UFR 919 Sciences de l'ingénieur
Master Informatique – Spécialité STL
UE CPS – Composants

Réseau de terrain spontanés éphémères en BCM4Java

Jacques Malenfant
professeur des universités

version 1.1 du 3/02/2021

Résumé

L'objectif de ce projet est de comprendre l'utilisation des concepts, des approches et des techniques du génie logiciel des architectures à base de composants concurrents et répartis. Cet objectif sera poursuivi par le développement d'une implantation en partie simulée d'un *réseau spontané éphémère*. Le projet se développe en plusieurs étapes permettant d'aborder successivement l'implantation d'un système minimal fonctionnel, l'intégration de mécanismes assurant une meilleure réutilisabilité logicielle, puis l'utilisation du parallélisme et de la synchronisation, la répartition sur plusieurs processus (voire sur plusieurs ordinateurs en réseau) et enfin la performance pour affiner ses paramètres de configuration au déploiement. Ces étapes se veulent représentatives d'un projet de développement de bout-en-bout.

1 Généralités

Les services d'urgence comme les militaires déploient régulièrement sur le terrain des *réseaux de terrain spontanés éphémères* qui vont fournir un service de communication numérique sur une zone géographique donnée pendant une durée limitée. Pour les militaires, il s'agit d'équiper les unités déployées sur un espace géographique donné d'une capacité de communication robuste et sûre. Pour les services d'urgence, un tel réseau permet la communication entre les personnels et les unités pendant une intervention mais aussi suppléer pour les sinistrés et pendant un temps limité la destruction d'équipements de communication classiques suite à une catastrophe naturelle, par exemple. D'autres exemples de réseaux de terrain sont les réseaux de capteurs qui peuvent être déployés à partir d'aéronefs en les lançant au sol, par exemple, voire les aéronefs eux-mêmes qui peuvent fournir collectivement un réseau spontané éphémère sur une zone qu'ils survolent.

L'objectif du projet est d'implanter une version simplifiée d'un réseau éphémère capable de s'auto-configurer en utilisant une approche pair-à-pair. Selon cette approche, les participants au réseau éphémère sont à la fois des nœuds de communication capables de router des messages et des entités de traitement. La multiplicité des nœuds dans un tel réseau permet une plus grande robustesse vis-à-vis des pannes. Elle permet aussi de tenir compte d'éléments fonctionnant de manière intermittente, par exemple lorsqu'ils n'ont plus assez d'énergie et qu'ils doivent être rechargés (avec des panneaux solaires, par exemple) avant de pouvoir se connecter à nouveau.

La figure 1 illustre le déploiement d'un tel réseau. Les nœuds ou éléments du réseau éphémère sont des appareils numériques (téléphones mobiles, tablettes numériques, ordinateurs, etc.) disposant d'une capacité de communication sans fil. Il y a trois types de nœuds dans le réseau éphémère :

1. Les nœuds terminaux se connectent au réseau éphémère pour émettre des messages mais n'ont pas une capacité suffisante (ou se déconnecte trop souvent) pour servir à router les messages.
2. Les nœuds de routage sont comme des nœuds terminaux mais ils sont capables de router les messages dans le réseau éphémère.
3. Les points d'accès sont des entités plus permanentes car ayant accès à une source d'énergie durable. Ces points d'accès sont capables à la fois de communiquer avec les nœuds du réseau éphémère mais aussi avec le réseau classique pour atteindre des éléments qui y sont connectés. Les points d'accès sont interconnectés entre eux via le réseau classique, ce qu'ils utilisent pour router des messages entre eux pour atteindre potentiellement des parties du réseau éphémère inaccessibles autrement.

