2020 17th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*.

Krishna Gummadi,Stefan Saroiu, S. D. G. (2002). King : Estimating Latency between Arbitrary Internet End Hosts. *Proceedings of SIGCOMM IMW 2002*.

Lakshmi Prasanna Chitra, R. S. (2017). Performance comparison and evaluation of Node.js and traditional web server (IIS). *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*.

M. Castro,P. Druschel, A.-M. K. A. (2002). Scribe : a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*.

Maaroufi, S. (2016). A DYNAMIC MESSAGING ARCHITECTURE FOR VEHICULAR SOCIAL NETWORKS.

Mahdi Ghorbani,Mohammad Reza Meybodi, A. M. S. (2019). An Architecture for Managing Internet of Things based on Cognitive Peer-to-peer Networks. *2019 5th International Conference on Web Research (ICWR)*.

Mehreen Ahmed,Rafia Mumtaz, S. M. H. Z. M. H. S. A. R. Z. M. A. (2020). Distributed Fog Computing for Internet of Things (IoT) Based Ambient Data Processing and Analysis. *Electronics 2020*, 1-20.

Mukesh Kumar, A. K. S. (2019). FDDS : An Integrated Conceptual FDDS Framework for DDS Based Middleware. *2019 International Conference on Communication and Electronics Systems (ICCES)*.

Mutsch, F. (2020). TalkyCars : A Distributed Software Platform for Cooperative Perception among Connected Autonomous Vehicles based on Cellular-V2X Communication. *Institut fur Technik der ¨Informationsverarbeitung (ITIV)*.

Nikolov, N. (2020). Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems. *2020 XXIX International Scientific Conference Electronics (ET)*.

Nuno Apolonia,Stefanos Antaris, S. G. G. P. M. D. (2018). SELECT : A Distributed Publish/Subscribe Notification System for Online Social Networks. *2018 IEEE International*

Parallel and Distributed Processing Symposium, 970-979.

Pardo-Castellote, G. (2003). OMG Data-Distribution Service : architectural overview. *Conference : Military Communications Conference, 2003. MILCOM 2003*.

Patrick Eugster,Pascal Felber, R. G. A.-M. K. (2003). The Many Faces of Publish/Subscribe. *ACM Computing Surveys*.

Patrick Eugster,Pascal Felber, R. G. A.-M. K. (2017). Considerations on IPv6 scalability for the Internet of Things — Towards an intergalactic Internet. *2017 Global Internet of Things Summit (GIoTS)*.

Rausch, T. (2017). Message-oriented middleware for edge computing applications. *the 18th Doctoral Symposium of the 18th International Middleware Conference*.

Roberto Baldoni,Leonardo Querzoni, A. V. (2005). Distributed Event Routing in Publish/Subscribe Communication Systems : a Survey.

Ryohei Banno,Jingyu Sun, M. F. S. T. K. S. (2017). Dissemination of Edge-Heavy Data on Heterogeneous MQTT Brokers. *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*.

ryukn. csv-writer. Repéré à https://www.npmjs.com/package/csv-writer.

Sherif Abdelwahab, B. H. (2016). FogMQ : A Message Broker System for Enabling Distributed, Internet-Scale IoT Applications over Heterogeneous Cloud Platforms. *Networking and Internet Architecture (cs.NI) ;*.

Soumyalatha Naveen, M. R. K. (2019). Key Technologies and challenges in IoT Edge Computing. *Proceedings of the Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC 2019)*.

Sven Akkermans,Rafael Bachiller, N. M. W. J. D. H. M. V. (2016). Towards efficient publish-subscribe middleware in the IoT with IPv6 multicast. *2016 IEEE International Conference on Communications (ICC)*.

Tarkoma, S. (2012). *Publish / Subscribe Systems : Design and Principles*.

Thomas Rausch,Stefan Nastic, S. D. (2018). EMMA : Distributed QoS-Aware MQTT Middleware for Edge Computing Applications. *2018 IEEE International Conference on Cloud Engineering (IC2E)*.

Urs Hunkeler,Hong Linh Truong, A. S.-C. (2008). MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*.

Vinay Setty,Maarten van Steen, R. V.-S. V. (2012). PolderCast : Fast, Robust, and Scalable Architecture for P2P Topic-Based Pub/Sub. *Proceedings of the 13th International Middleware Conference*.

Vinoski, S. (2006). Advanced Message Queuing Protocol. *IEEE Internet Computing*.

Vitenberg, C. C. Y. T. H.-A. J. R. (2015). Weighted Overlay Design for Topic-Based Publish/Subscribe Systems on Geo-Distributed Data Centers. *2015 IEEE 35th International Conference on Distributed Computing Systems*.

Ye Zhao,Kyungbaek Kim, V. N. (2013). DYNATOPS : A Dynamic Topic-based Publish/Subscribe Architecture. *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*, (11), 75-86.

Yuuichi Teranishi,Ryohei Banno, T. A. (2015). Scalable and Locality-Aware Distributed Topic-Based Pub/Sub Messaging for IoT. *2015 IEEE Global Communications Conference (GLOBECOM)*.

Zilhao,Morla, A. (2018). A Modular Tool for Benchmarking IoT Publish-Subscribe Middleware. *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*.