Compte Rendu Projet SMB111

Systèmes et applications répartis pour le cloud

TORTEVOIS Alexandre

RÉVISIONS				
Version	Nature de la modification	Auteur	Vérificateur	Date
01	Édition Originale	ALT	-	13/06/2020

Table des matières

1. Objectifs du projet

L'objectif du projet est de développer un système permettant de gérer des distributeurs de nourriture/boisson à distance situé dans des lieux différents. Pour réaliser le projet nous utiliserons :

- un algorithme d'adressage réparti sur un réseau suivant le schéma de nommage ZigBee,
- des objets distribués via Java RMI,
- de la communication réseau avec les Sockets UDP en Java.

Les déconnexions des distributeurs ne sont pas gérées dans ce projet.

2. Définition des acteurs du système

A. Le Gestionnaire de distributeurs (DistributorManager)

Cette entité correspond au « serveur » : c'est un objet distant accessible via Java RMI.
C'est lui qui fixe les paramètres de nommage du réseau et qui attribue les adresses des nœuds des points d'accès RMI, dont la description est ci-après.
Il répertorie tous les distributeurs connectés au réseau et permet ainsi d'exécuter des requêtes sur ceux-ci : par exemple, il peut interroger tous les distributeurs connectés pour connaitre l'état de leur stock, l'argent contenu dans la caisse, etc.
Il stock les alertes remontées par les distributeurs.

B. L'interface d'administration (AdminInterface)

Il s'agit d'une console qui propose une interface en ligne de commande qui permet d'interagir avec le gestionnaire de distributeur.

C. Distributeur point d'accès RMI (RMI Gateway)

Afin de limiter la communication avec le serveur, une communication directe en Java RMI est implémentée entre le serveur et certains distributeurs, ce sont donc les nœuds principaux du réseau.
Situé à la profondeur 1 du réseau, ils utilisent des Sockets UDP, pour communiquer avec les autres distributeurs du réseau.

D. Distributeur passerelle UDP (UDP Gateway)

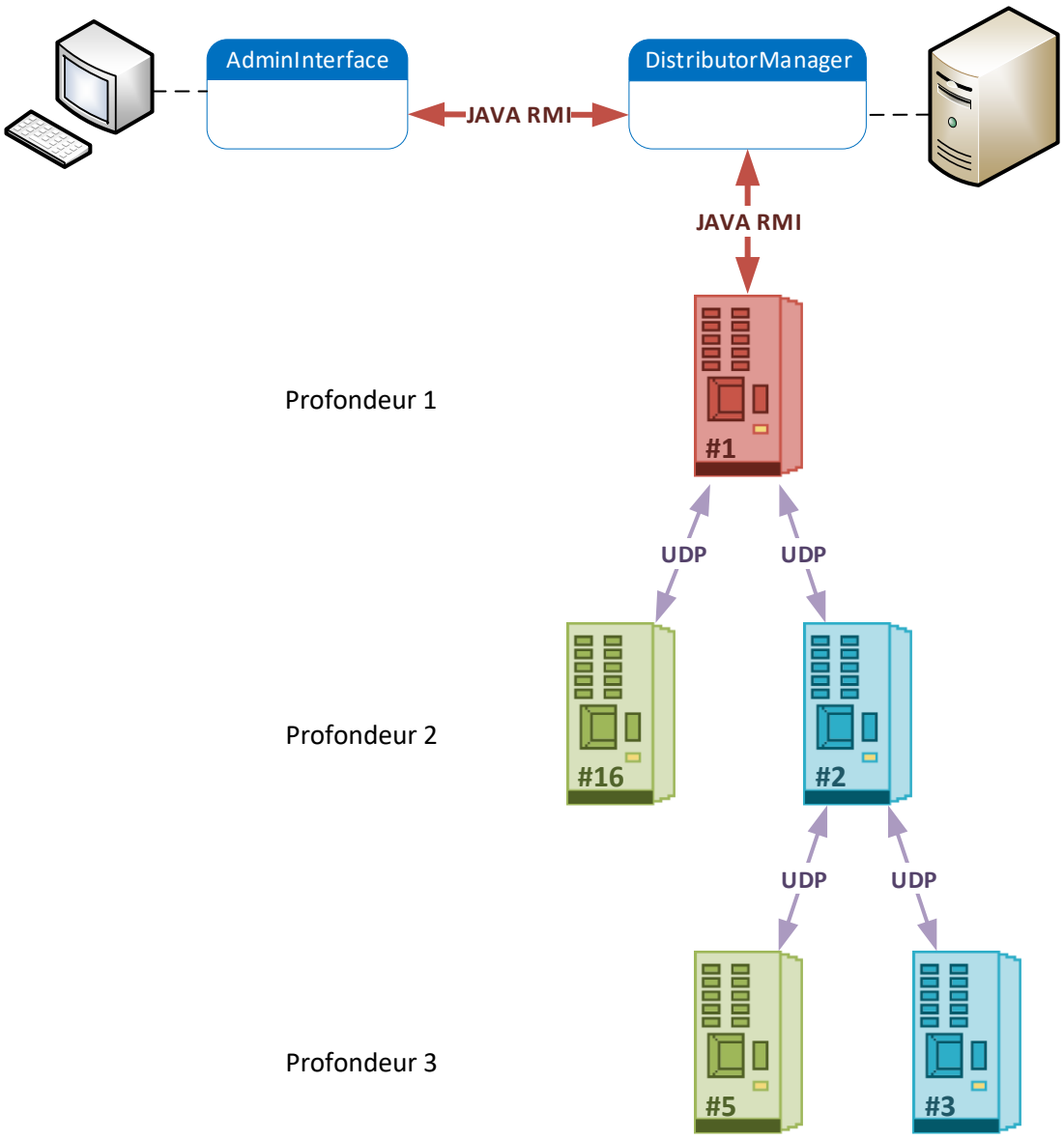
Situé à partir de la profondeur 2 du réseau, ces distributeurs permettre de réaliser l'interconnexion entre les couches du réseau (entre différentes profondeurs). Ces distributeurs peuvent se connecter sur un autre distributeur passerelle qu'il soit de type « RMI » ou « UDP ». Ils permettent la diffusion des messages aux différents équipements qui leur sont connectés.

E. Distributeur

Ce type de nœud n'implémente qu'une partie des fonctions réseaux. Il ne fait que répondre aux requêtes envoyées par les distributeurs passerelles.
Il peut être situé en profondeur 2 ou plus du réseau.
Dans le cadre du projet, il n'est pas prévu qu'un distributeur puisse communiquer directement avec le Gestionnaire de distributeurs, il devra impérativement passer par une passerelle.

