UNIVERSITÉ DE SHERBROOKE
Département d'informatique

Détection intelligente de la fumée

Par
Jean-Philippe Boivin, 13 042 450
Samuel Bouffard, 12 224 103

Travail présenté à Bessam Abdulrazak
Dans le cadre du cours
IFT 744 — Sujets avancés en télématique

13 avril 2018

1. Introduction **1**

2. Solution proposée **1**

2.1 Fonctionnalités *2*

2.1.1 Détection de la fumée et de gaz nocifs *2*

2.1.2 Détection des pannes *2*

2.1.2.1 Défaillance d'un noeud de détection *2*

2.1.2.2 Défaillance du noeud maître *2*

2.1.3 Alertes *2*

2.1.4 Auto-configuration des noeuds *3*

2.2 Protocoles *3*

2.2.1 Protocole de données *3*

2.2.1.1 Sujets *3*

2.2.1.2 Qualité de service *3*

2.2.1.3 Sécurité *3*

2.2.1.4 Messages *4*

2.2.2 Protocole de découverte *4*

2.2.2.1 Sécurité *4*

2.2.2.2 Messages *4*

2.2.2.3 Protocoles standards *4*

2.2.3 Protocole d'auto-configuration *4*

2.2.3.1 Sécurité *4*

2.2.3.2 Messages *5*

3. Implémentation et résultats **5**

3.1 Modules *5*

3.1.1 Noeud de détection *5*

3.1.1.1 Matériel *5*

3.1.2 Noeud maître *6*

3.1.2.1 Matériel *6*

3.2 Fonctionnalités *6*

3.2.1 Auto-configuration *6*

3.3 Protocoles *6*

3.3.1 Protocole de découverte *6*

3.3.2 Protocole d'auto-configuration *7*

3.3.3 Protocole des noeuds *7*
3.4 Résultats *7*
4. Limites de la solution proposée **7**
4.1 Réseau de communication préexistant nécessaire *7*
4.2 Point de défaillance central *7*
4.3 Géolocalisation des noeuds *7*
4.4 Format des messages (données et commandes) *8*
5. Conclusion **8**
Références **9**

Détection intelligente de la fumée

JEAN-PHILIPPE BOIVIN & SAMUEL BOUFFARD

1. Introduction

La détection de la fumée et des gaz nocifs est une problématique importante pour la sécurité des individus, et ce dans plusieurs secteurs d'activités. En effet, il peut être désirable de détecter la fumée dans un environnement familial (une maison), un environnement commercial (un immeuble), un environnement non contrôlé (une forêt), etc pour prévenir les incendies ou réagir rapidement. Plusieurs systèmes permettent la détection de la fumée et plus. Cependant, plusieurs nécessitent une installation par un professionnel. Or, plusieurs maisons n'ont toujours que de simple détecteur à batterie, et le coût d'installation de détecteurs connectés peut être prohibitif à cause des travaux nécessaires. Ainsi, la solution proposée vise la simplicité de configuration, ainsi que l'installation dans tous milieux possédant un simple réseau. Les simples détecteurs à batterie n'étant pas interconnectés, un bâtiment ayant une surface non négligeable peut nécessiter plusieurs détecteurs qui ne seront pas en mesure de notifier toutes les zones dudit bâtiment. La solution proposée, étant connectée, règle cette problématique.

2. Solution proposée

La solution qui est proposée à la problématique de la détection intelligente de la fumée et des gaz nocifs est représenté dans la figure 1. Il s'agit d'un réseau constitué de plusieurs noeuds de détection et d'un noeud de gestion. Le noeud de gestion est en charge du traitement des données provenant des capteurs, envoyées par les autres noeuds. C'est ce noeud central qui est responsable de détecter et réagir (alarme, notifications, appel des services d'urgence, etc).

