REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA

Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie

Département d'informatique

N° d'ordre : 28RTIC/M2/2017

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours: **Réseaux et Technologies d'Informations et Télécommunications**

EVALUATION DE PERFORMANCE DE PROTOCOLES DANS L'INTERNET DES OBJETS A BASE DE METHODES FORMELLES

Par:

BENAOUNE SIHAM

Soutenu le 06/05/2017, devant le jury composé de :

Boukhlouf djamaa	MCB	Président
Kahloul Laid	MCA	Rapporteur
Tibermacine Ahmed	MCB	Examinateur

2016-2017

Table des matières

Table des matières

Table des matières

Table des figures

Table des figures

Liste des tableaux

Introduction Générale

L'informatique et l'Internet deviennent de plus en plus une nécessité de la vie moderne, au fil du temps, on a intégré l'ordinateur dans différents objets de notre vie quotidienne. De plus, avec l'internet, ces objets peuvent se connecter et communiquer entre eux, en créant des possibilités d'intégration plus directe du monde physique dans les systèmes informatiques, et résultant en une meilleure efficacité, la précision et les avantages économiques en plus réduit l'intervention humaine.Le concept de lier les objets à l'Internet, connu de nos jours comme «Internet of Things [(1)]».

La fiabilité des produits, les questions d'interopérabilité et les failles de sécurité sont des problèmes encore très largement rencontrés dans le domaine de l'IoT. De par la nature des objets connectés forment des systèmes complexes qui rendent ces problèmes à la fois plus cruciaux et plus difficiles à appréhender. Pouvoir certifier efficacement ces produits apparait donc comme une nécessité majeure, le test est sans doute l'une des techniques de validation qui permet de répondre à ce besoin. Malgré le fait qu'il ne puisse apporter de garantie absolue sur l'absence d'imperfection, c'est un moyen efficace et relativement peu coûteux pour apporter un certain degré de confiance sur un système.

Pour assurer une bonne communication dans IoT, plusieurs protocoles ont été proposés. L'implémentation de ces protocoles doit respecter les normes et satisfaire les bonnes propriétés attendues et les méthodes formelles peuvent assurer cette dernière contrainte. Ces méthodes sont basées sur les mathématiques et elles mettent au point des outils informatiques pour faciliter la vérification. Cette dernière vise à certifier qu'un système satisfait bien une propriété donnée en modélisant et en spécifiant respectivement, le système et la propriété.

Le présent travail s'intéresse à l'étude formelle du protocole MQTT (Message Queue Telemetry Transport). Nous avons deux objectifs : la modélisation et la vérification du protocole. Le premier chapitre introduit le contexte et la problématique de la recherche proposée. Le deuxième chapitre concerne les méthodes formelles, dans le Chapitre 3, nous présentons en détail à partir de sa spécification le fonctionnement de MQTT ainsi que les différents aspects susceptibles d'être modélisés et étudiés. Chapitre 4 exposera notre expérimentation de modélisation du protocole MQTT, avec l'outil UPPAAL. Cet outil fait appel au model checker statistique et prend en charge les aspects temporels et stochastiques pour l'évaluation des performances du système.

(1) Nous utilisons tout au long de notre mémoire l'abréviation IoT pour « Internet of Things », qui se traduit en français par l'Internet des objets

Chapitre 1:

L'INTERNET DES OBJETS

I-1- Introduction

L'internet des objets et ses protocoles sont parmi les sujets les mieux financés dans l'industrie et étudiés dans le milieu universitaire. L'évolution rapide de l'Internet mobile, la fabrication de matériel mini, micro-informatique, et la machine à machine (M2M) ont permis aux technologies IoT d'être au sommet de sujets médiatiques, ce qui implique qu'une grande quantité d'argent est investi sur elle par l'industrie et la recherche en plus devrait venir dans les prochaines années. [1]

Les technologies IoT permettent à des choses ou des appareils qui ne sont pas des ordinateurs, d'agir intelligemment et de prendre des décisions de collaboration qui sont bénéfiques pour certaines applications.

Nous présentons dans ce chapitre des protocoles IoT fonctionnant à différentes couches de la pile réseau offerts par des organismes de normalisation. Ces normes ont été proposées au cours de la dernière demi-décennie pour répondre aux besoins actuels et aux besoins futurs de l'IoT.

I-2- Avantages de l'Internet des Objets(IoT)[2]

Les avantages de « IoT » s'étendent à tous les domaines du mode de vie et des affaires. Voici une liste de certains des avantages que l'IoT a à offrir:

I-2-a) Amélioration de l'engagement des clients: Les analyses actuelles souffrent de failles importantes dans l'exactitude; Et comme indiqué, l'engagement demeure passif. IoT aide pour atteindre un engagement plus riche et plus efficace avec le public.

I-2-b) Optimisation de la technologie: Les mêmes technologies et données qui améliorent l'expérience client améliorent également l'utilisation des périphériques et aident à améliorer la technologie. L'IoT déverrouille un monde de données fonctionnelles et de terrain critiques.

I-2-c) Réduction des déchets: L'IoT rend les zones d'amélioration claires. Les analyses actuelles nous donnent un aperçu superficiel, mais l'IoT fournit des informations réelles conduisant à une gestion plus efficace des ressources.

I-2-d) Amélioration de la collecte de données: La collecte de données moderne souffre de ses limites et de sa conception pour une utilisation passive. IoT le brise hors de ces espaces, et les place exactement où les humains veulent vraiment aller pour analyser notre monde. Il permet une image précise de tout.

I-3- Défis de l'Internet des Objets(IoT)[3]

Le développement d'une application réussie IoT n'est toujours pas une tâche facile en raison de multiples défis. Ces défis comprennent: la mobilité, la fiabilité, l'évolutivité, la

gestion, la disponibilité, l'interopérabilité et la sécurité et la vie privée. Dans ce qui suit, nous décrivons brièvement chacun de ces défis.

I-3-a) - **Mobilité**: Les périphériques IoT doivent se déplacer librement et changer leur adresse IP et leurs réseaux en fonction de leur emplacement. De plus, la mobilité peut entraîner un changement de fournisseur de services qui peut ajouter une autre couche de complexité en raison de l'interruption du service et de la modification de la passerelle.

I-3-b) - **Fiabilité**: Le système devrait fonctionner parfaitement et fournir toutes ses spécifications correctement. C'est une exigence très critique dans les applications qui nécessite des réponses d'urgence. Dans les applications IoT, le système doit être très fiable et rapide dans la collecte des données, de les communiquer et de prendre des décisions et éventuellement mauvaises décisions peuvent conduire à des scénarios désastreux.

I-3-c) - **Évolutivité**: L'évolutivité est un autre défi des applications IoT où des millions de périphériques pourraient être connectés sur le même réseau. Gérer leur distribution n'est pas une tâche facile. En outre, les applications IoT doivent être tolérantes à de nouveaux services et périphériques qui se connectent constamment au réseau et, par conséquent, doivent être conçus pour permettre des services et des opérations extensibles.

I-3-d) - **La gestion**: La gestion de tous ces périphériques et le suivi des défaillances, des configurations et des performances de ce grand nombre de périphériques est certainement un défi dans IoT. Les fournisseurs doivent gérer les défauts, la configuration, la comptabilité, la performance et la sécurité de leurs périphériques interconnectés et tenir compte de chaque aspect.

I-3-e) - Disponibilité: La disponibilité d'IoT inclut des niveaux de logiciel et de matériel fournis à tout moment et n'importe où pour les abonnés de service. La disponibilité du logiciel signifie que le service est fourni à toute personne autorisée à l'avoir. La disponibilité matérielle signifie que les périphériques existants sont faciles d'accès et sont compatibles avec la fonctionnalité IoT et les protocoles. En outre ces protocoles doivent être compacts pour être en mesure d'être intégrés dans les dispositifs IOT.

I-3-f) - L'interopérabilité: L'interopérabilité signifie que les dispositifs et les protocoles hétérogènes doivent pouvoir interagir les uns avec les autres. Ceci est difficile en raison du grand nombre de plates-formes différentes utilisées dans les systèmes IoT. L'interopérabilité doit être gérée par les développeurs d'applications et les fabricants de périphériques afin de fournir les

services indépendamment de la plate-forme ou des spécifications matérielles utilisées par le client, ce qui nous mène au point suivant la technologie.

I-3-g) - la technologie: l'Internet des Objets s'appuie sur une variété de technologies et de protocoles existants ou à définir qui, combinés ensemble ouvre la porte à de nouvelles et intéressantes perspectives. Pour illustrer les aspects technologies, on peut présenter la figure suivante:

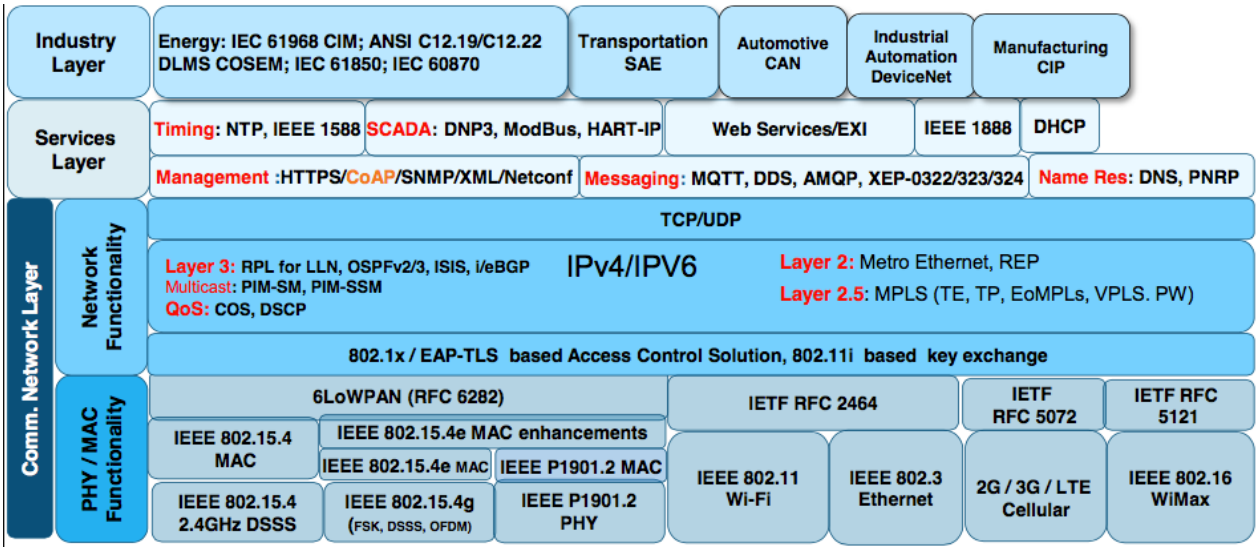


Figure 1-1: **IOT d'un point de vue technique** [4]

D'un point de vue de l'infrastructure de communication, on trouve: les technologies d'accès au réseau et le routage des informations.

Technologies d'Accès: On peut distinguer deux types de technologies d'accès:

- **accès filaires**: essentiellement pour l'industrie avec une adaptation de l'Ethernet pour ces besoins spécifiques,
- **accès sans fils**: via les technologies traditionnelles (Wi-Fi, 3G/4G, Bluetooth, etc.) ou via de nouvelles technologies (802.15.4 ou encore 802.15.6).

Routage: il existe de nouvelles techniques de compression pour le transport d'IPv6 sur ces nouvelles technologies sans fils. Avec le RFC 6550, l'IETF a défini un nouveau protocole de routage adapté pour ces réseaux dits « *Low Power & Lossy Networks* (LLN) ». Ce protocole prend en compte de nouveaux indicateurs pour définir la métrique, comme par exemple, la fiabilité des liens ou le niveau de batterie des nœuds.

I-3-h) - Sécurité des périphériques et services IoT [5]: assurer la sécurité implique la protection des dispositifs IoT et des services de l'accès non autorisé à partir de l'intérieur des dispositifs et à l'extérieur. La sécurité doit protéger les services, les ressources matérielles, les informations et les données, en transition et en stockage. On peut identifier les problèmes clés liés aux périphériques et services IoT suivant:

h-1- La confidentialité des données: représente un problème fondamental dans les dispositifs et les services IoT. Dans le contexte IoT, non seulement l'utilisateur peut accéder aux données mais également à l'objet autorisé. Cela nécessite d'aborder deux aspects importants: d'abord, le mécanisme de contrôle d'accès et d'autorisation et le second mécanisme d'authentification et de gestion d'identité. Le périphérique IoT doit pouvoir vérifier que l'entité (personne ou autre appareil) est autorisée à accéder au service.

h-2 - L'autorisation: permet de déterminer si, lors de l'identification, la personne ou l'appareil est autorisé à recevoir un service. Le contrôle d'accès consiste à contrôler l'accès aux ressources en accordant ou en refusant des moyens en utilisant un large éventail de critères. L'autorisation et le contrôle d'accès sont importants pour établir une connexion sécurisée entre un certain nombre d'appareils et de services. La principale question à traiter dans ce scénario est de rendre les règles de contrôle d'accès plus faciles à créer, à comprendre et à manipuler.

Un autre aspect qui doit être pris en considération lors de la gestion de la confidentialité est l'authentification et la gestion de l'identité. En fait, cette question est critique dans l'IoT, parce que plusieurs utilisateurs, objets / choses et périphériques doivent s'authentifier mutuellement à travers des services fiables. Le problème est de trouver une solution pour gérer l'identité de l'utilisateur, des choses / objets et des dispositifs d'une manière sécurisée.

h-3 - Intégrité: pour fournir des services fiables aux utilisateurs IoT, l'intégrité est une propriété de sécurité obligatoire dans la plupart des cas. Différents systèmes dans l'IoT ont diverses exigences d'intégrité. Par exemple, un système de surveillance à distance des patients aura une vérification d'intégrité élevée contre les erreurs aléatoires dues aux sensibilités de l'information. La perte ou la manipulation des données peuvent se produire en raison de la communication, ce qui peut entraîner la perte de vies humaine.

h-4 -Disponibilité: un utilisateur d'un périphérique (ou le périphérique lui-même) doit pouvoir accéder aux services à tout moment, chaque fois que nécessaire. Différents composants matériels et logiciels dans les dispositifs IoT doivent être robustes de manière à fournir des services même en présence d'entités malveillantes ou de situations défavorables. Différents

systèmes ont des exigences de disponibilité différentes. Par exemple, les systèmes de surveillance des incendies ou de surveillance des soins de santé auraient probablement des exigences de disponibilité plus élevées que les capteurs de pollution routière.

h -5 - Non-répudiation: la propriété de la non-répudiation produit certaines preuves dans les cas où l'utilisateur ou l'appareil ne peut pas refuser une action. La non-répudiation n'est pas considérée comme une propriété de sécurité importante pour la plupart des IoT. Il peut être applicable dans certains contextes, par exemple, les systèmes de paiement où les utilisateurs ou les fournisseurs ne peuvent pas refuser une action de paiement.

I-4 - Les Normes et la standardisation

Comme les dispositifs IoT continuent de saturer la société, la standardisation est essentielle pour atteindre des spécifications universellement acceptées et des protocoles pour une véritable interopérabilité entre les dispositifs IoT et les applications. Les normes publiées aujourd'hui marquent une étape majeure pour l'Internet des objets en offrant la proposition de valeur unique d'une seule plate-forme d'interfonctionnement pour tous les appareils activés, [6]

Près de 140 organismes dans le monde sont aujourd'hui concernés, directement ou indirectement, par la normalisation de la communication M2M. Cette phase de normalisation représente en effet l'un des facteurs cruciaux de l'évolution de l'Internet mobile vers l'Internet des Objets. Il existe des milliers de standards « spécifiques » à des contextes particuliers de l'IoT, parmi eux et en particulier ceux qui sont d'ores et déjà utilisés par l'industrie nous trouvant ceux offerts par Internet Engineering Task Force (**IETF**), Institut des ingénieurs électriciens et électroniciens (**IEEE**), Union internationale des télécommunications (**UIT**) et Global Standard1 (**GS1**), Organization for the Advancement of Structured Information Standards(**OASIS**), comme illustré dans le tableau 1-1[7] suivant:

Emetteur	Norme/ standard	Définition
UIT	UIT-T Y.2060	Concept IoT
	UIT-T Y.2061	Interface machine-application
IEEE	IEEE 802.15.4	Couche liaison
IETF	6LoWPAN	IPv6 over Low Power Wireless Personal Area Networks
	CoAP	Constrained Application Protocol
	RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks
GS1	ONS	Object Naming Service
	EPC	Electronic Product Code
OASIS	MQTT	Message Queue Telemetry Transport
	AMQP	Advanced Message Queuing Protocol
	DDS	Data Diffusion Service

Tableau 1-1: Organisme de normalisation de protocole IOT

I-4 -1- UIT: les deux recommandations**, l'UIT-T Y.2060** qui fournit une vue générale du concept de l'IoT **et l'UIT-T Y.2061** qui décrit les conditions relatives à l'interface machine orientée vers les applications de communications dans l'environnement NGN (réseaux de nouvelle génération).

I-4 -2- IEEE et l'IETF dans le domaine des réseaux de capteurs sur protocole IP. Ces efforts se sont d'abord concrétisés par la proposition d'un modèle de couches sur le modèle OSI ainsi que de protocoles plus adaptés aux réseaux industriels que le modèle TCP/IP sur Ethernet. On trouve notamment :

- au niveau de la couche de Liaison, le standard **IEEE 802.15.4 [8]** plus adapté que l'Ethernet aux environnements industriels difficiles ;
- au niveau Réseau, le standard **6LoWPAN** [9] (IPv6 over Low Power Wireless Personal Area Networks), qui a réussi à adapter le protocole IPv6 aux communications sans fil entre nœuds à très faible consommation. Ainsi 6LowPAN, est une couche d'adaptation qui permet de transporter des paquets IPv6 sur des liens 802.15.4. Sans 6LowPAN IPv6, les protocoles Internet ne fonctionneraient pas dans ces réseaux personnels sans fil à faible consommation d'énergie qui utilise IEEE 802.15.4.
- En ce qui concerne le routage **l'IETF** a publié en 201l le standard **RPL** (IPv6 Routing Protocol for Low-Power and Lossy Networks) ;
- au niveau Application le protocole **COAP** (Constrained Application Protocol) qui tente d'adapter **HTTP**, (beaucoup trop d'échange), aux contraintes des communications entre nœuds à faible consommation, ainsi que (**XMPP**) est un protocole de messagerie qui a été conçu à l'origine pour les applications de chat et d'échange de messages il a été standardisé par **l'IETF**.

I-4 -3- L'organisme GS1: a proposé le système **EPC** (Electronic Product Code) qui est un identifiant individuel unique permettant d'identifier un produit électronique ainsi que l'architecture **EPC global Network** qui définit l'organisation des systèmes d'informations destinés à assurer l'échange des informations EPC au niveau global. L'un de ses principaux composants, **l'ONS** (Object Naming Service), est directement fondé sur le DNS (Domain Name System).

I-4 -4- L'OASIS [10]**:** qui est un consortium sans but lucratif qui oriente les développements et l'adoption de standards « ouverts » pour la société de l'information. Les travaux de ce consortium sur l'internet des objets portent sur les technologies de réseau et de messagerie normalisées comme Message Queue Telemetry Transport (**MQTT**), Advanced Message Queuing Protocol (**AMQP**), ainsi que le service de diffusion de données (**DDS**). Ces protocoles se situent à la couche application, le tableau 1-2 montre une comparaison entre ces protocoles:

Protocoles	UDP / TCP	Architecture	Sécurité et qualité de service	Taille en-tête (octets)	Max Longueur (octets)
MQTT	TCP	Pub / Sub	Tous les deux	2	5
AMQP	TCP	Pub / Sub	Tous les deux	8	-
CoAP	UDP	Req / Res	Tous les deux	4	20 (typique)
XMPP	TCP	Tous les deux	Sécurité	-	-
DDS	TCP / UDP	Pub / Sub	QoS	-	-

Tableau 1-2: Comparaison entre des protocoles IoT [11]

I-5- Sécurité dans les protocoles IoT

La sécurité est un autre aspect des applications IoT qui est critique et peut être trouvé dans tous les couches des protocoles IoT.

Les mécanismes de sécurité intégrés dans les protocoles IoT:

. I-5 -1. MAC 802.15.4 [12]

MAC 802.15.4 offre différents modes de sécurité en utilisant le "Security Enabled Bit" dans le champ de Control de trame dans l'en-tête. Les exigences de sécurité incluent la confidentialité, l'authentification, l'intégrité, les mécanismes de contrôle d'accès et les communications sécurisées synchronisées dans le temps.

. I-5- 2. 6LoWPAN

Le 6LoWPAN ne propose pas par lui-même un mécanisme de sécurité. Cependant, Une extension du protocole SEND (RFC 3971[13]) (*SEcure Neighbor Discovery protocol*) qui permettant de sécuriser le mécanisme de découverte des voisins a été mise en place pour les réseaux 6LoWPAN, appelé **LSEND** (de l'anglais *Lightweight Secure Neighbor Discovery Protocol*, « SEND allégé »[14])

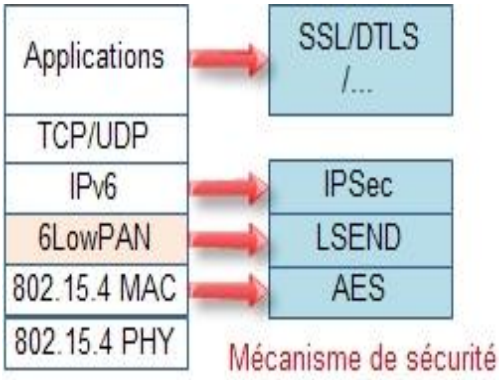


Figure 1-2 Sécurité dans les protocoles IoT [15]

I-5-3.RPL [16]

Offre différents niveaux de sécurité en utilisant un champ " Sécurité" après l'en-tête de message ICMPv6 de 4 octets. Les informations contenues dans ce champ indiquent le niveau de sécurité et l'algorithme de cryptographie utilisé pour chiffrer le message. RPL offre un support pour l'authentification des données, la sécurité sémantique, la protection contre les attaques de replay, la confidentialité et la gestion des clés. Les attaques RPL incluent les attaques de transfert sélectif, Sinkhole, Sybil, Hello Flood, Wormhole, Black hole et Denial of Service.

I-5-4-Couche d'application

Les applications peuvent fournir un niveau de sécurité supplémentaire en utilisant TLS ou SSL comme protocole de couche de transport De plus, des algorithmes de chiffrement et d'authentification de bout en bout peuvent être utilisés pour gérer différents niveaux de sécurité.

I-6- Protocoles de messagerie IoT

D'après une ressente étude publieé dans **IEEE Access, VOLUME 4, nov-2016[17],** sur les performances web pour les protocoles IoT, un ensemble de test est mené pour comparer les temps de latence et le débit des messages des protocoles de messagerie IoT, cette étude a montré que:

- Dans le premier cas de test, ils ont mesuré la latence du transfert de messages d'un éditeur à un abonné. Les temps mesurés sont indiqués dans le Tableau 1-3 et la Figure 1-3. La latence la plus courte est produite par le protocole MQTT, suivi par AMQP, alors que la différence entre XMPP et DDS est négligeable. Pour le cas de test d'un seul nœud capteur, le MQTT a obtenu une latence de seulement 2,53 ms. Les latences augmentent proportionnellement avec l'augmentation de la taille des messages.

Protocol	Number of Sensor Nodes	1 Sensor Node	2 Sensor Nodes	3 Sensor Nodes	4 Sensor Nodes	5 Sensor Nodes
	Message Payload Size [chars]	1475-1477	2896-2901	4316-4321	5734-5746	7157-7169
MQTT	*Ttransfer* [ms]	**2.53**	**3.38**	**3.65**	**3.84**	**4.33**
AMQP	*Ttransfer* [ms]	3.99	4.25	4.61	4.82	4.93
XMPP	*Ttransfer* [ms]	4.11	4.97	5.28	5.89	6.34
DDS	*Ttransfer* [ms]	4.3	5.04	5.48	5.56	5.72

Tableau 1 3: Mesure la latence du transfert de messages des protocoles de messagerie IoT [17]

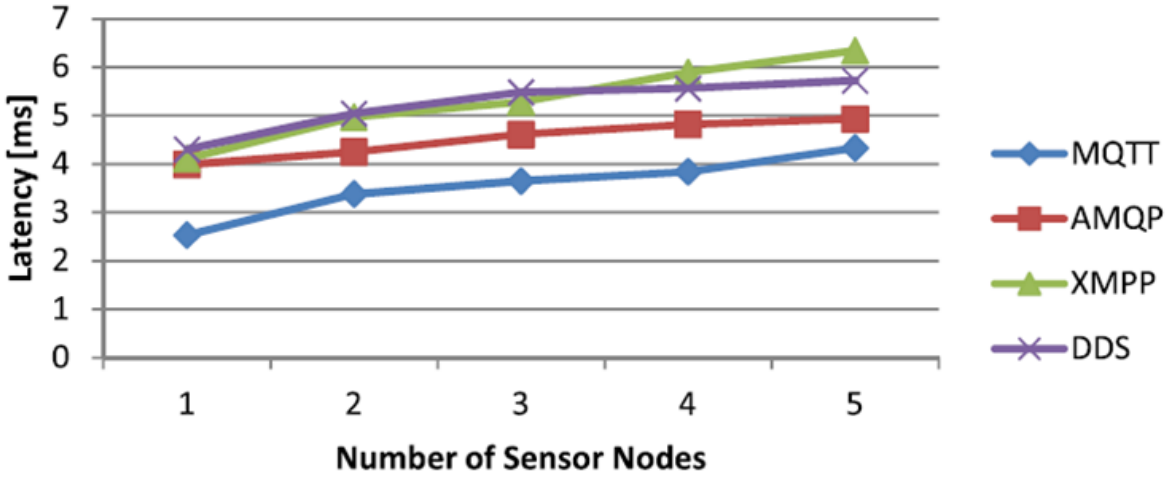


Figure 1-3: Mesure de latence des protocoles de messagerie d'application web IoT [17]

- Afin de minimiser les erreurs en raison de la résolution de la minuterie, ainsi que de mesurer la capacité maximale de chaque protocole, ils ont effectué un autre test dans lequel ils ont mesuré le nombre de messages transférés aller-retour, avec le scénario suivant:

L'éditeur publie un message et à sa réception, l'abonné renvoie immédiatement ce message à l'éditeur. Le processus se poursuit car l'éditeur renvoie également immédiatement le même message à l'abonné, etc. Les résultats montrent le nombre de messages aller-retour passés dans l'intervalle de 100s. Ils ont obtenu les résultats qui sont montrés dans le tableau 1-4 et Figure 1-4.

Ces tests ont en effet des résultats très différents. Dans le cas des protocoles AMQP et XMPP, les valeurs de débit sont alignées sur le test de latence: AMQP atteint un débit de message légèrement supérieur à XMPP, 229.38 msg/s à 187.87 msg/s, et les meilleurs résultats ont été mesurés immédiatement après que le broker de messages a commencé.

Pour le protocole MQTT, le taux de débit des messages dans le cas de test des clients Java est nettement plus élevé que pour les protocoles AMQP et XMPP avec 302,48 msg/s, comparativement à 266,97 msg/s et 196,04 msg/s respectivement.

Protocol	Java – JavaScript clients		Java – Java clients		JavaScript – JavaScript clients	
	Message Throughput Rate [msg/s]	*Ttransmission* [ms]	Message Throughput Rate [msg/s]	*Ttransmission* [ms]	Message Throughput Rate [msg/s]	*Ttransmission* [ms]
MQTT(Mosquitto)	9.85	50,76	302.48	1,65	3,33	150,15
MQTT(HiveMQ)	177.37	2,82	194.60	2,57	**354,64**	**1,41**
AMQP	**229.38**	**2,18**	266.97	1,87	208,97	2,39
XMPP	187.85	2,66	196.04	2,55	195,63	2,56
DDS	X	X	**463.94**	**1,08**	181,88	2,75

Tableau 1-4: Les mesures de taux de débit des messages des protocoles de messagerie d'application web IOT [17].

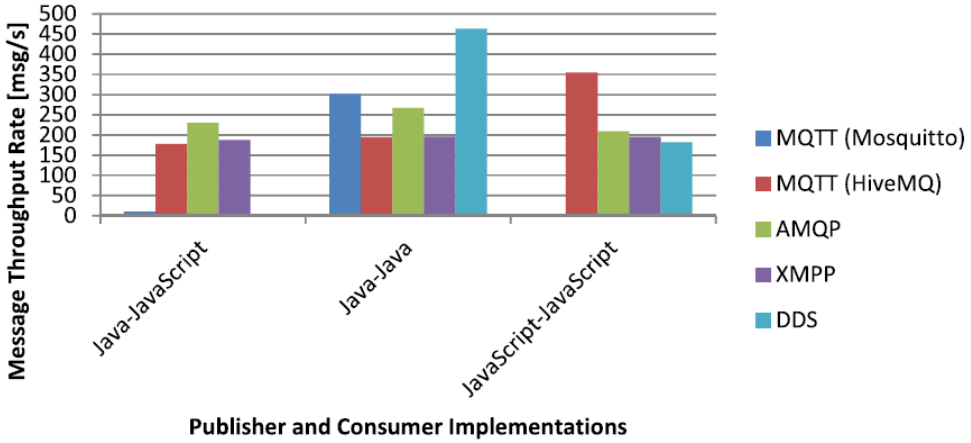


Figure 1-4: Les mesures de taux de débit des messages des protocoles de messagerie d'application web IOT. [17]

Le taux de rendement de message des clients **JavaScript <-> JavaScript** est le plus haut de tous les protocoles éprouvés, atteignant 354.60 Msg/s. Cependant, dans le test de latence, la configuration basée sur HiveMQ (1) montrait une latence sensiblement plus élevée que le test basé sur le courtier Mosquitto(2). Le deuxième résultat inattendu est un taux de transmission de messages de plus de 50% pour le protocole DDS que pour le deuxième meilleur protocole, qui est MQTT, dans la configuration avec des clients Java Java, 463.94 msg/s à 302.48 msg/s.