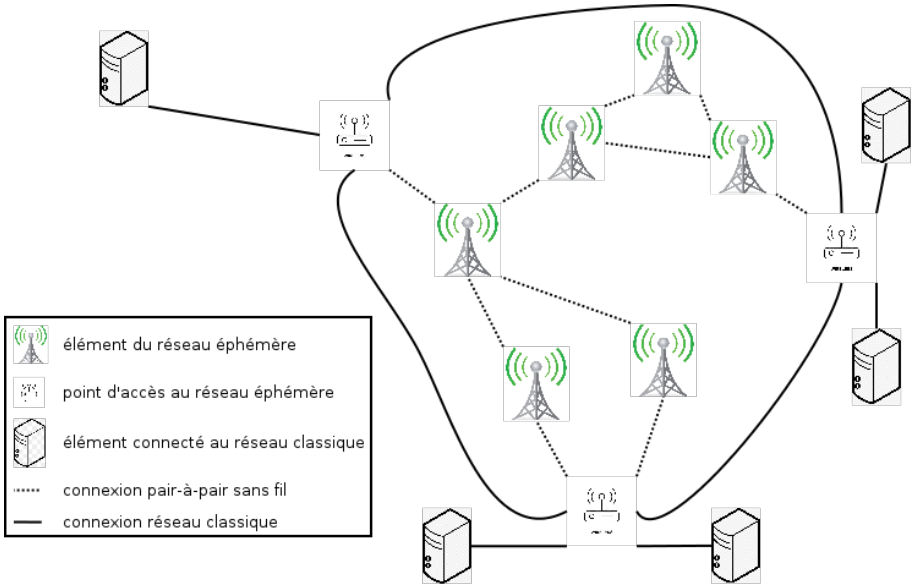


FIGURE 1 – Déploiement d'un réseau spontané éphémère.

Le réseau éphémère utilise un protocole très simple, beaucoup moins complet que le protocole IP. Les éléments du réseau éphémère et ses points d'accès peuvent découvrir leurs homologues dans un rayon de communication efficace donné (spécifique à chaque appareil et pouvant diminuer en fonction de la réserve d'énergie), se connecter à certains d'entre eux puis échanger des messages avec les homologues auxquels ils sont alors directement connectés que l'on nomme *voisins*. Ainsi, pour envoyer un message à un élément du réseau éphémère n'étant pas leur voisin, ils doivent passer par une séquence d'autres éléments qui pourront retransmettre le message à l'un de leur propre voisin et ainsi atteindre le destinataire. On va supposer que le nombre d'éléments sur le terrain et leur position relative aux uns et aux autres par rapport au rayon de communication sont suffisamment denses pour obtenir un graphe connexe, passant potentiellement par les points d'accès.

1.1 Interconnexion des pairs

Dans un réseau réel sur le terrain, les nœuds peuvent se découvrir à l'aide de la détection de présence sur la fréquence des ondes radio utilisées. Les protocoles de liaison sans fil comme le WiFi ou le Bluetooth permettent de découvrir les autres appareils dans leur portée. Le réseau spontané gère les connexions entre voisins en se servant de cette capacité. Lorsqu'on nœud veut se joindre au réseau, il doit détecter les autres nœuds dans sa portée puis leur demander un après l'autre de se connecter à lui comme voisins.

Dans le cadre du projet, nous allons bien sûr devoir simuler cette fonctionnalité. Le nœuds vont se voir attribuer à leur création une adresse unique (équivalente à une addresse IP), une position initiale (coordonnées x et y en mètres dans un plan) et une portée d'émission initiale en mètres. Pour joindre le réseau, chaque nœud va s'enregistrer auprès d'un composant gestionnaire de réseau chargé de simuler la détection par protocole radio. La figure 2 donne les interfaces à utiliser dans le projet, dont l'interface de composant `RegistrationCI` (interface à la fois offerte par le composant gestionnaire et requise par les composants nœuds).

Pour s'enregistrer, un nœud terminal ou de routage appelle le gestionnaire par la méthode `registerTerminalNode` en lui fournissant son adresse, l'URI de son port entrant de communication (voir ci-après), sa position et sa portée. Le gestionnaire enregistre le nouveau nœud dans ses tables puis il calcule la liste des nœuds capables de router des messages déjà enregistrés qui se situent dans la portée du nouveau nœud et lui retourne cette liste. Un nœud capable de router des messages s'enregistre avec la méthode `registerRoutingNode` similaire à celle des nœuds terminaux sauf qu'il ajoute un paramètre pour donner l'URI de son port entrant offrant l'interface `RoutingCI` (voir ci-après). Un point d'accès s'enregistre également mais en appelant la méthode `registerAccessPoint`, similaire à `registerRoutingNode`. Notez qu'un point d'accès est toujours capable de router les messages et il sera connecté à tous les autres points d'accès qui se comporteront comme ses voisins (peu importe leur distance puisque cette connexion peut passer par le réseau classique).