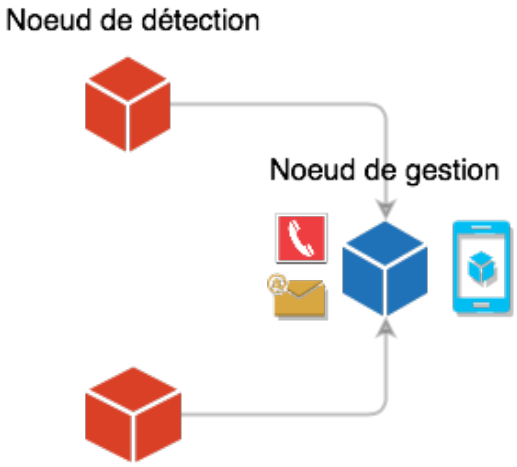

Figure 1. Modules du réseau

Les fonctionnalités minimales qui doivent être fournies par le système sont: la détection de la fumée, la détection du monoxyde de carbone, les alertes sonores, la détection des pannes. D'autres fonctionnalités d'un tel système pourraient être l'auto-configuration des noeuds, les alertes SMS, les alertes par courriel, la communication avec les services d'urgence, etc. En effet, le noeud de gestion possède toutes les informations nécessaires afin d'y greffer davantage de services.

L'architecture proposée suppose que tous les noeuds sont sur le même réseau. Cependant, la conception des protocoles est agnostique de la position des noeuds en général. En effet, seul le protocole de découverte requiert que tous les noeuds soient sur un même réseau local, mais ce protocole est facilement remplaçable. Il faut toutefois noter que si le noeud maître se trouve sur un autre réseau, des problèmes de pare-feu, traduction d'adresse réseau et d'accessibilité peuvent apparaître et ne sont pas couvert par cet article.

2.1 Fonctionnalités

2.1.1 Détection de la fumée et de gaz nocifs

Pour la détection de la fumée et des gaz nocifs, il est proposé que les noeuds de détection envoient la concentration en PPM au noeud maître. Afin de minimiser l'impact sur le réseau, il est vital que les noeuds n'envoient les données que lorsqu'un changement marqué est détecté. Le noeud maître pourra de son côté forcer les noeuds de détection à envoyer l'état actuel de leurs capteurs en envoyant un message. Ainsi, les messages de données servent également de *heartbeat*.

Le noeud maître est responsable de détecter un seuil alarmant de fumée ou de gaz. Pour ce faire, il doit périodiquement valider les différentes valeurs des noeuds. Lorsqu'une concentration alarmante est détectée, il doit réagir, par exemple en déclenchant une alerte. De même, si l'état actuel est en alerte et que la concentration se rétablit à des niveaux normaux, il doit arrêter les alertes.

2.1.2 Détection des pannes

Le système étant distribué, plusieurs composants peuvent montrer des défaillances. La défaillance des capteurs eux-mêmes ne sera cependant pas abordée par cet article puisque la détection de cette problématique dépend des capteurs eux-mêmes. De la redondance pourrait être utilisée, par exemple.

2.1.2.1 Défaillance d'un noeud de détection

Dans le cas où un noeud de détection est défaillant, le noeud maître ne recevra pas les données des capteurs, malgré ses tentatives de forcer l'envoie de celles-ci. Après un nombre défini de tentatives, le noeud maître doit se mettre en état d'alerte. Dans un contexte où l'alerte ne peut explicitement identifier le noeud défaillant (e.g. alerte sonore), alors tous les noeuds fonctionnels doivent se mettre en état d'alerte. Pour y arriver, le noeud maître doit leur envoyer une commande.

2.1.2.2 Défaillance du noeud maître

Dans le cas où le noeud maître serait défaillant, les noeuds de détection doivent être en mesure de détecter cette situation. À cause du protocole proposé à la section 2.2.1, les noeuds ne peuvent détecter directement cette défaillance. De plus, le noeud maître n'envoie aucune donnée périodique permettant de signaler sa présence. C'est pourquoi un *heartbeat* est nécessaire. Un noeud de détection ne recevant pas un *heartbeat* pendant N + S secondes (où N est la période à laquelle un *heartbeat* est envoyé et S une marge plus petite que cette dite période) doit entrer en alerte jusqu'à ce que la communication avec le noeud maître soit rétablie.