I-7- Conclusion

L'un des aspects essentiels d'une architecture IoT est celui de la communication entre les divers composants du système global. Il existe bien sûr de multiples moyens de connecter et de faire communiquer les composants en question, mais on assiste dans ce domaine à l'émergence de certains protocoles mieux adaptés que d'autres pour remplir ce besoin. Ainsi, avec son modèle Publish/Subscribe et sa très grande légèreté, le protocole MQTT est actuellement considéré comme l'un des candidats les plus sérieux pour assurer le transport des données au sein des architectures IoT. On en trouve de nombreuses implémentations dans divers langages de programmation, dont beaucoup sont open-source.

Nous avons considéré ce protocole comme étant à la fois suffisamment simple et intéressant pour constituer l'objet de notre étude.

(1) HiveMQ est le courtier MQTT pour l'entreprise connectée. http://www.hivemq.com/

(2) Mosquitto est un iot.eclipse.org projet, est un open source (EPL / EDL autorisé) courtier de messages qui implémente le protocole MQTT versions 3.1 et 3.1.1.

Chapitre 2:

L'ANALYSE DE PERFORMANCE DES SYSTEMES

II-1- Introduction

Ce chapitre introduit l'analyse de performances des systèmes. Il nous a paru indispensable de consacrer un chapitre à la définition des notions et des termes sur lesquels notre travail est basé. Ainsi, nous définissons ce qu'est la performance des systèmes, et nous présentons les méthodes permettant de l'analyser. Nous présentons également dans ce chapitre une méthodologie d'analyse de performance des systèmes basée sur les techniques de model checking.

II-2- Performance des systèmes [18]

La performance d'un système est habituellement considérée comme la qualité de service (QoS) que ce système fournit en mode opérationnel.

II-2-1 -Analyser les performances

Analyser les performances d'un système en mode opérationnel demande de pouvoir prédire la capacité de celui-ci à réaliser ses services, sous l'hypothèse que sa structure demeure intacte, c'est-à-dire qu'il n'est pas soumis à des dysfonctionnements ou des pannes de ses composants. Par ailleurs, analyser la sûreté de fonctionnement d'un système nécessite de représenter les changements qui peuvent affecter sa structure, et comment de telles modifications affectent le système global.

Nous pouvons distinguer deux catégories de méthodes d'analyse de Performance des systèmes:

II-2-1-1- L'expérimentation du système réel [19]

Il s'agit, par exemple, de concevoir un système différent répondant à de nouveaux objectifs, de tester un système dans des conditions anormales de fonctionnement, telles que des pannes de sous-systèmes, ou des surcharges de travail…etc.

L'expérimentation a l'avantage d'être la méthode d'analyse la plus précise, puisqu'elle utilise directement le système réel ou un système équivalent. Mais il s'agit d'une méthode coûteuse, qui n'est pas toujours possible à mettre en œuvre lorsque le système est utilisé, ou du moins, difficile à le faire.

II-2-1-2- La modélisation

Qui consiste à spécifier un modèle d'un système, dont l'analyse s'effectue généralement à l'aide d'un ordinateur. Parmi ces techniques, on distingue la simulation à événements discrets, les techniques de modélisation analytique, et les techniques de modélisation formelle, en particulier le model checking.

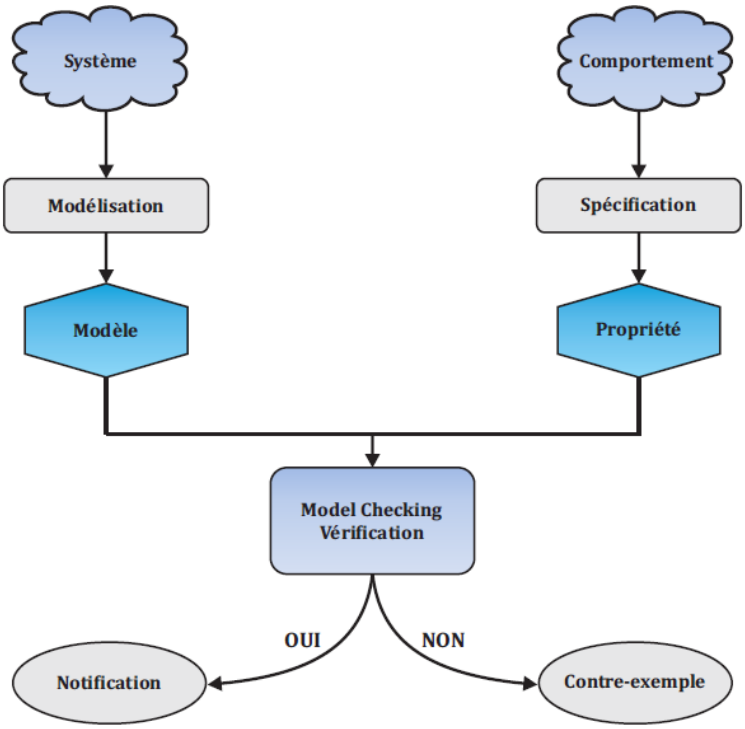


Figure 2-1: Procédure du model checking [20]

II-3- Vérification qualitative et analyse quantitative

A partir d'un modèle, l'analyse de Performance d'un système portent sur [21] :

_ La vérification qualitative

De ses propriétés structurelles et comportementales, telles que l'absence de blocage (vivacité), les invariants du système, les comportements finis ou bornés (stabilité), l'équité, l'inéluctabilité, etc.

_ L'analyse quantitative

C'est-à-dire l'évaluation, de ses paramètres de performance. Elle n'a de sens que si une vérification qualitative a été menée. Il est en effet inutile de vouloir obtenir les performances d'un système qui est bloqué.

La vérification qualitative et l'analyse quantitative se combinent en incluant un comportement stochastique, avec des aspects temps réels. De telles propriétés quantitatives permettent l'évaluation de la performance d'un système, en plus de sa correction. Par ailleurs, un modèle qui comprend des probabilités et des aspects temporels permet d'inclure des exigences quantitatives dans des propriétés liées à la sûreté de fonctionnement, telle que ***« le protocole doit assurer que la probabilité d'envoi d'un message avec succès en au plus 5 secondes est supérieure à 95% ».***

II-4- Techniques de modélisation

II-4-1- La simulation à événements discrets [22]

La simulation à événements discrets consiste à reproduire le comportement d'un système en étudiant une réalisation particulière de son modèle. L'avantage de la simulation est d'offrir une approche très générale permettant d'étudier n'importe quel modèle, du moment que l'outil de simulation est adapté au modèle considéré. Par contre, son inconvénient est de demander beaucoup de temps de calcul machine.

Il existe de nombreux outils de simulation à événements discrets. Parmi eux, citons les simulateurs de réseaux NS-2 , NS-3 (_ NS _ pour network simulator en anglais), OMNeT++ , qui permet de simuler différents types de réseaux dont les réseaux de files d'attente, et OPNET qui est un outil destiné aux études de performances.

II-4-2- Les méthodes analytiques [20]

Les méthodes analytiques proposent d'étudier le comportement d'un système en résolvant les équations mathématiques sous-jacentes à son modèle mathématique. L'intérêt des méthodes analytiques réside principalement dans la résolution des équations généralement peu coûteuse en temps de calcul. Par ailleurs, les méthodes analytiques permettent d'avoir une bonne compréhension du fonctionnement du système, car on est plus en mesure d'analyser certains de ses dysfonctionnements en résolvant son modèle, et donc de proposer des modifications pour les régler.

L'inconvénient des méthodes analytiques est qu'il est généralement nécessaire de faire des hypothèses restrictives sur le système réel pour pouvoir obtenir des modèles exploitables.

II-4-3- Les méthodes formelles [23]

Dans cette section, nous nous intéressons aux méthodes qui permettent d'établir formellement que le modèle d'un système respecte ses spécifications

Avant de présenter les différentes techniques existantes, il nous semble important de préciser les termes de validation et de vérification

***** La **validation :** est le processus qui permet d'assurer que le système répond bien aux attentes du maître d'ouvrage. La **validation** s'applique au processus de développement dans son ensemble pour être sûr que le résultat de ce développement est le produit attendu.

***** La **vérification :** permet de s'assurer qu'un système respecte une série de spécifications. Ce qui différencie ce processus de celui de la validation est le fait que les spécifications vérifiées, si elles ne sont pas validées, peuvent ne pas correspondre aux attentes du maître d'ouvrage. La vérification formelle d'un système nécessite sa modélisation en langage mathématique.

On distingue, dans la littérature scientifique, deux méthodes de vérification formelle généralistes : la preuve et le Model Checking.

II-4-3.1 La construction assistée de preuves (theorem proving)

Avec la preuve (ou preuve de théorème), le système et les propriétés sont modélisés grâce à des formules de logique mathématique. La logique comporte des axiomes (qui sont admis) et des règles d'inférences qui permettent de produire les théorèmes. La réalisation de preuve consiste donc à appliquer les règles d'inférences sur le modèle du système. Ce processus peut être réalisé à la main ou bien de manière semi-automatique grâce à des logiciels assistants de preuve. Parmi eux, citons Coq [24], PVS [25] et HOL [26].

La preuve formelle permet de vérifier des systèmes qui ont un nombre d'états infini (par exemple avec des variables qui peuvent prendre une infinité de valeurs).

Cependant, ce type de méthodes, non automatisées, demande une grande expertise de la part de l'utilisateur. De plus, le processus de preuve, souvent long, laisse la place à l'introduction d'erreurs subtiles [27].

II-4-3-2- La vérification de modèles (model checking)

Cette technique consiste à construire un modèle **M** d'un système **S**, et à spécifier les exigences de performabilité que ce système doit satisfaire sous la forme de propriétés écrites en logique temporelle $\{\phi i\}i\in I$, avant de vérifier automatiquement avec l'aide d'un outil appelé un **model checker,** que ce modèle respecte ces propriétés. Dans le cas contraire, l'outil fournit un contre-exemple. Les modèles et les propriétés sont ***qualitatifs*** ou, si l'outil de vérification utilisé le supporte ***quantitatifs***, exprimant des probabilités, des taux, des durées, des coûts, ou encore des récompenses. Les modèles finis, sont dérivés des systèmes à états-transitions étiquetées (LTS pour labeled transition system). La vérification s'effectue par l'exploration exhaustive des états du modèle de manière algorithmique. Les utilisateurs spécifient généralement les modèles à l'aide de formalismes de haut-niveau permettant d'abstraire les LTSs, par exemple des automates [28], des réseaux de Petri [29], ou encore des algèbres de processus [30].

Les principaux avantages du Model Checking sont [31]

• ***L'exhaustivité de la vérification*** : absolument tous les comportements du modèle sont explorés, cela permet de détecter toutes les violations de spécification.

• ***Le fait que la méthode soit automatique*** : cela évite les erreurs humaines évoquées dans le cas des preuves et permet de se concentrer sur la qualité de la modélisation.

• ***La capacité à fournir des contre-exemples*** : lorsqu'une propriété n'est pas vérifiée, le *Model Checker* est capable de fournir une trace qui permet au concepteur du système de comprendre le problème.

En revanche, la principale limitation du Model Checking est l'explosion combinatoire.

En effet, lorsque le système est complexe, distribué et/ou large échelle, le nombre de comportements possibles devient très important. Il est alors impossible de stocker ou d'explorer

en un temps raisonnable le modèle des comportements du système, ce qui rend la vérification infaisable en pratique.

II-5- Exigences de performance [32]

Les exigences de performabilité d'un système que l'on cherche à vérifier concernent habituellement:

II-5-1- L'accessibilité

L'accessibilité d'une certaine configuration du système, c'est-à-dire que l'état correspondant dans son modèle peut être atteint. Cette propriété peut être exprimée en utilisant l'opérateur temporel **EF**.

II-5-2- L'invariance de propriété

C'est-à-dire que tous les états du modèle étudié satisfont cette propriété.

II-5-3- La **sûreté**

C'est-à-dire que quelque chose de mauvais pour le système ne peut pas se produire. Cette propriété peut être exprimée en utilisant l'opérateur temporel G (toujours).

II-5-4- La **vivacité**

Le concept de vivacité pour s'assurer qu'une action finira par avoir lieu. Ce type de propriété peut être exprimé en logique LTL en utilisant l'opérateur **F** (éventuellement) et l'opérateur **U** (jusqu'à). Cette propriété peut être également exprimée en logique CTL en utilisant l'opérateur **AF**. Exemple : AF GOAL, cette propriété permet de vérifier que le processus va atteindre fatalement son état cible GOAL.

II-5-5- L'équité

C'est-à-dire que quelque chose se répétera infiniment souvent. On distingue l'équité forte (si un processus demande continuellement son exécution, il finira par l'avoir) L'équité peut être exprimée en utilisant la logique temporelle LTL.

II-5-6- L'absence de blocage

C'est-à-dire que le système ne peut pas se trouver dans une configuration à partir de laquelle il ne peut plus évoluer. Une propriété d'absence de blocage peut être exprimée par la formule CTL : AG (EX true).

II-6- Le model checking

II-6- 1- Le model checking classique

II-6- 1-1- Les logiques temporelles (LTL – CTL – CTL*)

Les logiques temporelles sont des formalismes permettant de décrire l'évolution comportementale d'un système au cours du temps. Ces logiques sont plus expressives que le langage naturel, trop ambigu.

- **Logique LTL**: L'idée d'utiliser une logique temporelle pour raisonner sur l'évolution comportementale des programmes concurrents est due à Amir Pnueli dans The temporal logic of programs [33]. Il utilisa une logique temporelle comprenant les opérateurs temporels de base F (un jour) et G (toujours). Enrichie avec X (la prochaine fois), et U (jusqu'à), elle est maintenant connue sous le nom de **LTL (Linear Time Logic)**. Les formules de LTL sont :

$$\Psi ::= true \mid p \mid \neg \Psi \mid \Psi \wedge \Psi \mid \Psi \vee \Psi \mid X\Psi \mid F\Psi \mid G\Psi \mid \Psi \, U \, \Psi$$

Où **p** est une proposition atomique. Le domaine d'interprétation est un ensemble de couples

M, σ associant un modèle **M** du système (c'est-à-dire une structure de Kripke [(1)], et un chemin **σ** dans cette structure. On appelle un chemin dans une structure de Kripke, une séquence infinie.

σ = q0q1q2 . . . d'états q ∈ Q. On note σi le chemin extrait de σ à partir du ième état. Par exemple σ2 = q2q3 . . . . On note σ(k) le k-ième état de σ. La relation de satisfaction
M, σ |= Φ est définie par :

_ M, σ |= true (toujours) ;
_ M, σ |= p ⇔ p ∈ l(σ(0)) (p est vraie dans l'état courant) ;
_ M, σ |= ¬ Ψ ⇔ M, σ |= Ψ est faux ;
_ M, σ |= Ψ 1 ∨ Ψ 2 ⇔ M, σ |=Ψ 1 ou M, σ |= Ψ 2 ;
_ M, σ |= Ψ 1 ∧Ψ 2 ⇔ M, σ |=Ψ 1 et M, σ |= Ψ 2 ;
_ M, σ |= X Ψ ⇔ M, σ1 |= Ψ (cf. figure 2-2 (a)) ;
_ M, σ |= F Ψ ⇔ ∃ k ≥ 0 M, σk |= Ψ (cf. figure 2-2 (b)) ;
_ M, σ |= G Ψ ⇔ ∀ k ≥ 0 M, σk |= Ψ (cf. figure 2-2 (c)) ;
_ M, σ |= Ψ1U Ψ2 ⇔ ∃k≥ 0 M, σk |= Ψ2 et ∀j, 0 ≤ j < k, M, σj |= Ψ 1 (cf. figure 2-2 (d)).

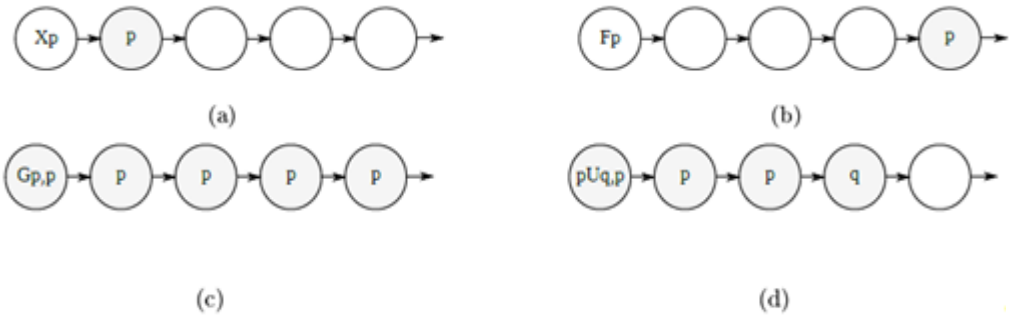


Figure. 2-2: Illustration des opérateurs de LTL.

(1) D'après Saul Aaron Kripke, un logicien et philosophe américain né en 1940 qui a proposé ces structures

- **Logique CTL*:** On distingue dans CTL* des formules d'états (Ψs), interprétées sur les états d'une structure de Kripke, et des formules de chemins (Ψp), interprétées sur ses chemins.

Les formules CTL* sont alors définies par la grammaire suivante :

Ψs ::= true | p | ¬ Ψ s | Ψs ∨ Ψs | Ψs ∧ Ψs | A Ψp | E Ψp

Ψp ::= Ψs | ¬ Ψp | Ψ p ∨ Ψp | Ψp ∧ Ψp | F Ψp | G Ψp | Ψ pU Ψp

Où **p ∈ PA** est une proposition atomique. Le domaine d'interprétation est un couple (M, s) associant une structure de Kripke et un état (pour les formules d'états), et un couple (M, σ) associant une structure de Kripke et un chemin (pour les formules de chemins). On définit
M, s |= par :

1. Les formules d'états (relation |=s) :

_ M, s |=s p ⇔ p ∈ l(s) (p est vraie dans l'état courant) ;
_ M, s |=s ¬ ⇔ M, s |=s est faux ;
_ M, s |=s 1 ∨ 2 ⇔ M, s |=s 1 ou M, s |=s 2 ;
_ M, s |=s 1 ∧ 2 ⇔ M, s |=s 1 et M, s |=s 2 ;
_ M, s |=s Af ⇔ tous les chemins σ partant de s vérifient M, σ |=p f ;
_ M, s |=s Ef ⇔ il existe un chemin σ partant de s qui vérifie M, σ |=p f.

2. Les formules de chemins (relation |=p) :

_ M, σ |=p f ⇔ M, σ(0) |=s f (f étant une formule d'états) ;
_ M, σ |=p ¬ ⇔ M, σ |=p ;
_ M, σ |=p 1 ∧ 2 ⇔ M, σ |=p 1 ou M, σ |=p 2 ;
_ M, σ |=p 1 ∨ 2 ⇔ M, σ |=p 1 et M, σ |=p 2,
_ M, σ |=p X ⇔ M, σ1 |=p ;
_ M, σ |=p F ⇔ ∃k > 0, M, σk |=p ;
_ M, σ |=p G ⇔ ∀k > 0, M, σk |=p ;
_ M, σ |=p 1U 2 ⇔ ∃k > 0 M, σk |=p 2 ∀j, 0 6 j < k, M, σj |=p 1.

- **Logique CTL**. La logique CTL (Computation Tree Logic) est une restriction de CTL* dans laquelle les connecteurs temporels X, F, G et U doivent être directement précédés d'un quantificateur de chemin A ou E. Une formule CTL s'obtient à partir de ¬,∨,∧ et des huit opérateurs AX, EX, AF, EF, AG, EG, AU et EU. Par exemple :

_ EFp : il existe un chemin partant de s qui atteint p (figure 2-3 (c)).

_ AFp : tous les chemins partant de s finissent par atteindre p (figure 2-3 (a)).

_ EGp : il existe un chemin partant de s tel que p est vrai tout au long du chemin, y compris en s (figure 2-3 (d)).

_ AGp : p est toujours vrai en partant de s, y compris en s (figure 2-3 (b)).

L'influence des logiques LTL, CTL, CTL* a été très importante, et elles ont engendré de nombreuses extensions utilisées dans l'industrie. C'est le cas par exemple d'IBM Sugar basé sur CTL, Intel ForSpec basé sur LTL, et PSL (norme IEEE-1850) qui comprend des caractéristiques de CTL*.

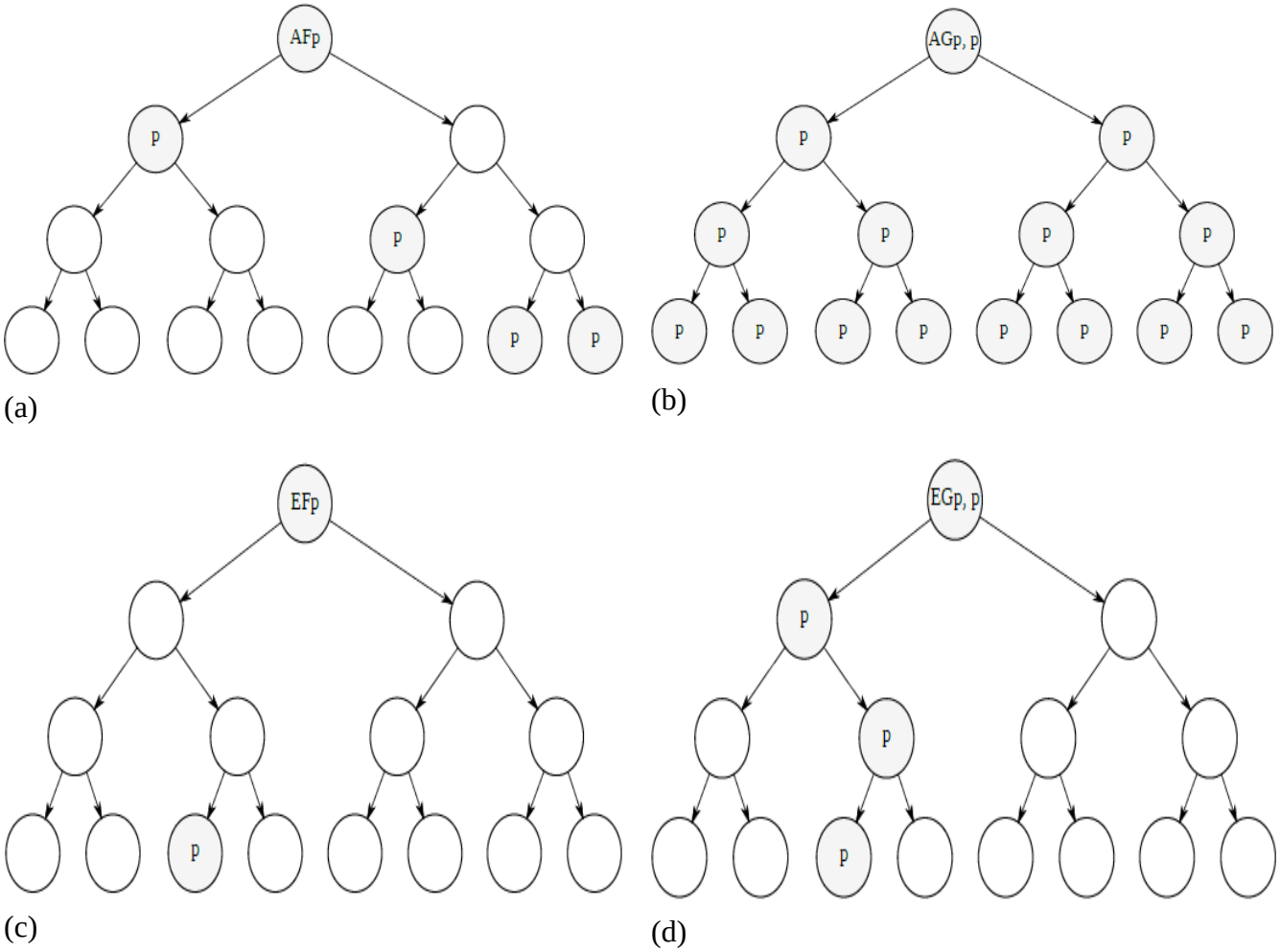


Figure. 2-3: Illustration de quelques formules de CTL.

II-6-2- Le model checking stochastique

Si le système à analyser présente un caractère aléatoire que l'on souhaite prendre en compte, ce système sera modélisé sous la forme d'un processus stochastique. Les processus stochastiques dont les comportements sont discrets sont souvent représentés par des chaînes de Markov à temps discrets (DTMCs pour discrete time Markov chains), ou des processus de décision markoviens (MDPs pour Markov decision processes), si l'on souhaite les analyser selon un temps discret, ou encore des chaînes de Markov à temps continu (CTMCs pour continuous time Markov chains). Il est alors possible d'évaluer la probabilité de satisfaction d'une exigence. Dans ce cas, une logique temporelle permettant d'exprimer des probabilités est nécessaire.

II-6.2-1 Notion de processus stochastique

Un processus stochastique (Xt)t∈T est une famille de variables aléatoires, indexée sur le temps T. Si le temps est discret (T ⊂ N), il s'agit d'un processus à temps discret, alors que lorsque le temps est continu (T ⊂ R), on parle de processus à temps continu. Pour ***t*** fixé, Xt est une variable aléatoire prenant ces valeurs dans un certain espace d'états **Ɛ** suivant des distributions qui peuvent dépendre de ***t***. L'ensemble des états possibles d'un processus stochastique peut, lui-même, être discret ou continu.

II-6-2.2 Chaînes de Markov à temps discret (DTMC)

Les chaînes de Markov à temps discret (DTMCs) modélisent des systèmes à événements discrets, dont les comportements sont déterministes et probabilistes, et sont observés à des intervalles temporels discrets. De manière informelle, une DTMC est une structure de Kripke étendue, où les transitions entre états sont pondérées par leur probabilité d'occurrence respective.

Chaque état représente une configuration possible du système modélisé, et les transitions entre les états représentent les évolutions possibles de la configuration de ce système, c'est-à-dire son comportement. Sachant que chaque transition est pondérée par une probabilité, il est possible de calculer la probabilité d'occurrence d'un comportement particulier. **Parmi** les systèmes modélisables par des DTMCs, nous trouvons:

- les algorithmes stochastiques dont les comportements sont déterministes,
- la transmission d'un message sur une voie de communication non fiable dont la probabilité de panne est connue, etc.

II-6-2.3 Logique temporelles stochastiques (PCTL)

La logique temporelle PCTL (Probabilistic Computation Tree Logic) permet de spécifier des propriétés des chaînes de Markov à temps discrets. Elle a été définie en 1994 par Hans Hansson and Bengt [34]. Elle étend la logique temporelle CTL par l'ajout de l'opérateur probabiliste P, et par les extensions quantitatives des opérateurs A et E.

- **La syntaxe de PCTL est définie par**

_ Des formules d'états : $\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg \Phi \mid P{\sim}p[\Phi]$; et

_ des formules de chemins : $\Psi ::= X\,\Phi \mid \Phi U{\leq}k\,\Phi \mid \Phi\, U\, \Phi$

Où :

_ a est une proposition atomique utilisée pour identifier les états qui intéressent l'analyse;

_ $\sim \in \{<, >, \leq, \geq\}$ et $p \in [0, 1]$ est une probabilité ;

_ $k \in N$;

_ X Φ signifie «Φ est vraie à la prochaine étape » (opérateur « next») ;

_ Φ1U≤k Φ2 signifie « Φ2 est vraie en k étapes et Φ1 est vraie jusqu'à ce point » ;

_ Φ1U Φ2 signifie «Φ2 est finalement vraie et Φ1 est vraie jusque-là ».

II-6-3 Le model checking temporisé

Le model checking temporisé considère les aspects liés au temps dans le comportement des systèmes. Ce peut être un paramètre important. Par exemple, beaucoup de protocoles de communication utilisent des timeout pour permettre aux nœuds qui communiquent de ne pas rester bloqués en attente d'une transmission qui ne vient pas. D'autres systèmes dits temps réels imposent des délais maximaux pour les tâches qu'ils accomplissent avec, éventuellement, leur suspension. Il est alors utile d'analyser les exigences de performabilité en considérant le temps, en particulier les temps de réponse des tâches temps réel.

II-6-3.1 Logiques temporelles temporisées

Il existe de nombreuses extensions de logiques temporelles permettant d'exprimer des contraintes quantitatives sur les délais des actions d'un système à analyser.