Pour chaque nœud ainsi fourni aux nouvel enregistré, la liste contient l'adresse et l'URI de son port de

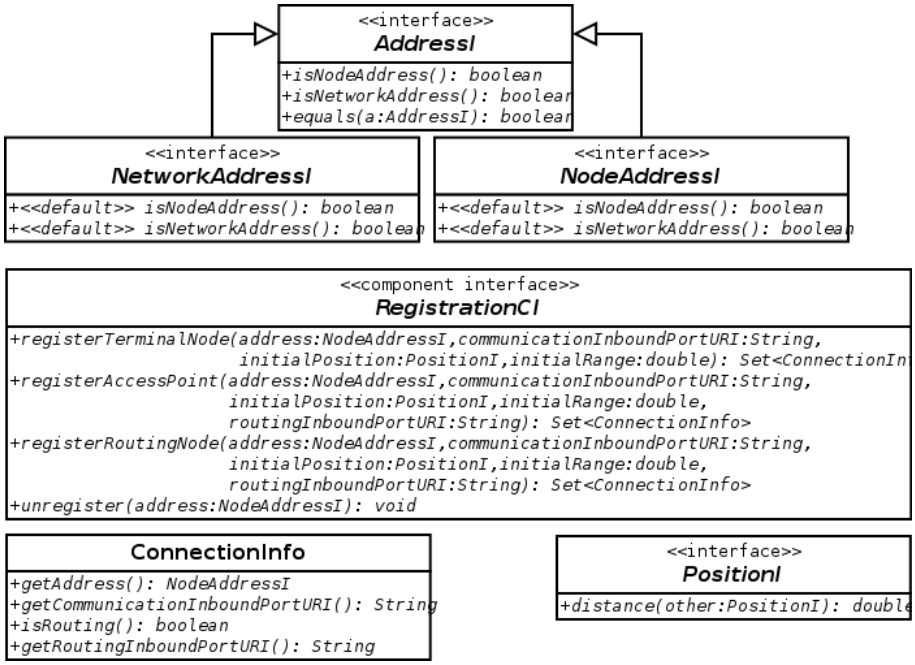


FIGURE 2 – Interfaces et classes impliquées dans l'interconnexion du réseau. Les adresses de type `NodeAddressI` identifient les nœuds du réseau éphémère alors que celles du type `NetworkAddressI` identifient les nœuds du réseau classique. Les nœuds du réseau éphémère ont une position de type `PositionI` avec une mesure de distance calculable à l'aide de la méthode `distance`.

communication entrant. Le nouvel enregistré peut alors se connecter avec chacun des nœuds fournis qui vont devenir ses voisins et qui se connecteront à lui en retour. La figure 3 présente les interfaces concernées. Pour se connecter, les nœuds requièrent et offrent l'interface `CommunicationCI`. Un nœud terminal se connecte à ses voisins en appelant la méthode `connect` en fournissant son adresse et l'URI de son port de communication entrant (celui par lequel il offre l'interface `CommunicationCI`). Le nœud appelé note alors l'appelant comme son nouveau voisin et utilise l'URI de son port de communication entrant de ce dernier pour se connecter à lui en retour (par simple connexion de ports dans ce cas, puisque le nouveau nœud connaît déjà les informations concernant son nouveau voisin).

Un nœud qui a la capacité à router des messages se connecte en appelant la méthode `connectRouting`, similaire à `connect` mais ajoutant l'URI de son port entrant de routage (par lequel il offre l'interface `RoutingCI`, voir ci-après).

Notons que les nœuds peuvent se désenregister du gestionnaire de réseau en appelant la méthode `unregister` de l'interface `RegisterCI`. Nous nous servirons de cette fonctionnalité pour simuler les déconnexions du réseau suite à des pannes ou lorsque les nœuds doivent s'éteindre par manque d'énergie (voir plus loin).

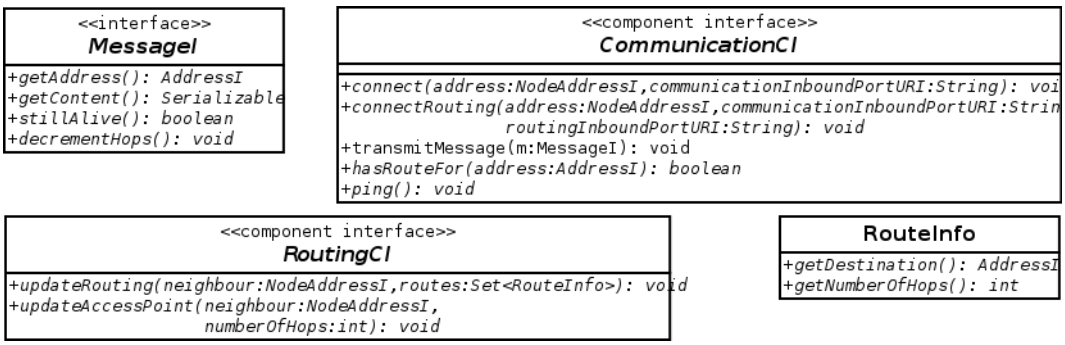


FIGURE 3 – Interfaces et classes impliquées dans l'échange des messages et le routage.