F. Schématisation de l'architecture

L'architecture déployée peut-être schématisée comme suit :



Légende :

Distributeur point d'accès RMI (RMI Gateway)

Distributeur passerelle UDP (UDP Gateway)

Distributeur

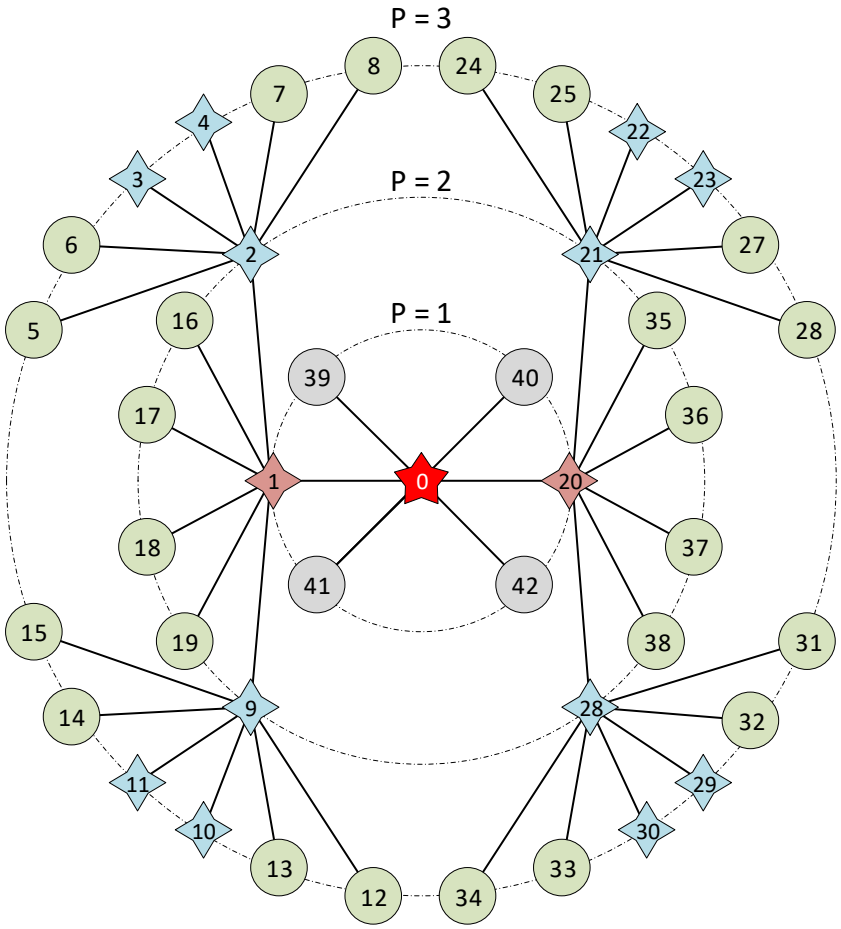
3. Topologie du Réseau

Les adresses sont attribuées suivant le schéma de nommage du protocole ZigBee proposé dans le cours.
Les paramètres de ce type de réseau sont :

- P : la profondeur maximale du réseau.
- R : le nombre maximum de passerelles pouvant se connecter à une autre passerelle parente.
- D : le nombre maximum de distributeurs (end-devices) pouvant se connecter à un autre routeur parent.

Les paramètres qui ont été choisi pour le projet sont : P = 3 / R = 2 / D = 4

L'arbre d'adressage se défini donc comme suit :



Légende :

- Coordinateur / Gestionnaire de distributeurs (DistributorManager)
- Distributeur point d'accès RMI (RMIGateway)
- Distributeur passerelle UDP (UDPGateway)
- Distributeur

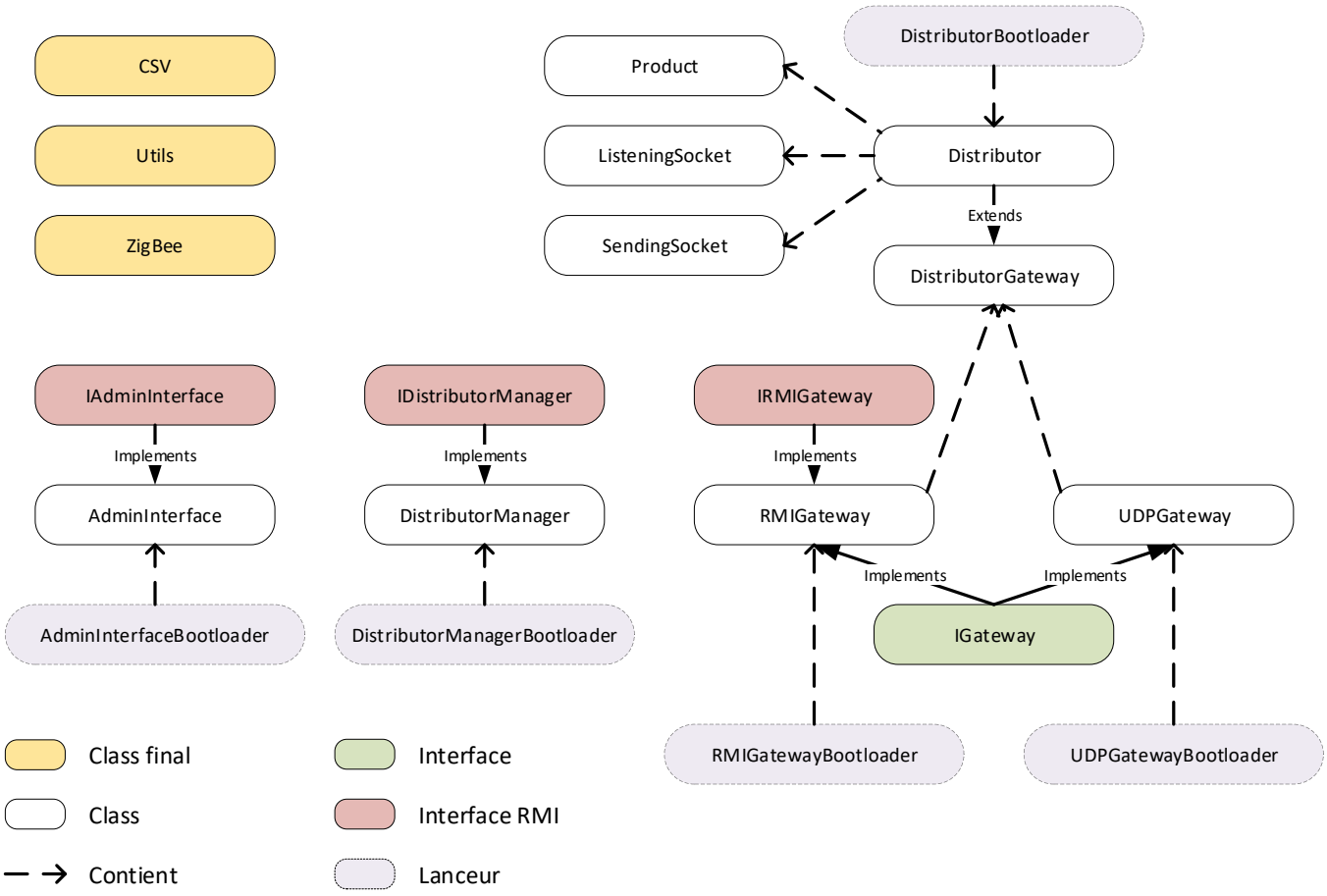
L'algorithme de calcul des adresses sera développé au point 6.

Note : Les adresses 39 à 42 ne seront pas utilisé dans le projet, un distributeur ne pouvant pas se connecter directement au Gestionnaire de distributeurs.

4. La structure du projet

A. Les classes implémentées

Voici une représentation schématique des différentes classes qui ont été implémentée pour la réalisation de ce projet :



B. Les classes finales

Pour éviter une redondance du code, ces classes fournissent un ensemble de méthodes (statiques) qui sont communes et utiliser dans tout le projet.