II-6-3.1.1 Extension temporisée de CTL (La logique TCTL)

- **La logique TCTL** est interprétée sur des systèmes de transitions temporisées U est assorti d'une contrainte ∼ c où **∼** représente un opérateur de comparaison parmi {<,>=,=,>=,>}, et **c** une constante entière (c ∈ N). Ce qui permet de contraindre les dates d'occurrence de la propriété ψ.
    - **La syntaxe de TCTL est définie par :**

$\varphi, \psi ::= P \mid \neg\ \varphi \mid \varphi \wedge \psi \mid \varphi\ U_{\rhd\lhd c}\ \psi \mid A\ \varphi U_{\rhd\lhd c}\psi$

où P est une proposition atomique.

II-6-3.1.2 Extension temporisée de LTL (La logique MTL)

- **La logique MTL** [35] est une extension temporisée de LTL.
    - **La syntaxe de MTL est définie par :**

$\varphi, \psi ::= P \mid \neg\ \varphi \mid \varphi \wedge \psi \mid \varphi\ U_I\ \psi$

Où P est une proposition atomique et I est un intervalle.

II-6-4- Model checking probabiliste et temporisé PCTL

Les comportements des systèmes probabilistes et temps réel se modélisent avec des automates temporisés probabilistes présentés au paragraphe II-6-2-3. Les propriétés de ces systèmes peuvent se spécifier avec la logique temporelle PTCTL (probabilistic timed CTL) qui est dérivée des logiques **PCTL** [36] et de **TCTL** [37].

La syntaxe de PTCTL est :

$\Phi ::= true \mid a \mid \zeta \mid z.\Phi \mid \Phi \wedge \Phi \mid \neg \Phi \mid P{\sim}p[\Phi U \Phi]$

Où :

_ a est une proposition atomique ;

_ $p \in [0, 1]$;

_ $\sim \in \{<, >, \leq, \geq\}$;

_ $z.\Phi$ est le «quantificateur de gel ».

_ Z est un ensemble de formules d'horloges

_ ζ est la zone sur $X \cup Z$; $\zeta \in Zones(X \cup Z)$, $z \in Z$.

II-7- Outils de Model Checking

Parmi les outils qui possèdent des caractéristiques ayant un pouvoir d'analyse qui permet de vérifier à la fois des propriétés temporelles et probabilistes sur des PTA (Probabilistic Timed Automata), on trouve:

II-7-.1- PRISM

PRISM est un vérificateur de modèle probabiliste, il permet la modélisation et l'analyse des systèmes qui présentent un comportement aléatoire ou probabiliste formelle. Il a été utilisé pour analyser les systèmes de nombreux domaines d'application différents, y compris les protocoles de communication et multimédia, algorithmes distribués aléatoires, protocoles de sécurité, les systèmes biologiques et de nombreux autres. [38]

II-7.2- FORTUNA

Fortuna est le premier outil de model checking probabiliste introduisant la notion de coût. Il peut gérer la combinaison de temps réel ainsi que des caractéristiques probabilistes et de coûts. Ceci est nécessaire pour concevoir de nombreuses applications telles que les protocoles zero-conf, Bluetooth, Fire-wire, IEEE802.11 et les protocoles pour réseaux de capteurs, ou des problèmes d'ordonnancement avec les échecs. [39]

II-7-3- UPPAAL [40]

L'outil Uppaal est certainement le plus connu des outils de vérification temporisés [41]. Il est un environnement d'outils intégrés pour la modélisation, la simulation et la vérification des systèmes temps réel, développé conjointement par la recherche fondamentale en sciences informatiques à l'Université d'Alborg au Danemark et le ministère des technologies de l'information à l'Université d'Uppsala en Suède. Il convient pour les systèmes qui peuvent être modélisés comme une collection de processus non déterministes avec structure de contrôle finie

et horloges valeurs réelles, communiquant par des canaux ou des variables partagées. Domaines d'application typiques incluent les contrôleurs en temps réel et des protocoles de communication, en particulier, les aspects de synchronisation où sont essentiels.

II-8 Conclusion

Le model checking est aujourd'hui une technique reconnue. ***Quels facteurs ont contribué à cette reconnaissance ? E. Allen Emerson*** en note trois [42]. D'abord, son cadre théorique réalisable et compréhensible qui s'appuie sur les logiques temporelles et sur l'algorithmique. Ensuite son automaticité : il permet de détecter la présence d'erreurs de conception des systèmes de façon automatique. Par ailleurs, en ce qui concerne la vérification des programmes informatiques, il sépare nettement leur conception, de leur débogage, le troisième facteur qui a contribué à la reconnaissance du model checking est certainement le gain en puissance des ordinateurs depuis le début des années 1980. Des machines plus puissantes ont permis de supporter des outils de model checking demandant plus de ressources pour fonctionner.

A ces trois facteurs, on pourrait ajouter le facteur recherche et développement, qui est largement orienté vers la résolution du problème majeur du model checking, à savoir l'explosion combinatoire des états. Cette recherche a permis de repousser les limites de l'applicabilité du model checking, et a aussi contribué, avec les trois facteurs précédents, à son succès. Aujourd'hui, la vérification par model checking se fait systématiquement à une grande échelle sur beaucoup de systèmes, tant matériel (en particulier pour vérifier des puces électroniques) que logiciels, y compris des systèmes industriels. Les grandes organisations, depuis les fournisseurs de matériel, jusqu'aux organismes gouvernementaux adoptent la technique du model checking.[43].

Dans ce chapitre nous avons fait un tour d'horizon des déférentes méthodes, formalismes et outils disponibles pour la vérification formelle. Dans le chapitre 4 nous allons spécifier et vérifier un protocole en utilisant les automates temporisés probabilistes comme un langage de spécification et le model-checking statistique sous l'outil UPPAAL 4.1.19.

Chapitre 3:

LE PROTOCOLE MQTT

III-1- Introduction

MQTT (Message Queue Telemetry Transport) est un protocole M2M (Machine-to-Machine) open source [44]. C'est un protocole de messagerie basé sur le publish/subscrib à la fois extrêmement simple et léger idéal pour l'Internet des objets. Il a été inventé par Andy Stanford-Clark (IBM) et Arlen Nipper (Arcom, maintenant Cirrus Link) en 1999, lorsque leur cas d'utilisation était de créer un protocole avec les objectifs suivants: [45]

1- Simple à mettre en œuvre.

2- Fournir une qualité de service de livraison de données.

3- Léger et faible consommation de la bande passante.

Par la suite, la version 3.1 a été libérée de droits en 2010, et le protocole est standardisé par l'OASIS en 2014 dans sa **version 3.1.1**. : [46]. Il devient un standard ISO (ISO/IEC 20922) en 2016 [47] .

Aujourd'hui, avec l'essor de l'IOT, MQTT suscite un intérêt grandissant, dont on peut se rendre compte en observant l'évolution du nombre de recherches pour MQTT sur Google Trends (Figure 3-1).

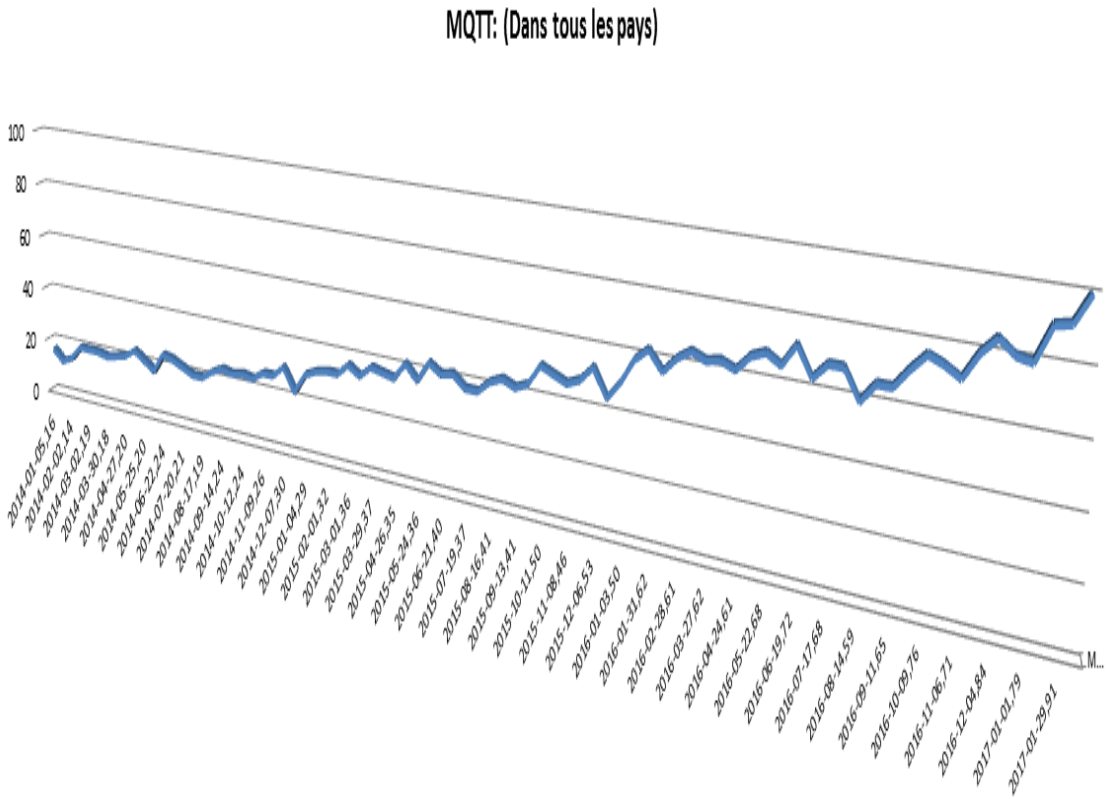


Figure 3-1 Google Trends pour MQTT (18/02/2017)

Les qualités de MQTT font de lui un excellent candidat pour les communications au sein de l'Internet des Objets [48], mais il a également été utilisé avec succès par Facebook pour être au cœur de leur système de messagerie à cause de sa performance [49], comme il est utilisé par amazon web services dans ses passerelles pour permettre la diffusion des données à plusieurs abonnés sur un sujet donné [50].

La norme MQTT décrit le comportement attendu des brokers et clients MQTT, ainsi que les détails du format binaire employé pour les échanges.

La suite de ce chapitre synthétisera l'essentiel de cette norme afin de mieux appréhender la question de la normalisation par la suite. Nous expliquerons en détail le fonctionnement du protocole MQTT. La partie suivante sera consacrée au format binaire défini par la norme pour les paquets MQTT.

III-2- Mode de fonctionnement

MQTT est un protocole de publication et de souscription de message. Les clients ne communiquent pas directement entre eux, ils publient des messages sur un broker ***(appelé aussi courtier),*** les messages sont composés d'un contenu et d'un sujet (topic) [51]. Le broker garde en mémoire le dernier message pour chaque sujet. Les clients qui sont intéressés par les messages d'un sujet peuvent les récupérer en se connectant au broker. L'avantage de cette solution est qu'elle permet à plusieurs clients de communiquer même s'ils ne sont jamais connectés en même temps au broker.

Ce patterne consiste à organiser les messages par sujets (topics) et à gérer leur distribution selon le principe d'abonnement. Ainsi, toute application peut publier ses messages sur les topics de son choix, tandis que les applications intéressées par les messages d'un topic donné peuvent s'abonner pour recevoir tous les nouveaux messages publiés sur ce topic.

III-2-1 Topologies MQTT

Le standard ne définit pas de topologie particulière d'utilisation, la plus simple étant celle avec un seul broker et un certain nombre de clients. D'autres organisations restent évidement possible, avec en particulier les systèmes à base de bridge dans lesquels plusieurs brokers sont utilisés, chacun étant client des autres brokers, comme sur la Figure .3-2.

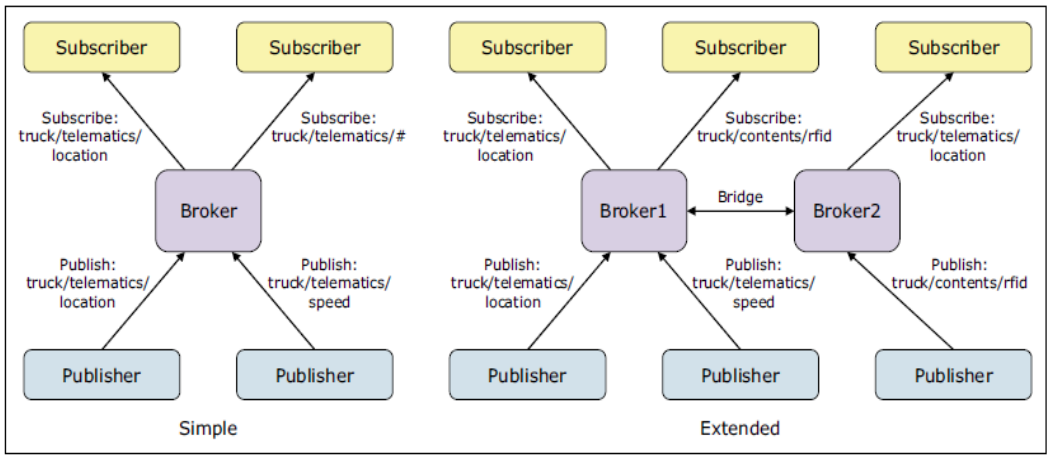


Figure .3-2 – Exemples de topologies MQTT [52]

III-2-2 Fonctionnement

III-2-2-1 Connexion et Déconnexion

MQTT se sert de connexions persistantes entre les clients et le broker, et exploite pour cela les protocoles de réseau garantissant un bon niveau de fiabilité comme TCP.

Avant de pouvoir envoyer des commandes de contrôle, un client doit au préalable **s'enregistrer** auprès du broker, ce qui se fait avec la commande **CONNECT**.

Divers **paramètres** de connexion peuvent alors être échangés, comme les identifiants du client ou encore le mode de persistance souhaité. Le broker doit confirmer au client que son **inscription** a bien été prise en compte, soit indiquer qu'une erreur est survenue en renvoyant un **CONNACK** accompagné d'un code de retour.

Il existe une commande **PINGREQ** permettant de faire savoir au broker que le client est toujours actif, le broker répondra de son côté avec un **PINGRESP** pour indiquer au client que la connexion est toujours active.

Lorsque le client veut se déconnecter, il envoie au préalable une commande **DISCONNECT** au broker pour lui faire part de son intention. Dans le cas contraire, le broker considérera la déconnexion comme anormale et agira en conséquence, [53] en envoyant le message de volonté **WILL** au nom du client à tous les abonnés. Un message que Le client peut spécifier et conservera l'indicateur dans la charge utile de message CONNECT.

III-2-2-2 Abonnements et Publications

Chaque message publié est nécessairement associé à un sujet, ce qui permet sa distribution aux abonnés. Les sujets peuvent être organisés en hiérarchie arborescente, ainsi les abonnements peuvent porter sur des motifs de filtrage que nous détaillerons plus loin.

La gestion des abonnements est très simple et consiste en trois commandes essentielles :

1- **SUBSCRIBE** Permet à un client de s'abonner à un topic (voir section **III-2-3**), une fois abonné il recevra par la suite toutes les publications concernant ce sujet. Un abonnement définit également un niveau de qualité de service, dont il sera question plus bas (section **III-2-4**).
La bonne réception de cette commande est confirmée par le broker par un **SUBACK** portant le même identifiant de paquet.

2- **UNSUBSCRIBE** Donne la possibilité d'annuler un abonnement, et ainsi ne plus recevoir les publications ultérieures. La bonne réception de cette commande est confirmée par le broker par un **UNSUBACK** portant le même identifiant de paquet.

3- **PUBLISH** Initié par un client, permet de publier un message qui sera transmis par le broker aux abonnés éventuels. La même commande sera envoyée par le broker aux abonnés pour délivrer le message.

Si la qualité de service requise est supérieure à zéro, des messages seront échangés pour confirmer la prise en charge de la publication.

III-2-3- Topic et motifs de filtrage

Un sujet est une chaîne UTF-8, qui est utilisé par le broker pour filtrer les messages pour chaque client connecté. Un sujet est constitué d'un ou plusieurs niveaux de sujet. Chaque niveau de sujet est séparé par une barre oblique. Voici quelques exemples de sujets:

sport/tennis/player1/score/wimbledon

myhome / RdC / living / température

5ff4a2ce-e485-40f4-826c-b1a5d81be9b6 / statut

Allemagne / Bavière / voiture / 2382340923453 / latitude

A noter que chaque sujet doit avoir au moins 1 caractères pour être valide et il peut également contenir des espaces. Aussi un sujet est sensible à la casse, ce qui rend **myhome / température** et **MyHome / Température** deux sujets individuels. En outre, la barre oblique seule est un sujet valable. Un client peut souscrire à plus de sujets à la fois comme illustré dans la figure qui suit:

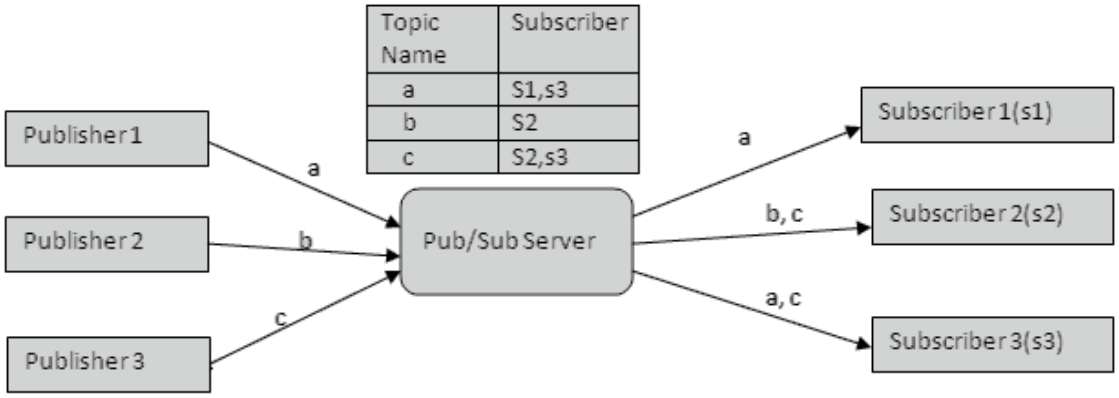


Figure 3-3: Souscription au topic

Afin de proposer un système de filtrage efficace sur les sujets, il est possible de définir une arborescence à l'aide du séparateur /.

Deux jokers sont réservés pour représenter un et plusieurs niveaux d'arborescence:

- + représente un niveau d'arborescence, ainsi T1/T2/T3 peut être mis en correspondance avec divers filtres tels que T1/T2/+, T1/+/T3 ou T1/+/+.
- # représente autant de niveaux que possible, et ne peut être utilisé qu'à la fin d'un motif de filtrage ; ainsi T**1/#** permettra de filtrer tous les sujets dont le premier niveau est T1, et # permet de recevoir tous les sujets publiés par le broker (A l'exception des sujets spéciaux commençant par $).

III-2-4- Qualité de service [54]

Trois niveaux de qualité de service (QoS) sont définis pour la publication des messages:

QoS 0: Livraison *une fois maximum*.

Les messages sont envoyés en fonction des capacités maximum du réseau TCP/IP sous-jacent. Aucune réponse n'est attendue. Aucune sémantique de relance n'est définie dans le protocole. Par conséquent, le message ne parvient pas du tout ou une seule fois au broker de destination. Le niveau QoS 0 est également connu comme *fire and forget*.

QoS 1: Livraison *au moins une fois*.

L'arrivée d'un message QoS 1 au broker est reconnue. En cas d'échec identifiable de la liaison ou de l'unité d'envoi, ou bien après une certaine période de temps sans réception du message d'accusé de réception, l'expéditeur envoie à nouveau une copie du message. Par conséquent, le message est sûr d'arriver, mais il peut le faire plusieurs fois.

QoS 2: Livraison *exactement une fois*.

Pour le niveau QoS 2, d'autres flux de protocoles sont utilisés au-delà de QoS 1 pour que des messages ne soient pas envoyés en double à l'application de réception. Il s'agit du niveau de service le plus élevé qui sert lorsque des messages en double ne sont pas appropriés. Il existe évidemment des conséquences en matière de trafic réseau, mais cet impact est souvent acceptable sachant l'importance du contenu du message.

La qualité de service peut être sélectionnée message par message, ce qui permet la publication des messages d'importance mineure avec le niveau QoS 0 et la distribution des messages plus importants avec QoS 2.

Ces niveaux sont mis en œuvre par des échanges supplémentaires entre l'expéditeur et le récepteur, et plus la qualité demandée est élevée, plus il faudra d'échanges pour valider une publication comme illustré dans la figure.

Pour tous les niveaux supérieurs à zéro, un identifiant est associé au message pour permettre son suivi, MQTT prévoyant la possibilité d'avoir au plus 65535 messages en attente (l'identifiant de message tenant sur 16 bits)

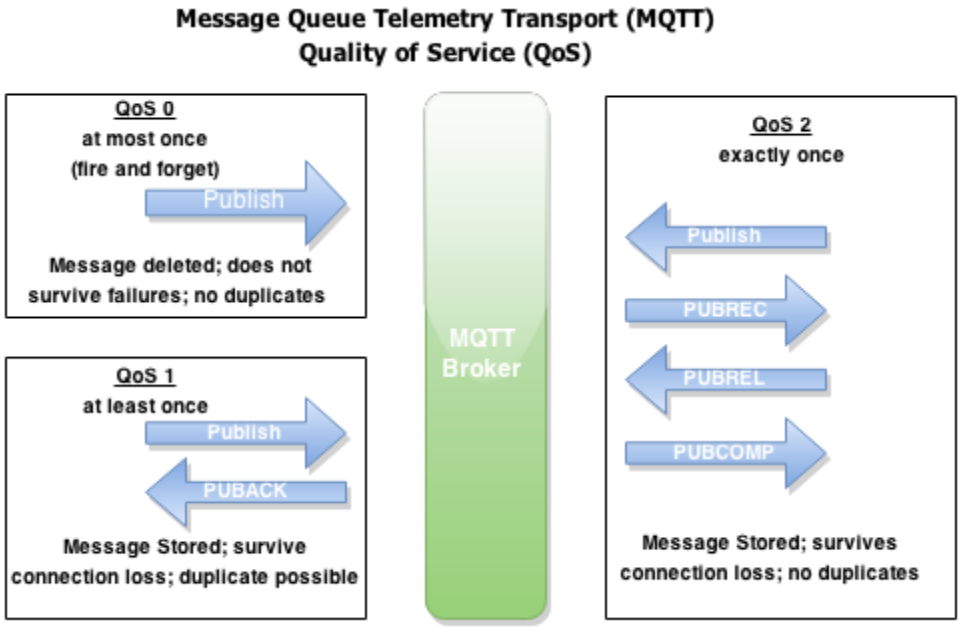


Figure 3-4: Niveaux de Qualité de Service MQTT [55]

III-2-5- Dernière volonté et testament (Last Will & Testament)[56]

Permet de notifier une déconnexion anormale, ils ont défini lors de la connexion.

*** Dernière volonté (Last Will)***:*

Dans le protocole MQTT parce qu'il n'y a pas de file d'attente le message n'est pas mis en cache. Pour éviter ce problème, les utilisateurs peuvent définir une (*retenue dans le* drapeau) lors de la publication. Ce drapeau invite le broker pour stocker le message et à l'offrir aux abonnés futurs dès leur inscription comme dernière publication connue pour un sujet donné.

*** Testament:**

Est une caractéristique importante de MQTT. Un client indique au broker au cours de la configuration de la connexion que: «En cas de ma mort, s'il vous plaît envoyer le message suivant pour ce sujet à mes abonnées». Si le broker détecte alors que ce client est plus en vie, il le fait comme il a été dit et publie le message pour les abonnées pour le sujet désiré.

III-2-6- Sécurité [57]

Trois concepts sont fondamentaux pour la sécurité MQTT : identification, authentification et autorisation. ·

- **L'identification:** consiste à nommer le broker et le client auquel on donne des droits d'accès.
- **L'authentification:** cherche à prouver mutuellement l'identité du client et du broker.
- **L'autorisation**: consiste à gérer les droits du client.

Le broker MQTT s'identifie auprès du client avec son adresse IP et son certificat numérique. Le client MQTT utilise le protocole SSL pour authentifier le certificat envoyé par le broker.

Un client authentifie un broker à l'aide du protocole SSL.

Un broker MQTT authentifie un client à l'aide du protocole SSL, d'un mot de passe, ou des deux.

L'autorisation ne fait pas partie du protocole MQTT. Elle est fournie par les brokers MQTT. Ce qui est autorisé ou non dépend de ce que fait le broker.

III-3- Description informelle du protocole [1]

Dans cette section, nous définissons le format binaire des paquets spécifié par la norme MQTT.

III-3-1- Représentations de données

III-3-1-1 Bits

Les bits dans un octet sont étiquetés de 7 à 0. Le bit numéro 7 est le bit le plus important, le bit le moins significatif est affecté au bit 0.

III-3-1-2 Valeurs de données entières

Les valeurs de données entières sont 16 bits: l'octet d'ordre supérieur précède l'octet d'ordre inférieur. Cela signifie qu'un mot de 16 bits est présenté sur le réseau comme octet le plus significatif **(MSB),** suivi de l'octet le moins important **(LSB).**

III-3-1-3 chaînes codées en UTF-8

Les chaines de caractères doivent toutes être encodées en utf8, et préfixées par leur longueur sur deux octets, ces chaines sont donc limitées à une longueur de

$2^{16} - 1 = 65535$ octets.

III-3-2- Structure d'un paquet de contrôle MQTT

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

Un paquet de contrôle MQTT comprend jusqu'à trois parties, toujours dans l'ordre suivant:

En-tête fixe, En-tête variable, Charge utile,

III-3-2-1- Entête fixe

Tous les packets MQTT commencent par un format d'entête fixe, contenant le code de commande MQTT (sur 4 bits), un drapeau de 4 bits et la remaining length.

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

Figure 3-5: Entête fixe du protocole MQTT.

(1) Tous les tableaux de cette section sont extraits du ref -46.

III-3-2-1-1 Control Packet type:

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Client request to connect to Server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server/Server to Client	Publish message
PUBACK	4	Client to Server/Server to Client	Publish acknowledgment
PUBREC	5	Client to Server/Server to Client	Publish received (assured delivery part 1)
PUBREL	6	Client to Server/Server to Client	Publish release (assured delivery part 2)
PUBCOMP	7	Client to Server/Server to Client	Publish complete (assured delivery part 3)
SUBSCRIBE	8	Client to Server	Client subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server	Client is disconnecting
Reserved	15	Forbidden	Reserved

Tableau 3.1 – Type des paquets MQTT

III-3-2-1-2- remaining length: Une particularité du champ **remaining length** est qu'il est de longueur variable, et peut utiliser de 1 à 4 octets en fonction de la taille encodée c'est-à-dire : La taille maximale d'un paquet MQTT est donc de 256 Mo.

III-3-2-1-3- Flag: Les bits restants [3-0] de l'octet 1 de l'en-tête fixe contiennent des drapeaux spécifiques à chaque type de paquet de contrôle MQTT. Lorsqu'un bit de drapeau est marqué comme «Réservé» , il est réservé pour une utilisation future**.**

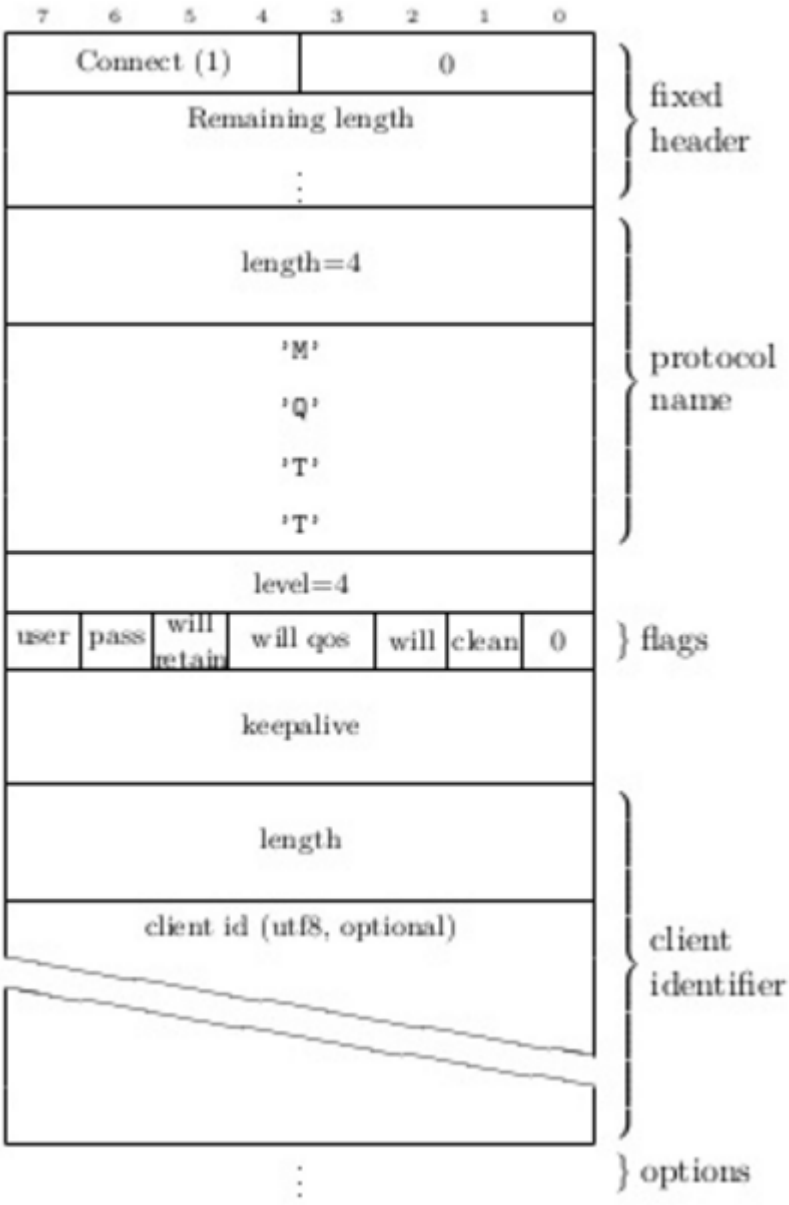
Control Packet	Fixed header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT 3.1.1	DUP[1]	QoS[2]	QoS[2]	RETAIN[3]
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved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

Tableau 3-2: Description du drapeau dans l'entête fixe du protocole MQTT

Voir la section III-3-2-4- pour une description des drapeaux DUP, QoS et RETAIN dans le paquet de contrôle PUBLISH.

