Toutes les routes mènent à Mido.

Projet Java avancé M1 Miage Apprentissage 2014/2015

1 Dates importantes

— Envoi des binômes par email : 9 février 2015.
— Rendu du projet : 12 avril 2015, 23h.
— Soutenances : 14 avril 2015, à partir de 8h30.

2 Versions

Ce sujet est susceptible d'être modifié.
— 30 janvier : Ajout de questions/réponses.
— 23 janvier 2015 : Mise en ligne du sujet.

3 Description

Depuis un noeud d'Internet, la commande traceroute permet de connaitre la liste des noeuds formant un des chemins du réseau internet vers un autre noeud. Par exemple :

```
traceroute to www.google.fr (173.194.66.94), 30 hops max, 60 byte packets
 1  10.1.3.254 (10.1.3.254)  0.782 ms  0.784 ms  0.777 ms
 2  lamsade-gw.lamsade.dauphine.fr (193.48.71.254)  1.661 ms  1.683 ms  1.659 ms
 3  192.168.2.1 (192.168.2.1)  1.658 ms  1.656 ms  1.932 ms
 4  interco-9.01-auteuil.rap.prd.fr (195.221.127.153)  2.389 ms  2.841 ms  2.385 ms
 5  pe-odeon.rap.prd.fr (195.221.125.25)  2.839 ms  2.836 ms  2.832 ms
 6  vl260-te4-4-paris1-rtr-021.noc.renater.fr (193.51.186.102)  3.534 ms  2.754 ms  2.703 ms
 7  te0-0-0-3-paris1-rtr-001.noc.renater.fr (193.51.177.24)  6.088 ms * *
 8  te0-6-0-2-paris2-rtr-001.noc.renater.fr (193.51.177.89)  4.067 ms  4.046 ms  195.760 ms
 9  * * *
10  google-te1-6-paris2-rtr-021.noc.renater.fr (193.51.182.197)  194.530 ms  193.455 ms  193.392 ms
11  72.14.238.234 (72.14.238.234)  196.899 ms  202.685 ms  204.217 ms
12  * 209.85.245.70 (209.85.245.70)  4.357 ms  2.992 ms
13  216.239.51.198 (216.239.51.198)  8.168 ms 209.85.242.229 (209.85.242.229)  18.170 ms 209.85.248.202 (209.85.248.202)  8.000 ms
14  72.14.238.215 (72.14.238.215)  7.659 ms 66.249.95.238 (66.249.95.238)  8.824 ms 216.239.49.45 (216.239.49.45)  8.721 ms
15  * * *
16  we-in-f94.1e100.net (173.194.66.94)  8.000 ms  7.446 ms  7.780 ms
```

Chaque ligne correspond à un routeur. On y trouve un numéro de ligne, une adresse IP (4 octets !), parfois un nom (pas sur les lignes 1, 3...), des temps de réponse (ignorés dans ce projet). Il y a parfois plusieurs adresses pour une ligne (par ex. ligne 13) – comme traceroute fait 3 essais pour chaque étape, il y a parfois 3 résultats. À vous de voir et de justifier comment vous prenez en compte ceci. Les étoiles correspondent à des absences de réponse. Un même traceroute peut donner des résultats différents – le graphe d'internet n'est pas figé. Les chemins donnés ne sont pas forcément les plus courts non plus.

4 Ce qui est demandé

Le but est de constituer, visualiser et localiser en temps réel le graphe d'internet en effectuant des traceroute automatiquement. Chaque sommet du graphe sera une adresse IP pour laquelle on voudra pouvoir connaitre ses voisins, sa distance *minimale* (cela peut être différent pour des traceroute différents) à l'origine (la machine sur laquelle sont lancés les traceroute). On voudra en plus visualiser ce graphe sur un globe terrestre en géolocalisant chaque sommet du graphe (les adresses IP) grâce à des services externes – comme ces derniers ne sont pas forcément très pertinents, l'utilisateur veut pouvoir choisir le service utilisé.

Le projet doit s'exécuter sous Linux (les machines du CRIO UNIX) et optionnellement sur les autres plateformes où Java et les services externes fonctionnent (*a priori* Windows et MacOS), sans avoir à modifier le programme.