node terminal
node routeur
point d'accès
tout node
message

1.2 Messages et routage

Le routage des messages est la procédure par laquelle chaque nœud émettant lui-même un message ou relayant un message reçu d'un de ses voisins va décider vers quel autre de ses voisins ce message doit être relayé pour rejoindre son destinataire final. Les messages échangés entre les nœuds identifient leur destinataire par leur adresse et le routage va se faire grâce à des tables maintenues dans chaque nœud.

Ainsi, pour envoyer un message, il faut nécessairement connaître l'adresse du nœud auquel il est destiné. L'émetteur crée une instance de message implantant l'interface `MessageI`. Un message possède donc une adresse de destinataire et un contenu (un objet sérialisable). Puisqu'il ne possède pas d'information pour router, un nœud terminal souhaitant émettre un message interroge d'abord tous ses voisins grâce à la méthode `hasRouteFor` de l'interface `CommunicationCI`, auquel chacun va répondre par un nombre de sauts estimés entre nœuds du réseau qui seront nécessaires pour atteindre le destinataire si le message transite par lui. L'émetteur choisit alors le voisin qui lui propose le plus petit nombre de sauts et lui transmet le message en appelant la méthode `transmitMessage`.

Il peut cependant arriver qu'aucun des nœuds voisins ne sachent vers où router un message pour un destinataire donné. Dans ce cas, il répond à `hasRouteFor` par un entier négatif. Si tous les voisins de l'émetteur lui retourne un entier négatif, l'émetteur va tenter un acheminement par inondation. Il choisit M voisins parmi ses N voisins courants et il envoie le message à chacun d'entre eux dans l'espoir que cela mène au destinataire. M sera d'abord fixé arbitrairement, puis dans la dernière partie du projet, il faudra étudier son impact sur la performance du réseau pour le fixer de manière moins arbitraire.

Cette procédure d'inondation pose une sévère difficulté : des copies d'un message risquent de tourner indéfiniment dans le réseau. Pour éviter cela, chaque message est créé avec un nombre de sauts allloués qui est borné et attribué à sa création. À chaque fois qu'un message est relayé d'un nœud à un autre, son nombre de sauts restant est décrémenté en appelant la méthode `decrementHops` de l'interface `MessageI`. La méthode `stillAlive` permet de vérifier si le message a atteint sa limite de sauts autorisés, et alors un nœud recevant ce message ne doit plus le relayer. S'il n'en est pas destinataire, il le détruit, tout simplement.

Le nœuds ayant une capacité de routage possède des tables de routage qui associé à l'adresse de destinataires possibles le voisin à qui relayer le message. Ces nœuds possèdent aussi une route vers un point d'accès vers lequel router les messages destinés aux nœuds sur le réseau classique. Pour cela, il note l'adresse de son voisin qui lui offre un chemin vers un point d'accès dans un minimum de sauts. Lorsqu'il reçoit ou émet lui-même un message ayant une adresse de type `NetworkAddressI` et qu'il connaît un voisin proposant une route vers un point d'accès, il envoie le message à ce voisin. S'il n'a pas encore d'information pour atteindre un point d'accès, il utilise la même procédure que le nœud terminal pour inonder ses voisins ayant une capacité de routage avec ce message.

La table de routage d'un nœud contient des entrées qui associent l'addresse d'un destinataire dans le réseau éphémère (type `NodeAddressI`) au voisin vers lequel les messages visant ce destinataire doivent être relayés ainsi que le nombre de sauts nécessaires pour arriver à sa destination en passant par ce voisin. Lors de la connexion d'un nouveau nœud, le nœud acceptant cette connexion entre dans sa table de routage une entrée pour ce voisin. Si le voisin est lui-même capable de router, il lui envoie toutes les informations de sa table de routage pour peupler la table de routage du nouveau nœud. L'interface `RoutingCI` prévoit les méthodes nécessaires. La méthode `updateRouting` transmet une addresse et un nombre de sauts nécessaire pour l'atteindre vers un voisin qui met à jour sa table de routage si cette route est plus intéressante que la précédente qu'il avait pour ce destinataire (ou s'il n'en avait pas, bien sûr). La méthode `updateAccessPoint` joue un rôle similaire pour mettre à jour la route vers le point d'accès le plus proche : le nœud candidat fournit son adresse et le nombre de sauts qu'il offre jusqu'à un point d'accès. Le nœud receveur met à jour sa route si cette proposition de route est meilleure que celle qu'il a préalablement enregistrée.