III-3-2-2- CONNECT

L'entête de **CONNECT** dans la version 3.1.1 se présente sous la forme suivante :



	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0
Protocol Level									
	Description	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
byte 7	**Level (4)**	0	0	0	0	0	1	0	0
Connect Flags									
byte 8	**User Name Flag (1)** **Password Flag (1)** **Will Retain (0)** **Will QoS (01)** **Will Flag (1)** **Clean Session (1)** ***Reserved* (0)**	1	1	0	0	1	1	1	0
Keep Alive									
byte 9	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
byte 10	Keep Alive LSB (10)	0	0	0	0	1	0	1	0

Figure .3-6: Format d'un paquet CONNECT

Tout d'abord le nom du protocole (préfixé par sa taille), puis le niveau du protocole, des drapeaux indiquant la présence ou non de paramètres optionnels ainsi que la persistance de la session (clean indique que la session est non persistante, et constitue en même temps une directive pour effacer une éventuelle session existante), et enfin la valeur du keepalive

III-3-2-2-1- Level

La valeur non signée de 8 bits qui représente **le niveau de révision du protocole** utilisé par le Client.. La valeur du champ Level de protocole pour la version 3.1.1 du protocole est **4 (0x04)**.

Le broker DOIT répondre au paquet CONNECT avec un code de retour CONNACK 0x01 (niveau de protocole inacceptable), puis déconnecter le client si le niveau de protocole n'est pas pris en charge par le broker

III-3-2-2 2- Connect Flags

L'octet Connect Flags contient un certain nombre de paramètres spécifiant le comportement de la connexion MQTT. Il indique également la présence ou l'absence de champs dans la charge utile.

- Le broker doit valider que le drapeau **Reserved** dans le paquet de Control **CONNECT** est mis à **zéro** et déconnecter le client s'il n'est pas zéro.

III-3-2-2 -2-1 Nom de l'utilisateur

Position: bit **7** de l'octet Connect Flags.

- Si le drapeau de nom d'utilisateur est défini sur 0, un nom d'utilisateur NE DOIT PAS être présent dans la charge utile.
- Si le drapeau de nom d'utilisateur est défini sur 1, un nom d'utilisateur DOIT être présent dans la charge utile.

III-3-2-2-2-2- Indicateur de mot de passe

Position: bit **6** de l'octet Connect Flags.

- Si le drapeau du mot de passe est défini sur 0, un mot de passe NE DOIT PAS être présent dans la charge utile.
- Si le drapeau de mot de passe est défini sur 1, un mot de passe DOIT être présent dans la charge utile.
- Si le drapeau de nom d'utilisateur est défini sur 0, l'indicateur de mot de passe DOIT être réglé sur 0.

III-3-2-2-2-3- Will Retain

Position: bit **5** de l'octet Connect Flags.

Ce bit spécifie si le message de volonté doit être conservé lorsqu'il est publié.

- Si le drapeau Will est défini sur 0, alors le drapeau Will Retain doit être réglé sur 0.
- Si le drapeau de volonté WILL est défini sur 1:
  - Si Will Retain est défini sur 0, le broker DOIT publier le message Will comme un message non retenu.
  - Si Will Retain est défini sur 1, le broker DOIT publier le message Will comme message conservé.

III-3-2-2-2-4- Will QoS

Position: bits 4 et 3 de l'octet Connect Flags

Ces deux bits spécifient le niveau QoS à utiliser lors de la publication du message Will.

Si le drapeau de Will est défini sur 0, alors la QoS de will DOIT être réglée sur 0 (0x00).

Si le drapeau Will est défini sur 1, la valeur de Will QoS peut être 0 (0x00), 1 (0x01) ou 2 (0x02). Il NE DOIT PAS être 3.

III-3-2-2-2-5- Will (volonté)

Position: bit 2 de l'octet Connect Flags.

Si le drapeau Will est défini sur 1, cela indique que, si la demande de connexion est acceptée, un message de volonté DOIT être stocké sur le broker et associé à la connexion réseau. Le message de volonté DOIT être publié lorsque la connexion réseau est fermée par la suite, sauf si le message de volonté a été supprimé par le broker lors de la réception d'un paquet DISCONNECT.

Les situations dans lesquelles le message de volonté est publié comprennent, sans s'y limiter:

- Une erreur d'E / S ou une panne de réseau détectée par le broker.
- Le Client ne parvient pas à communiquer dans le temps de Keep Alive.
- Le client ferme la connexion réseau sans d'abord envoyer un paquet DISCONNECT.
- Le broker ferme la connexion réseau en raison d'une erreur de protocole.

- Si le drapeau de volonté est défini sur 1, les champs Will QoS et Keep Retain dans les Flags Connect seront utilisés par le broker et les champs Will Topic et Will Message DOIVENT être présents dans la charge utile.

Le message de volonté DOIT être supprimé de l'état de session stocké dans le broker une fois qu'il a été publié ou le broker a reçu un paquet DISCONNECT du client

- Si le drapeau de volonté est défini sur 0, les champs Will QoS et Keep Retain dans les drapeaux de connexion DOIVENT être mis à zéro et les champs Will Topic et Will Message NE DOIVENT PAS être présents dans la charge utile .
- Si le drapeau de volonté est défini sur 0, un message de volonté NE DOIT PAS être publié lorsque cette connexion réseau se termine.
- Le broker DEVRAIT publier Will Messages rapidement. Dans le cas d'un arrêt ou d'une panne du broker, le broker PEUT différer la publication des messages de volonté jusqu'à un redémarrage ultérieur. Si cela se produit, il peut y avoir un délai entre le moment où le broker a éprouvé une panne et un message de volonté en cours de publication.

III-3-2-2-2-6- clean Session

Position: bit 1 de l'octet Connect Flags. Ce bit spécifie la gestion de l'état Session.

Le drapeau clean session indique au broker, que le client souhaite établir une session persistante ou non. Le client et le broker peuvent stocker l'état de session pour permettre une messagerie fiable de continuer à travers une séquence de connexions réseau. Ce bit est utilisé pour contrôler la durée de vie de l'état Session.

- Si CleanSession est défini sur 0, le broker DOIT reprendre les communications avec le client en fonction de l'état de la session en cours (identifié par l'identifiant du client). S'il n'y a pas de session associée à l'identifiant client, le broker DOIT créer une nouvelle session. Le client et le broker DOIVENT enregistrer la session après que le client et le broker soient déconnectés. Après la déconnexion d'une session qui avait CleanSession réglé sur 0, le broker DOIT stocker plus de messages QoS 1 et QoS 2 qui correspondent aux souscriptions que le client avait au moment de la déconnexion dans le cadre de l'état Session. Il PEUT également stocker des messages QoS 0 répondant aux mêmes critères.
- Si CleanSession est défini sur 1, le Client et le Broker DOIVENT renoncer à toute Session précédente et en créer une nouvelle. Cette session dure aussi longtemps que la connexion réseau. Les données d'état associées à cette session NE DOIVENT PAS être réutilisées lors d'une session ultérieure.

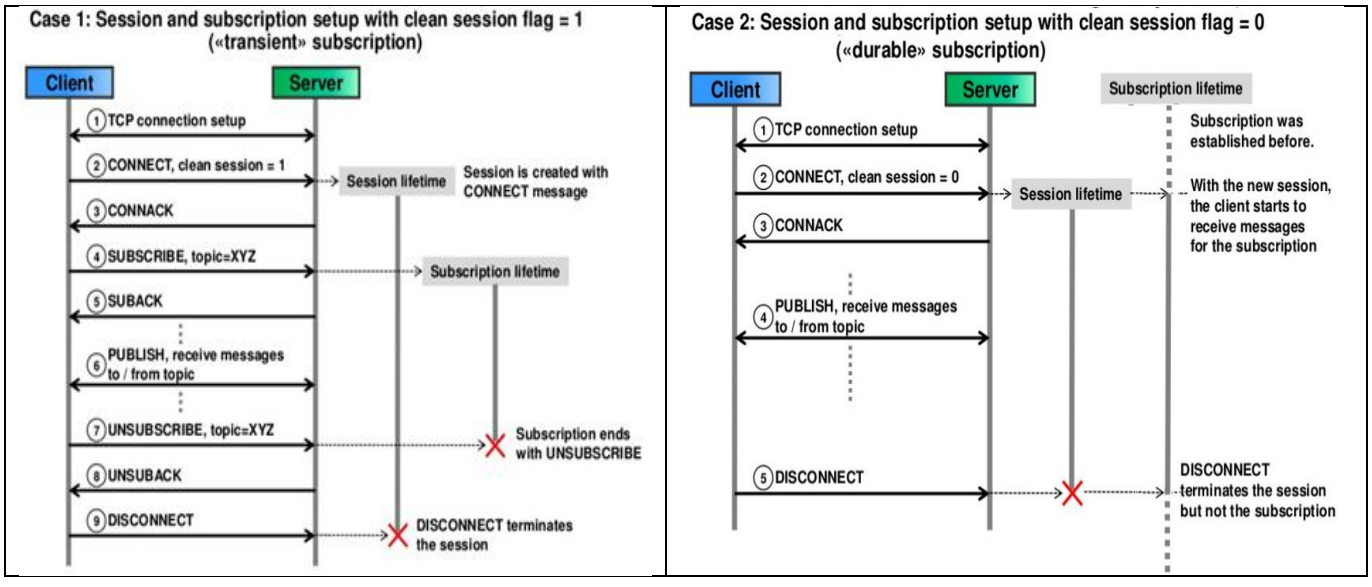


Figure 3-7: Exemple d'utilisation du drapeau clean session [58]

III-3-2-2-3- Keep Alive

Keep Alive est un intervalle de temps mesuré en secondes. Exprimé en tant que mot de 16 bits, c'est l'intervalle de temps maximum qui peut s'écouler entre le point auquel le client finit de transmettre un paquet de contrôle et le point qu'il commence à envoyer le prochain.

Il revient au client de s'assurer que l'intervalle entre les paquets de contrôle envoyés ne dépasse pas la valeur Keep Alive. En l'absence d'envoi d'autres paquets de contrôle, le client DOIT envoyer un paquet PINGREQ.

Le client peut envoyer PINGREQ à tout moment, indépendamment de la valeur Keep Alive, et utiliser le PINGRESP pour déterminer si le réseau et le broker fonctionnent.

Si la valeur Keep Alive est non nulle et que le broker ne reçoit pas un paquet de contrôle du client en une fois et demi la période de temps de Keep Alive, il DOIT ANNULER la connexion réseau au client comme si le réseau avait échoué.

Si un client ne reçoit pas un paquet PINGRESP dans un délai raisonnable après avoir envoyé un PINGREQ, il DOIT fermer la connexion réseau au broker.

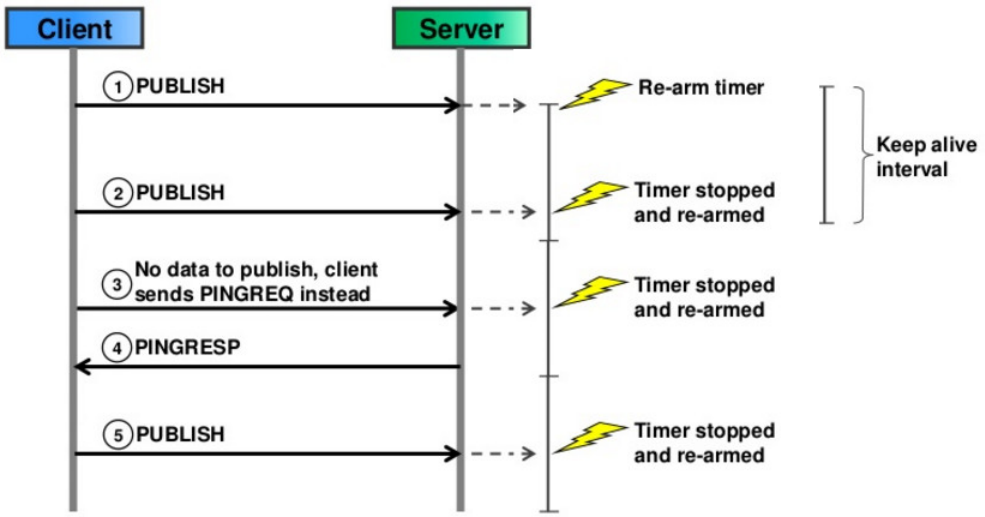
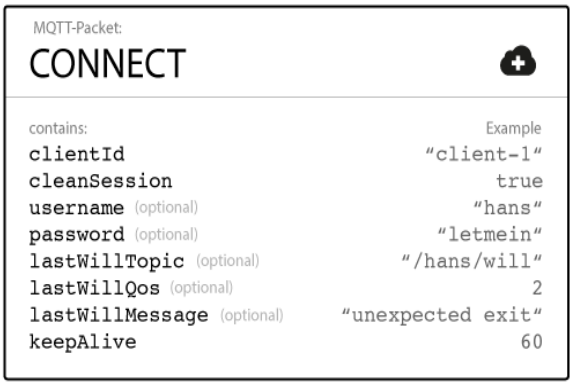


Figure 3-8: Exemple d'utilisation du drapeau Keep alive [58]

III-3-2-2-4 Payload (Charge utile) : La charge utile du paquet CONNECT contient un ou plusieurs champs préfixés en longueur, dont la présence est déterminée par les indicateurs dans l'en-tête de la variable. Ces champs, le cas échéant, DOIVENT apparaître dans l'ordre: Identifiant du client, Nom d'utilisateur, Mot de passe, Sujet (Will topic), Will Message,

Exemple: de paquet MQTT- CONNECT [59]



III-3-.2-3- CONNACK

L'entête du CONNACK est constitué d'un bit (SP) pour indiquer si le broker possède déjà une session pour ce client, dans le cas où le client demande à restaurer une session existante, et d'un code de retour sur un octet (seuls les codes de 0 à 5 étant actuellement définis comme illustré dans le tableau 3-3).

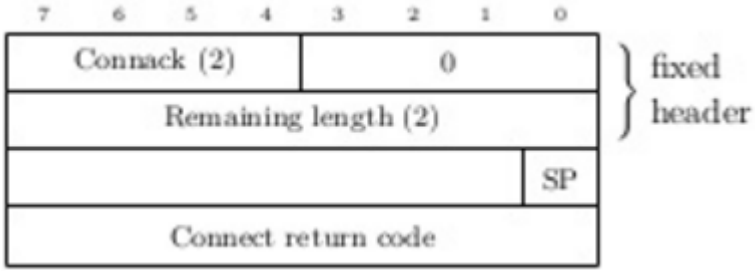


Figure 3-9: Format d'un paquet CONNACK

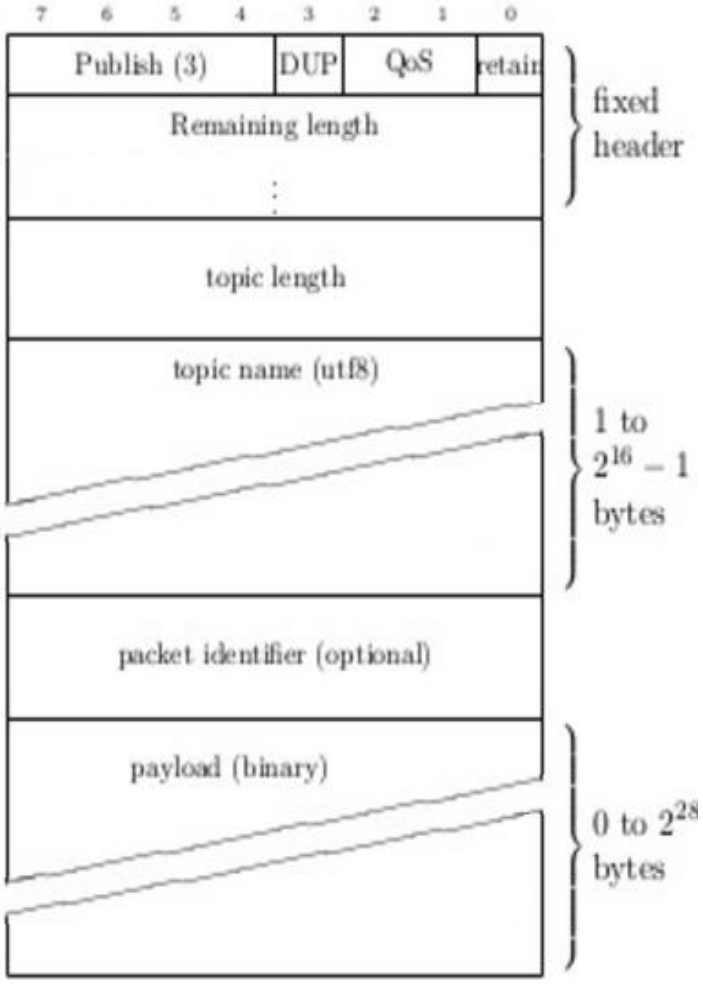
III-3-.2-3-1 : Les valeurs du code de retour

Valeur	Réponse du code de retour	Description
0	0x00 Connexion acceptée	Connexion acceptée
1	0x01 Connexion rejetée, Version de protocole inacceptable	Le broker ne prend pas en charge le niveau du protocole MQTT demandé par le client
2	0x02 Connexion refusée, identifiant rejeté	L'identifiant du client est correct UTF-8 mais pas autorisé par le broker
3	0x03 Connexion refusée, broker indisponible	La connexion réseau a été effectuée mais le service MQTT n'est pas disponible
4	0x04 Connexion refusée, mauvais nom d'utilisateur ou mot de passe	Les données dans le nom d'utilisateur ou le mot de passe sont mal formées
5	0x05 Connexion Refusée, non autorisée	Le Client n'est pas autorisé à se connecter
6-255		Réservé pour une utilisation future

Tableau 3-3: Les valeurs du code de retour d'un paquet CONNACK.

III-3-2-4- PUBLISH

Un paquet PUBLISH est comme illustrer sur la figure suivante:



Exemple

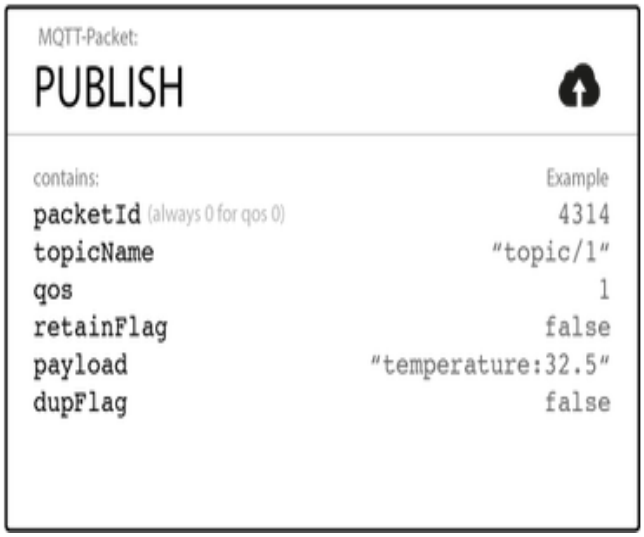


Figure 3-10: Format d'un paquet PUBLISH

Un paquet PUBLISH contient les valeurs:

III-3-2-4-1 Fixed header

a) DUP

- Si l'indicateur DUP est réglé sur 0, cela indique que c'est la première fois que le Client ou le Broker a tenté d'envoyer ce paquet MQTT PUBLISH.
- Si l'indicateur DUP est défini sur 1, cela indique que cela pourrait être la remise à neuf d'une tentative antérieure d'envoi du paquet.

b) QoS: Ce champ indique le niveau d'assurance pour la livraison d'un message d'application.

- Si un broker ou un client reçoit un paquet PUBLISH qui a deux bits QoS réglés sur 1, il DOIT fermer la connexion réseau.

c) RETENIR

- Si l'indicateur RETAIN est réglé sur 1, dans un paquet PUBLISH envoyé par un client à un broker, le broker DOIT stocker le message d'application et sa QoS, afin qu'il puisse être envoyé aux futurs abonnés dont les abonnements correspondent à son nom de sujet.
- Si le broker reçoit un message QoS 0 avec l'indicateur RETAIN réglé sur 1, il DOIT rejeter tout message précédemment retenu pour ce sujet. Il DOIT stocker le nouveau message QoS 0 comme nouveau message retenu pour ce sujet.

d) Champ de longueur restante

Il s'agit de la longueur de l'en-tête variable plus la longueur de la charge utile.

III-3-2-4-2 En-tête variable

L'en-tête variable contient les champs suivants dans l'ordre: Nom du sujet, Identificateur de paquets.

a) Nom du sujet (topic name)

Le nom du sujet identifie le canal d'information auquel les données de la charge utile sont publiées.

Le nom du sujet DOIT être présent comme premier champ dans l'en-tête variable de paquet PUBLISH. Il DOIT être une chaîne codée UTF-8 telle que définie dans la section **III-2-3**.

b) Identificateur de paquet

Le champ Identificateur de paquets n'est présent que dans les paquets PUBLISH où le niveau QoS est 1 ou 2.

III-3-2-5- PUBACK, PUBREC, PUBREL, PUBCOMP

Tous ces paquets ont la même structure, seules les valeurs du code de commande et du champ **Reserved** peuvent varier.(voir tableau 3-1).

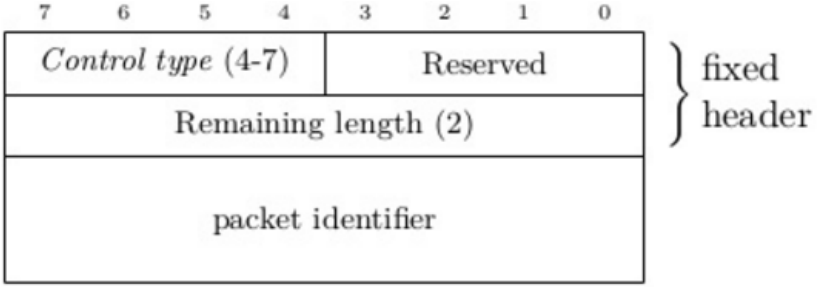


Figure 3-11: – Format d'un paquet PUB [ACK/REC/REL/COMP]

III-3-2-6- SUBSCRIBE

Un paquet SUBSCRIBE envoyé par un client comprend un identifiant pour la réponse, puis une **liste** de souscriptions d'au moins un élément. La structure d'une souscription est composée d'un filtre (longueur et chaine de caractères) et de la qualité de service demandée.

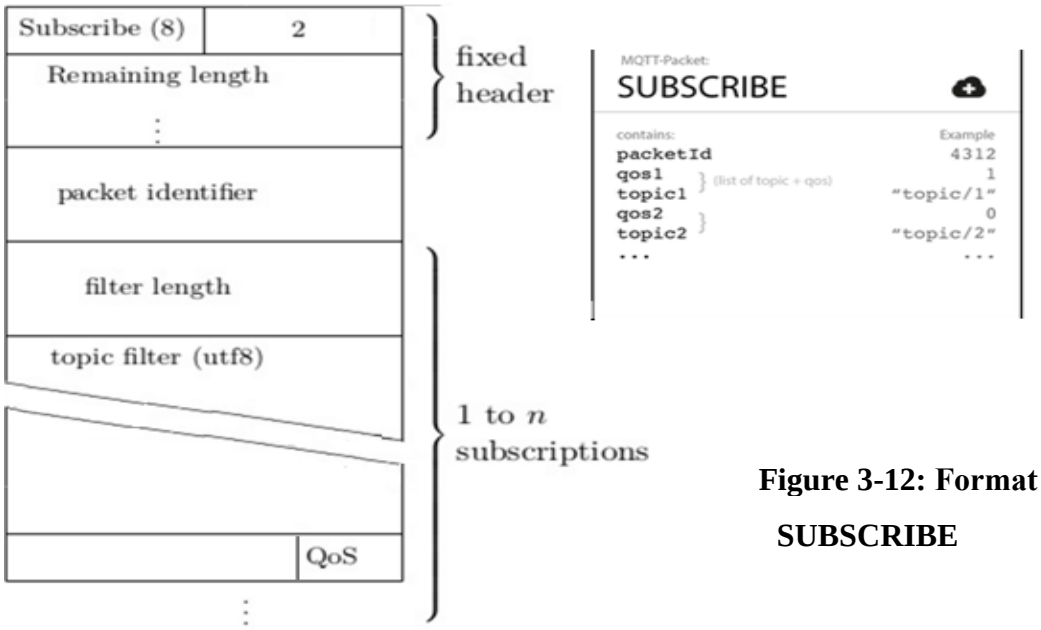


Figure 3-12: Format d'un paquet SUBSCRIBE

Les bits 3,2,1 et 0 de l'en-tête fixe du paquet de contrôle SUBSCRIBE sont réservés et DOIVENT être réglés respectivement à 0,0,1 et 0. Le broker DOIT traiter toute autre valeur comme mal formée et fermer la connexion réseau

Exemple de paquet SUBSCRIBE

Topic Name	"a/b"
Requested QoS	0x01
Topic Name	"c/d"
Requested QoS	0x02

:

	Description	7	6	5	4	3	2	1	0
Topic Filter									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Requested QoS									
byte 6	Requested QoS(1)	0	0	0	0	0	0	0	1
Topic Filter									
byte 7	Length MSB (0)	0	0	0	0	0	0	0	0
byte 8	Length LSB (3)	0	0	0	0	0	0	1	1
byte 9	'c' (0x63)	0	1	1	0	0	0	1	1
byte 10	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 11	'd' (0x64)	0	1	1	0	0	1	0	0
Requested QoS									
byte 12	Requested QoS(2)	0	0	0	0	0	0	1	0

III-3-2-7- SUBACK

Un paquet SUBACK est envoyé par le broker au client pour confirmer la réception et le traitement d'un paquet SUBSCRIBE.

Dans un paquet SUBACK, le broker répond avec le même identifiant de paquet, puis retourne les codes de retour dans le même ordre que les souscriptions du paquet SUBSCRIBE correspondant.

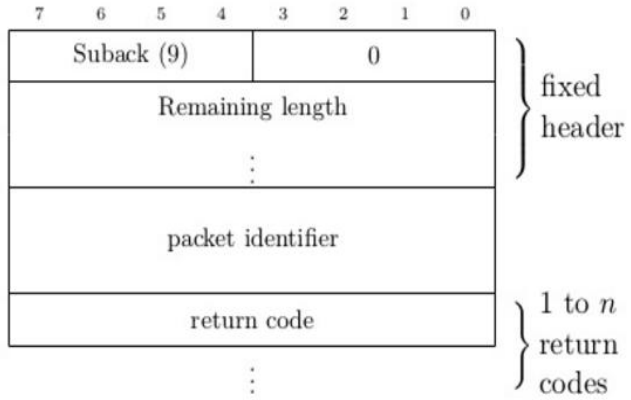


Figure 3-13 – Format d'un paquet SUBACK

Un paquet SUBACK contient une liste de codes de retour, qui spécifient le niveau de QoS maximum qui a été accordé dans chaque abonnement qui a été demandé par le SUBSCRIBE.

Ces codes de retour correspondent à la qualité de service garantie par le broker (qui peut être inférieure à celle initialement demandée), ou le code d'erreur 0x80 en cas de problème.

Description de code de retour			**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0x00 Success - Maximum QoS 0	0		0	0	0	0	0	0	0	0
0x01 Success - Maximum QoS 1	1		0	0	0	0	0	0	0	1
0x02 Success - Maximum QoS 2	2		0	0	0	0	0	0	1	0
0x80 Failure	128		1	0	0	0	0	0	0	0

Tableau 3-4: Code de retour de paquet SUBACK

III-3-2-8- UNSUBSCRIBE

Un paquet UNSUBSCRIBE est envoyé par le client au broker, pour se désabonner des sujets, il ressemble à SUBSCRIBE, à ceci près que le client ne spécifié pas de qualité de service.