2.1.3 Alertes

La solution proposée utilise le son comme moyen d'alerter. Il faut noter que cela ne serait pas suffisant dans un contexte extérieur ou éloigné. Ainsi, la fonctionnalité d'alertes est abordée de manière générique, soit comme un ensemble de drapeaux. Plusieurs services peuvent donc s'y rattacher, que ce soit des alertes sonores, par courriel ou des SMS.

Comme spécifié dans les sections précédentes, les situations problématiques sont les suivantes:

1. La connexion avec le *broker* est non-fonctionnelle;
2. La connexion avec un ou plusieurs des noeuds de détection est non-fonctionnelle;
3. La connexion avec le noeud maître est non-fonctionnelle;
4. Un noeud détecte une concentration anormale de fumée;
5. Un noeud détecte une concentration anormale d'un gaz nocif.

L'ordre des situations est important dans un contexte où une seule situation peut être rapportée à la fois, par exemple, par des alertes sonores. En effet, si la connexion avec le *broker* est non fonctionnel, même si le drapeau pour la concentration élevée de fumée est levé, ce drapeau

n'est pas fiable puisqu'il se peut que des messages ait été manqué entre-temps.

2.1.4 Auto-configuration des noeuds

La solution proposée vise à être facilement déployable. Ainsi, il est désirable qu'un noeud, lorsque placé dans un environnement où un noeud maître est déjà présent, puisse s'auto-configurer avec celui-ci.

Pour y arriver, le noeud de détection doit tout d'abord découvrir le noeud maître. Un protocole de découverte spécifié plus tard est utilisé à cet effet. Ensuite, connaissant l'adresse et les services offerts par le noeud maître, il peut s'y connecter. Le noeud maître générera alors la configuration.

Sachant que le processus de découverte et de connexion peut ne pas toujours être fiable, il est proposé d'effectuer ce processus au maximum trois fois.

2.2 Protocoles

Il est proposé d'utiliser quatre protocoles différents pour les différents services offerts par le noeud de gestion. Afin de simplifier la définition des messages des différents protocoles, il est proposé d'utiliser Google Protobuf, qui permet de définir les messages dans une syntaxe neutre, et qui offre également la sérialisation binaire optimisée.

2.2.1 Protocole de données

Afin de publier les données des capteurs au noeud maître, il est proposé d'utiliser le protocole MQTT puisque ce protocole est adapté au domaine de l'IoT. Le *broker*, dans le solution proposée, se retrouve sur le noeud maître. Cela permet au noeud maître de garantir que celui-ci est toujours connecté au *broker*, ainsi que d'éviter une configuration du maître.

2.2.1.1 Sujets

Les sujets suivants sont utilisés dans le protocole proposé:

- fsh/master/heartbeat
- fsh/master/command
- fsh/master/command/{SUID}
- fsh/slaves/{SUID}/smoke
- fsh/slaves/{SUID}/carbon_monoxide

2.2.1.2 Qualité de service

Le protocole MQTT offre trois nivaux de qualité de service au niveau du *broker*:

- Niveau 0: Corresponds à l'envoi d'un message sans se soucier de sa réception;
- Niveau 1: Corresponds à l'envoi d'un message en s'assurant qu'il soit reçu au moins une fois;
- Niveau 2: Corresponds à l'envoi d'un message en s'assurant qu'il soit reçu une et seulement une fois.

Il est proposé d'utiliser les niveaux 0 et 1 afin de minimiser le coût de l'envoi des données. Les données de capteurs peuvent être perdues puisqu'elles sont envoyées lors des changements et que le maître peut forcer l'envoi de celles-ci à l'aide d'une commande. Le niveau zéro est donc adapté. Ensuite, les commandes doivent être reçues, ainsi le niveau 1 est requis.

Comme certains messages peuvent être reçus plusieurs fois, il est proposé de les horodater afin d'ignorer les duplicatas.