On veut pouvoir :

- exécuter plusieurs traceroute en parallèle pour assembler plus rapidement les informations,
- choisir à la volée combien de traceroute s'effectuent simultanément,
- choisir combien de traceroute sont effectués par minute (de 1 à infini) (optionnel),
- ajouter à la volée des nouvelles cibles pour traceroute (par IP, par nom, par tranche d'adresses ou depuis un fichier contenant des adresses à explorer),
- ajouter des cibles aléatoires (des adresses IP aléatoires (mais pas les adresses réservées – comment les traitez-vous ?)),
- changer de service de géolocalisation,
- visualiser des statistiques sur le graphe construit (nombre de sommets vus, de traceroute lancés, classement par distance, etc),
- indiquer les voisins d'un sommet du graphe,
- visualiser au fur et à mesure, en temps réel, le graphe (y compris pendant qu'un traceroute s'exécute) sur le globe,
- déterminer le plus court chemin dans votre graphe entre un point et l'origine (l'ordinateur où s'exécute le programme), et entre deux points (optionnel)
- exporter et importer (sans écraser l'existant) le graphe construit sous forme de 2 fichiers CSV (séparé par des virgules) : un premier avec la liste des sommets, leur distance à l'origine et leurs coordonnées GPS (1 par ligne), un second avec l'ensemble des couples de sommets reliés (1 couple par ligne) (optionnel),
- importer les résultats de traceroute effectués depuis une autre machine (on ignore alors les distances à l'origine) (optionnel).

Le projet utilisera la commande traceroute (ou équivalent) (voir la classe `ProcessBuilder`). L'utilisateur pourra choisir le nombre maximum de sauts (hop), le temps maximum d'attente par saut. Le traceroute est plus rapide si on désactive la résolution de nom DNS. Les résultats peuvent être plus ou moins intéressants si traceroute est utilisé en mode UDP, TCP ou ICMP.

4.1 Services externes

- Géolocalisation :
  - GeoIP. Télécharger la base de données et l'API Java.
  - HostIP. Faire un GET HTTP avec `URLConnection` pour avoir un retour en XML et utiliser SAX pour le parser.
  - Des services supplémentaires optionnellement.

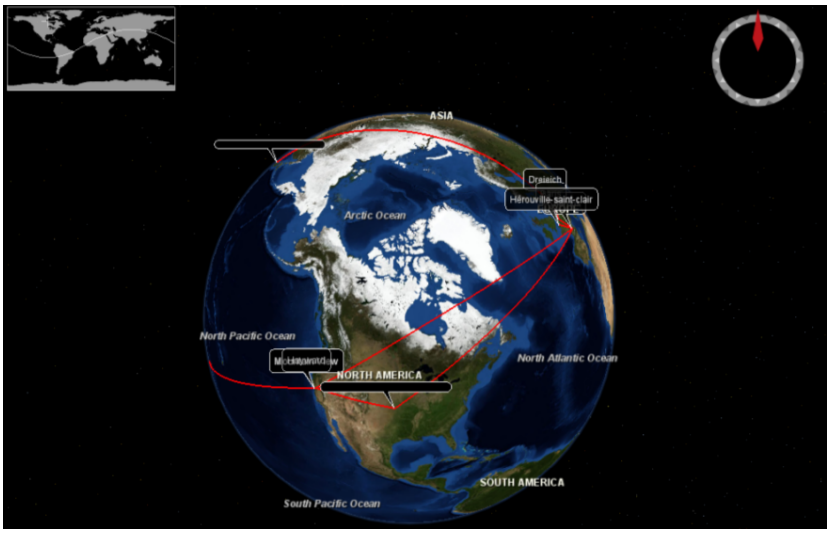

FIGURE 1 – Illustration non contractuelle

— Visualisation :
  — Nasa WorldWind. Voir ici, et notamment les classes `RenderableLayer, AnnotationLayer, GlobeAnnotation` dans la javadoc.

Aucune autre bibliothèque externe (au JDK) est autorisée.

Il est conseillé de faire des tests unitaires (pertinents) tout au long du projet avec JUnit. "Tests should be written before the code. Test-driven development is a lot more fun than writing tests after the code seems to be working. Give it a try !"

La propreté du code, sa lisibilité et l'architecture choisie comptera largement dans la note finale.