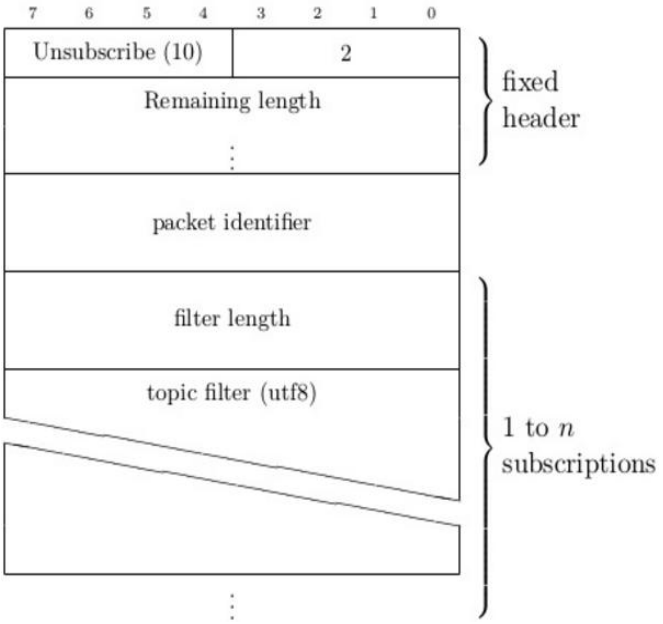


Figure 3-14: Format d'un paquet UNSUBSCRIBE

III-3-2-9- UNSUBACK

UNSUBACK sert de confirmation pour UNSUBSCRIBE, et contient simplement l'identifiant de paquet de la requête correspondante.

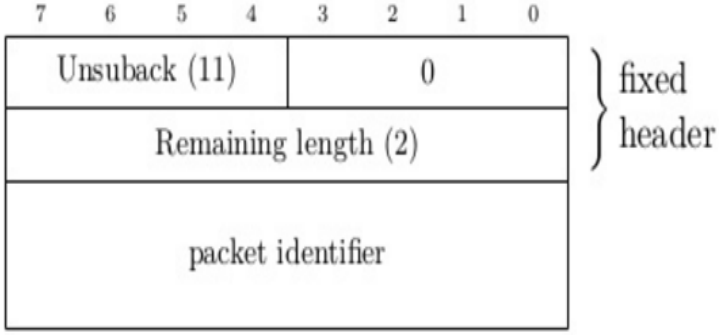


Figure3-15: Format d'un paquet UNSUBACK

III-3-2-10- PINGREQ, PINGRESP

Ces deux paquets sont très simples, et sont constitués uniquement de l'entête fixe du protocole, tenant ainsi sur deux octets.

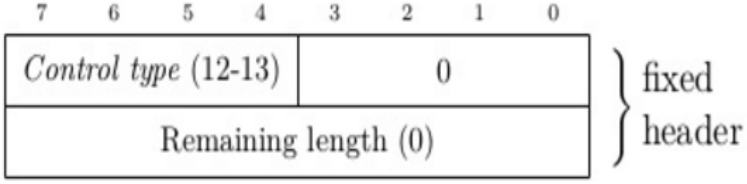


Figure 3-16 – Format d'un paquet PING [REQ/RESP]

*- Le paquet **PINGREQ** est envoyé d'un client au broker. Il peut être utilisé pour:

1. Indiquez au broker que le client est en vie en l'absence de tout autre paquet de contrôle envoyé par le client au broker.
2. Demandez au Broker de répondre pour confirmer qu'il est en vie.
3. Exercer le réseau pour indiquer que la connexion réseau est active.

*- Un paquet **PINGRESP** est envoyé par le broker au client en réponse à un paquet PINGREQ. Cela indique que le broker est en vie.

Ces paquets sont utilisés dans le traitement **Keep Alive**, voir la section **III-3-2-2-3**

III-3-2-11- DISCONNECT

Le paquet DISCONNECT est le paquet de contrôle final envoyé du client au broker. Cela indique que le client a l'intention de se déconnecter, et comporte uniquement l'entête fixe.

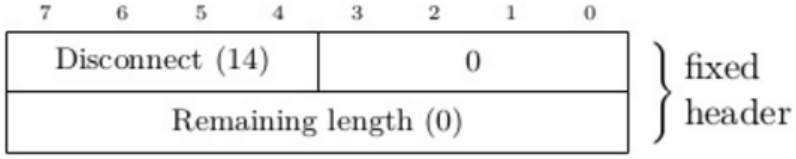


Figure 3-17 – Format d'un paquet DISCONNECT

Sur réception de DISCONNECT le Broker DOIT rejeter tout message de volonté (Wil message) associé à la connexion actuelle sans le publier.

III-3-3 La synchronisation des requêtes envoyées au broker [60]

Le tableau 3-5 décrit les méthodes du client MQTT qui envoient une demande au broker.À moins que le client d'application fixe un délai d'attente indéfinie, le client attend indéfiniment jamais pour le broker. Si le client se bloque, il est soit un problème de programmation d'application, ou un défaut dans le client MQTT.

Méthode	**Synchronisation**	**intervalle délai d'attente**
Connect	Temps d'attente pour une connexion à établir avec le broker	La valeur par défaut de 30 secondes, ou tel que défini par un paramètre, lance alors une exception
Disconnect	Temps d'attente pour le client MQTT pour terminer les travaux qu'il doit faire, et pour la session TCP / IP pour déconnecter	
Subscribe	Attend la fin de la méthode abonner ou désabonner	
UnSubscribe		
Publish	Retourne immédiatement au fil d'application après avoir passé la demande au client MQTT.	Aucun

Tableau 3-5: Comportement de synchronisation des méthodes qui donnent lieu à des requêtes au broker

III-3-4 L’identifiant de client MQTT

Le protocole MQTT définit un «identifiant client» (ID client) qui identifie de manière unique un client dans un réseau. En termes simples, lors de la connexion à un broker, un client doit spécifier une chaîne unique qui n'est pas utilisée actuellement et ne sera pas utilisée par un autre client qui se connectera au broker MQTT. Il existe plusieurs façons de choisir un identifiant client. Voici quelques exemples:

- Un capteur installé dans un emplacement particulier peut utiliser le code d'emplacement comme ID du client.

- Un appareil mobile possédant une capacité de réseau peut choisir l'adresse MAC ou un identifiant d'appareil unique comme ID de client mais MQTT restreint la longueur d'ID de client à 23 caractères. Par conséquent, il y aura une situation où l'identification du client doit être raccourcie.

En raccourcissant l'identifiant du client, il faut s'assurer que l'ID client n'est pas identique à toute autre ID client utilisée dans le réseau. Afin de garder l'identifiant court et unique, il faut présenter un mécanisme de génération d'identifiant fiable. Par exemple, créer un identifiant client à partir du périphérique MAC 48 bits adresse. Si la taille de la transmission n'est pas un problème critique, et utiliser les 17 octets restants pour rendre l'adresse plus facile à administrer, comme un texte lisible par l'utilisateur dans l'identifiant.

Maintenant, essayons de comprendre les implications de deux clients obtenant le même identifiant client.

Le broker MQTT surveille les messages en attente d'être envoyés à un client en fonction de l'identifiant du client. Ainsi, si un client utilise QoS 1 ou QoS 2 et abonné à n'importe quel sujet et déconnecté du broker, le broker sauvera les messages arrivés pour le client pendant qu'il était déconnecté. Une fois que le client se reconnecte, le broker enverra ces messages au client. Si un autre appareil MQTT utilise le même ID de client et se connecte au broker, le broker enverra les messages qu'il avait sauvegardés.

Un autre scénario lié aux identifiants clients est la connexion en double. Disons qu'un périphérique particulier à l'aide de l'ID client DeviceA est connecté au courtier. Si un autre client vient avec le même ID de client DeviceA, le broker peut décider s'il doit permettre au nouveau client de se connecter et de déconnecter le client existant ou de garder l'ancienne connexion en vie et interdire le nouveau client. Il s'agit d'une fonctionnalité facultative d'un broker MQTT.

III-3-5 - Callbacks [60]

Le modèle de programmation client MQTT utilise des threads en profondeur. Les threads découplent une application client MQTT, autant qu'ils le peuvent, des retards dans la transmission de messages vers et depuis le broker. Les publications, les jetons de livraison et les événements perdus de connexion sont livrés aux méthodes d'une classe de rappel qui implémente MqttCallback.

L'interface MqttCallback comporte trois méthodes de rappel

III-3-5 -1 ConnectionLost

- ConnectionLost est appelé lorsqu'une erreur de communication entraîne la sortie de la connexion. Il est également appelé si le broker abandonne la connexion à la suite d'une erreur sur le broker une fois la connexion établie. Les erreurs de broker sont enregistrées dans le journal des erreurs du gestionnaire de files d'attente. Le broker supprime la connexion au client et le client appelle MqttCallback.connectionLost.
- Les seules erreurs distantes jetées comme exceptions sur le même thread que l'application client sont des exceptions de MqttClient.connect. Les erreurs détectées par le broker après que la connexion est établie sont un rapport à la MqttCallback.connectionLost méthode de rappel comme Throwables.
- Les erreurs typiques du broker qui résultent en connectionLost sont des erreurs d'autorisation. Par exemple, le broker de télémétrie tente de publier sur un sujet pour le compte d'un client qui n'est pas autorisé à publier sur le sujet. Tout ce qui se traduit par uneMQCC_FAIL Le code d'état retourné au broker de télémétrie peut entraîner la chute de la connexion.

III-3-5 2-DeliveryComplete

- DeliveryComplete est appelé par le client MQTT pour passer un jeton de livraison à la demande client, En utilisant le jeton de livraison, le rappel peut accéder au message publié avec la méthode **token.getMessage**.
- Lorsque le retour d'appel de l'application renvoie le contrôle au client MQTT après avoir été appelé par la méthode **deliveryComplete**, la livraison est terminée. Jusqu'à ce que la livraison soit terminée, les messages avec QoS 1 ou 2 Sont conservés par la classe de persistance.
- L'appel à DeliveryComplete est un point de synchronisation entre l'application et la classe de persistance. La méthode DeliveryComplete n'est jamais appelée deux fois pour le même message.

- Lorsque le retour d'appel de l'application revient de DeliveryComplete au client MQTT, le client appelle **MqttClientPersistence.remove** pour les messages avec QoS1 ou 2. **MqttClientPersistence.remove** supprime la copie stockée localement du message publié.
- Du point de vue du traitement des transactions, l'appel à deliveryComplete est une transaction monophasée qui engage la livraison. Si le traitement échoue pendant le rappel, lors du redémarrage du client, **MqttClientPersistence.remove** est appelé à nouveau pour supprimer la copie locale du message publié. Le rappel n'est pas encore appelé. Si vous utilisez le rappel pour stocker un journal des messages livrés, vous ne pouvez pas synchroniser le journal avec le client MQTT. Si vous souhaitez stocker un journal de manière fiable, mettez à jour le journal dans la **classe MqttClientPersistence**.
- Le jeton de livraison et le message sont référencés par le thread d'application principal et le client MQTT. Le client MQTT désélectionne l'objet MqttMessage lorsque la livraison est terminée et l'objet token de livraison lorsque le client se déconnecte. **L'objet MqttMessage** peut être récupéré lorsque la livraison est terminée si l'application cliente la détermine.Le jeton de livraison peut être récupéré après la déconnexion de la session.
- Vous pouvez obtenir MqttDeliveryToken et MqttMessage attributs après un message a été publié. Si vous tentez de définir des attributs MqttMessage après la publication du message, le résultat est indéfini.
- Le client MQTT continue de traiter les remerciements de la livraison si le client se reconnecte à la session précédente avec le même ClientIdentifier. L'application client MQTT doit configurer **MqttClient.CleanSession** pour faux Pour la session précédente, et la configurer faux Dans la nouvelle session. Le client MQTT crée de nouveaux jetons de livraison et des objets de message dans la nouvelle session pour les livraisons en attente. Il récupère les objets à l'aide de la classe MqttClientPersistence. Le rappel d'application est appelé dans la nouvelle session pour toutes les livraisons initiées dans la session précédente et complétées dans cette session.
- Le rappel de l'application est appelé après la connexion du client de l'application, lorsqu'une livraison en attente est terminée. Avant que le client de l'application ne se connecte, il peut récupérer les livraisons en attente en utilisant la méthode MqttClient.get Pending Delivery Tokens..

III-3-5-3 MessageArrived

- MessageArrived s'appelle lorsqu'une publication arrive pour le client correspondant à un sujet d'abonnement. Le sujet est le sujet de publication, pas le filtre d'abonnement.Les deux peuvent être différents si le filtre contient des caractères génériques.

- Si le sujet correspond à plusieurs abonnements créés par le client, le client reçoit plusieurs copies de la publication. Si un client publie sur un sujet auquel il s'inscrit également, il reçoit une copie de sa propre publication.
- Si un message est envoyé avec une QoS de1 ou 2, Le message est mémorisé par la **classe MqttClientPersistence** avant que le client MQTT n'appelle message. **MessageArrived** se comporte comme deliveryComplete: il est appelé uniquement une fois pour une publication, et la copie locale de la publication est supprimée par **MqttClientPersistence.remove** lorsque **messageArrived** retourne au client MQTT.

III-4 Conclusion

Nous avons vu que MQTT est un protocole tout à la fois minimaliste et très simple à mettre en œuvre (en particulier du côté client). Malgré tout, il reste suffisamment souple et extensible pour suffire aux besoins d'un bon nombre d'applications, et est donc particulièrement adapté au contexte de l'IoT. C'est la raison pour laquelle nous avons choisi ce protocole pour l'analyser avec le model checker , avec pour objectif de tester ses performances qualitatives et quantitatives pour réaliser une modélisation du protocole capable de s'assurer des propriétés que l'on doit être en mesure d'attendre des futurs « objets » d'Internet :

- Conformité avec la norme, interopérabilité des différentes implémentations,
- Respect des exigences de sécurité et tolérance aux erreurs.

Chapitre 4:

CONTRIBUTION

IV-1- Introduction

Après avoir présenté une description informelle du protocole dans le troisième chapitre, nous allons dans ce chapitre l'étudier formallementen passant par: une modélisation semi-formelle à base du langage UML suivie par une modélisation formelle et une vérification de quelques propriétés et une évaluation probabiliste de performance du protocole en utilisant l'outil de vérification UPPAAL.

IV-2 Proposition d'un modèle d'étude:

Pour valider un protocole, on le modélise à l'aide d'un formalisme, choisi en fonction de trois critères majeurs : le pouvoir d'expression (capacité du modèle à exprimer les caractéristiques du système à étudier), le pouvoir de modélisation (pouvoir les exprimer simplement, critère assez subjectif, qui dépend beaucoup de l'expertise de la personne qui modélise) et le pouvoir d'analyse ou de vérification (capacité à exprimer et à vérifier des propriétés sur le système).

Une fois le modèle formel construit, une exploration exhaustive de tous les chemins peut être menée par model-checking. Dans un premier temps, les contraintes que devra satisfaire le système doivent être définies formellement et seront utilisées comme propriétés au cours du model-checking. Il s'agit à la fois de contraintes de comportement et de contraintes temporelles. Parallèlement, un jeu de scénarios est créé, sur lequel les propriétés seront vérifiées. Un moteur de model-checking effectue la validation formelle de l'ensemble des propriétés sur le modèle, en s'appuyant sur le jeu de scénarios. La méthodologie est synthétisée dans la figure 4-1[61] suivante:

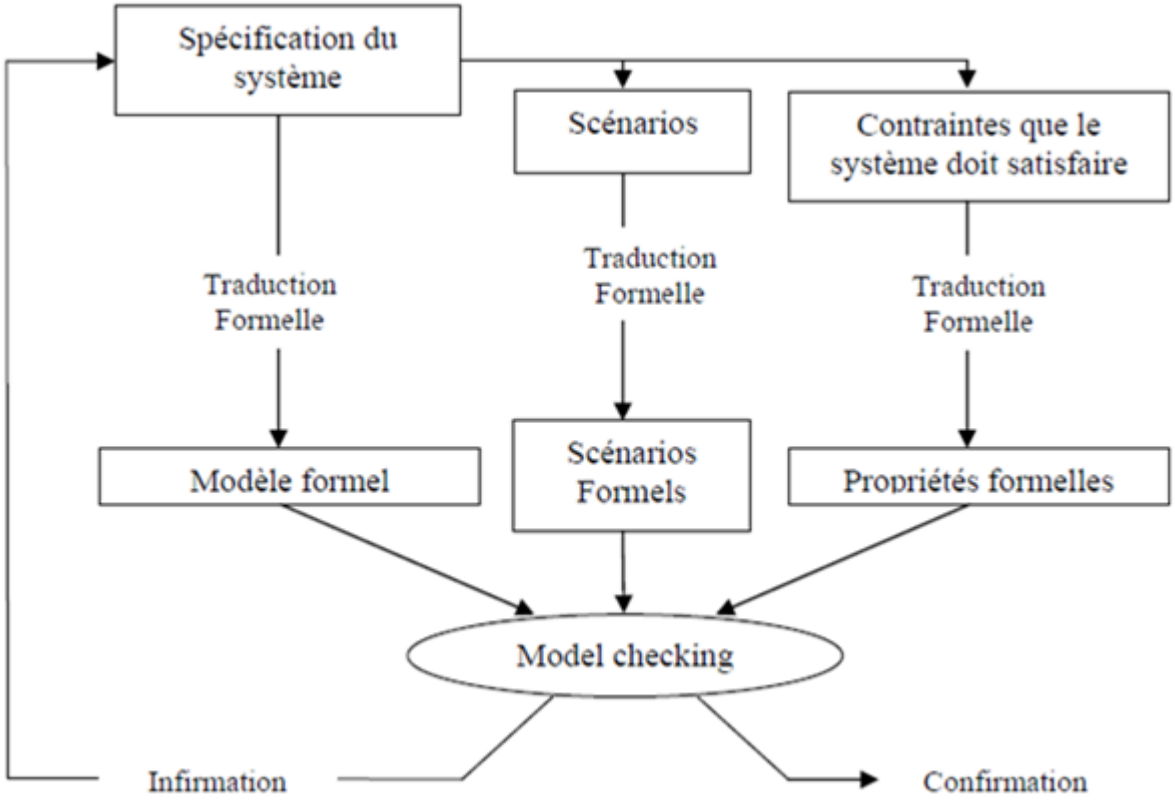


Figure 4.1: Méthodologie adoptée

Justification du choix d'UPPAAL. Pour ce qui est du pouvoir d'expression, nos besoins sont ceux de tout protocole de communication. Il doit être possible d'exprimer le parallélisme et la concurrence (plusieurs nœuds fonctionnent en parallèle), une communication entre ceux-ci doit être possible avec un mécanisme d'envoi de messages. Il doit de plus être possible d'exprimer des choix, ainsi que des contraintes temporelles. Pour ce qui est de l'analyse, nous voulons pouvoir vérifier un comportement et obtenir des bornes temporelles.

Les propriétés à vérifier sont exprimées dans le langage de logique temporelle TCTL (Timed Computation Tree Logic, [62]). La vérification se fait en parcourant de manière exhaustive tous les chemins possibles lors de l'exécution du graphe ; les propriétés sont donc exprimées sur ces chemins d'exécution. Les formules logiques suivantes sont utilisées :

E Il existe un chemin tel que …

A Pour tous les chemins …

[] … dans tous les états de ce chemin.

<> … dans au moins un état de ce chemin.

Dans la suite de ce travail, nous allons donc essayer de proposer un modèle pour le protocole MQTT V-3.1.1 et de le valider, c'est-à-dire prouver formellement qu'il respecte des contraintes temps-réel.

Dans un premier temps nous avons proposé un scénario « affichage de température » par le déploiement des capteurs qui vont être les Publishers et qui sont sensé capter la température. Les Subscribers vont être (soit: micro portable, un smart phone ou tablette.), tous connectés à l'internet.

Le nombre minimal pour étudier les Publishers et les Subscribers et 3 car un subscriber pout être inscrit à 1 ou plusieurs sujets (topic). D'un autre coté pour le bon fonctionnement d'UPPAAL, il faut rester dans la limite de 10 car au plus de 10 modèles le nombre de possibilité d'états va augmenter au plus de 10 millions d'états, cas bloquant pour le système.

IV-3- Modélisation semi formelle du protocole MQTT

Pour comprendre le fonctionnement du protocole MQTT 3-1.1 et comme première étape on a opté pour: La modélisation semi formelle du protocole, qui est réalisée en utilisant des diagrammes du langage UML. Qui sont le diagramme de séquence pour montrer les interactions entre les clients et le broker, ainsi que le diagramme de classe pour formaliser les aspects fonctionnels du protocole.

IV-3-1 Diagramme de classe

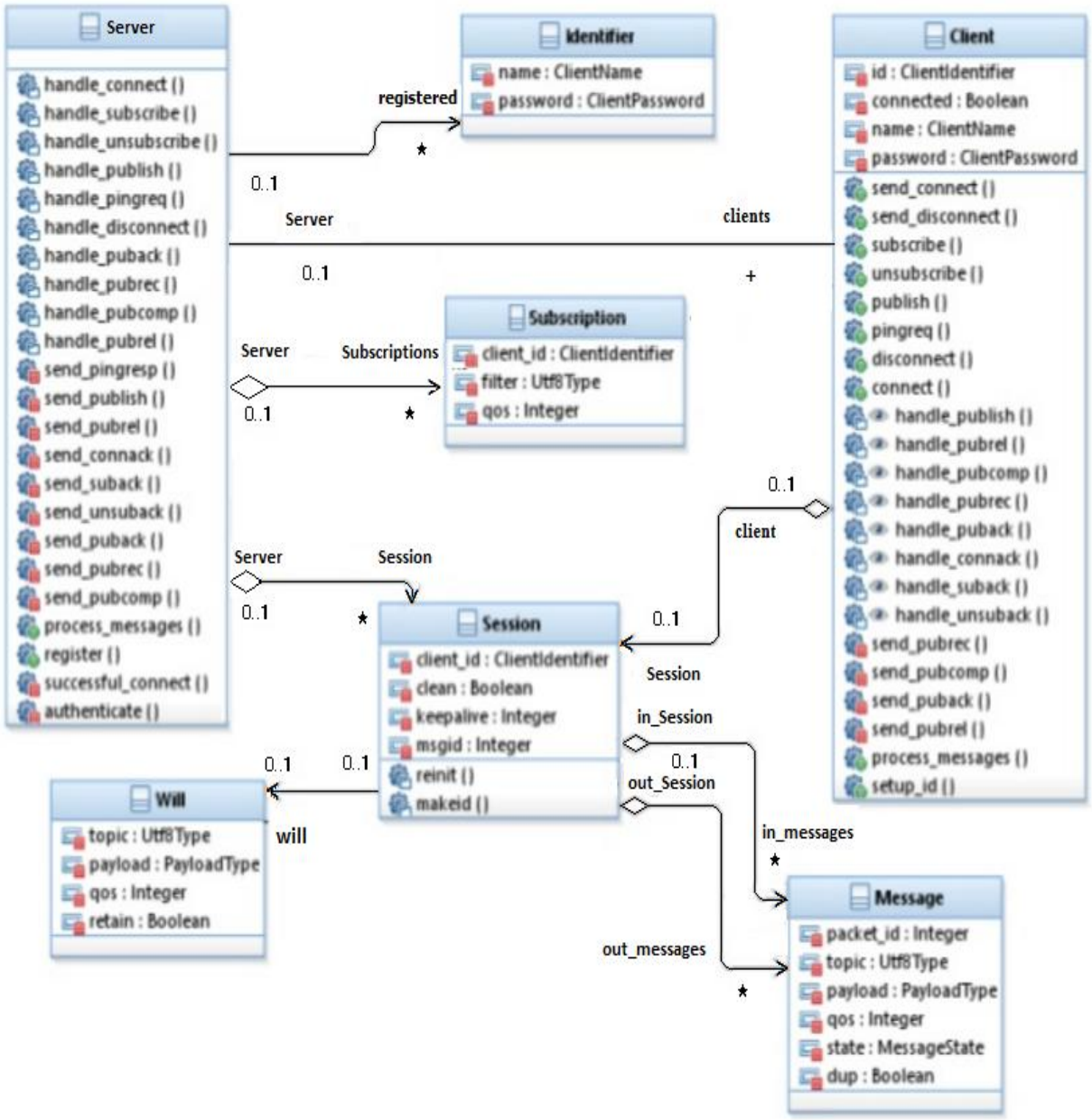


Figure 4-2 Diagramme de classe [63]

Les composants principaux sont les classes broker et Client, le broker pouvant prendre en charge simultanément la connexion de plusieurs clients. Chaque client connecté maintient une

Session lui permettant de contrôler le transit des messages, tandis que le broker maintient une Session pour chaque client.

Un client peut être soit **déconnecté**, soit **connecté** au broker sans avoir initié un échange MQTT, soit connecté avec une session en cours.

Coté broker, la gestion des identités des clients se fait à l'aide de la classe Identifier qui représente simplement un couple login / mot de passe. Une méthode **register** permet de modéliser l'ajout de nouveaux couples d'identifiants (login et mot de passe). Quant à la gestion des abonnements, elle est gérée par le broker avec la classe **Subscription**,

Lorsqu'un client envoie une demande d'abonnement avec la méthode subscribe, le broker crée un nouvel abonnement avec l'identifiant du client, le filtre de sujet et la qualité de service requise, et l'ajoute à sa liste d'abonnements. La méthode **unsubscribe** provoque à l'inverse la suppression d'un abonnement existant.

La classe Message modélise les messages véhiculés par le protocole et contient les informations permettant de suivre l'évolution de l'état de transmission d'un message.

IV-3-2 Diagramme de sequences du protocole

Le diagramme de sequence est l'un des diagrammes dynamiques du langage UML. C'est une représentation graphique qui permet de montrer des interactions entre les objets du système selon un ordre chronologique.

La figure 4-3 présente le diagramme de séquence du protocole. Cette figure montre un résumé des interactions nécessaires pour la transmission d'une trame selon le protocole déjà décrit. Ces interactions font intervenir trois entités: Le Publisher, le Subscriber et un broker.

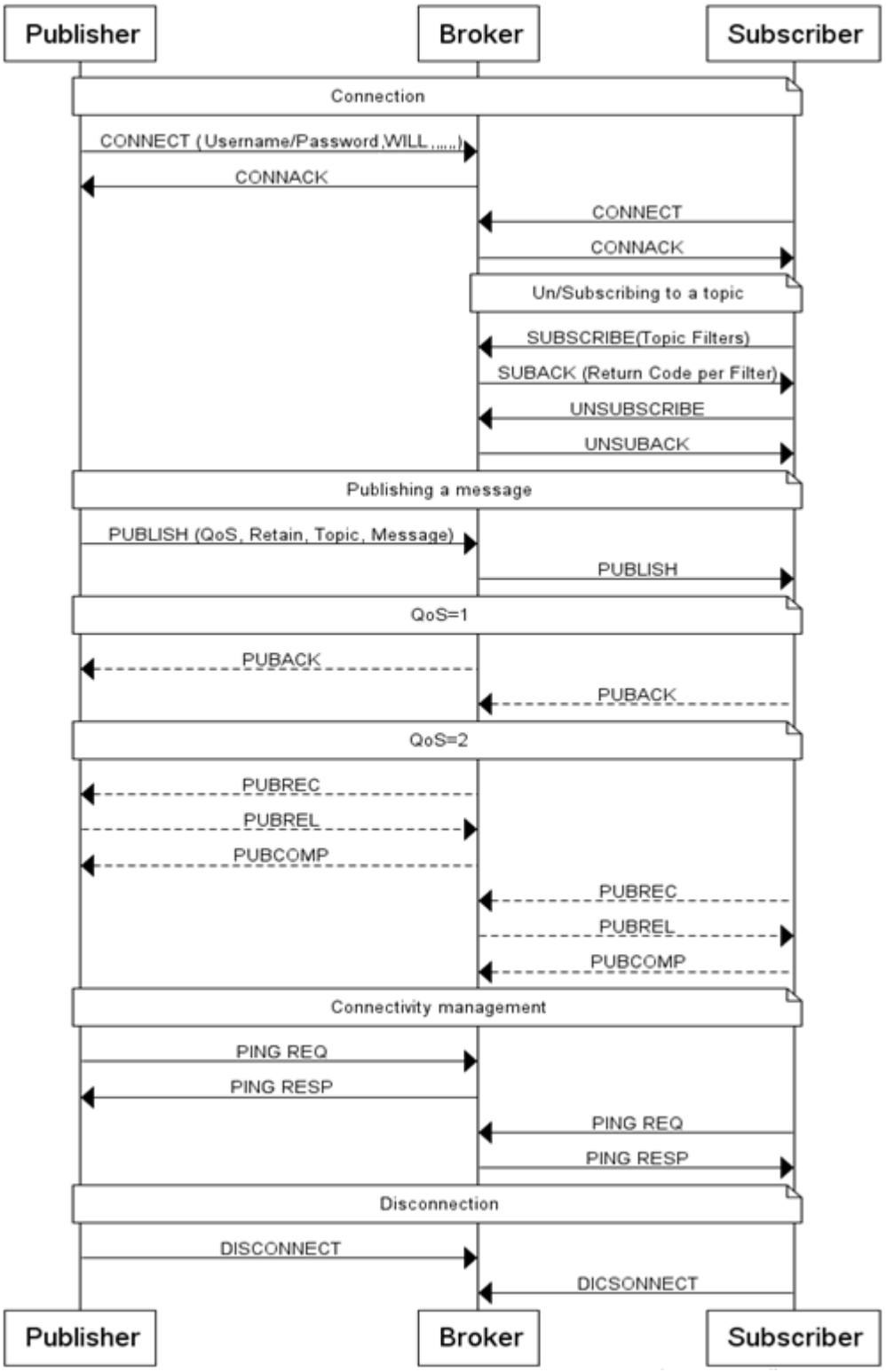


Figure 4-3: Diagramme de sequence du protocole MQTT 3-1.1.