Quand un nœud de routage reçoit un message, il y a quatre possibilités :

1. Il est le destinataire du message, auquel cas il le traite tout simplement.
2. Il s'agit d'un message destiné à un nœud en dehors du réseau éphémère (adresse de type `NetworkAddressI`), alors s'il connaît une route vers un point d'accès, il relaie le message vers le voisin qui lui a proposé cette route.
3. Il s'agit d'un message destiné à un nœud du réseau éphémère (adresse de type `NodeAddressI`), alors s'il possède une entrée dans sa table de routage lui disant auquel de ses voisins il doit relayer le message, il relaie le message à ce voisin.
4. Il ne possède pas d'entrée pour ce destinataire dans sa table de routage ou vers un point d'accès, auquel cas il va tenter un routage par inondation comme décrit précédemment.

Un point d'accès se comporte comme un nœud de routage sauf pour les messages destinés à un nœud en dehors du réseau éphémère. Dans la réalité, le message serait alors réexpédier via le réseau classique. Dans le cadre du projet, les nœuds sur le réseau classique sont connectés directement à un point d'accès via l'interface `CommunicationCI` et une table de routage distincte permet au point d'accès de leur envoyer directement leurs messages.

Propagation dynamique des informations des tables de routage

Pour obtenir une bonne performance, les nœuds de routage du réseau éphémère vont devoir se propager les informations de leurs tables de routage au fur et à mesure qu'ils acquièrent de nouvelles informations. Ainsi, à chaque fois qu'un nœud modifie sa table de routage ou sa route vers un point d'accès, il propage à ses voisins la nouvelle information pour qu'ils mettent eux-mêmes à jour leur propres tables (en prenant soin en propageant d'augmenter de 1 le nombre de sauts). En plus, à intervalle régulier, les nœuds vont échanger leurs informations des tables de routage. Par une sorte de percolation, les informations vont se propager progressivement vers des nœuds en plus en plus éloignés et aboutir à des tables complètes dans tous les nœuds au bout d'un certain nombre d'itérations de cette procédure de propagation.

1.3 Connexions et déconnexions dynamiques

Dans un réseau spontané éphémère, les connexions et déconnexions dynamiques sont courantes. Elles peuvent résulter d'un manque d'énergie du nœud qui doit alors s'éteindre ou, s'il est mobile, de son éloignement par rapport à ses voisins au-delà de leurs portées ou encore d'une panne. Il est donc important de savoir les gérer.

Pour la connexion, l'enregistrement et la connexion aux voisins déjà présentés peuvent déjà se faire dynamiquement. Pour la déconnexion, on va supposer ici que les nœuds ne savent pas prédire leur propre déconnexion. Chaque nœud devra donc régulièrement vérifier si ses voisins répondent toujours à ses messages et, si ce n'est pas le cas, toutes les entrées dans les tables de routage qui indiquent ce voisin comme relais seront éliminées. Pour vérifier qu'un voisin répond toujours, l'interface `CommunicationCI` propose la méthode `ping` qui retourne normalement si le voisin est encore présent mais lance l'exception `java.rmi.ConnectException` sinon (cette exception est lancée en RMI lorsqu'un processus ne répond plus aux appels rmi ; c'est pour cette raison que nous allons l'utiliser pour simuler la déconnexion dans ce projet).

Lorsqu'un nœud veut simuler sa déconnexion du réseau éphémère, il se désenregistre du gestionnaire du réseau, se déconnecte de ses voisins et provoque le lancement de l'exception `ConnectException` lors des prochains appels à la méthode `ping` par ses voisins qui vont alors se déconnecter de lui. La reconnexion potentielle des nœuds préalablement déconnectés est plus complexe à gérer (comme dans la réalité d'ailleurs). Ce sera considéré comme un bonus pour les étudiants qui décideront de le faire.