La classe ***CSV*** fournie une méthode qui retourne les lignes contenues dans le fichier dans une liste. Elle est appelée à l'initialisation du Gestionnaire de distributeur et d'un distributeur pour charger en mémoire la liste des produits.

La classe ***Utils*** fournie un ensemble de fonctions « utilitaires ».

La classe ***ZigBee*** fournie les méthodes permettant l'implémentation du protocole. Elle permet de réaliser différents calculs suivant les paramètres réseaux fournis :

- calcul du nombre total d'adresses,
- calcul du prochain l'intervalle d'adresse suivant une adresse de nœud,
- récupération d'informations à partir d'une adresse de nœud (nœud parent, type, profondeur dans le réseau).

C. Le Gestionnaire de distributeurs (DistributorManager)

La classe ***DistributorManager*** est instancié suivant le Design Patern du singleton. En effet, je suis parti du principe qu'il ne devrait pas être possible de créer plusieurs Gestionnaires de distributeurs sur le même serveur, sinon il faudrait spécifier à quel gestionnaire on veut s'adresser en Java RMI. Pour simplifier le projet, il n'y a donc qu'une seule instance de créée, donc un seul objet *DistributorManager* en mémoire.

Cette classe implémente l'interface ***IDistributorManager*** qui fournit les méthodes accessibles via Java RMI :

- *initDistributorManager* : Initialisation de l'objet *DistributorManager* et de ces variables internes. Cette méthode est appelée juste après la création de l'objet dans le *DistributorManagerBootloader*.
- *getAdminFromRMI* : Cette méthode permet de récupérer l'objet *AdminInterface* créé et disponible via Java RMI. Elle est appelée par l'*AdminInterface* après sa création.
- *getAddressCount* : retourne le nombre totale d'adresses disponibles sur le réseaux, suivant les paramètres qui ont été définis. Le nombre d'adresses est stocké en mémoire, pour ne pas le recalculer à chaque fois qu'on en a besoin. En effet il ne change pas au cours de l'exécution du *DistributorManager*.
- *getConnectedDistributors* : retourne la liste des adresses de tous les distributeurs connectés.
- *getAvailableNodeID* : retourne une adresse de nœud disponible quand un point d'accès RMI demande à se connecter.
- *getNetworkParameters* : retourne les paramètres du réseau au point d'accès RMI.
- *sendMessageToRMI* : envoie un message au point d'accès RMI cible, un filtrage est réalisé suivant l'adresse du destinataire du message.
- *readMessageFromRMI* : réception et traitement des messages transmis par les points d'accès RMI.
- *waitForEndQuery* : cette méthode permet d'attendre la fin d'une requête (ou un timeout) avant de pouvoir saisir une nouvelle requête sur l'*AdminInterface*.
- *freeNodeID* : réinitialise les adresses des distributeurs et point d'accès connectés. Très pratique pour ne pas relancer le serveur et l'*AdminInterface* à chaque test !
- *displayLogsHistory* : affiche l'historique des logs des distributeurs sur l'*AdminInterface*.

La classe ***DistributorManagerBootloader*** créer l'objet *DistributorManager* et le rends disponible sur le Java RMI Registry. Elle prend comme paramètres D, R, P (les paramètres du réseau souhaité tel qu'exposé au point 3).

D. L'interface d'Administration (AdminInterface)

Pour les mêmes raisons que le *DistributorManager,* la classe ***AdminInterface*** est un singleton qui implémente l'interface ***IAdminInterface.*** Elle fournie les méthodes suivantes accessible via Java RMI :

- *printTraceOnAdminInterface* : appelée depuis le *DistributorManager* cette méthode permet de réaliser l'affichage sur la console de l'*AdminInterface.*

- *waitForEndQuery* : transmet la méthode *waitForEndQuery* du *DistributorManager* permettant d'attendre la fin d'une requête avant la saisie d'une nouvelle.

La classe ***AdminInterfaceBootloader*** créer l'objet l'*AdminInterface* et le rends disponible sur le Java RMI Registry.
Elle réalise l'association de l'*AdminInterface* sur le *DistributorManager* par l'appel de la méthode *getAdminFromRMI*. L'interface est ainsi automatiquement liée lors de sa création.

E. Le distributeur (Distributor)

La classe ***Distributor*** permet d'instancier un objet possédant les fonctionnalités minimales attendues pour un distributeur :

- Charger une liste de produits (classe ***Product***) au démarrage du distributeur.
- Proposer un menu de choix de produits.
- Gérer le stock de produits, émettre une alerte en dessous d'un certain seuil de produit, connaitre les fonds disponibles dans le distributeur.
- Création des sockets de communications pour l'écoute et l'envoie des messages, par l'intermédiaire des classe ***ListeningSocket*** et ***SendingSocket***.
  Les sockets restent ouvertes en permanence, la socket d'écoute est placée dans un thread et reste en permanence en attente d'un message.
- Traiter les messages reçus et envoyer une réponse.

Cette classe dispose de deux constructeurs par défaut, suivant que le distributeur est un Distributeur (passerelle ou simple) UDP ou un Distributeur point d'accès RMI.

La classe ***DistributorBootloader*** permet d'instancier et de démarrer un nouveau *Distributor*. Elle prend en paramètres l'adresse et le port de la passerelle sur laquelle on veut connecter le distributeur.
S'ils ne sont pas renseignés, l'adresse par défaut est « localhost » et le port est demandé via l'interface en ligne de commande.

F. Le distributeur « passerelle » (DistributorGateway)

La classe ***DistributorGateway*** étends la classe *Distributor* en implémentant les fonctions supplémentaires permettant à un simple distributeur de devenir un distributeur passerelle.
Elle ajoute les méthodes qui permettent :

- De récupérer les paramètres du réseau.
- De définir les adresses de nœuds disponibles au niveau de la passerelle.
- De donner une adresse de nœud disponible à un nouveau distributeur (simple ou passerelle) qui veut se connecter au réseau.
- De propager les messages (dans les deux sens) sur le réseau, avec un filtrage suivant les adresses des nœuds cibles.

Elle surcharge la méthode de traitement des messages reçus et d'envoie des réponses, puisqu'il y plus de cas à gérer à ce niveau.

G. Le point d'accès RMI (RMIGateway)

La classe ***RMIGateway*** permet de créer un Distributeur point d'accès RMI.
Elle implémente l'interface ***IGateway*** qui spécifie les méthodes communes aux distributeurs point d'accès :

- *setNetworkParameters* : récupère les paramètres réseau auprès du *DistributorManager* via Java RMI et les définis au niveau du *DistributorGateway*.

- *sendMessageToGateway* : envoie un message via Java RMI vers le *DistributorManager*.

Elle implémente également l'interface ***IRMIGateway*** qui fournit une méthode accessible via Java RMI :

- *readMessageFromRMI* : permet la transmission d'un message depuis le *DistriburtorManager* vers les *Distributor* connectés.

La classe ***RMIGatewayBootloader*** permet d'instancier et de démarrer un nouveau *RMIDistributor*. Elle prend en paramètres l'adresse et le port de la passerelle sur laquelle on veut connecter le distributeur.
S'ils ne sont pas renseignés, l'adresse par défaut est « localhost » et le port est demandé via l'interface en ligne de commande.

H. Le distributeur passerelle UPD (UDPGateway)

La classe ***UDPGateway*** permet de créer un Distributeur passerelle UDP. Elle implémente l'interface ***IGateway*** :

- *setNetworkParameters* : construit la requête de demande des paramètres réseaux à la passerelle parente sur le réseau et traite la réponse.