IV-3-3 Le client MQTT

Le client MQTT est conçu essentiellement comme suit:

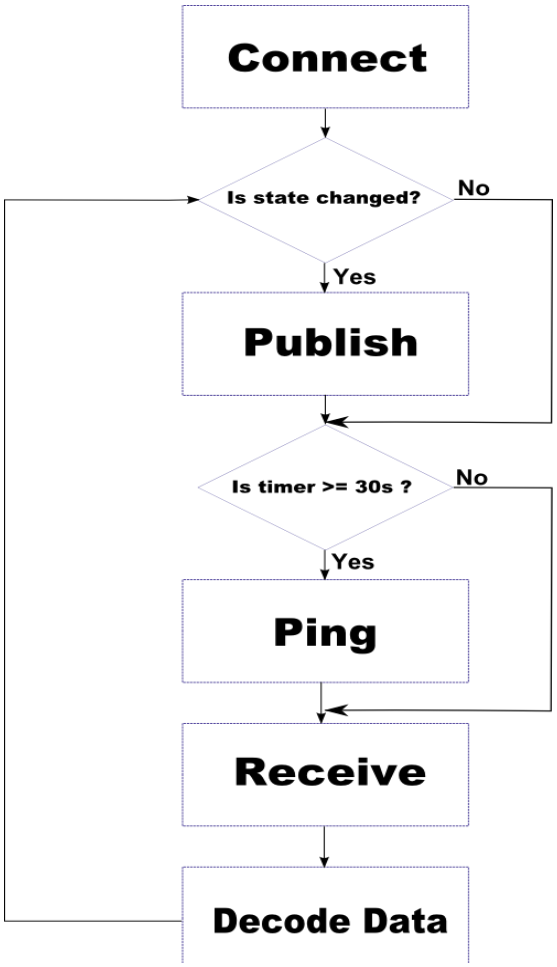


Figure 4-4- Le Diagramme de seveur MQTT

1- Connect: La fonction: connect suit le modèle de la connexion TCP;

2- **Publish:** La publication est lancée seulement après avoir reçu un message de CONNACK de broker (sans erreur);

3- **Ping:** La fonction est appelée toutes les 30 secondes pour maintenir en vie la connexion**;**

4- **Receive:** La fonction receiveData est appelée à chaque fois dans la boucle. Il vérifie si une commande a été reçue**;**

- Tout d'abord, il vérifie si un ou plusieurs octets ont été reçus. Ensuite, il les décode pour connaître le type de message.
- Quand il reçoit le reste du message, il décode la taille et analyse le message complet.
- Une fois le message complet a été reçu, il extrait de la commande le message d'MQTT complet et fournir la charge utile pour le rappel.

5- Decode Data: Cette fonction est utilisée pour décoder le message reçu à partir du broker.

Il peut être un CONNACK, un ***PING_RESP ou un PUBACK, PUBREC, PUBCOMP.***

IV-3-4 Le Broker MQTT

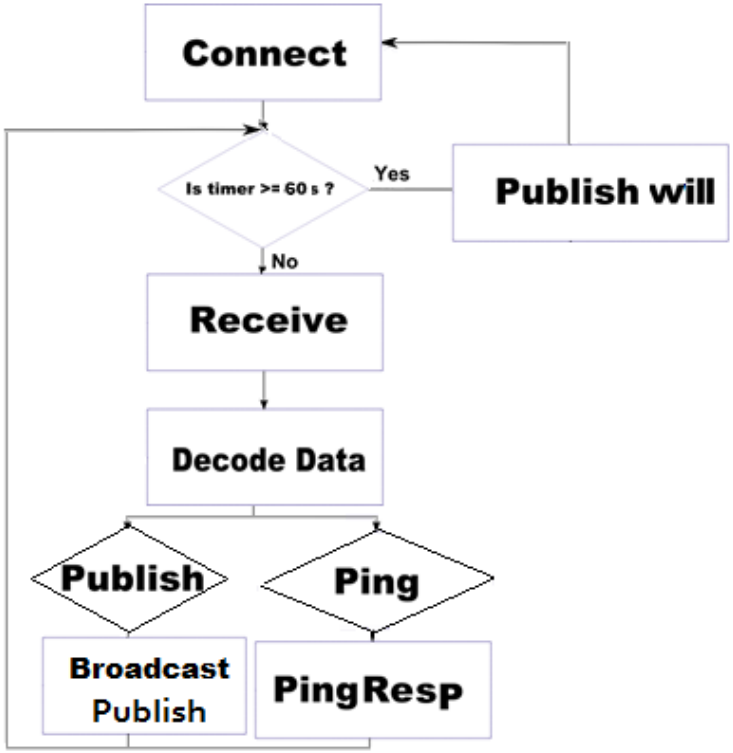


Figure 4-5- Le Diagramme de Broker MQTT

1-Publish will: La publication de message will est lancée seulement si le temps keep alive est expiré.

2-Broadcast publish: envoie le message a tous les abonnés consernés.

IV-4 Modélisation formelle avec les automates temporisés

Dans cette section, nous allons voir les modèles du système que nous avons réalisés en utilisant les automates temporisés probabilistes. Ce système est modélisé sous forme de six modèles:

- Le modèle de Publisher,
- Le modèle du Subscriber,
- Le modèle de connexion,
- Le modèle de la phase Souscription,
- le modèle de la phase désinscription,
- Le modèle du broker dans la phase de publication côté Publisher,
- Le modèle du broker dans la phase de publication côté Subscriber,

Ces deux dernier malgré qu'ils se ressemblent et d'après le protocole MQTT-3-1.1, les Publishers et les Subscribers sont asynchrones et fonctionnent indépendamment les uns des autres.

La simulation de l'envoi d'une trame est modélisée par des actions synchrones. Par exemple: L'action (PUBLISH) échangée entre le Publisher et le broker ou le broker et le subscriber dans le cas d'envoi de trame d'information. Même chose pour la réception; d'après la qualité de services exigée par le Publisher ou le subscriber:

- dans le cas de QS=1 : l'action est (PUBACK).
- dans le cas de QS=2 : les trois actions synchrones sont (PUBACK, PUBREPL, PUBCOMP).

Un exemple d'envoi d'une trame de publication est montré dans la figure 4.5.

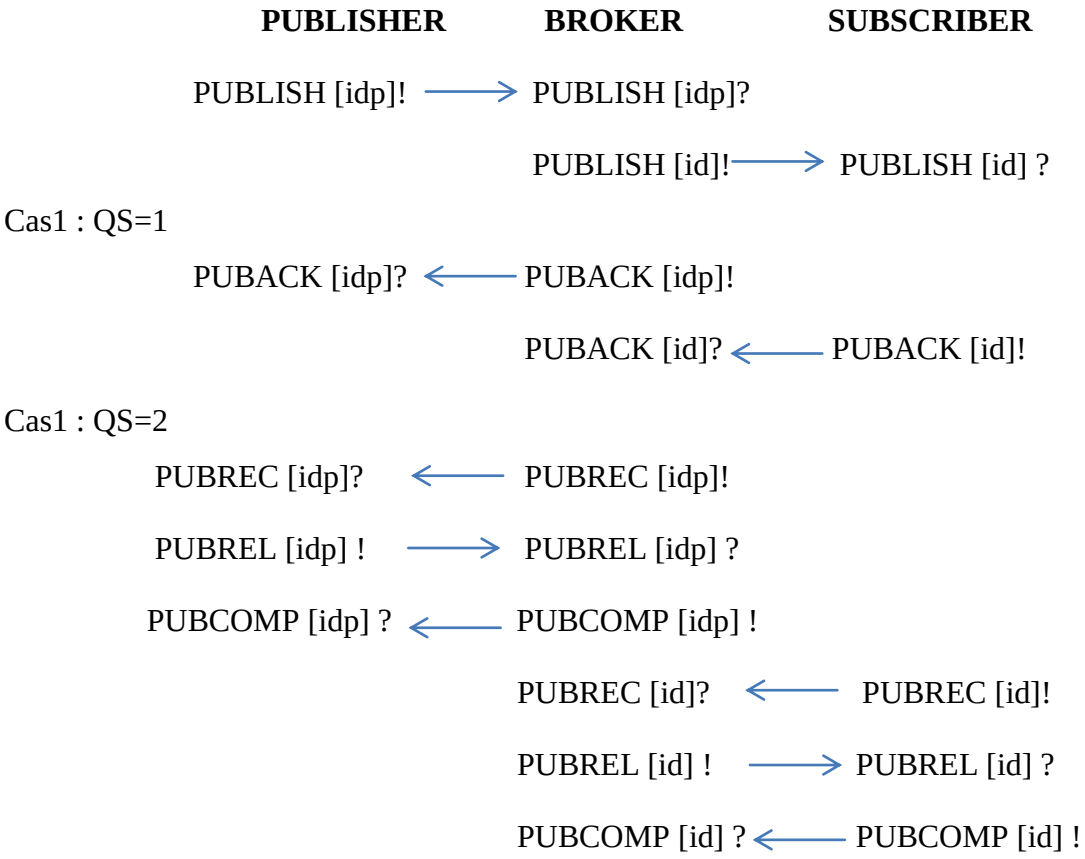


Figure 4.6: Scénario d'une transmission

Les cas d'échecs d'une transmission peuvent se produire a cause des "échecs" dans le réseau ne sont pas modélisés, ainsi que le cas de retrasmission suite a la non reception d'un ACK avec; comme le filtrage des topics, lui aussi n'est pas modélisé faute de temps.

Les sous-sections suivantes présenteront les modèles réalisés.

IV-4 .1 Modèle d'un broker dans la phase de publication côté Publisher

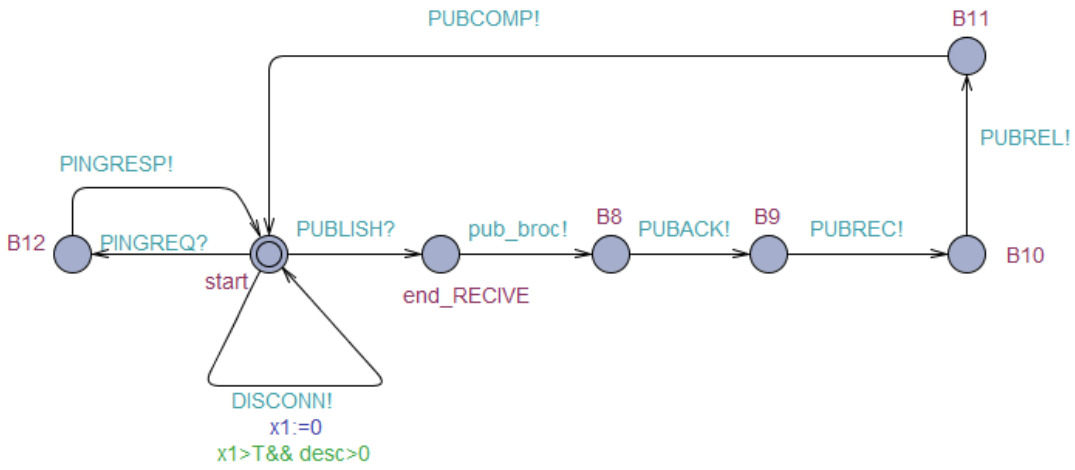


Figure 4-7: Modèle d'un broker phase de publication côté Publisher

Le broker attend durant une période qui ne dépasse pas Time keep_alive la reception d'une publication ou un ping pour maintenir la session ouverte. La réception d'un paquet publish est représentée par l'action synchrone (PUBLISH) et le paquet ping par l'action synchrone (PINGREQ). Si le Time keep_alive et dépassé et le broker n'a rien reçu, il détermine que il y'a un problème (au niveau du réseau ou le publisher) et donc déclenche la procédure de déconnexion qui est l'action synchrone(DISCONN) qui lance l'envoi de paquet publish will aux clients abonnés à ce Publisher et sauvegarde ses informations pour une ultérieure connexion si le flag de clean session =0, si non il les supprime.

Deux liaisons entre le broker dans la phase de publication côté Publisher et le broker dans la phase de publication côté Subscriber:

- Une action synchrone (pub_broc) qui représente une nouvelle publication.
- Une action synchrone (DISCONN), dans le cas où le Publisher est déconnecté.

IV-4 .2 Modèle d'un broker dans la phase de publication côté Subscriber

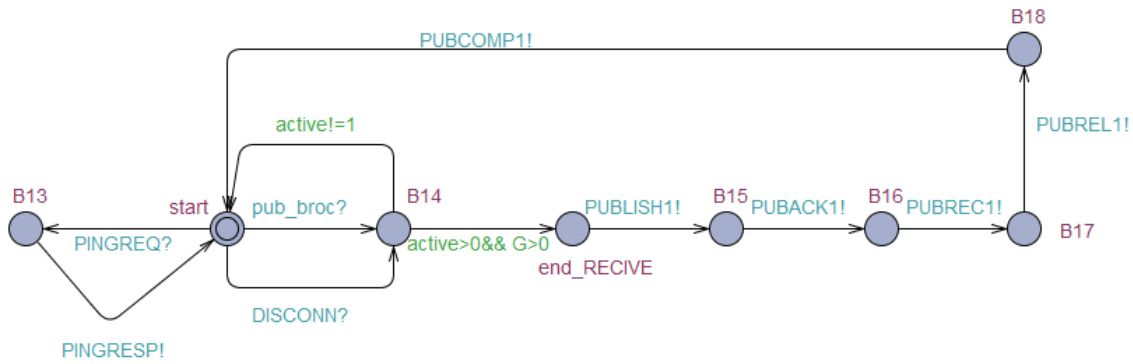


Figure 4-8: Modèle d'un broker dans la phase de publication côté Subscriber

Le broker dans la phase de publication côté Subscriber attend une action synchrone (pub_broc) annonçant que le Publisher a soumis une nouvelle publication ou une action synchrone (DISCONN) pour lancer le broadcast du testament du Publisher correspondant.

Ces deux actions sont attendues pour une période qui ne dépasse pas Time keep_alive durant ce temps le broker peut recevoir un pingreq des subscribers pour maintenir la session ouverte.

La réception de l'une des deux actions synchrones présidentes lance une autre action synchrone (PUBLISHS) attendue par les subscribers suivie par au plus 3 actions synchrones d'après la qualité de service exigé pas les subscribers pour ce topic

IV-4 .3 Modèle d'un Subscriber

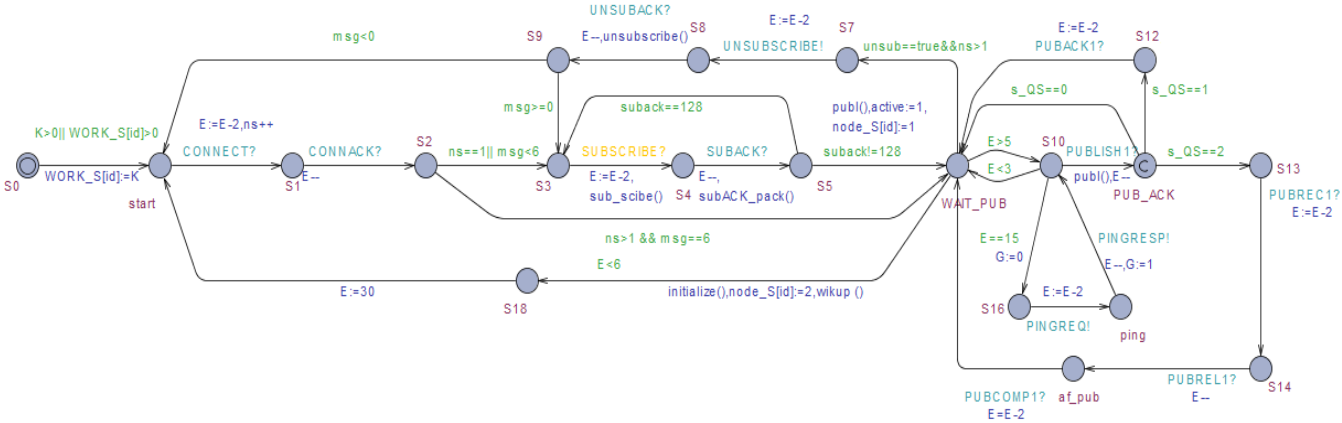


Figure 4-9: Modèle d'un Subscriber

Le modèle du Subscriber et en interaction avec trois modèles de broker:

1. Avec le modèle de broker en phase de connexion par les deux actions synchrones (CONNECT, CONNACK).
2. Avec le broker en phase de souscription pour qu'il enregistre les topics qui lui intéresse avec les actions synchrones (SUBSCRIBE, SUBACK) ou les deux actions synchrones (UNSUBSCIBE, USUBACK) pour se désinscrive.
3. Avec le broker en phase de publication côté Subscriber avec:
   - L'action synchrone (PUBLISH1) et pour la réception les actions synchrones (PUBACK1, PUBREC1, PUBREL1, PUBCOMP1).
4. L'action synchrone (PINGREQ) et pour la réception l'action synchrone (PINGRESP) pour maintenir la session ouverte.

Pour l'énergie en a utilisé une variable E avec une valeur initiale qui se décrémente de deux unités en cas d'envoie et d'une unité en cas d'attente d'un évènement.

IV-4 .4 Modèle d'un Publisher

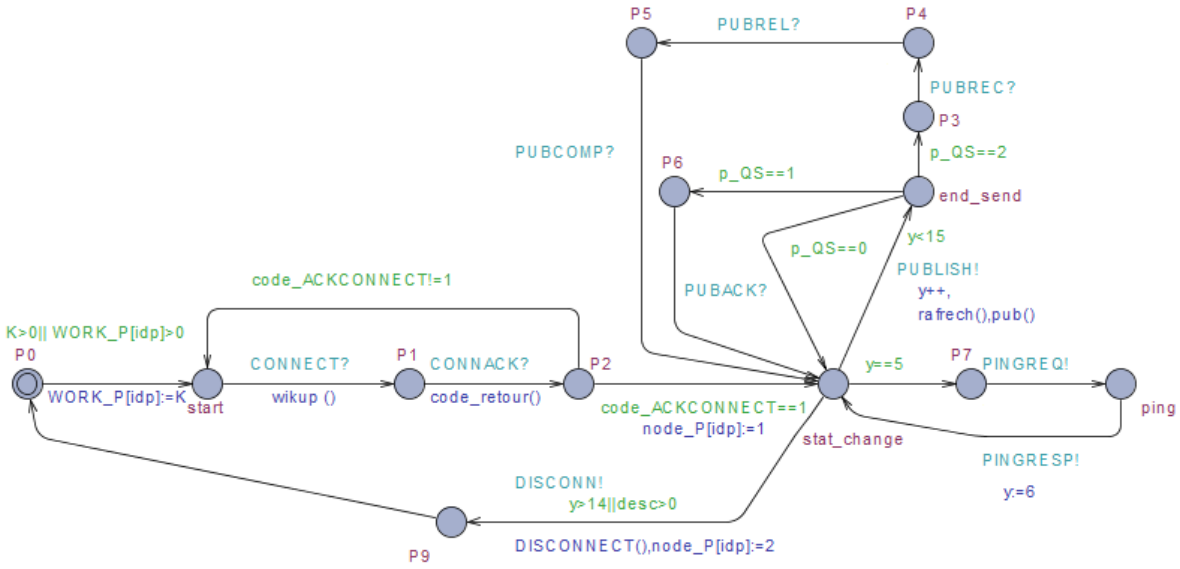


Figure 4-10: Modèle d'un Publisher

Le modèle du Publisher et en interaction avec deux modèles de broker:

1- Avec le modèle de connexion par les deux actions synchrones (CONNECT, CONNACK). Sa requête pour se connecter comme mentionné dans la section « III-3-.2-2 » Englobe aussi des informations sur le topic sur lequel il va publier et le message de testament en cas de sa déconnexion. .

La repense par l'action synchrone (CONNACK) porte un code de retour : 0 pour la connexion est acceptée ou une valeur de (1-4) pour « connexion refusé ».

2- Avec le broker en phase de publication côté Publisher pour qu'il publie sur le topic lui correspondant avec : les actions synchrones (PUBLISH) et pour la réception les actions synchrones (PUBACK, PUBREC, PUBREL, PUBCOMP)

3- L'action synchrone (PINGREQ) et pour la réception l'action synchrone (PINGRESP) pour maintenir la cession ouverte.

Pour l'énergie en a utilisé une variable Y pour simuler que le Publisher tombe en panne après 14 publications.

IV-4 .5 Modèle de Souscription/ Désinscription

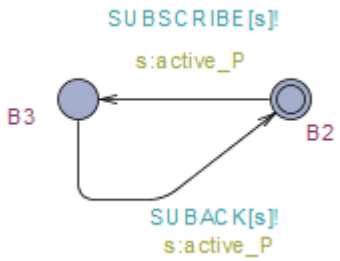
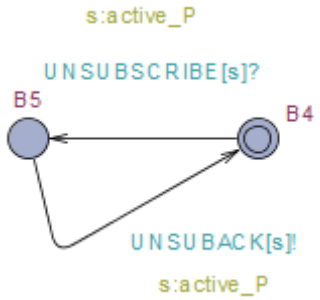
Modèle de Souscription	Modèle Désinscription
	

Figure 4-11: Modèle de la phase Souscription/ Désinscription

Ces modèles et en interaction avec le Subscriber par les deux actions synchrones (SUBSCRIBE, USUBSCRIBE) et pour la réponse avec les deux actions synchrones (SUBACK, UNSUBACK) déjà discutés dans le modèle de Subscriber.

IV-4 .6 Modèle de la phase de Connexion

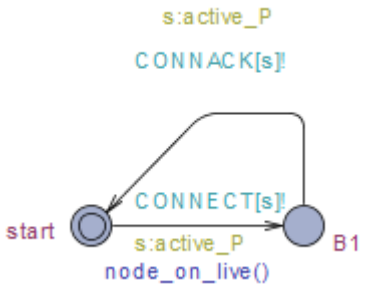


Figure 4-12: Modèle de la phase connexion

Ce modèle est dédié pour la phase de connexion des Publishers et Subscribers avec l'action synchrone (CONNECT, CONNACK).

IV-5 Vérification des propriétés qualitatives:

Nous avons vérifié les propriétés d'accessibilité, de sureté et de vivacité*:* Nous avons trouvé les résultats suivants:

IV-5-1. Propriétés d'accessibilité:

La formule utilisé pour vérifié les propriétés d'accessibilité est de la forme: E<> ϕ.

Dans notre cas on a examiné qu'un subscriber peut arriver a un état d'inisialisation «S18» par la formule **E<> sub1.S18.**

La figure 4-13 montre la satisfaction de la propriété vérific

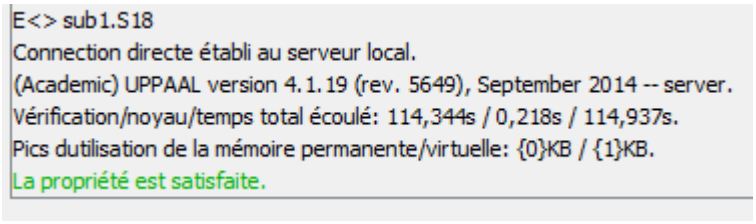


Figure 4-13:Propriété d'accessibilité satisfaite

Dans le « cas de 3subscribers et 3 Publishers» le model_checker a pris beaucoup de temps sans réponse ce qui rend la propriété indécidable malgré qu'il génère presque 8 millions d'états.

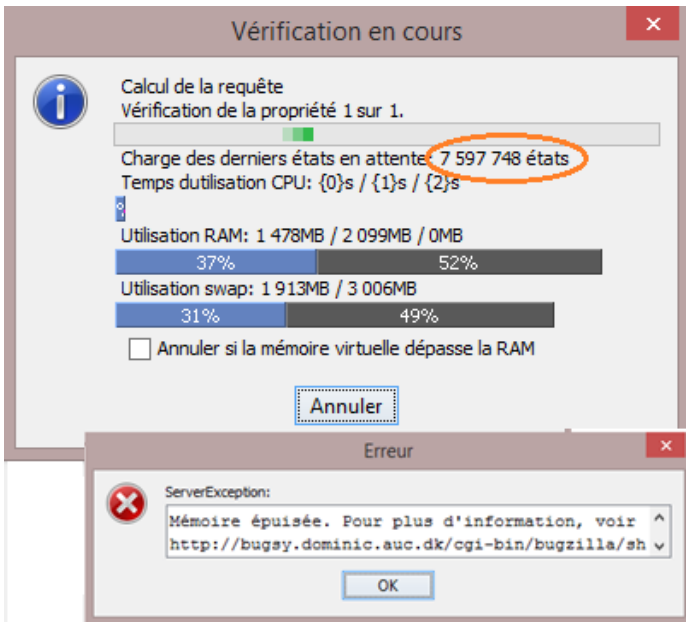


Figure 4-14: Propriété d'accessibilité indécidable

Dans le « cas de 3 Subscribers et 3 Publishers»

On a aussi examiné qu'un publisher peut arriver a l'état d'initialisation par la formule

E<> pub1.P9.

La figure 4-15 montre la satisfaction de la propriété

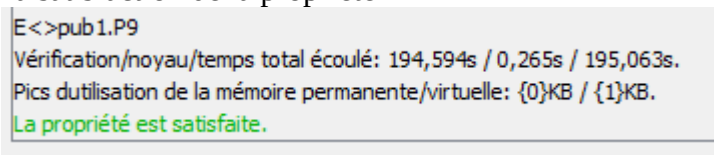


Figure 4-15:Propriété d'accessibilité satisfaite : atteindre l'état de déconnexion

IV-5-2. Propriétés de sureté

Ces propriétés sont de la forme: une certaine mauvaise chose n'arrivera jamais, par la formule:

A [] not deadlock qui veut dire qu'un blocage ne devra jamais se produire dans le modèle.

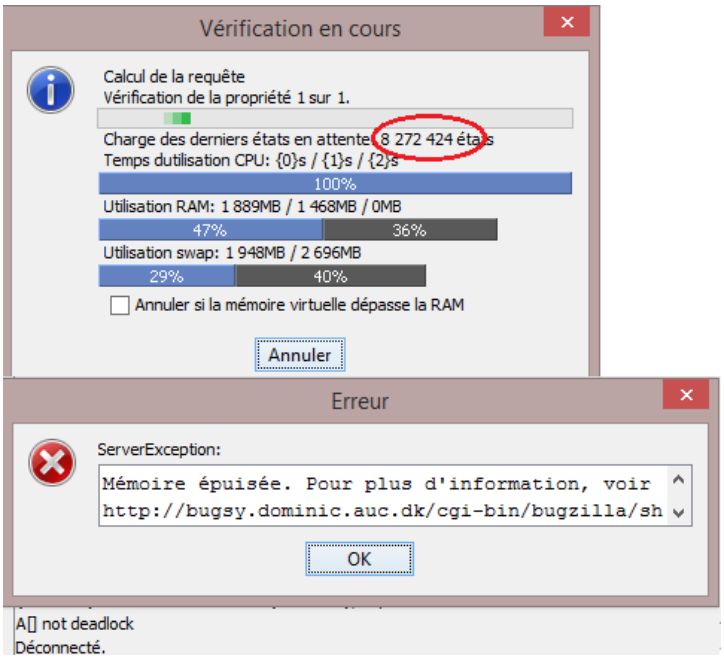


Figure 4-16:Propriété de sureté indécidable

IV-5-3. Propriétés de vivacité

La formule, pour vérifier ces propriétés, est de la forme**:**

A<> ϕ. ***« Quelque chose finira par arriver »***

Par exemple: Dans notre modèle, tout message qui a été envoyé par la suite doit être reçue.

A<>publish_sub.END_RECIVE

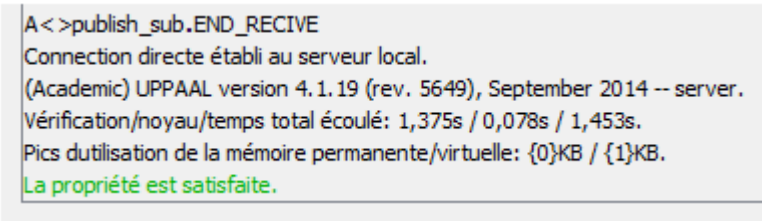


Figure 4-17:Propriété de vivacité satisfaite

IV-6 Vérification des propriétés qualitatives temporisées

La logique TCTL nous donne la possibilité d'ajouter des contraintes temporelles aux formules des propriétés à vérifier. UPPAAL permet d'utiliser cette logique.

Dans notre cas, nous avons déjà une horloge X dans le modèle de **broker dans la phase de publication côté Publisher** pour présenter la propriété keep alive.

Pour vérifier si le subscriber peut recevoir une publication avant que le temps d'attente se termine. Ou bien X1 atteint un certain seuil. La figure 4-18, montre que les 3 propriétés sont satisfaites (pour les seuils T= 45000,1000 et 100 respectivement).