En théorie, pour réparer le routage après la déconnexion d'un nœud, il suffirait que les voisins ayant perdu des routes passent en mode inondation pour les messages concernant tous les destinataires dont les messages devaient passer par le voisin déconnecté, et ce en attendant de rétablir une route via un autre de leurs propres voisins. Mais alors, le routage serait loin d'être optimal en attendant cette mise à jour. Pour éviter cela, les tables de routage sont modifiées pour inclure non pas une seule entrée pour chaque destinataire mais autant d'entrées que le nœud a pu recevoir de ses voisins triées en ordre croissant de nombre de sauts. Ainsi, l'élimination d'une entrée pour une destination qui aurait le nombre minimal de sauts cèdera simplement la première place à l'éventuelle entrée suivante.

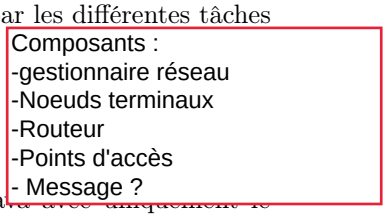
2 Étapes de réalisation du projet et évaluations

Le projet va se dérouler en quatre étapes correspondant aux quatre évaluations prévues (audit 1, soutenance de mi-semestre, audit 2 et soutenance finale) :

1. Première version interconnectant les composants et réalisant le routage systématiquement par inondation qui devra être terminée pour l'audit 1.
2. Seconde version intégrant les tables de routage, leur gestion et leur utilisation dans le routage qui devra fonctionner pour la soutenance de mi-semestre.
3. Troisième version intégrant l'utilisation de *threads* et de la synchronisation à l'interieur des composants pour paralléliser le routage des messages, la connexion des nouveaux voisins et la gestion des tables de

routage. Il s'agira aussi d'intégrer la gestion des connexions et déconnexions dynamiques des nœuds.

4. Quatrième et dernière version où une exécution en multi-JVM devra être réalisée et où un réglage de la performance en fonction des paramètres de configuration des composants (paramètres du routage, comme le nombre de voisins utilisés dans l'inondation, et le nombre de *threads* utilisés par les différentes tâches de différents types de nœuds).



Étape 1 — Première version avec routage par inondation

La première étape du projet consiste à implanter le réseau éphémère en BCM4Java avec uniquement le routage par inondation. Pour cela, il faut s'appuyer sur la spécification exposée dans les sections précédentes puis des les tester pour leurs fonctionnalités. Il s'agira donc de réaliser les éléments suivants :

— des types de composants représentant les entités du réseau éphémère comme des ordinateurs fixes et portables, des téléphones et des tablettes qui vont servir à simuler des envois et réception de messages à partir du réseau éphémère. Certains joueront le rôle de nœuds terminaux, d'autres de nœuds de routage et d'autres de points d'accès ;
— le composant gestionnaire du réseau éphémère qui va servir à enregistrer les nœuds et leur fournir les informaitons pour se connecter avec leurs voisins ;
— les interfaces requises et offertes (en grande partie reprises du présent cahier des charges) ainsi que tous les ports entrants et sortants ainsi que les connecteurs utilisés pour les échanges entre ces composants ;
— des composants représentant les éléments connectés au réseau classique et qui vont servir à simuler des envois et réception de messages à partir de ce réseau.

Vous devez également programmer des tests unitaires et d'intégration démontrant le bon fonctionnement de ces éléments. Votre démonstration pour l'audit 1 devrait déployer au moins huit composants représentant les nœuds (deux terminaux, deux de routage, deux points d'accès et deux sur le réseau classique) et il doit montrer des nœuds qui se joignent au réseau éphémère et s'échangent quelques messages nécessitant plus d'un saut dans le réseau éphémère, et d'autres allant vers et venant de nœuds sur le réseau classique.

Étape 2 — Intégration du routage par tables et greffons

Pour cette seconde étape, il s'agira d'ajouter les fonctionnalités de routage et les tables de routage au sein des nœuds de routage. Cela inclut la procédure de mise à jour des tables entre voisins et la procédure de routage des messages grâce aux tables.

De plus, le cours 3 introduit le mécanisme de greffons (*plug-ins*) permettant de réutiliser facilement des comportements sur différents composants. À cette étape, vous devrez créer des greffons (*plug-ins*) pour chacun des rôles de d'élément et point d'accès du réseau éphémère à installer par les composants participant au réseau éphémère.

— le rôle de nœud terminal,
— le rôle de nœud de routage,
— le rôle de point d'accès et
— le rôle de nœud sur le réseau classique.

Vous pourrez alors créer des composants ordinateurs portables, tablettes, téléphone, etc. et leur attribuer leur rôle en installant le bon greffon au moment de la création du composant.