- *sendMessageToGateway* : envoie un message à la passerelle parente sur le réseau via la *SendingSocket*

La classe ***UDPGatewayBootloader*** permet d'instancier et de démarrer un nouveau *UDPDistributor*. Elle prend en paramètres l'adresse et le port de la passerelle sur laquelle on veut connecter le distributeur.
S'ils ne sont pas renseignés, l'adresse par défaut est « localhost » et le port est demandé via l'interface en ligne de commande.

5. Interaction entre les acteurs

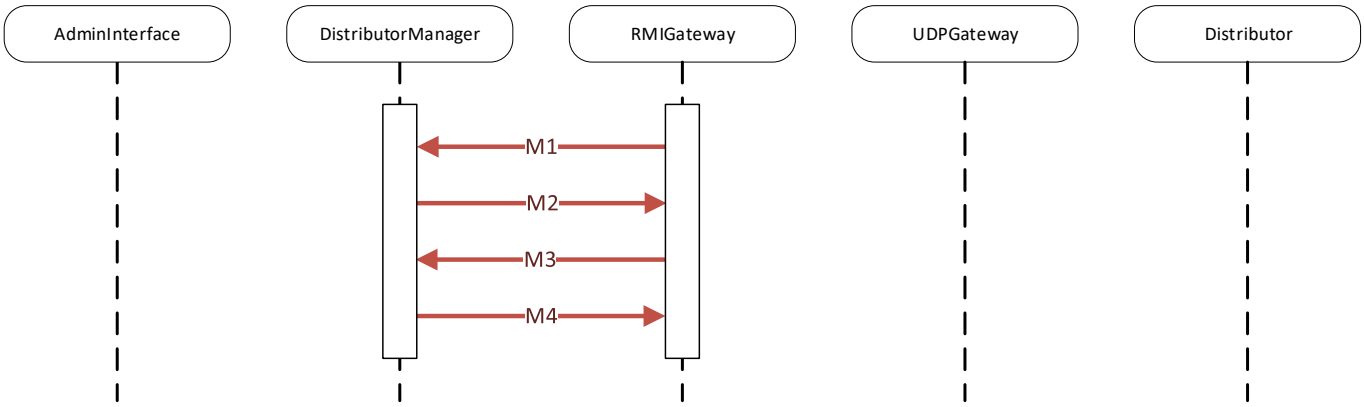
Pour la suite du développement, je me suis basé sur le schéma d'architecture de la page 4.
Il est a noté que pour simplifier, j'ai choisi d'attribuer comme numéro d'identifiant au distributeur le même numéro que son adresse (numéro du nœud dans le réseau). Il aurait été possible d'ajouter une table de correspondance supplémentaire pour faire le lien numéro d'identifiant ⇔ adresse.
De plus, les numéros de port de communication en écoute sont choisis en fonction de l'adresse du distributeur dans le réseau. Le numéro de port de base est 6000 auquel on ajoute l'adresse. Ainsi pour le distributeur n°1 la socket d'écoute est 6001, pour le distributeur n°2 la socket d'écoute est 6002, etc…

Les échanges entre les nœuds en Java RMI sont réalisés grâce aux appels des méthodes disponibles sur les objets distants et au passage en paramètres du nom de la requête et des données liés à ces échanges.
Les échanges entre les nœuds au travers des sockets UDP sont réalisés au travers de messages envoyés/reçus formaté en JSON. J'ai choisi ce formatage car il est communément employé dans le développement d'application IOT (voir Annexe 7.B pour le détail du formatage des messages).

Les mêmes chaines de caractères ont été utilisé pour identifier les requêtes et traiter les réponses en Java RMI et en JSON.

A. Connexion d'un point d'accès RMI au Gestionnaire de Distributeur (exemple nœud 1)

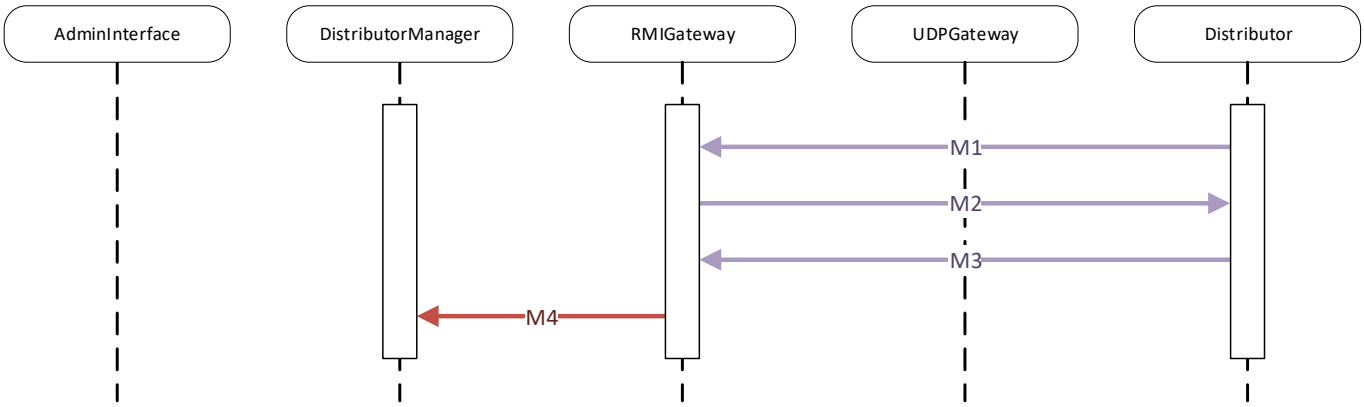
Cet échange se fait exclusivement en utilisant les méthodes disponibles via Java RMI entre les objets *RMIGateway* et *DistributorManager*.
Il peut être schématisé comme suit :



M1 : Requête de connexion : demande un numéro de nœud disponible au Gestionnaire.
M2 : Réponse : envoie d'un numéro de nœud disponible (1), l'adresse 1 est maintenant réservée sur le Gestionnaire et le distributeur n°1 est enregistré dans la liste des distributeurs connectés du Gestionnaire. A la réception du message le point d'accès créer le distributeur n°1 et la *ListeningSocket* sur le port 6001.
M3 : Requête des paramètres du réseau.
M4 : Réponse : envoie du tableau de paramètres du réseau ([4, 2, 3]). A la réception du message le point d'accès initialise les paramètres réseaux et la liste des adresses disponibles.