2.2.1.3 Sécurité

Les données transmises (des concentrations de gaz) ne sont pas confidentielles. Cependant, il est vital de s'assurer de protéger le réseau de noeuds de messages forgés. La solution proposée pour garantir l'intégrité et l'authenticité des messages est l'utilisation de HMAC (avec SHA-256). Cela implique un échange préalable d'un secret partagé. Cet échange n'est pas couvert par le protocole de données. Afin de minimiser l'impact sur la taille des messages, la signature sera tronquée de moitié. La troncation de la signature sera effectuée comme spécifié par la section 5 du RFC 2104. Il faut noter que la troncation expose moins de données du hash

à un attaquant, mais réduit la quantité de bits à forger.

L'horodatage des messages offre une certaine protection contre les attaques de rejeu. En effet, un attaquant ne pourrait renvoyer un ancien message de données si le maître a déjà reçu un message plus récent.

2.2.1.4 Messages

Tout d'abord, peu importe le format utilisé pour sérialiser les données applicatives, il est nécessaire d'avoir l'horodatage et la signature dans les en-têtes. En effet, il faut pouvoir ignorer les messages à coût presque nul. En mettant ces deux informations dans les en-têtes, il est par exemple possible de savoir si le message doit être désérialisé ou non, et ainsi éviter du temps de traitement.

Le protocole est constitué de trois messages pour envoyer un *heartbeat*, des commandes ou des données de capteurs. Le *heartbeat* n'a pas besoin de données. Le message de commande contient simplement un identifiant, et finalement le message de capteur doit contenir le type de capteur ainsi qu'une valeur.

2.2.2 Protocole de découverte

Le protocole de découverte qui est proposé suppose que tous les noeuds se trouvent sur un même réseau local, et qu'il est possible de diffuser des paquets. En effet, le noeud de détection ou l'application désirant communiquer avec le noeud maître devra diffuser une requête sur toutes ses interfaces réseau, à laquelle le noeud maître répondra avec l'information des services qu'il offre.

2.2.2.1 Sécurité

Le protocole de découverte retourne des informations publiques, comme le fait le protocole DNS. Il n'est donc pas nécessaire d'offrir la confidentialité. Cependant, comme il est basé sur le protocole UDP, la première entité à répondre sera le noeud maître découvert. Cela ouvre la porte à une tierce partie étant sur le réseau d'usurper l'identité du noeud maître. Ainsi, il serait nécessaire d'offrir l'authentification et l'intégrité dans le protocole. Cette sécurité additionnelle n'est cependant pas proposée dans cet article puisqu'il est supposé que tous les noeuds se situent sur un réseau local, qui peut être considéré comme étant fiable.

Des protocoles de sécurité comme DNSSEC et TLS peuvent d'inspiration pour ajouter l'intégrité et l'authentification au protocole.

2.2.2.2 Messages

Le protocole de découverte est constitué de deux messages: la requête et la réponse. La requête ne contient aucune donnée, elle ne fait que signaler le désir de découvrir un noeud maître, tandis que la réponse doit contenir les informations suivantes: l'identifiant du noeud, les différents ports des services.

2.2.2.3 Protocoles standards

Plusieurs protocoles existent déjà et permettent la découverte de services. Par exemple, zeroconf (Bonjour) aurait permis de découvrir. Cependant, ces protocoles sont généralement plus lourds et complexes. Le protocole proposé est simplifié à son maximum afin de minimiser le trafic et le temps d'antenne.

2.2.3 Protocole d'auto-configuration

Le protocole d'auto-configuration qui est proposé ne couvre pas la découverte du noeud, qui fait partie du protocole de découverte. Le protocole proposé se base sur TLS 1.2. En effet, il est important d'échanger les données de manière confidentielle puisqu'elles incluent un secret partagé, mais aussi que la fiabilité du protocole TCP permet de simplifier l'implémentation d'un tel système. La connexion étant de courte durée et peu fréquente, l'overhead par rapport à un protocole plus léger comme DTLS sera négligeable.