Lors de la soutenance de mi-semestre, vos tests unitaires et d'intégration de l'étape 1 devront montrer le bon fonctionnement de cette nouvelle version.

Étape 3 — Intégration du parallélisme dans les nœuds et de la déconnexion dynamique

À cette étape, les fonctionnalités développées précédemment doivent être attribuées à différents groupes de *threads* dans les composants de manière à pouvoir s'exécuter en parallèle. Par exemple, un composant de routage doit pouvoir exécuter sur différents groupes de *threads* la connexion des nouveaux voisins, la mise à jour des tables de routage et le routage des messages.

Outre le fait de faire s'exécuter ces tâches sur différents groupes de *threads*, il faudra prendre soin des accès concurrents aux structures de données utilisés pour gérer ces fonctionnalités. Il s'agira donc de choisir différents

mécanismes de synchronisation (collections synchronisées, verrous, etc.) pour assurer des accès cohérents aux structures de données.

Lors de l'audit 2, vos tests unitaires et d'intégration de l'étape 1 devront montrer le bon fonctionnement de cette nouvelle version.

Étape 4 — Exécution répartie et réglage de performance

L'étape 4 aura deux objectifs : réaliser une exécution répartie sur plusieurs JVM et explorer la problématique de la configuration pour obtenir une meilleure performance. Pour le premier objectif, si vos composants sont programmés en suivant les préceptes de BCM vus en cours, la tâche consistera à les déployer dans plusieurs JVM grâce aux mécanismes de BCM qui sont vus en cours.

Tout au long de son développement, le projet devra inclure des tests fonctionnels complets, incluant des tests unitaires implantés avec JUnit quand c'est possible et des tests d'intégration implantés comme des applications en BCM4Java. Lors de la soutenance finale, vos tests unitaires et d'intégration des étapes précédentes devront montrer le bon fonctionnement de cette exécution répartie. Vous y ajouterez des scénarios montrant la déconnexion dynamique de nœuds.

Pour cette quatrième étape, le projet devra également inclure des tests de performance (aussi implantés sous la forme d'applications BCM4Java). Ces tests devront montrer comment se comporte le réseau éphémère en fonction de la pression imposée par les clients, incluant un nombre croissant de messages émis par seconde, un nombre croissant de nœuds, le nombre moyen de voisins et d'autres paramètres de configuration comme le nombre de *threads* dans les groupes de chaque type de composants et le nombre de voisins vers lesquels est émis un message en procédure d'inondation. L'objectif de ces tests est de déterminer comment fixer les choix d'implantation (interconnexion des colposants courtiers répartis) et les paramètres de déploiement (nombre de *threads* utilisés dans chaque courtier pour exécuter ses différentes fonctionnalités – réception et stockage des messages publiés, traitement des souscriptions, filtrage et transmission des messages aux souscripteurs).

Les résultats de cette étape seront :
- un (petit) plan d'expérimentation fixant les cas à traiter et les mesures de performance à prendre (à décrire dans la Javadoc du paquetage regroupant tous ces tests) ;
- des résultats (par exemple sous la forme de tableaux) des mesures de performance obtenues (à présenter lors de la soutenance finale) ;
- des recommandations faites aux installateurs du réseau éphémère pour la bonne configuration des paramètres selon leur situation (à inclure dans la Javadoc).

3 Modalités générales de réalisation et calendrier des évaluations

ok — Le projet se fait **obligatoirement** en **équipe de deux étudiant·e·s**. Tous les fichiers sources du projet doivent comporter les noms (balise `authors`) de tous les auteurs en Javadoc. Lors de sa formation, chaque équipe devra se donner un nom et me le transmettre avec les noms des étudiant·e·s la formant au plus tard le **12 février 2021** à minuit.

... — Le projet doit être réalisé avec **Java SE 8**. Attention, peu importe le système d'exploitation sur lequel vous travaillez, il faudra que votre projet s'exécute correctement sous Eclipse et sous **Mac Os X/Unix** (que j'utilise et sur lequel je devrai pouvoir refaire s'exécuter vos tests).