B. Connexion d'un distributeur à un point d'accès RMI (exemple nœud 16)

Pour réaliser cet échange, il faut avoir préalablement terminé l'échange du point A.
L'échange peut être schématisé comme suit :



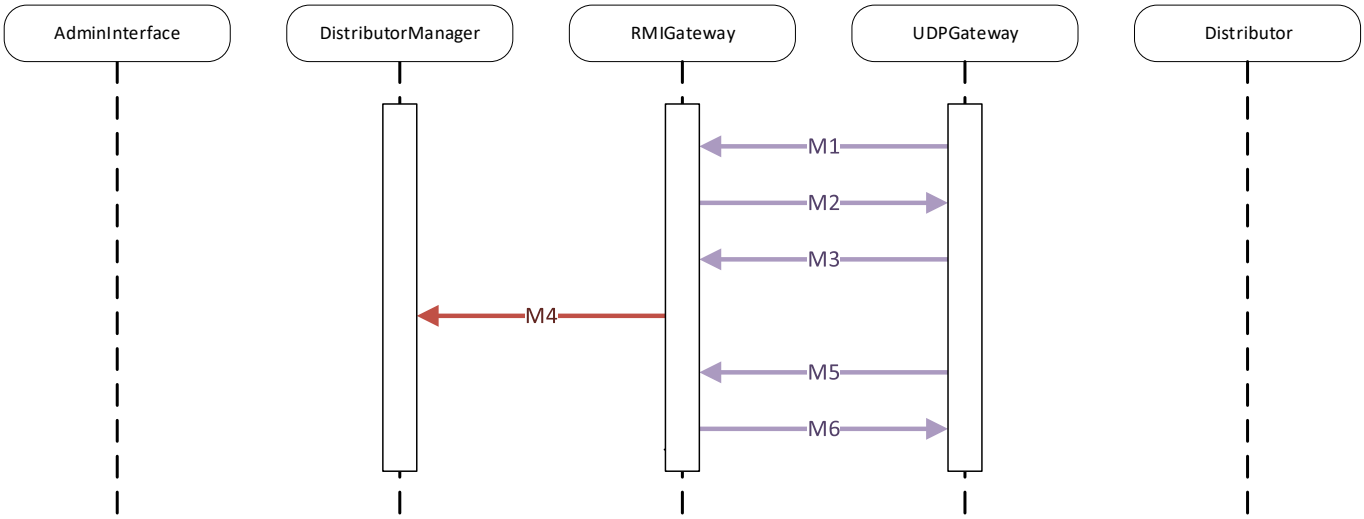
M1 : Requête de connexion : demande un numéro de nœud disponible au point d'accès sur le port 6001. L'adresse, le numéro du port (8080) et le type *Distributor*, sont spécifiés dans le message.
M2 : Réponse : envoie d'un numéro de nœud de type *Distributor* disponible (16) sur le port 8080. L'adresse 16 est maintenant réservée sur le point d'accès. A la réception du message le distributeur prend l'adresse 16.
M3 : Confirmation de la connexion. Création de la *ListeningSocket* sur le port 6016.
M4 : Propagation de la connexion. Le distributeur n°1 est enregistré dans la liste des distributeurs connectés sur le Gestionnaire.

Les échanges M1, M2, M3 sont réalisés au travers des sockets. Les messages sont :
M1 : `{"query":"query_get_node_id","reply_port":8080,"reply_address":"localhost","device_type":3}`
M2 : `{"query":"reply_get_node_id","node_id":16}`
M3 : `{"query":"new_node_connexion","node_id":16}`

L'échange M4 est réalisé en utilisant les méthodes disponibles via Java RMI.

C. Connexion d'un distributeur passerelle UPD à un point d'accès RMI (exemple nœud 2)

Pour réaliser cet échange, il faut avoir préalablement terminé l'échange du point A.
L'échange peut être schématisé comme suit :



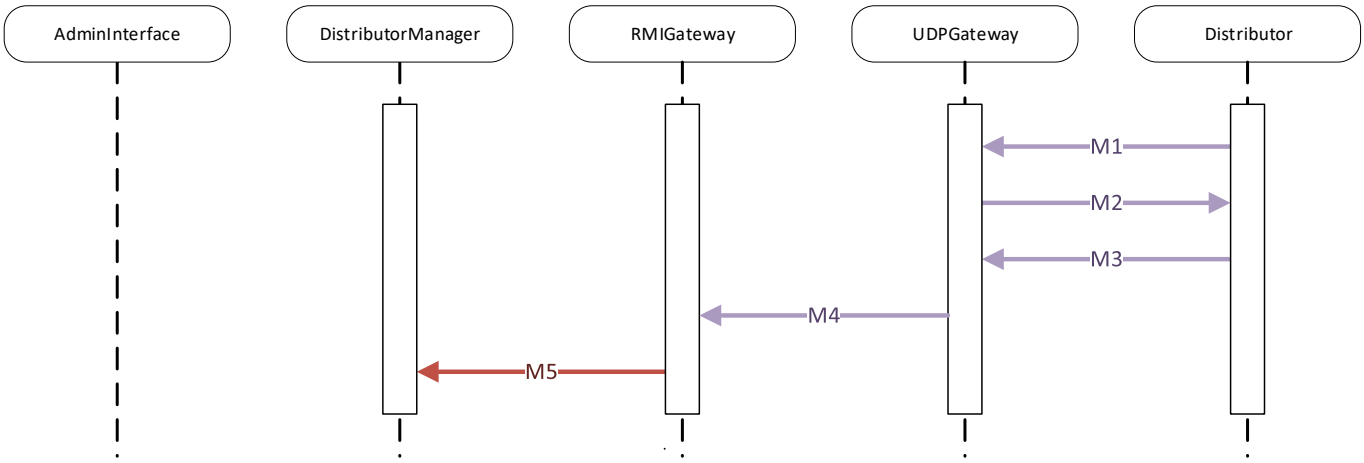
M1 : Requête de connexion : demande un numéro de nœud disponible au point d'accès sur le port 6001. L'adresse, le numéro du port (8080) et le type *DistributorGateway*, sont spécifiés dans le message.
M2 : Réponse : envoie d'un numéro de nœud de type *DistributorGateway* disponible (2) sur le port 8080. L'adresse 2 est maintenant réservée sur le point d'accès. A la réception du message le distributeur prend l'adresse 2.
M3 : Confirmation de la connexion. Création de la *ListeningSocket* sur le port 6002.
M4 : Propagation de la connexion. Le distributeur n°1 est enregistré dans la liste des distributeurs connectés sur le Gestionnaire.
M5 : Requête des paramètres du réseau.
M6 : Réponse : envoie du tableau de paramètres du réseau ([4, 2, 3]). A la réception du message la passerelle initialise les paramètres réseaux et la liste des adresses disponibles. La réponse est envoyée sur le port 6002.

Les échanges M1, M2, M3, M5, M6 sont réalisés au travers des sockets. Les messages sont :
M1 : `{"query":"query_get_node_id","reply_port":8080,"reply_address":"localhost","device_type":2}`
M2 : `{"query":"reply_get_node_id","node_id":2}`
M3 : `{"query":"new_node_connexion","node_id":2}`
M5 : `{"query":"query_network_parameters","node_id":2}`
M6 : `{"query":"reply_network_parameters","gateways":2,"depth":3,"devices":4}`

L'échange M4 est réalisé en utilisant les méthodes disponibles via Java RMI.