2.2.3.1 Sécurité

Comme mentionné, la confidentialité est primordiale pour la configuration, d'où l'utilisation

de TLS. Il est fortement recommandé d'utiliser des suites fortes, soit: ECDHE-RSA-AES-GCM-SHA384 ou ECDHE-ECDSA-AES-GCM-SHA384. Ensuite, le protocole TLS offre la validation d'un certificat client en option. Cette option doit être activée. En effet, il est essentiel autant pour le client que le serveur de s'assurer que l'autre entité est valide et fiable. Sinon, toute personne découvrant le protocole pourrait s'auto-configurer avec le noeud maître, et envoyer des données falsifiées. Chaque noeud devrait avoir un certificat différent, signé par une même autorité de certification. Celle-ci n'a pas besoin d'être publique, mais identique pour un même ensemble de noeuds. Cela permet de valider l'authenticité de chaque noeud du réseau.

2.2.3.2 Messages

Le protocole est constitué d'un seul message, qui doit être envoyé par le serveur dès qu'une nouvelle connexion est complétée. Ce message doit contenir l'identifiant unique du noeud, son secret partagé et le secret partagé du maître. En générant toutes les informations sur le maître, cela évite un échange de messages.

Le secret partagé doit être généré à partir d'un générateur de nombres aléatoires fiable, et doit avoir une longueur de 32 octets.

3. Implémentation et résultats

L'implémentation qui a été développée n'est que partielle, mais elle couvre les fonctionnalités minimales. Il a été décidé d'utiliser des Raspberry Pi pour les différents noeuds, cependant les modules développés en C++ sont fonctionnels sur macOS et Windows également. L'implémentation se divise en deux modules, soit le noeud de détection (fsh-node) et le noeud maître (fsh-master).

Les deux modules développés sont divisés en gestionnaires qui communiquent entre eux par événements, à travers un répartiteur d'événements. Les événements proviennent d'un réservoir d'objets. Ainsi, le travail exécuté périodiquement par chaque gestionnaire minimise les allocations mémoires et la synchronisation.

3.1 Modules

3.1.1 Noeud de détection

Le noeud de détection a pour mandat principal d'effectuer l'acquisition des données provenant de différents capteurs.

3.1.1.1 Matériel

L'acquisition des données des capteurs se fait à travers les ports GPIO du Raspberry Pi.

Afin de détecter la concentration de fumée dans l'air, un capteur MQ-2 a été utilisé, tandis qu'un capteur MQ-9 a été utilisé pour le monoxyde de carbone. Ces capteurs offrent une lecture analogique variant entre 0 V et 5 V, représentant la concentration en PPM de certains gaz. Il est vital de calibrer les capteurs afin d'associer la tension à la concentration. La plateforme choisie n'ayant que des entrées digitales, il a fallu utiliser un convertisseur analogique-numérique de 16 bits, soit l'ADS1115. Ce convertisseur offre quatre canaux, ou deux canaux différentiels et offre une sortie digitale utilisant le protocole I2C pour la communication avec l'hôte. Le Raspberry Pi supporte nativement le protocole I2C. Une autre limitation de la plateforme choisie est la tension maximale des GPIO. En effet, ceux-ci ne permettent pas une tension supérieure à 3.3 V, or, la sortie du convertisseur analogique-numérique est de 5 V. Il a donc été nécessaire d'utiliser un convertisseur de niveau logique bidirectionnel.

Afin d'offrir une fonctionnalité d'auto-configuration, un bouton poussoir est connecté à un port GPIO du Raspberry Pi. Lorsqu'il est enfoncé pour une période de 5 secondes, le mode d'auto-configuration est activé.