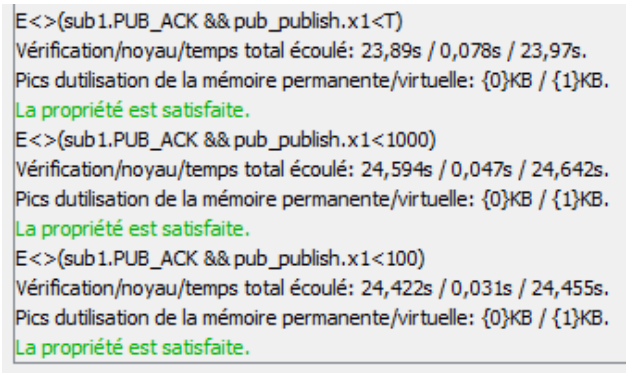


Figure4-18: Propriété d'accessibilité - réception de publication

Nous avons aussi vérifié si le Publisher peut arriver à l'état de la fin de réception d'un acquittement avant que C atteint un certain seuil. La figure 4-19 montre que la propriété est satisfaite (pour les seuils T et 100 respectivement)..

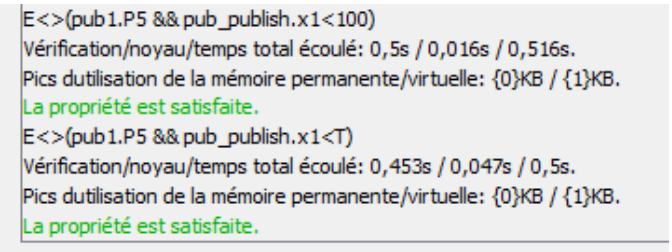


Figure4-19: Propriété d'accessibilité - réception d'acquittement

IV-7 Modélisation formelle avec les automates temporisés probabilistes

Afin de faire une évaluation de performance du protocole, nous étions obligés de considérer les aspects stochastiques qui régissent certains évènements dans le protocole. En effet, le Model-checker statistique associé avec UPPAAL 4.1.19 exige de rendre l'ensemble des canaux des **(broadcast chanels)** et de définir les lois de probabilité régissant les délais de séjour dans chaque localité ayant des transitions de sortie. Dans une première modélisation, nous avons associé un taux=1 pour toutes les localités sans invariants sur les horloges. Les deux figures, ci-dessous, montrent les modèles obtenus pour le Subscriber et le Publisher.

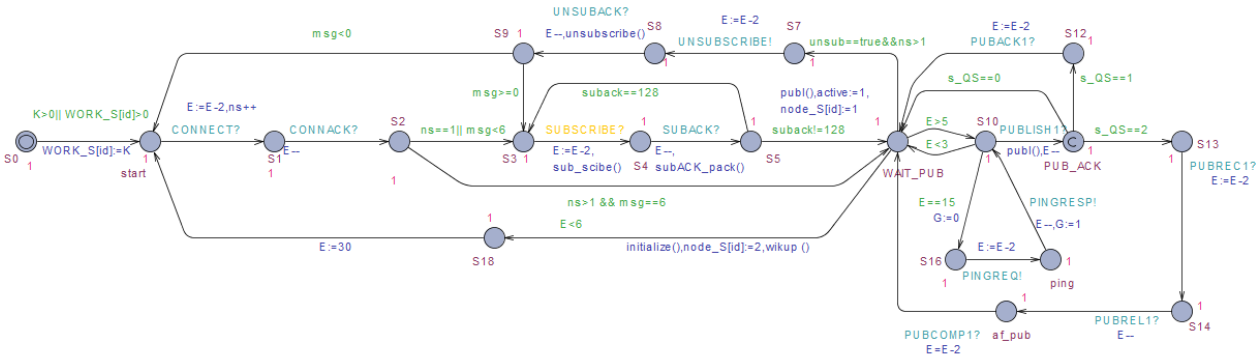


Figure 4-20: Modèle de Subsciber

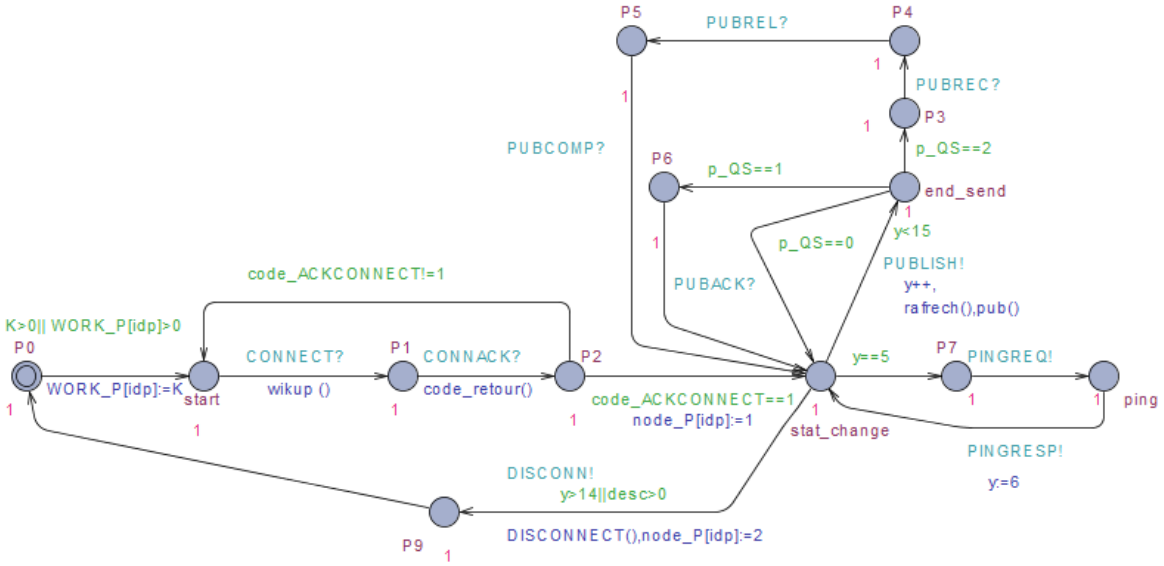


Figure 4-21: Modèle de Publisher

IV-7.1 Vérification des propriétés quantitatives

Nous avons examiné si un subscriber peut atteindre l'état d'initialisation avec Ces modèles probabilistes et avec la formule Pr[<= 100](<> sub1.S18), résultat: Cette propriété est satisfaite avec une probabilité dans l'intervalle [0.303, 0.403], comme il est montré dans la figure 4.22. Cette même propriété qu'on a trouvé au paravent indécidable avec la formule

E <>sub.S18 **pour 3pub et 3 sub**

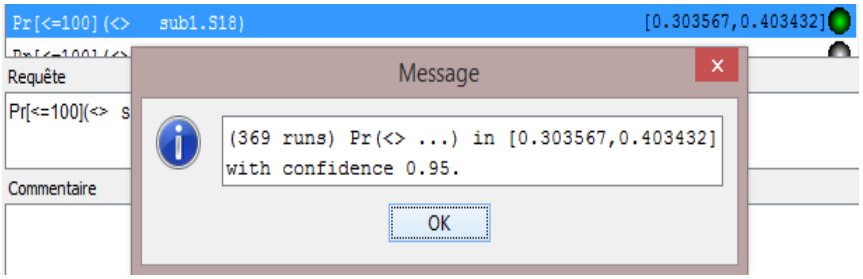


Figure 4-22: .Propriété probabiliste d'accessibilité -atteindre l'état d'initialisation pour le Subscriber pour un runtime <= 10^2 unité de temps

Et avec la formule Pr [<= 1000](<> sub1.S18), le résultat: changement de l'intervalle [0.410, 0.510].

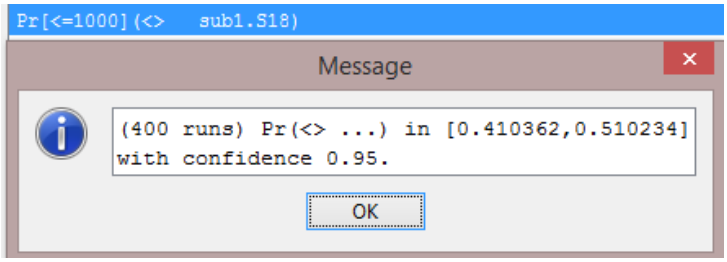


Figure 4-23 .Propriété probabiliste d'accessibilité -atteindre l'état d'initialisation pour le Subscriber pour un runtime <= 10^3 unité de temps

Et avec la formule Pr [<= 1000](<> sub1.af_pub): possibilité de recevoir un acquittement avec QS=2.

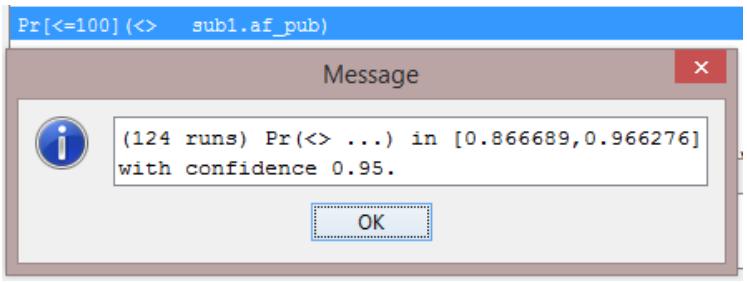


Figure 4-24 Propriété probabiliste d'accessibilité: réception d'une publication.

- **Et avec la formule** Pr [<=100](<> NNI_P>=1 && NNL_P>=2&& NND_P<=2): nombre de pub inactives, morts,active sur un total de 9 pub et 9 sub.

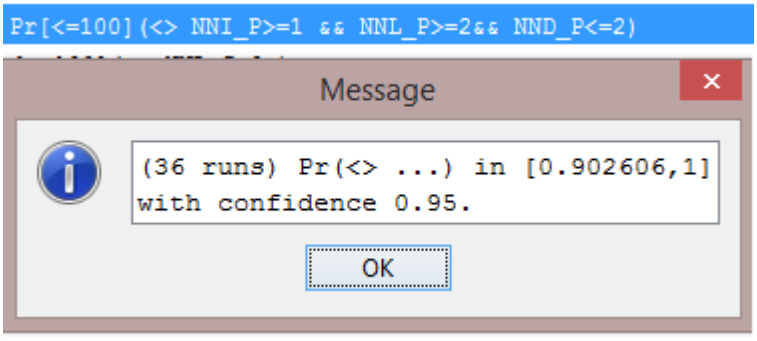


Figure 4-25 Propriété probabiliste d'accessibilité nombre de Publishers actives pour un runtime <= 10^2 unité de temps

Nous pouvons tracer la fonction caractéristique de la probabilité en utilisant SMC-UPPAAL

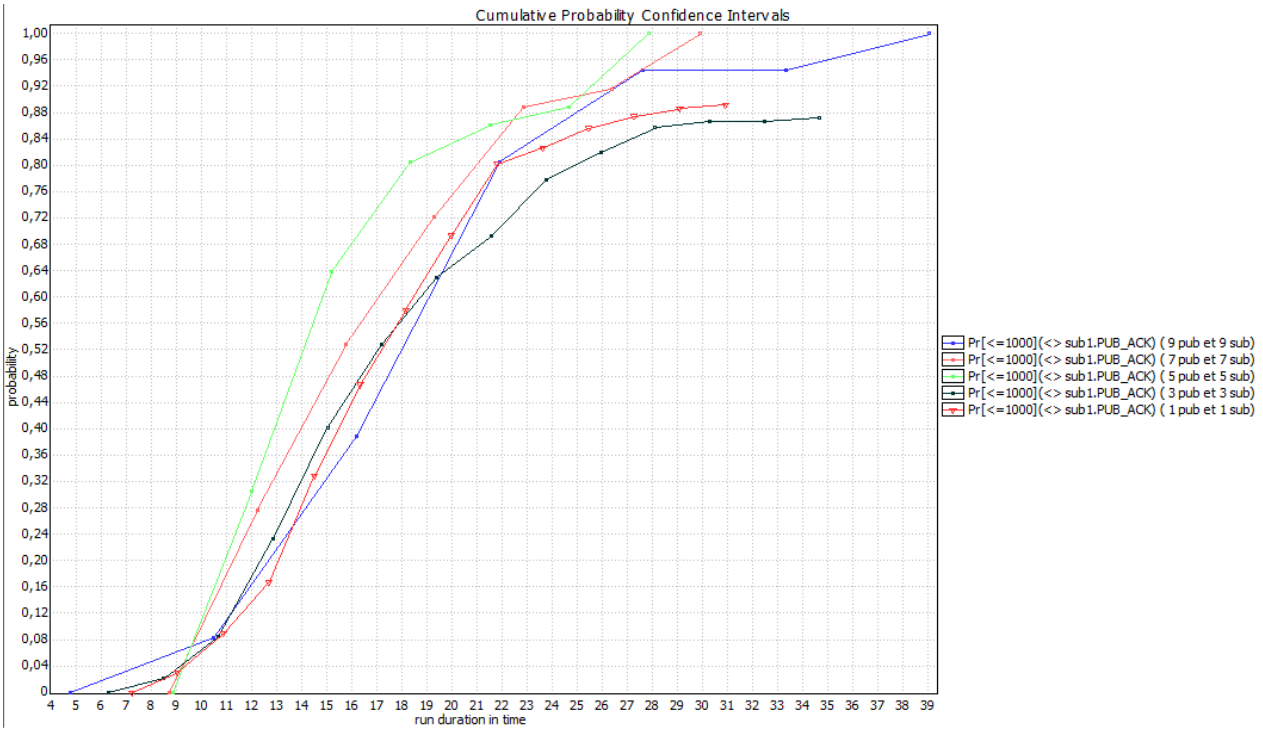


Figure 4-26: Probabilités de finir la réception d'une publication en fonction du nombre de nœuds pour un runtime <= 10^3 unité de temps

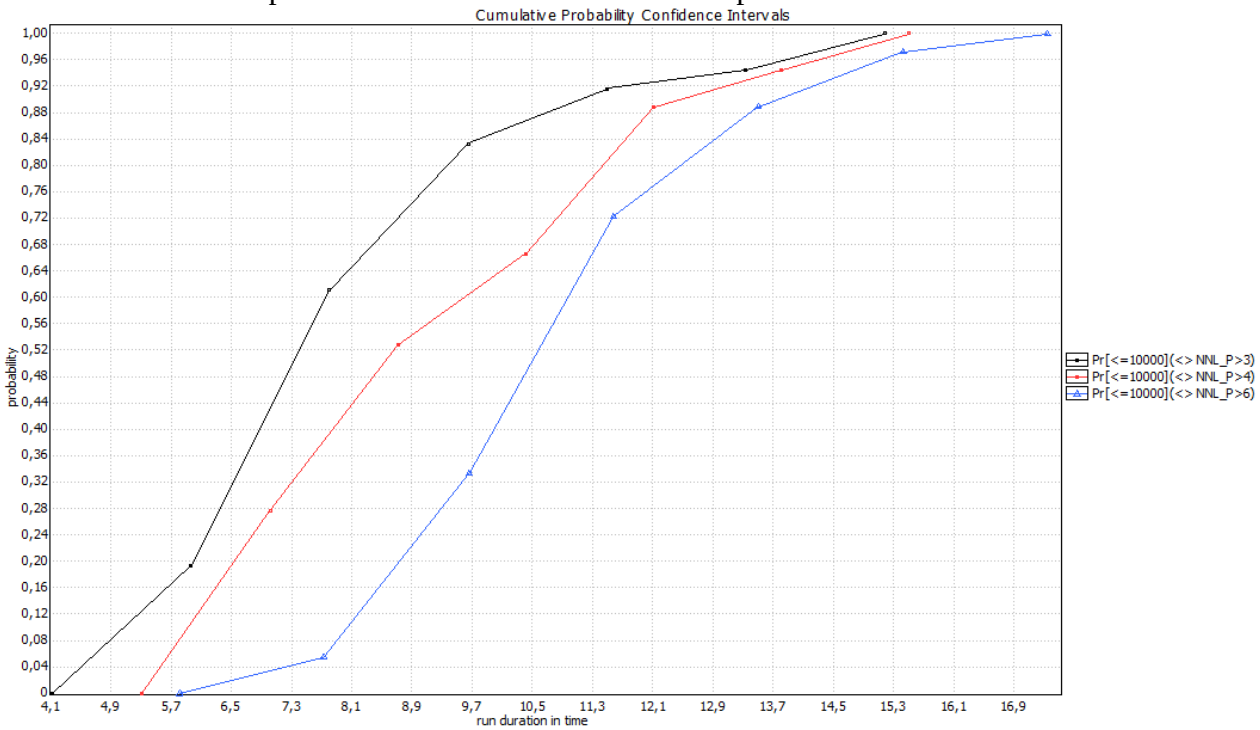


Figure 4-27: Probabilités d'avoir des Publishers actives pour un runtime <= 10^4 unité de temps

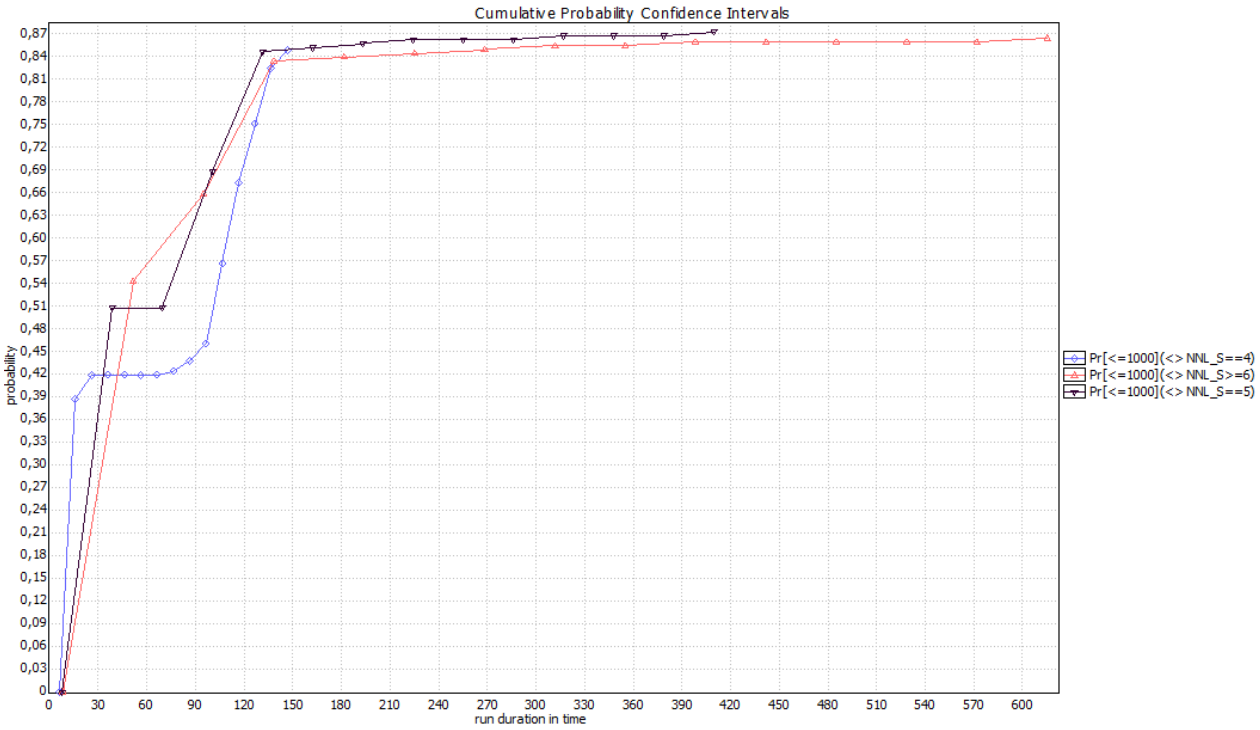


Figure 4-28: Probabilités d'avoir des Subscribers actives pour un runtime <= 10^3 unité de temps

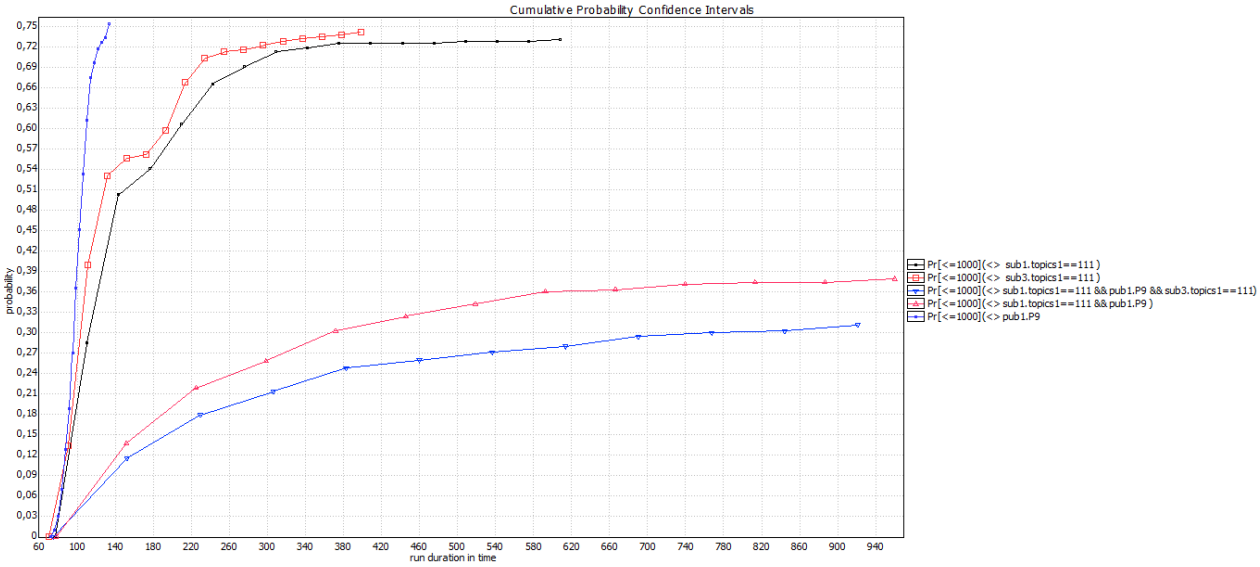


Figure 4-29: Probabilités qu'un Publisher se déconnecte et ses abonnés reçoivent son testament pour un runtime <= 10^3 unité de temps

Conclusion

Dans ce chapitre, nous avons proposé un modèle abstrait décrivant un protocole de la couche application dédié pour l'échange de messages. Nous avons présenté la modélisation formelle de ce protocole de messagerie nommé "MQTT", nous avons tout d'abord commencé par l'étudier exhaustivement, le modéliser semi-formellement avec UML, et enfin le modéliser formellement avec les automates temporisés et probabilistes. Afin de vérifier ce protocole, on a fait appel à l'outil UPPAAL avec son SMC (Statistical Model-checking) pour une analyse qualitative et quantitative de ce protocole.

Plusieurs propriétés ont été analysées (Accessibilité, Sûreté, Vivacité) et certains métriques étaient analysées (nombre de nœuds actives, inactives, taux de réussite de transfère et la réception de message, ...).

Conclusion générale:

Nous nous sommes intéressés, au cours de ce mémoire, à l'étude formelle d'un protocole utilisé dans l'internet des objets qui est le protocole MQTT (Message Queue Telemetry Transport). Cette étude fait appel aux méthodes formelles qui sont exploitées pour assurer la fiabilité des systèmes conçus ou déjà réalisés. Ces méthodes offrent en même temps: des langages pour la spécification des systèmes, des langages pour la spécification des propriétés que l'utilisateur cherche à avoir, et en fin les algorithmes permettant de vérifier ces propriétés sur ces systèmes.

Parmi ces méthodes formelles pour la vérification: Model-checking statistique (Stastical Model Checking SMC) permet de rendre certaines propriétés décidables, en limitant l'exploration de l'espace d'états d'un système. Cette méthode permet aussi de produire des probabilités sur la satisfaction ou non de certaines propriétés et donc peut être considérée comme une solution là où le model-checking classique se rend incapable.

Ce travail était réalisé en quatre phases:

1. Etudier les caractéristiques essentielles et les notions fondamentales des protocoles de l'internet des objets;
2. Explorer le domaine des méthodes formelles;
3. Etudier les caractéristiques essentielles et les notions fondamentales du protocole sujet de cette étude MQTT V.3.1.1
4. Modéliser et vérifier le protocole choisi en évaluant ses performances avec SMC-UPPAAL.

Nous avons rencontré plusieurs obstacles mais, la description du protocole par ses auteurs était trop riche d'après notre point de vue et les forums en répondus à certaines lacunes.

Nous sommes limités dans le développement à construire un prototype qui permet de traduire les variables MQTT et de générer un squelette de modélisation UPPAAL.

Le présent travail nous a donné plusieurs idées à développer dans le futur et comme perspectives, nous proposons :

- D'améliorer le modèle proposé en prenant en considération les fonctions non modélisé comme le filtrage des topics ou la gestion de la retransmission dans le cas de non réception d'acquittement
- D'Etudier approfondiment des algorithmes exploités dans l'outil UPPAAL et proposition d'amélioration possible, ou l'implémentation d'autres algorithmes non encore implémentés; par éxemple

- D’étendre ce travail pour pouvoir déceler les fragments de modélisation qui sont la source au non satisfaction des propriétés à vérifier, et pour qu’il puisse prendre en compte d’autres fragments de modélisations que nous n’avons pas pris en considération et de même pour élargir les classes des propriétés à vérifier.
- Vérifier d'autres protocoles en utilisant le modèle checking statistique; et surtout établissement de comparaison raffinée avec d’autres protocoles de l’IOT comme COAP (Constrained Application Protocol) qui tente d’adapter HTTP a l’IOT..

À la fin de ce mémoire, nous considérons que nous avions développé plusieurs acquis sur le domaine des techniques de vérification formelle.

Bibliographie:

Bibliographie

[1] "Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business", August 2014, http://www.gartner.com/newsroom/id/2819918

[2] Internet of Things tutorialspoint, simply easy learning, 2016.

[3] Tara Salman (A paper written under the guidance of Prof. Raj Jain), Networking Protocols and Standards for Internet of Things , p 20

[4] Internet Of Things: Et d'un point de vue technique,

URL: http://www.iplogos.fr/wp-content/uploads/2015/08/IoT-Stack.png

[5] Mohamed Abomhara and Geir M. Køien , Cyber Security and the Internet of Things:Vulnerabilities, Threats, Intruders and Attacks ,*Journal of Cyber Security, Vol. 4,* 65–88

[6] Dr Omar Elloumi, président de la plénière technique OneM2M,

URL : http://www.epdtonthenet.net/article/124327/IoT-ecosystem-expands-significantly-with-new-global-standards.aspx.

[7] Sécurité des objets connectés - Travaux des auditeurs, p 16,
URL : http://images.cigref.fr/Publication/2014-INHESJ-CIGREF-securite_objets_connectes.pdf

[8] Stéphane Lohier,Dominique Présent, Réseaux et transmissions: Protocoles, infrastructures et services-6 éditionn,DUNOD,2016, p 122.

[9] 6LoWPAN, RFC 6568; https://datatracker.ietf.org/doc/html/rfc6568#page-25

[10] B. Babovic *et al.*: Web Performance Evaluation for IoT Applications,IEEE Access, volume 4-nov 2016, page 6989-6990.

[11] Tara Salman (A paper written under the guidance of Prof. Raj Jain),Networking Protocols and Standards for Internet of Things, page 18:

[12] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva, Security for the Internet of Things: A Survey of Existing Protocols and Open Research issues, IEEE COMMUNICATION SURVEYS & TUTORIALS, 2014.

[13] RFC 3971: SEcure Neighbor Discovery (SEND) ,URL: https://tools.ietf.org/html/rfc3971

[14] Lightweight Secure Neighbor Discovery for Low-power and Lossy Networks, URL : https://trac.tools.ietf.org/html/draft-sarikaya-6lowpan-cgand-01

[15] 6LoWPAN, https://fr.wikipedia.org/wiki/6LoWPAN

[16] RPL: IPv6 Routing Protocol for Low power and Lossy Networks, RFC 6550, 2012

URL: https://tools.ietf.org/html/rfc6550

[17] Z. B. Babovic *et al.*: Web Performance Evaluation for IoT Applications,IEEE Access, volume 4-nov 2016, page 6989-6990

[18] Jean-Claude Laprie, Guide de la sûreté de fonctionnement, Cépaduès-Editions, 1996.

[19] Robert Abo, Approches formelles pour l'analyse de la performabilité des systèmes communicants mobiles : Applications aux réseaux de capteurs sans fil, 2011.

[20] MAJDA MOUSSA. Vérification et configuration automatiques de pare-feux par model checking et synthèse de contrôleur, Université De Montréal,Mémoire De Maitrisées Sciences Appliquées, 2014, p 6.

Bibliographie

[21] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns et Joost-Pieter Katoen : Performance evaluation and model checking join forces. Communications of the ACM, 53(9):76_85, 2010

[22] Gérard Fleury, Philippe Lacomme - Alain Tanguy, Simulation à événements discrets, Collection *Algorithmes* dirigée par Gérard Dreyfus, Groupe Eyrolles, 2007,

[23] Pierre-Alain Masson, Vérification par Model-Checking Modulaire de Propriétés Dynamiques PLTL exprimées dans le cadre de Spécifications B événementielles, Thèse de Doctorat, Université de Franche-Comté-2001

[24] Site internet de COQ :https://coq.inria.fr/

[25] specification and Verification System, Site internet de PVS .URL: http://pvs.csl.sri.com

[26] higher-order logic: Site internet de HOL. URL: http://hol.sourceforge.net/

[27] Clarke, E. and Wing, J. (1996). Formal methods : State of the art and future directions. *ACM Computing Surveys*, 28(4) :626–643.