D. Connexion d'un distributeur à distributeur passerelle UPD (exemple nœud 5)

Pour réaliser cet échange, il faut avoir préalablement terminé l'échange du point A puis du point C.
L'échange peut être schématisé comme suit :



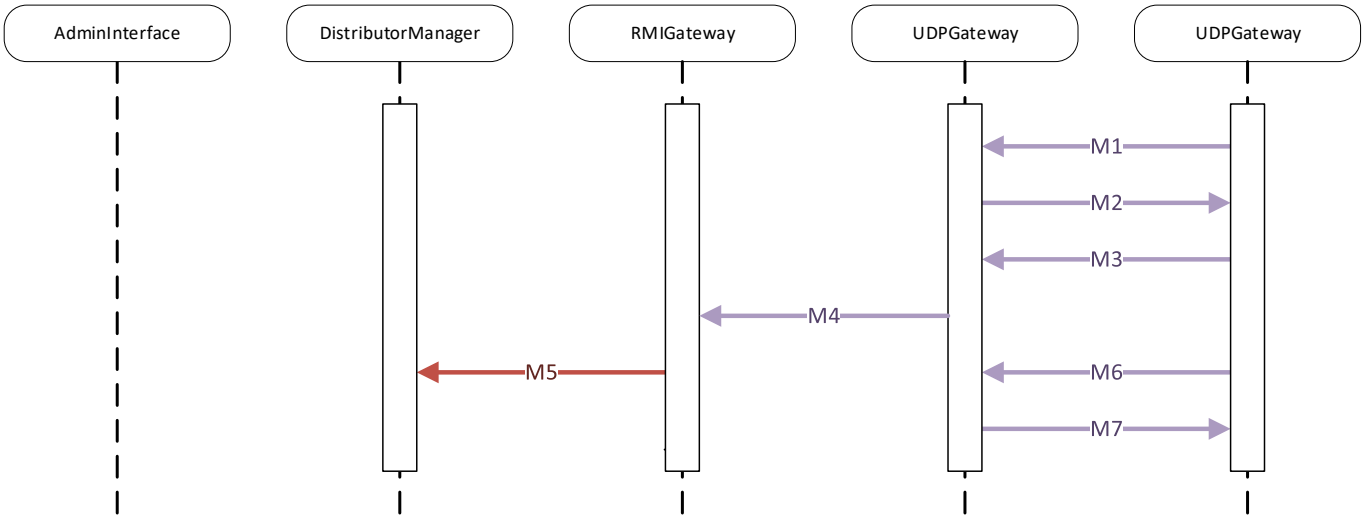
M1 : Requête de connexion : demande un numéro de nœud disponible au distributeur passerelle sur le port 6002. L'adresse, le numéro du port (8080) et le type *Distributor*, sont spécifiés dans le message.
M2 : Réponse : envoie d'un numéro de nœud de type *Distributor* disponible (5) sur le port 8080. L'adresse 5 est maintenant réservée sur le distributeur passerelle. A la réception du message le distributeur prend l'adresse 5.
M3 : Confirmation de la connexion. Création de la *ListeningSocket* sur le port 6005.
M4 : Propagation de la connexion sur le point d'accès.
M5 : Propagation de la connexion. Le distributeur n°5 est enregistré dans la liste des distributeurs connectés sur le Gestionnaire.

Les échanges M1, M2, M3, M4 sont réalisés au travers des sockets. Les messages sont :
M1 : `{"query":"query_get_node_id","reply_port":8080,"reply_address":"localhost","device_type":3}`
M2 : `{"query":"reply_get_node_id","node_id":5}`
M3 : `{"query":"new_node_connexion","node_id":5}`
M4 : `{"query":"new_node_connexion","node_id":5}`

L'échange M5 est réalisé en utilisant les méthodes disponibles via Java RMI.

E. Connexion d'un distributeur passerelle UPD à distributeur passerelle UPD (exemple nœud 3)

Pour réaliser cet échange, il faut avoir préalablement terminé l'échange du point A puis du point C.
L'échange peut être schématisé comme suit :



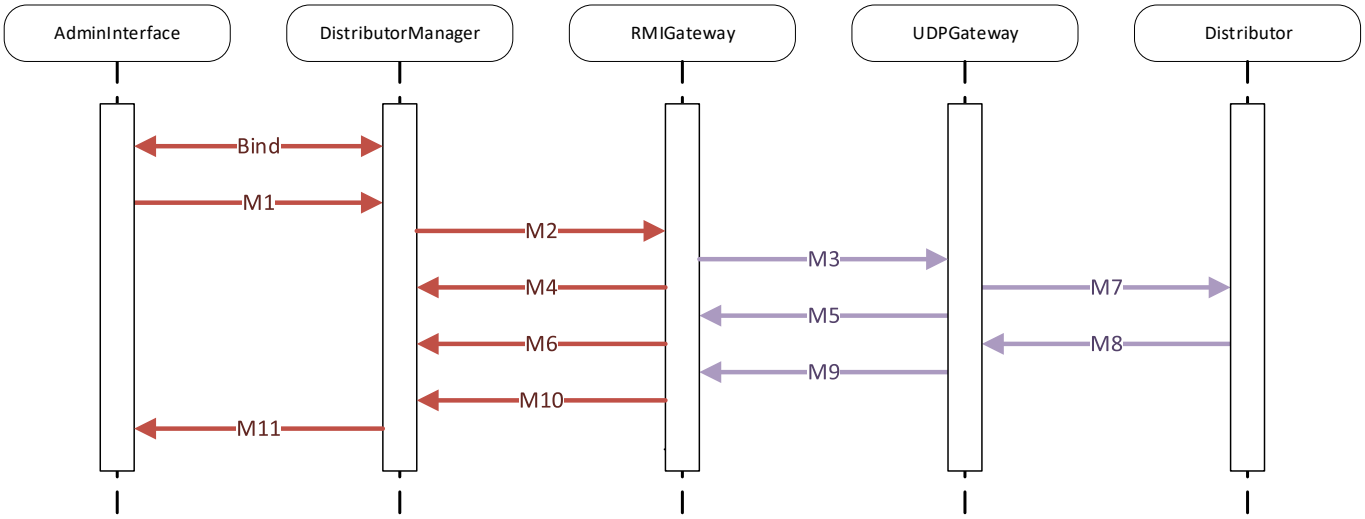
M1 : Requête de connexion : demande un numéro de nœud disponible au distributeur passerelle sur le port 6002. L'adresse, le numéro du port (8080) et le type *DistributorGateway*, sont spécifiés dans le message.
M2 : Réponse : envoie d'un numéro de nœud de type *DistributorGateway* disponible (3) sur le port 8080. L'adresse 3 est maintenant réservée sur le distributeur passerelle parent. A la réception du message le distributeur prend l'adresse 3.
M3 : Confirmation de la connexion. Création de la *ListeningSocket* sur le port 6003.
M4 : Propagation de la connexion sur le point d'accès.
M5 : Propagation de la connexion. Le distributeur n°3 est enregistré dans la liste des distributeurs connectés sur le Gestionnaire.
M6 : Requête des paramètres du réseau.
M7 : Réponse : envoie du tableau de paramètres du réseau ([4, 2, 3]). A la réception du message la passerelle initialise les paramètres réseaux et la liste des adresses disponibles (ici aucune puisqu'on est à la profondeur 3). La réponse est envoyée sur le port 6003.

Les échanges M1, M2, M3, M4, M6, M7 sont réalisés au travers des sockets. Les messages sont :
M1 : `{"query":"query_get_node_id","reply_port":8080,"reply_address":"localhost","device_type":2}`
M2 : `{"query":"reply_get_node_id","node_id":3}`
M3 : `{"query":"new_node_connexion","node_id":3}`
M4 : `{"query":"new_node_connexion","node_id":3}`
M6 : `{"query":"query_network_parameters","node_id":3}`
M7 : `{"query":"reply_network_parameters","gateways":2,"depth":3,"devices":4}`

L'échange M5 est réalisé en utilisant les méthodes disponibles via Java RMI.