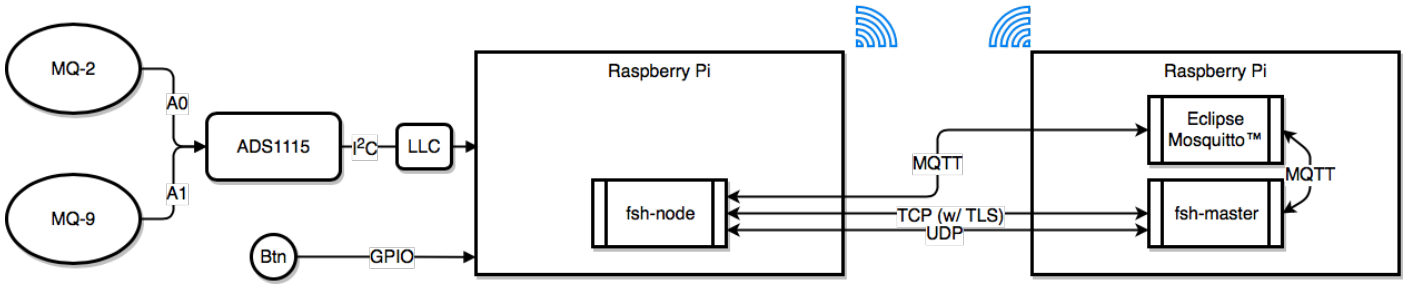

Figure 2. Architecture matérielle et logicielle du système développé

Finalement, le Raspberry Pi possède un port audio qui est utilisé pour y raccorder un haut-parleur pour les alertes sonores.

3.1.2 Noeud maître

Le mandat principal du noeud maître est de réagir selon les données reçues des autres noeuds. En plus du service développé, un *broker* MQTT (Eclipse Mosquitto™) est exécuté.

3.1.2.1 Matériel

Comme pour les noeuds de détection, le port audio du Raspberry Pi est utilisé pour y raccorder un haut-parleur afin d'émettre les alertes sonores.

3.2 Fonctionnalités

Les fonctionnalités implémentés pour ce prototype sont la détection de la fumée, la détection du monoxyde de carbone, les alertes sonores, la détection des pannes et l'auto-configuration des noeuds.

3.2.1 Auto-configuration

Le mode d'auto-configuration a été implémenté pour être lancé automatiquement lors du lancement d'un noeud, si celui-ci n'est pas préalablement configuré. Il est également possible de forcer la configuration en maintenant un bouton enfoncé.

L'auto-configuration fonctionne comme suit:

1. Le noeud tente de découvrir un noeud maître à l'aide du protocole de découverte
2. Le noeud établit une connexion TLS avec le serveur (cela implique la validation des certificats)
3. Le serveur génère la configuration du noeud et l'écrit dans un fichier INI
4. Le serveur envoie la configuration au noeud
5. Le noeud écrit la configuration dans un fichier INI

Si aucun noeud n'est découvert pendant une période de 3 secondes ou que la connexion avec le maître échoue, une nouvelle tentative sera essayé. Après trois échecs, le mode d'auto-configuration est annulé.

Il faut noter que dans une implémentation complète, le noeud maître devrait stocker les configurations dans une base de données (par exemple SQLite), plutôt que des fichiers textes non-optimaux.

3.3 Protocoles

3.3.1 Protocole de découverte

Le protocole de découverte étant basé sur UDP, il a été décidé d'utiliser une implémentation expérimentale du Networking TS[1], basée sur Boost.ASIO. Il est ainsi possible d'utiliser le même code pour toutes les plateformes. Les données sont sérialisées à l'aide de Protobuf comme recommandé.

[1] http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/n4734.pdf

3.3.2 Protocole d'auto-configuration

Le protocole d'auto-configuration étant basé sur TLS 1.2, il a été décidé d'utiliser LibreSSL (2.6.4) qui offre un API plus moderne à travers leur librairie libtls. Le client TLS utilise une lecture bloquante sur un fil séparé, tandis que le serveur TLS utilise des API événementiels propres à chaque plateforme (à l'exception de Windows), également sur un fil séparé. Cela offre de meilleures performances que d'utiliser poll ou select. Sur GNU/Linux, l'API asynchrone epoll est utilisé, tandis que sur macOS il s'agit de kqueue. L'implémentation sur Windows n'utilise pas IOCP, elle effectue plutôt des validations périodiques. L'implémentation sur Windows devrait être retravaillée.

Au niveau des données, elles sont sérialisées à l'aide de Protobuf.

Finalement, le secret partagé est généré à l'aide du générateur de nombres aléatoires (RNG) de LibreSSL. Celui-ci utilise la meilleure source d'entropie disponible sur le système, qu'elle soit logicielle ou matérielle. L'identifiant unique (UUID version 4) est quant à lui généré en utilisant un générateur propre à chaque plateforme (WinAPI, CoreFoundation, libuuid).