... — L'évaluation comportera quatre épreuves : deux audits intermédiaires, une soutenance à mi-semestre et une finale, ces dernières accompagnées d'un rendu de code et de documentation. Ces épreuves se dérouleront selon les modalités suivantes :

1. Les deux audits intermédiaires dureront 15 minutes (par équipe) et se dérouleront lors des séances de TD/TME. Le premier audit se tiendra pendant la séance 3 et il examinera plus particulièrement votre avancement sur l'étape 1. Le second audit se déroulera lors de la séance 8 et il examinera plus particulièrement votre avancement sur l'étape 3. Ils compteront chacun pour 5% de la note finale de l'UE.
2. La **soutenance à mi-parcours** d'une durée de 20 minutes portera sur l'atteinte des objectifs des *deux premières étapes* du projet. Elle se tiendra pendant la semaine des premiers examens répartis du **15 au 19 mars 2021** selon un ordre de passage et des créneaux qui seront annoncés sur le site de l'UE. Elle comptera pour 35% de la note finale de l'UE. Elle comportera une discussion des réalisations pendant une quinzaine de minutes (devant l'écran sous Eclipse) et une courte démonstration de cinq

minutes. Les rendus à mi-parcours se feront le **dimanche 14 mars 2021 à minuit** au plus tard (des pénalités de retard seront appliquées).

3. La **soutenance finale** d'une durée de 30 minutes portera sur l'ensemble du projet mais avec un accent sur les *troisième et quatrième étapes*. Elle aura lieu dans la semaine des seconds examens répartis du **17 au 21 mai 2021** selon un ordre de passage et des créneaux qui seront annoncés sur le site de l'UE. Elle comptera pour 55% de la note finale de l'UE. Elle comportera une présentation d'une douzaine de minutes (en utilisant des transparents), une discussion d'une dizaine de minutes également devant écran sur les réalisations suivie d'une démonstration. Les rendus finaux se feront le **dimanche 16 mai 2021 à minuit** au plus tard (des pénalités de retard seront appliquées).

— Lors des soutenances, les points suivants seront évalués :
  — le respect du cahier des charges et la qualité de votre programmation ;
  — l'exécution correcte de tests unitaires (JUnit pour les classes et les objets Java, scénario de tests pour les composants) ;
  — l'exécution correcte de tests d'intégration mettant en œuvre des composants de tous les types ;
  — la qualité et l'exécution correcte des scénarios de tests de performance, conçus pour mettre à l'épreuve les choix d'implantation versus la montée en charge ;
  — la qualité de votre code (votre code doit être commenté, être lisible - choix pertinents des identifiants, . . .– et correctement présenté – indentation, . . .– etc.) ;
  — la qualité de votre plan d'expérimentation et tests de performance (couverture de différents cas, isolation des effets pour identifier leurs impacts sur la performance globale, etc.) ;
  — la qualité de la documentation (vos rendus devront inclure une *documentation Javadoc* des différents paquetages et classes de votre projet générée et incluse dans votre livraison dans un répertoire `doc` au même niveau que votre répertoire `src`).
— Bien que les audits et les soutenances se fassent par équipe, l'évaluation reste à chaque fois **individuelle**. Lors des audits et des soutenances, *chaque étudiant·e* devra se montrer capable d'expliquer différentes parties du projet, et selon la qualité de ses explications et de ses réponses, sa note peut être supérieure, égale ou inférieure à celle de l'autre membre de son équipe.
— Lors des soutenances, **tout retard** non justifié d'un·e ou des membres de l'équipe de plus d'un tiers de la durée de la soutenance (7 minutes à la soutenance à mi-semestre, 10 minutes à la soutenance finale) entraînera une **absence** et une note de 0 attribuée au(x) membre(s) retardataire(s) pour l'épreuve concernée. Si un·e des membres d'une équipe arrive à l'heure ou avec un retard de moins d'un tiers de la durée de la soutenance, il·elle passera l'épreuve seul·e.
— Le rendu à mi-parcours et le rendu final se font sous la forme d'une archive `tgz` si vous travaillez sous Unix ou `zip` si vous travaillez sous Windows que vous m'enverrez à `Jacques.Malenfant@lip6.fr` comme attachement fait proprement avec votre programme de gestion de courrier préféré ou encore par téléchargement avec un lien envoyé par courrier électronique (en lieu et place du fichier). Donnez pour nom au répertoire de projet et à votre archive celui de votre équipe (ex. : équipe LionDeBelfort, répertoire de projet `LionDeBelfort` et archive `LionDeBelfort.tgz`).
— **Tout manquement à ces règles élémentaires entraînera une pénalité dans la note des épreuves concernées !**
— Pour la deuxième session, si elle s'avérait nécessaire, elle consiste à poursuivre le développement du projet pour résoudre ses insuffisances constatées à la première session et donnera lieu à un rendu du code et de documentation puis à une soutenance dont les dates seront déterminées en fonction du calendrier du master.