F. Envoie d'une requête depuis l'interface d'administration

Pour réaliser cet échange, il faut avoir préalablement terminé les échanges des points A, C et D.
On considère ici un échange entre l'*AdminInterface*, le *Gestionnaire*, le point d'accès 1 et les nœuds 2 et 5.
Cet échange peut être schématisé comme suit :



L'*AdminInterface* peut être exécuter à tout moment, après que le *DistributorManager* a été lancé.
Cette étape correspond au « Bind » sur le schéma.

M1 : Envoie d'une commande. Par exemple : *get_stock all*. Cette commande se traduit par l'interrogation de tous les nœuds connectés, avec la requête *get_stock* et la liste de tous les nœuds.
M2 : Transmission de la requête au point d'accès.
M3 : Transmission de la requête à la passerelle. Le message est envoyé par le point d'accès sur le port 6002 de la passerelle.
M4 : Réponse du distributeur point d'accès (nœud 1) au Gestionnaire.
M5 : Réponse de la passerelle. Envoie du message sur le port 6001 du point d'accès.
M6 : Transmission de la réponse au Gestionnaire.
M7 : Transmission de la requête au distributeur. Le message est envoyé par la passerelle sur le port 6005 du distributeur.
M8 : Réponse du distributeur. Envoie du message sur le port 6002 de la passerelle.
M9 : Transmission de la réponse au point d'accès. Le message est envoyé par la passerelle sur le port 6001 du point d'accès.
M10 : Transmission de la réponse au Gestionnaire.
M11 : Traitement des réponses à la requête et affichage du résultat sur la console de l'*AdminInterface*.

Les échanges M1, M2, M4, M6, M10, M11 sont réalisés en utilisant les méthodes disponibles via Java RMI.

Les échanges M3, M5, M7, M8, M9 sont réalisés au travers des sockets. Les messages sont :
M3 : `{"query":"query_get_stock","querying_nodes_id":[1,2,5]}`
M5 : `{"query":"reply_get_stock","status":200, "distributor_id":2,"distributor_stock":{...}}`
M7 : `{"query":"query_get_stock","querying_nodes_id":[1,2,5]}`
M8 : `{"query":"reply_get_stock","status":200, "distributor_id":5,"distributor_stock":{...}}`
M9 : `{"query":"reply_get_stock","status":200, "distributor_id":5,"distributor_stock":{...}}`

6. Algorithme

A. Obtenir les informations sur un nœud

C'est le premier algorithme que j'ai écrit dans le cadre de ce projet. Il permet de savoir où se situe un nœud donné dans l'arbre d'adressage (nœud parent, profondeur) et son type.

```
Paramètres :
NœudRecherché
D // paramètre du réseau : nombre de end-devices maximum par passerelle parente
R // paramètre du réseau : nombre de passerelles maximum par passerelle parente
P // paramètre du réseau : profondeur

Variables :
nb_adresses  // nombre d'adresses disponible sur le réseau
parent       // adresse du nœud parent
profonteur   // profondeur parcourue
type         // type du nœud
lim_inf      // limite inférieur de la plage d'adresses parcourue
lim_sup      // dernière adresse de passerelle disponible sur la plage d'adresses parcourue
pas          // intervalle entre les adresses des passerelles de la plage
nœud         // nœud trouvé

Initialisation :
nb_adresses <- obtenir_total_adresses() // voir les formules Annexe 7.A
parent <- 0
profonteur <- 0
type <- "racine"

Recherche des informations :
Si NœudRecherché != 0 Alors
       nœud <- 1
       profonteur <- 1
       lim_inf <- 1
       lim_sup <- (nb_adresses - D)
       pas <- (nb_adresses - D - 1 ) / G

       TantQue nœud < NœudRecherché && NœudRecherché < lim_sup && profonteur != P Faire
              Si NœudRecherché < (nœud + pas) Alors
                     parent <- nœud
                     profonteur <- profonteur + 1
                     lim_inf <- nœud + 1
                     lim_sup <- nœud + pas - D
                     pas <- (pas - D - 1) / G
                     nœud <- nœud + 1
              Sinon
                     nœud <- nœud + pas
              FinSi
       FinTantQue

       Si NœudRecherché >= lim_inf && NœudRecherché < lim_sup Alors
              type <- "passerelle"
       FinSi

       Si NœudRecherché >= lim_sup && NœudRecherché < (lim_sup + D) Alors
              type <- "distributeur"
       FinSi
FinSi
```

B. Calculer les adresses d'un nœud passerelle disponibles à attribuer

Pour réaliser ce calcul, j'ai réutilisé l'algorithme précédent. Il suffit alors de recalculer les limites et le pas et la profondeur avec une itération supplémentaire.
Cela revient à changer la condition `nœud < NœudRecherché` par `nœud <= NœudRecherché`
Il suffira alors de contrôler en retour des variables la profondeur. Si celle-ci est supérieur à la profondeur du réseau, nous sommes sur un distributeur passerelle « final » qui n'a aucune plage d'adresses de disponible pour l'attribution.

7. Conclusion

Une vidéo complémentaire montrant l'application en action est disponible sur Youtube :
https://www.youtube.com/watch?v=So7VVoToQmU

Ce projet fut fort intéressant à développer.
La recherche de l'algorithme d'adressage a été un petit casse-tête très sympathique à résoudre.
N'étant pas familier du développement en Java, que je n'ai pu côtoyer qu'au travers des unités NFP121 et SMB116, j'ai également pu acquérir de nouvelles connaissances et compétences au travers de l'utilisation du Java RMI, des sockets en Java et de l'IDE IntelliJ.

8. Annexes

A. Notes de calcul des adresses sur le réseau suivant le protocole ZigBee

On définit :
- P : la profondeur
- R : le nombre de nœud de type FFD (nombres de routeurs maximums connectés par routeurs)
- D : le nombre de nœud de type RFD (nombres de end-devices maximums connectés par routeurs)

Exemple, une application numérique avec : $P = 3$; $R = 2$; $D = 4$; calcul du nombre de nœuds :

	$P = 1$	$P = 2$	$P = 3$	Total
Nombre de « R »	2	$2 \times 2 = 4$ #1	$2 \times 2 \times 2 = 8$ #3	14
Nombre de « D »	4	$2 \times 4 = 8$ #2	$2 \times 2 \times 4 = 16$ #4	28
Nombre de Nœuds	6	12	24	42

#1 : Pour $P = 2$, il y a 2 routeurs et pour chaque routeur il y a 2 routeurs maximum connectés.
#2 : Pour $P = 2$, il y a 2 routeurs et pour chaque routeur il y a 4 end-devices maximum connectés.
#3 : Pour $P = 3$, il y a 4 routeurs et pour chaque routeur il y a 2 routeurs maximum connectés.
#4 : Pour $P = 3$, il y a 4 routeurs et pour chaque routeur il y a 4 end-devices maximum connectés.

On observe une récursivité dans les calculs.
En ajoutant l'adresse « 0 » du coordinateur, le réseau comprendra un total de 43 adresses.

Généralisation des calculs :

	$P = 1$	$P = 2$	$P = 3$	Total
Nombre de « R »	R	$R \times R = R^2$	$R \times R \times R = R^3$	S_R
Nombre de « D »	D	$D \times R$	$D \times R \times R = D \times R^2$	S_D
Nombre de Nœuds	$R + D$	$R \times (R + D)$	$R^2 \times (R + D)$	S_N

On observe un lien entre la profondeur P et l'élévation à la puissance.