3.3.3 Protocole des noeuds

Le protocole des noeuds étant basé sur MQTT, il a été décidé d'utiliser mosquitto. Les en-têtes sont codés dans une structure de données pouvant être mappées directement à la mémoire, ce qui simplifie le traitement de ceux-ci. Les données sont quant à elles sérialisées à l'aide de protobuf.

La signature (généré à l'aide de HMAC) est implémentée en utilisant la librairie LibreSSL, qui était déjà utilisée dans le projet. LibreSSL n'utilise cependant pas la carte graphique pour l'algorithme SHA-256.

3.4 Résultats

En bref, il a été possible d'implémenter la solution proposée sur des Raspberry Pi. Le système développé détecte avec succès les concentrations anormales de fumés et de monoxyde de carbone, et réagit en conséquence en émettant une alarme sonore. De même, la panne d'un noeud est détectée par le système. Globalement, ce système utilise une quantité raisonnable de ressources (temps de calcul, mémoire et réseau).

4. Limites de la solution proposée

La solution proposée possède certaines limitations.

4.1 Réseau de communication préexistant nécessaire

La solution proposée suppose un réseau de communication IP préexistant. En effet, le protocole principal du système est MQTT sur TCP/IP. Il serait cependant possible d'utiliser MQTT-SN et de modifier l'architecture en conséquence. Cela permettrait de créer un réseau (par exemple ZigBee) totalement indépendant.

4.2 Point de défaillance central

La solution proposée crée un point de défaillance central, c'est-à-dire le noeud maître. Si ce noeud est défaillant, la détection devient non-fonctionnelle. Pour y remédier, il faudrait évaluer la possibilité d'avoir de la redondance, ou d'élire un nouveau maître.

4.3 Géolocalisation des noeuds

La solution proposée gère l'auto-configuration des noeuds de détection. Cependant, pour que cette auto-configuration soit fonctionnelle, il est nécessaire que le système n'ait pas besoin de connaître la position des noeuds. En effet, rien dans les technologies proposées ou utilisées ne permet de localiser un noeud. Il faudrait donc manuellement spécifier cette information.

4.4 Format des messages (données et commandes)

La solution proposée n'offre aucune métadonnée aux noeuds lors de l'envoi des données et des commandes. Ainsi, le noeud récepteur doit absolument connaître la commande ou le type de capteur pour savoir quelles actions entreprendre.

5. Conclusion

En conclusion, la solution proposée est fonctionnelle. Elles possèdent certaines limitations qui peuvent être facilement contournées en adaptant certains modules puisque ceux-ci sont relativement indépendants. La solution n'est pas adaptée à toutes les situations, par exemple elle n'est pas adaptée à la détection des feux de forêt, mais cette base pourrait tout de même être utilisée dans un tel contexte en continuant le travail.

Références

« MQTT Version 3.1.1 », OASIS Standard, Octobre 2014.
<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf>

« MQTT For Sensor Networks (MQTT-SN) Version 1.2 », Novembre 2013.
<http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf>

E. Rescorla. « The Transport Layer Security (TLS) Protocol Version 1.2 », RFC 5246, Août 2008.
<https://tools.ietf.org/html/rfc5246>

H. Krawczyk, M. Bellare, R. Canetti. « HMAC: Keyed-Hashing for Message Authentication », RFC 2104, Février 1997.
<https://tools.ietf.org/html/rfc2104>

P. Leach, M. Mealling, R. Salz. « A Universally Unique IDentifier (UUID) URN Namespace », RFC 4122, Juillet 2005.
<https://tools.ietf.org/html/rfc4122>

Julia Evans. « Async IO on Linux: select, poll, and epoll », Juin 2017.
<https://jvns.ca/blog/2017/06/03/async-io-on-linux--select--poll--and-epoll/>

Google Protocol Buffers:
Documentation: https://developers.google.com/protocol-buffers/docs/overview
Benchmarks: https://google.github.io/flatbuffers/flatbuffers_benchmarks.html

LibreSSL:
Documentation: http://www.libressl.org