5 Conditions de rendu

Le projet est à effectuer en binôme, *i.e.* par 2 personnes. En cas de nombre impair d'élèves, un seul groupe sera autorisé à effectuer le projet en trinôme (noté plus sévèrement). Pour éviter les mauvaises surprises, envoyez votre binôme par email (florian.sikora@dauphine.fr) avant la date précisée plus haut.

Votre projet est à rendre avant la date précisée plus haut, sur l'espace MyCourse dédié. Une fois votre fichier envoyé, vous devez avoir un écran de confirmation avec votre fichier et la date de l'envoi. **Il y aura un point en moins par heure de retard, une heure entamée est due.** Le format de rendu est une archive au format **ZIP** contenant :

— Un répertoire *src* avec les sources de votre implémentation java.
— Un répertoire *classes* vide dans l'archive, devant contenir les classes du projet une fois compilé.
— Un répertoire *docs* contenant :
  — Une documentation pour l'utilisateur *user.pdf* décrivant à un utilisateur comment se servir de votre projet.
  — Une documentation pour le développeur *dev.pdf*, devant justifier les choix effectués, présenter l'architecture choisie, indiquer quelles ont été les difficultés rencontrées au cours du projet ainsi que la répartition du travail entre les membres du binôme.

— Un répertoire vide *doc* pour la javadoc.
— Un répertoire *lib* contenant le minimum de bibliothèques externes nécessaires à votre projet pour s'exécuter.
— Un fichier ant *build.xml* permettant de compiler, créer le jar, générer la javadoc, etc.
— Votre jar exécutable pouvant être lancés au moyen de la commande `java -jar`.

Votre projet doit pouvoir s'exécuter sans utiliser eclipse ! (Vous avez le droit d'utiliser java.library.path).

L'archive aura pour nom Nom1Nom2.zip, où Nom1 et Nom2 sont les noms des membres du binôme par ordre alphabétique. L'extraction de l'archive devra créer un dossier Nom1Nom2 contenant les élements précisés ci-dessus.

Il va sans dire que les différents points suivants doivent être pris en compte :
— Uniformité de la langue utilisée dans le code (anglais conseillé).
— Uniformité des conventions de nommage et de code (convention Java obligatoire).
— Projet compilant sans erreur et fonctionnant sur les machines de l'université.
— Gestion propre des différentes exceptions.
— La documentation ne doit pas être un copié-collé du code source du projet.
— Les sources doivent être commentées, dans une unique langue, de manière pertinente (pas de commentaire "fait un test" avant un if.).
— Le nom des variables, classes et méthodes doivent être choisis judicieusement.
— Le code devra être propre, les exceptions correctement gérées, les classes correctement organisées en packages. La visibilité des méthodes et champs doit être pertinente (privée ou non...).
— Le projet doit évidement être propre à chaque binôme. Un détecteur automatique de plagiat sera utilisé.

La documentation (rapports, commentaires...) compte pour un quart de la note finale. On préférera un projet qui fonctionne bien avec peu de fonctionnalités qu'un projet bancal avec plus de fonctionnalités.

L'utilisation d'un gestionnaire de version type CVS, SVN ou GIT est conseillée (attention toutefois aux sites où le code est public). Riouxsvn semble être gratuit et privé mais je ne le connais pas spécialement.

5.1 Soutenance

Une soutenance d'un quart d'heure aura lieu binôme par binôme à la date précisée plus haut. Elle doit être préparée et menée par le binôme (i.e. fonctionnant parfaitement du premier coup, avec des jeux de tests pertinents etc).

5.2 Binômes

1. André PRAGOSA Chimon SULTAN.
2. Bastien Leca et Ait-Enceur Hakim.
3. Vincent RAULIC Medhi AGLIM
4. Quentin COMBIER Thomas BETHELOT
5. Sarah Chichportich Samy Barboucha
6. Roland Srong Chartrain Laurent

6 Questions/Réponses

— **Les sommets sont seulement les IP machines de destinations ou toutes les machines et IP rencontrées lors du traceroute ?** Tout ce qui a été rencontré.
— **Les deux services de géolocalisation sont à utiliser au minimum ?** Oui, mais l'utilisateur choisit l'un ou l'autre à un instant t pour ne pas avoir de conflit.