Ainsi on en déduit :

	$P = 1$	$P = 2$	$P = 3$	Total
Nombre de « R »	R^P	R^P	R^P	S_R
Nombre de « D »	$D \times R^{P-1}$	$D \times R^{P-1}$	$D \times R^{P-1}$	S_D
Nombre de Nœuds	$(R + D) \times R^{P-1}$	$(R + D) \times R^{P-1}$	$(R + D) \times R^{P-1}$	S_N

Les formules généralisées sont donc les suivantes :

$$S_R = \sum_{i=1}^{i=P} R^P$$

$$S_D = \sum_{i=1}^{i=P} D \times R^{P-1}$$

$$S_N = S_R + S_D = \sum_{i=1}^{i=P} (R + D) \times R^{P-1}$$

Remarque : on observe que S_N est une suite géométrique : c'est la somme des P premiers termes d'une suite de premier terme $(R + D)$ de raison R.

- Calcul du nombre total de nœuds « R » :

$$S_R = \sum_{i=1}^{i=P} R^P = R^1 + R^2 + \cdots + R^P$$

Si on multiplie par« R » :

$$RS_R = R^2 + R^3 + \cdots + R^P + R^{P+1}$$

En faisant ensuite la différence $RS_R - S_R$:

$$RS_R - S_R = R^2 + R^3 + \cdots + R^P + R^{P+1} - (R^1 + R^2 \ldots + R^P)$$

$$RS_R - S_R = R^{P+1} - R^1$$

$$S_R(R-1) = R^{P+1} - R$$

On obtient donc :

$$S_R = \frac{(R^{P+1} - R)}{R-1}$$

(Ce qui implique que $R \geq 2$)

- Calcul du nombre total de nœuds « D » :

$$S_D = \sum_{i=1}^{i=P} D \times R^{P-1} = D \times R^0 + D \times R^1 + \cdots + D \times R^{P-1}$$

$$S_D = D \times (1 + R^1 + \cdots + R^{P-1})$$

Si on multiplie par « R » :

$$RS_D = D \times (R + R^2 + \cdots + R^{P-1} + R^P)$$

En faisant ensuite la différence $RS_R - S_R$:

$$RS_D - S_D = D \times (R + R^2 + \cdots + R^{P-1} + R^P) - D \times (1 + R^1 + \cdots + R^{P-1})$$

$$RS_D - S_D = D \times (R^P - 1)$$

$$S_D(R-1) = D \times (R^P - 1)$$

On obtient donc :

$$S_D = \frac{D \times (R^P - 1)}{R - 1}$$

(Ce qui implique que $R \geq 2$)

- Calcul du nombre total d'adresses du réseau :

$$S_A = S_R + S_D + 1$$

$$S_A = \frac{(R^{P+1} - R)}{R-1} + \frac{D \times (R^P - 1)}{R-1} + 1$$

$$S_A = \frac{R^{P+1} - R + D \times R^P - D}{R-1} + 1$$

$$S_A = \frac{R \times R^P - R + D \times R^P - D}{R-1} + 1$$

$$S_A = \frac{R^P(R+D) - R - D}{R-1} + 1$$

$$S_A = \frac{R^P(R+D) - (R+D)}{R-1} + 1$$

$$S_A = \frac{(R^P - 1) \times (R+D)}{R-1} + 1$$

(Ce qui implique que $R \geq 2$)
Remarque : on retrouve ici la formule attendue de la suite géométrique S_N à laquelle on ajoute l'adresse « 0 » du coordinateur du réseau.

Calcul du nombre d'adresse si $R = 1$:

- Calcul du nombre total de nœuds « R » :

$$S_R = R \times P$$

- Calcul du nombre total de nœuds « D » :

$$S_D = D \times P$$

- Calcul du nombre total d'adresses sur le réseau si $R = 1$:

$$S_A = S_R + S_D + 1$$

$$S_A = R \times P + D \times P + 1 = P \times (R + D) + 1$$

Ressource : monclasseurdemaths.fr : calcul de la somme des n+1 premières puissances de q

B. Description de la composition des messages JSON

Chaque message échangé est composé à minima d'un champ (clé) « query » qui permet d'orienter celui-ci.
Les valeurs possibles sont décrites ci-après, ainsi que les autres champs qui lui sont associés.
Elles sont classées en deux catégories : les requêtes et les réponses.

1) Les requêtes (Query)

- *query_get_node_id*

Demande d'un numéro de nœud à la passerelle parente.

Champs associés :
reply_address : adresse IP sur laquelle répondre
reply_port : numéro du port sur lequel répondre
device_type : Type du nœud réalisant la demande (passerelle ou simple distributeur)

- *query_network_parameters*

Demande les paramètres réseau à la passerelle parente.

Champs associés :
node_id : numéro du nœud réalisant la requête, pour savoir à qui répondre.

- *query_get_stock*

Demande le contenu du stock d'un distributeur.

Champs associés :
querying_nodes_id : un tableau contenant les numéros des distributeurs interrogés.

- *query_get_money*

Demande l'argent contenu dans la caisse d'un distributeur.

Champs associés :
querying_nodes_id : un tableau contenant les numéros des distributeurs interrogés.

2) Les réponses (Reply)

- *reply_get_node_id*

Réponse à la demande *query_get_node_id*.

Champs associés :
node_id : numéro de nœud disponible.

- *new_node_connexion*

Propagation d'une nouvelle connexion jusqu'au Gestionnaire, à la suite de la réception de la réponse *reply_get_node_id.*

Champs associés :
node_id : numéro du nœud connecté.

- *reply_network_parameters*

Réponse à la demande *query_network_parameters*.

Champs associés :
devices : le nombre maximum de distributeurs (end-devices) pouvant se connecter à un autre routeur parent.
gateways : le nombre maximum de passerelles pouvant se connecter à une autre passerelle parente.
depth : la profondeur maximale du réseau.

- *reply_get_stock*

Réponse à la demande *query_get_stock*.

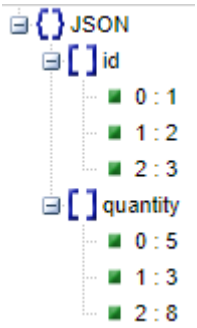
Champs associés :
status : 200 si le distributeur est bien connecté sinon 400.
distributor_id : l'identifiant du distributeur qui réponds.
distributor_stock : un objet contenant tout les produits en stock.

Le stock est construit comme suit :
- Champs *id* : un tableau de tous les ID des produits du stock
- Champs *quantity* : la quantité de tous les produits en stock

L'association est réalisée ensuite entre les clés de chaque tableau.

Exemple :

Soit le stock suivant : `{"id": [1,2,3], "quantity": [5,3,8]}`
Le produit avec l'ID n°1 (clé 0) est associé à la quantité 5 (clé 0)
Le produit avec l'ID n°2 (clé 1) est associé à la quantité 3 (clé 1)
Le produit avec l'ID n°3 (clé 2) est associé à la quantité 8 (clé 2)



Ci-contre une représentation plus graphique du stock formaté en JSON obtenue sur le site http://jsonviewer.stack.hu/

- *reply_get_money*

Réponse à la demande *query_get_money*.

Champs associés :
status : 200 si le distributeur est bien connecté sinon 400.
distributor_id : l'identifiant du distributeur qui réponds.
distributor_money : l'argent contenu dans la caisse