[28] Jacques Sakarovitch : Elements of Automata Theory, Cambridge University Press;2009

[29] Wolfgang Reisig : Petri Nets : An Introduction, volume 4 de Monographs inTheoretical Computer Science. An EATCS Series. Springer, 1985

[30] Jos C. M. Baeten: A brief history of process algebra. Journal of Theoretical, Computer Science, 335(2-3):131_146, 2005.

[31] Alexandre Mouradian, Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil, thèse de doctorat à l'école doctorale : Informatique et Mathématiques de Lyon, 2013, p 131.

[32] MAJDA MOUSSA. Vérification et configuration automatiques de pare-feux par model checking et synthèse de contrôleur, Université De Montréal, Mémoire De Maitrisées Sciences Appliquées, 2014, pp 14-18.

[33] Amir Pnueli : Verification engineering : A future profession (a. m. turing award lecture). In PODC, page 7, 1997

[34] H. Hansson et B. Jonsson : A logic for reasoning about time and reliability. Formal Aspects of Computing, 6(5):512_535, 1994.

[35] Ron Koymans : Specifying real-time properties with metric temporal logic. Real-Time Systems, 2(4):255_299, 1990.

[36] Andrea Bianco et Luca de Alfaro : Model checking of probabalistic and nondeterministic systems. In P. S. Thiagarajan, éditeur : FSTTCS, volume 1026 de Lecture Notes in Computer Science, pages 499_513. Springer, 1995.

[37] Rajeev Alur et David L. Dill : A theory of timed automata. Journal of Theoretical Computer Science, 126:183_235, April 1994

[38] Le model-checker prism est accessible au site: http://www.prismmodelchecker.org/

[39] Christel Baier and Joost-Pieter Katoen, Principles of model checking, the mit press, ambridge, Massachusetts London, England, 2008, pp 673-880.

[40] Le model-checker UPPAAL est accessible du site: http://www.uppaal.org/

[41] Kim Guldstrand Larsen, Paul Pettersson etWang Yi : Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer, 1(1-2):134_152, 1997.

Bibliographie

[42] Edmund M. Clarke, E. Allen Emerson et Joseph Sifakis : Model checking :algorithmic veri_cation and debugging. Communications of the ACM, 52(11):74_84, 2009.

[43] B. Berard M. Bidoit A. Finkel F. Laroussinie,A. Petit L. Petrucci Ph. Schnoebelen, with P. McKenzie, Systems and Software Verification- Model-Checking Techniques and Tools With 67 Figures, Springer 2001

[44] Carles Anton-Haro and Mischa Dohler,Machine-to-machine (M2M) Communications Achitecture, Performance and Applications, Elsevier,2015, p 91.

[45] http://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt

[46] MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS

[47] https://www.oasis-open.org/news/pr/oasis-mqtt-internet-of-things-standard-now-approved-by-iso-iec-jtc1

[48] Protocol Analysis - oneM2M, page 21, URL:

http://www.onem2m.org/images/files/deliverables/TR-0009-Protocol_Analysis-V0_7_0_1.pdf

[49] Lucy Zhang,Building Facebook Messenger, août 2011,

URL:https://www.facebook.com/notes/facebook-engineering/building-facebook-messenger/10150259350998920/

[50] https://aws.amazon.com/fr/iot-platform/how-it-works/

[51] mqtt-topics-best-practices, URL: http://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices.

[52] Building Smarter Planet Solutions with MQTT and IBM WebSphere MQ Telemetry, IBM Redbooks, septembre 2012.p 26.

[53] mqtt-essentials-part-9-last-will-and-testament, URL:http://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament.

[54] Eric J. Bruno,MQTT Programming In Depth, March 2016,

URL: http://programmingwithreason.com/article-mqtt-in-depth.html.

[55] Internet of Things MQTT Quality of Service Levels

URL:http://www.ossmentor.com/2015/04/internet-of-things-mqtt-quality-of.html

[56] Jan-Piet Mens ,Using-the-MQTT-IoT-protocol-for-unusual-but-useful-purposes

URL : http://www.admin-magazine.com/Archive/2015/30/Using-the-MQTT-IoT-protocol-for-unusual-but-useful-purposes

[57] Travaux de la 4e promotion (2013-2014) du Cycle « Sécurité des usages numériques »,p 28

[58] Peter R. Egli, MQTT - MQ Telemetry Transport for Message Queueing,2016, p25,

URL: https://www.slideshare.net/PeterREgli/mq-telemetry-transport

[59] http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment.

[60] Copyright IBM Corporation , URL :

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_7.1.0/com.ibm.mq.doc/tt60390_.htm

Bibliographie

[61] Thomas Watteyne, Proposition et validation formelle d'un protocole MAC temps réel pour réseaux de capteurs linéaires sans fils, Mémoire de Master Recherche Informatique-Spécialité Réseaux, Télécommunications et Services-l'INSA de Lyon,2004/2005

[62] R. Alur, C. Courcoubetis et D.L. Dill. Model-checking in dense real-time. Journal of Information and Computation, 104(1):2-34, 1993.

[63] Josué Melka, Model Based Testing des applications du protocole MQTT Mémoire de Master Recherche Informatique, Université franche comté-france, mis en ligne le 13-fivrier 2017.

ANNEXE:

Bibliographie

ANNEXE A : Déclaration nomative du Protocole MQTT-3-1.1

Number	Normative Statement
[MQTT-1.5.3-1]	The character data in a UTF-8 encoded string MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular this data MUST NOT include encodings of code points between U+D800 and U+DFFF. If a Server or Client receives a Control Packet containing ill-formed UTF-8 it MUST close the Network Connection.
[MQTT-1.5.3-2]	A UTF-8 encoded string MUST NOT include an encoding of the null character U+0000. If a receiver (Server or Client) receives a Control Packet containing U+0000 it MUST close the Network Connection.
[MQTT-1.5.3-3]	A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always to be interpreted to mean U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver.
[MQTT-2.2.2-1]	Where a flag bit is marked as “Reserved” in Table 2.2 - Flag Bits, it is reserved for future use and MUST be set to the value listed in that table.
[MQTT-2.2.2-2]	If invalid flags are received, the receiver MUST close the Network Connection.
[MQTT-2.3.1-1]	SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) Control Packets MUST contain a non-zero 16-bit Packet Identifier.
[MQTT-2.3.1-2]	Each time a Client sends a new packet of one of these types it MUST assign it a currently unused Packet Identifier.
[MQTT-2.3.1-3]	If a Client re-sends a particular Control Packet, then it MUST use the same Packet Identifier in subsequent re-sends of that packet. The Packet Identifier becomes available for reuse after the Client has processed the corresponding acknowledgement packet. In the case of a QoS 1 PUBLISH this is the corresponding PUBACK; in the case of QO2 it is PUBCOMP. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK.
[MQTT-2.3.1-4]	The same conditions [MQTT-2.3.1-3] apply to a Server when it sends a PUBLISH with QoS >0.
[MQTT-2.3.1-5]	A PUBLISH Packet MUST NOT contain a Packet Identifier if its QoS value is set to 0.
[MQTT-2.3.1-6]	A PUBACK, PUBREC or PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH Packet that was originally sent.
[MQTT-2.3.1-7]	Similarly to [MQTT-2.3.1-6], SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE Packet respectively.
[MQTT-3.1.0-1]	After a Network Connection is established by a Client to a Server, the first Packet sent from the Client to the Server MUST be a CONNECT Packet.
[MQTT-3.1.0-2]	The Server MUST process a second CONNECT Packet sent from a Client as a protocol violation and disconnect the Client.
[MQTT-3.1.2-1]	If the protocol name is incorrect the Server MAY disconnect the Client, or it MAY continue processing the CONNECT packet in accordance with some other specification. In the latter case, the Server MUST NOT continue to process the CONNECT packet in line with this specification.
[MQTT-3.1.2-2]	The Server MUST respond to the CONNECT Packet with a CONNACK return code 0x01 (unacceptable protocol level) and then disconnect the Client if the Protocol Level is not supported by the Server.
[MQTT-3.1.2-3]	The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero.
[MQTT-3.1.2-4]	If CleanSession is set to 0, the Server MUST resume communications with the Client based on state from the current Session (as identified by the Client identifier). If there is no Session associated with the Client identifier the Server MUST create a new Session. The Client and Server MUST store the Session after the Client and Server are disconnected.
[MQTT-3.1.2-5]	After the disconnection of a Session that had CleanSession set to 0, the Server MUST store further QoS 1 and QoS 2 messages that match any subscriptions that the client had at

	the time of disconnection as part of the Session state.
[MQTT-3.1.2-6]	If CleanSession is set to 1, the Client and Server MUST discard any previous Session and start a new one. This Session lasts as long as the Network Connection. State data associated with this Session MUST NOT be reused in any subsequent Session.
[MQTT-3.1.2.7]	Retained messages do not form part of the Session state in the Server, they MUST NOT be deleted when the Session ends.
[MQTT-3.1.2-8]	If the Will Flag is set to 1 this indicates that, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Network Connection. The Will Message MUST be published when the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of a DISCONNECT Packet.
[MQTT-3.1.2-9]	If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the Server, and the Will Topic and Will Message fields MUST be present in the payload.
[MQTT-3.1.2-10]	The Will Message MUST be removed from the stored Session state in the Server once it has been published or the Server has received a DISCONNECT packet from the Client.
[MQTT-3.1.2-11]	If the Will Flag is set to 0 the Will QoS and Will Retain fields in the Connect Flags MUST be set to zero and the Will Topic and Will Message fields MUST NOT be present in the payload.
[MQTT-3.1.2-12]	If the Will Flag is set to 0, a Will Message MUST NOT be published when this Network Connection ends.
[MQTT-3.1.2-13]	If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00).
[MQTT-3.1.2-14]	If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). It MUST NOT be 3 (0x03).
[MQTT-3.1.2-15]	If the Will Flag is set to 0, then the Will Retain Flag MUST be set to 0.
[MQTT-3.1.2-16]	If the Will Flag is set to 1 and If Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message.
[MQTT-3.1.2-17]	If the Will Flag is set to 1 and If Will Retain is set to 1, the Server MUST publish the Will Message as a retained message.
[MQTT-3.1.2-18]	If the User Name Flag is set to 0, a user name MUST NOT be present in the payload.
[MQTT-3.1.2-19]	If the User Name Flag is set to 1, a user name MUST be present in the payload.
[MQTT-3.1.2-20]	If the Password Flag is set to 0, a password MUST NOT be present in the payload.
[MQTT-3.1.2-21]	If the Password Flag is set to 1, a password MUST be present in the payload.
[MQTT-3.1.2-22]	If the User Name Flag is set to 0, the Password Flag MUST be set to 0.
[MQTT-3.1.2-23]	It is the responsibility of the Client to ensure that the interval between Control Packets being sent does not exceed the Keep Alive value. In the absence of sending any other Control Packets, the Client MUST send a PINGREQ Packet.
[MQTT-3.1.2-24]	If the Keep Alive value is non-zero and the Server does not receive a Control Packet from the Client within one and a half times the Keep Alive time period, it MUST disconnect the Network Connection to the Client as if the network had failed.
[MQTT-3.1.3-1]	These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password.
[MQTT-3.1.3-2]	Each Client connecting to the Server has a unique ClientId. The ClientId MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session between the Client and the Server.
[MQTT-3.1.3-3]	The Client Identifier (ClientId) MUST be present and MUST be the first field in the CONNECT packet payload.
[MQTT-3.1.3-4]	The ClientId MUST be a UTF-8 encoded string as defined in Section **Erreur ! Source du renvoi introuvable.**.
[MQTT-3.1.3-5]	The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".
[MQTT-3.1.3-6]	A Server MAY allow a Client to supply a ClientId that has a length of zero bytes. However if it does so the Server MUST treat this as a special case and assign a unique ClientId to that Client. It MUST then process the CONNECT packet as if the Client had

	provided that unique ClientId.
[MQTT-3.1.3-7]	If the Client supplies a zero-byte ClientId, the Client MUST also set CleanSession to 1.
[MQTT-3.1.3-8]	If the Client supplies a zero-byte ClientId with CleanSession set to 0, the Server MUST respond to the CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network Connection.
[MQTT-3.1.3-9]	If the Server rejects the ClientId it MUST respond to the CONNECT Packet with a CONNACK return code 0x02 (Identifier rejected) and then close the Network Connection.
[MQTT-3.1.3-10]	The Will Topic MUST be a UTF-8 encoded string as defined in Section 1.5.3.
[MQTT-3.1.3-11]	The User Name MUST be a UTF-8 encoded string as defined in Section 1.5.3.
[MQTT-3.1.4-1]	The Server MUST validate that the CONNECT Packet conforms to section **Erreur ! Source du renvoi introuvable.** and close the Network Connection without sending a CONNACK if it does not conform.
[MQTT-3.1.4-2]	If the ClientId represents a Client already connected to the Server then the Server MUST disconnect the existing Client.
[MQTT-3.1.4-3]	If CONNECT validation is successful the Server MUST perform the processing of CleanSession that is described in section 3.1.2.4.
[MQTT-3.1.4-4]	If CONNECT validation is successful the Server MUST acknowledge the CONNECT Packet with a CONNACK Packet containing a zero return code.
[MQTT-3.1.4-5]	If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT Packet.
[MQTT-3.2.0-1]	The first packet sent from the Server to the Client MUST be a CONNACK Packet.
[MQTT-3.2.2-1]	If the Server accepts a connection with CleanSession set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a zero return code in the CONNACK packet.
[MQTT-3.2.2-2]	If the Server accepts a connection with CleanSession set to 0, the value set in Session Present depends on whether the Server already has stored Session state for the supplied client ID. If the Server has stored Session state, it MUST set Session Present to 1 in the CONNACK packet.
[MQTT-3.2.2-3]	If the Server does not have stored Session state, it MUST set Session Present to 0 in the CONNACK packet. This is in addition to setting a zero return code in the CONNACK packet.
[MQTT-3.2.2-4]	If a server sends a CONNACK packet containing a non-zero return code it MUST set Session Present to 0.
[MQTT-3.2.2-5]	If a server sends a CONNACK packet containing a non-zero return code it MUST then close the Network Connection.
[MQTT-3.2.2-6]	If none of the return codes listed in **Erreur ! Source du renvoi introuvable.** are deemed applicable, then the Server MUST close the Network Connection without sending a CONNACK.
[MQTT-3.3.1-1]	The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet.
[MQTT-3.3.1-2]	The DUP flag MUST be set to 0 for all QoS 0 messages.
[MQTT-3.3.1-3]	The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH Packet is sent to subscribers by the Server. The DUP flag in the outgoing PUBLISH packet is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission.
[MQTT-3.3.1-4]	A PUBLISH Packet MUST NOT have both QoS bits set to 1. If a Server or Client receives a PUBLISH Packet which has both QoS bits set to 1 it MUST close the Network Connection.
[MQTT-3.3.1-5]	If the RETAIN flag is set to 1, in a PUBLISH Packet sent by a Client to a Server, the Server MUST store the Application Message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name.
[MQTT-3.3.1-6]	When a new subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber.
[MQTT-3.3.1-7]	If the Server receives a QoS 0 message with the RETAIN flag set to 1 it MUST discard any message previously retained for that topic. It SHOULD store the new QoS 0 message

Bibliographie

	as the new retained message for that topic, but MAY choose to discard it at any time - if this happens there will be no retained message for that topic.
[MQTT-3.3.1-8]	When sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag to 1 if a message is sent as a result of a new subscription being made by a Client.
[MQTT-3.3.1-9]	It MUST set the RETAIN flag to 0 when a PUBLISH Packet is sent to a Client because it matches an established subscription regardless of how the flag was set in the message it received.
[MQTT-3.3.1-10]	A PUBLISH Packet with a RETAIN flag set to 1 and a payload containing zero bytes will be processed as normal by the Server and sent to Clients with a subscription matching the topic name. Additionally any existing retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message.
[MQTT-3.3.1-11]	A zero byte retained message MUST NOT be stored as a retained message on the Server.
[MQTT-3.3.1-12]	If the RETAIN flag is 0, in a PUBLISH Packet sent by a Client to a Server, the Server MUST NOT store the message and MUST NOT remove or replace any existing retained message.
[MQTT-3.3.2-1]	The Topic Name MUST be present as the first field in the PUBLISH Packet Variable header. It MUST be a UTF-8 encoded string.
[MQTT-3.3.2-2]	The Topic Name in the PUBLISH Packet MUST NOT contain wildcard characters.
[MQTT-3.3.2-3]	The Topic Name in a PUBLISH Packet sent by a Server to a subscribing Client MUST match the Subscription's Topic Filter according to the matching process defined in Section 4.7.
[MQTT-3.3.4-1]	The receiver of a PUBLISH Packet MUST respond according to Table 3.4 - Expected Publish Packet response as determined by the QoS in the PUBLISH Packet.
[MQTT-3.3.5-1]	The Server MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions.
[MQTT-3.3.5-2]	If a Server implementation does not authorize a PUBLISH to be performed by a Client; it has no way of informing that Client. It MUST either make a positive acknowledgement, according to the normal QoS rules, or close the Network Connection.
[MQTT-3.6.1-1]	Bits 3,2,1 and 0 of the fixed header in the PUBREL Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.8.1-1]	Bits 3,2,1 and 0 of the fixed header of the SUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.8.3-1]	The Topic Filters in a SUBSCRIBE packet payload MUST be UTF-8 encoded strings as defined in Section 1.5.3.
[MQTT-3.8.3-2]	If the Server chooses not to support topic filters that contain wildcard characters it MUST reject any Subscription request whose filter contains them.
[MQTT-3.8.3-3]	The payload of a SUBSCRIBE packet MUST contain at least one Topic Filter / QoS pair. A SUBSCRIBE packet with no payload is a protocol violation.
[MQTT-3-8.3-4]	The Server MUST treat a SUBSCRIBE packet as malformed and close the Network Connection if any of Reserved bits in the payload are non-zero, or QoS is not 0,1 or 2.
[MQTT-3.8.4-1]	When the Server receives a SUBSCRIBE Packet from a Client, the Server MUST respond with a SUBACK Packet.
[MQTT-3.8.4-2]	The SUBACK Packet MUST have the same Packet Identifier as the SUBSCRIBE Packet that it is acknowledging.
[MQTT-3.8.4-3]	If a Server receives a SUBSCRIBE Packet containing a Topic Filter that is identical to an existing Subscription's Topic Filter then it MUST completely replace that existing Subscription with a new Subscription. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its maximum QoS value could be different. Any existing retained messages matching the Topic Filter MUST be re-sent, but the flow of publications MUST NOT be interrupted.
[MQTT-3.8.4-4]	If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses into a single SUBACK response.

Bibliographie

[MQTT-3.8.4-5]	The SUBACK Packet sent by the Server to the Client MUST contain a return code for each Topic Filter/QoS pair. This return code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed.
[MQTT-3.8.4-6]	The Server might grant a lower maximum QoS than the subscriber requested. The QoS of Payload Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published message and the maximum QoS granted by the Server. The server is permitted to send duplicate copies of a message to a subscriber in the case where the original message was published with QoS 1 and the maximum QoS granted was QoS 0.
[MQTT-3.9.3-1]	The order of return codes in the SUBACK Packet MUST match the order of Topic Filters in the SUBSCRIBE Packet.
[MQTT-3.9.3-2]	SUBACK return codes other than 0x00, 0x01, 0x02 and 0x80 are reserved and MUST NOT be used.
[MQTT-3.10.1-1]	Bits 3,2,1 and 0 of the fixed header of the UNSUBSCRIBE Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.10.3-1]	The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 encoded strings as defined in Section **Erreur ! Source du renvoi introuvable.**, packed contiguously.
[MQTT-3.10.3-2]	The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter. An UNSUBSCRIBE packet with no payload is a protocol violation.
[MQTT-3.10.4-1]	The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription is deleted, otherwise no additional processing occurs.
[MQTT-3.10.4-2]	If a Server deletes a Subscription It MUST stop adding any new messages for delivery to the Client.
[MQTT-3.10.4-3]	If a Server deletes a Subscription It MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client.
[MQTT-3.10.4-4]	The Server MUST respond to an UNSUBSUBCRIBE request by sending an UNSUBACK packet. The UNSUBACK Packet MUST have the same Packet Identifier as the UNSUBSCRIBE Packet.
[MQTT-3.10.4-5]	Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK.
[MQTT-3.10.4-6]	If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one UNSUBACK response.
[MQTT-3.12.4-1]	The Server MUST send a PINGRESP Packet in response to a PINGREQ packet.
[MQTT-3.14.1-1]	The Server MUST validate that reserved bits are set to zero and disconnect the Client if they are not zero.
[MQTT-3.14.4-1]	After sending a DISCONNECT Packet the Client MUST close the Network Connection.
[MQTT-3.14.4-2]	After sending a DISCONNECT Packet the Client MUST NOT send any more Control Packets on that Network Connection.
[MQTT-3.14.4-3]	On receipt of DISCONNECT the Server MUST discard any Will Message associated with the current connection without publishing it, as described in Section **Erreur ! Source du renvoi introuvable.**.
[MQTT-4.1.0-1]	The Client and Server MUST store Session state for the entire duration of the Session.
[MQTT-4.1.0-2]	A Session MUST last at least as long it has an active Network Connection.
[MQTT-4.3.1-1]	In the QoS 0 delivery protocol, the Sender • MUST send a PUBLISH packet with QoS=0, DUP=0.
[MQTT-4.3.2-1]	In the QoS 1 delivery protocol, the Sender • MUST assign an unused Packet Identifier each time it has a new Application Message to publish. • MUST send a PUBLISH Packet containing this Packet Identifier with QoS=1, DUP=0. • MUST treat the PUBLISH Packet as "unacknowledged" until it has received the

	corresponding PUBACK packet from the receiver. See Section **Erreur ! Source du renvoi introuvable.** for a discussion of unacknowledged messages.
[MQTT-4.3.2-2]	In the QoS 1 delivery protocol, the Receiver • MUST respond with a PUBACK Packet containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message. • After it has sent a PUBACK Packet the Receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new publication, irrespective of the setting of its DUP flag.
[MQTT-4.3.3-1]	In the QoS 2 delivery protocol, the Sender • MUST assign an unused Packet Identifier when it has a new Application Message to publish. • MUST send a PUBLISH packet containing this Packet Identifier with QoS=2, DUP=0. • MUST treat the PUBLISH packet as "unacknowledged" until it has received the corresponding PUBREC packet from the receiver. See Section **Erreur ! Source du renvoi introuvable.** for a discussion of unacknowledged messages. • MUST send a PUBREL packet when it receives a PUBREC packet from the receiver. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet. • MUST treat the PUBREL packet as "unacknowledged" until it has received the corresponding PUBCOMP packet from the receiver. • MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet.
[MQTT-4.3.3-2]	In the QoS 2 delivery protocol, the Receiver • MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message. • Until it has received the corresponding PUBREL packet, the Receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case. • MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL. • After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication.
[MQTT-4.4.0-1]	When a Client reconnects with CleanSession set to 0, both the Client and Server MUST re-send any unacknowledged PUBLISH Packets (where QoS > 0) and PUBREL Packets using their original Packet Identifiers.
[MQTT-4.5.0-1]	When a Server takes ownership of an incoming Application Message it MUST add it to the Session state of those clients that have matching Subscriptions. Matching rules are defined in Section 4.7.
[MQTT-4.5.0-2]	The Client MUST acknowledge any Publish Packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains.
[MQTT-4.6.0-1]	When it re-sends any PUBLISH packets, it MUST re-send them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages).
[MQTT-4.6.0-2]	Client MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages).
[MQTT-4.6.0-3]	Client MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages).
[MQTT-4.6.0-4]	Client MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages).
[MQTT-4.6.0-5]	A Server MUST by default treat each Topic as an "Ordered Topic". It MAY provide an administrative or other mechanism to allow one or more Topics to be treated as an

Bibliographie

	"Unordered Topic".
[MQTT-4.6.0-6]	When a Server processes a message that has been published to an Ordered Topic, it MUST follow the rules listed above when delivering messages to each of its subscribers. In addition it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client.
[MQTT-4.7.1-1]	The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name.
[MQTT-4.7.1-2]	The multi-level wildcard character MUST be specified either on its own or following a topic level separator. In either case it MUST be the last character specified in the Topic Filter.
[MQTT-4.7.1-3]	The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where it is used it MUST occupy an entire level of the filter.
[MQTT-4.7.2-1]	The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character.
[MQTT-4.7.3-1]	All Topic Names and Topic Filters MUST be at least one character long.
[MQTT-4.7.3-2]	Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000).
[MQTT-4.7.3-3]	Topic Names and Topic Filters are UTF-8 encoded strings, they MUST NOT encode to more than 65535 bytes.
[MQTT-4.7.3-4]	When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters.
[MQTT-4.8.0-1]	Unless stated otherwise, if either the Server or Client encounters a protocol violation, it MUST close the Network Connection on which it received that Control Packet which caused the protocol violation.
[MQTT-4.8.0-2]	If the Client or Server encounters a Transient Error while processing an inbound Control Packet it MUST close the Network Connection on which it received that Control Packet.

Source: MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS

Bibliographie

ANNEXE B : Methods for using message queuing telemetry transport for sensor networks to support sleeping devices

Numéro de publication	US20090172117 A1
Type de publication	Demande
Numéro de demande	US 11/968,466
Date de publication	2 juil. 2009
Date de dépôt	2 janv. 2008
Date de priorité	2 janv. 2008
Inventeurs	Bharat V. Bedi, 6 autre(s) »
Cessionnaire d'origine	International Business Machines Corporation

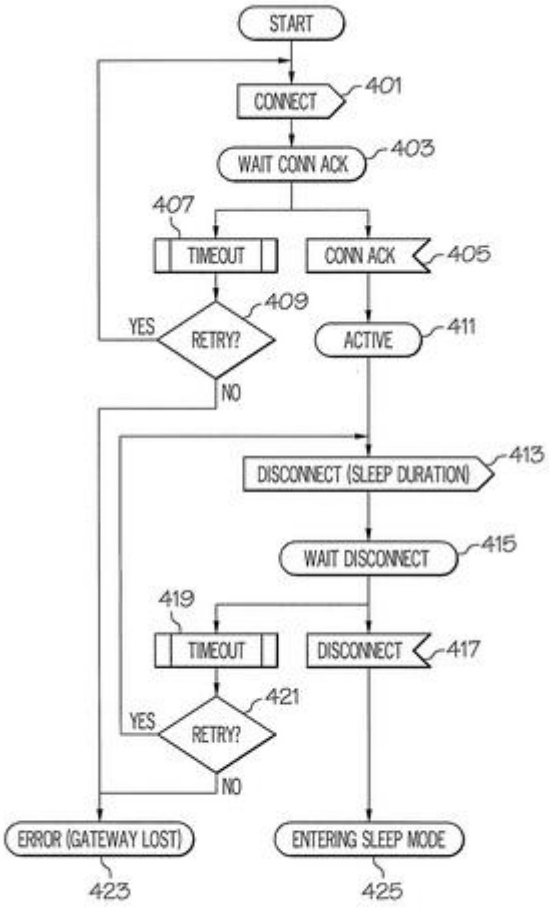


FIG. 4

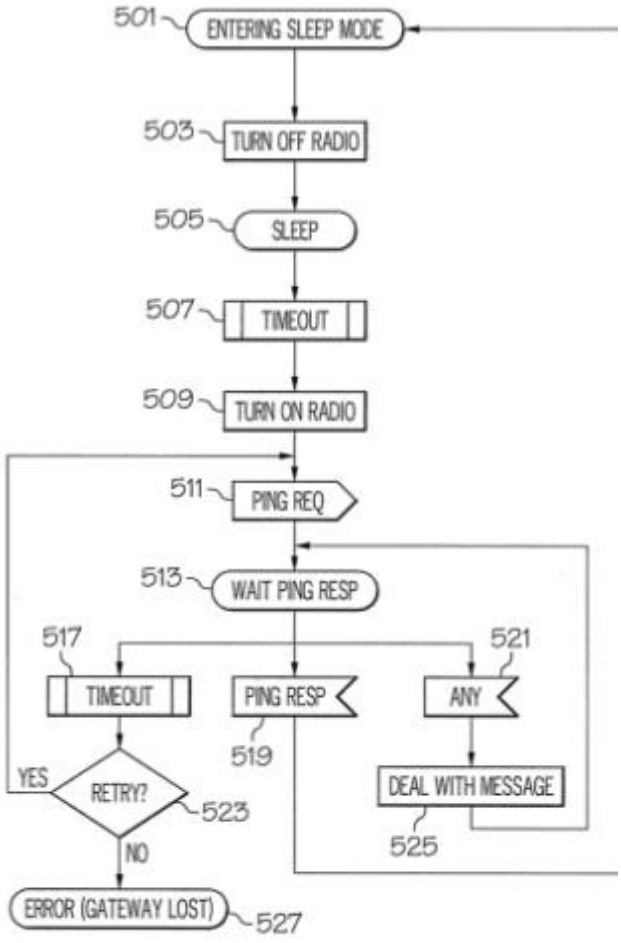


FIG. 5

Source: http://www.google.com/patents/US20090172117.