MINI-PROJET JAVA: SIMULATION D'UN LAN

Buts du projet

1. Comprendre une application orientée objet existante ;
2. Expérimenter avec le modèle simulé ;
3. Étendre le modèle : exploitation de l'héritage.

Description de l'application LAN

Le but de l'application est de *simuler* le fonctionnement d'un réseau local (ou LAN : *Local Area Network*). Un LAN contient des nœuds, qui sont connectés entre eux, formant un réseau, et s'envoyant des paquets.

C'est un *exercice de conception objet*, donc on ne recherche ni l'exactitude technique ni à interagir sur un réseau physique.

Le modèle objet

Le paragraphe précédent en indique les entités principales :

La classe `Network` *:* une instance de `Network` représente un *réseau* complet et en gère l'assemblage, en particulier les connexions et déconnexions entre nœuds.

La classe `Node` *:* ses instances représentent des *nœuds* ; chaque nœud a un nom, et peut envoyer et recevoir des paquets. Pour simplifier, on utilise le nom des nœuds en place des adresses MAC ou IP, et on oublie le DNS.

La classe `Link` *:* chacune de ses instances représente une *connection* point à point entre deux nœuds voisins d'un réseau, comprenant la liaison physique mais aussi une partie de la pile réseau à chaque extrémité.

La classe `Packet` *:* ses instances représentent les *paquets* transitant sur le réseau, chacun ayant une destination et une charge utile.

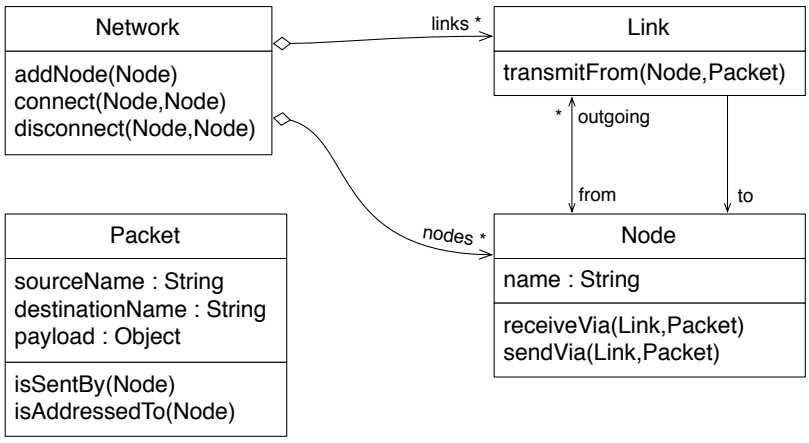

FIGURE 1: Diagramme de classes du LAN tel que fourni initialement (paquet `a43.lan.core`). Quelques détails sont omis, à vous de compléter en suivant votre bon sens.

Partie 1 — Construction d'un réseau

Découverte du code

T1 Lisez très rapidement l'ensemble du code fourni (un quart d'heure). Ne vous arrêtez pas au code des méthodes, faites vous simplement une carte mentale de l'organisation du projet, identifiez les éléments du diagramme en figure 1.

Construction de réseaux

T2 Créez une classe avec une méthode `main`, comme premier programme de tests. Implémentez `toString` dans Node, pour obtenir une représentation affichable qui vous soit utile, et testez en affichant une instance (la méthode `println` repose sur `toString`) :

```
Node n = new Node("n");
System.out.println(n);
```

Continuez similairement pour `Link` et `Network`. La représentation d'un lien entre deux nœuds A et B peut être A<−>B, et celle d'un réseau peut être une liste de ses liens (un par ligne).

T3 Dans un second programme de test, construisez et affichez un réseau en étoile, correspondant à la figure 2 ci-contre. Faites attention à la création des liens : à quelle classe cette tâche revient-t-elle ?

T4 Dans un troisième programme de test, construisez et affichez un réseau correspondant à la figure 3 ci-contre. Ce réseau vous servira lors de la partie sur le routage, page 5.

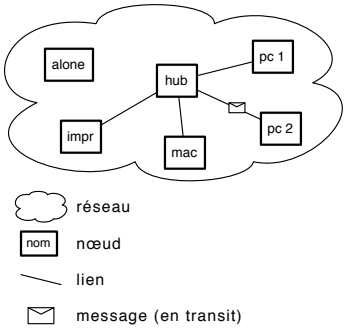

FIGURE 2: Un exemple de réseau

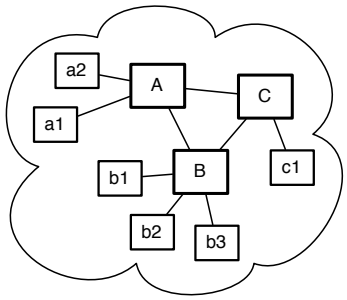

FIGURE 3: Un réseau plus complexe

Lecture approfondie

T5 On s'intéresse maintenant aux paquets et à leur acheminement. Pour commencer, examinez le petit programme d'essai LauncherMacPc. Voici quelques questions pour vous guider :

1. Une instance de `Link` représente-t-elle une connexion unidirectionnelle ou bidirectionnelle ?
2. Dans le code, qu'est-ce qui vous permet de déduire cela ? (il y a au moins 3 endroits)
3. Parmi les attributs de la classe `Packet`, lesquels ont une valeur fixe après initialisation, et lesquels représentent un état qui pourra changer au cours de la vie d'une de ses instances ? Faites le diagramme d'états correspondant.
4. De sa création dans la méthode `originatePacket`, à son traitement dans méthode `consume`, à travers quel enchaînement d'appels de méthodes une instance de `Packet` sera-t-elle acheminée ? (généraliser à partir LauncherMacPc et des réseaux construits précédemment).

Envoi d'un paquet et traçage

T6 Dans la méthode `buildNetwork` de la classe `BasicNetworkTest`, copiez votre construction du réseau en étoile (figure 2), cette fois en utilisant les variables d'instance déjà définies. Lancez ensuite cette classe en tant que suite de tests unitaires.

Pour lancer une suite de tests JUnit : Clic droit sur la classe, «Run as», «JUnit test».

À cette étape du projet, certains tests ne passent pas ! Un bug a été laissé par l'auteur du sujet de TP, à vous de le corriger.

T7 Insérez des instructions d'affichage où c'est nécessaire dans le code pour obtenir dans la console une trace du parcours du paquet jusqu'à sa réception. Vous pouvez également utiliser l'exécution dans le debugger.

T8 Notez le parcours précis de l'objet représentant le paquet à travers le code ; faites le diagramme de séquence UML correspondant. Quel est le problème ?

Partie 2 — Spécialisation des nœuds

Dans cette partie, vous allez corriger et étendre le modèle en faisant des sous-classes de Node.

Loopback

T 9 Examinez et lancez la suite de tests `BasicNetworkTest`. Parmi les trois tests unitaires définis, deux échouent ; commençons par `testScenarioSelfSend` :

— Faites-vous une idée du problème : soit le test est incorrect, soit il *devrait* passer en l'état, auquel cas c'est un bug de l'application.

— Corrigez le code en conséquence.

Retransmission

T 10 Pour `testScenarioHelloMacPc`, la situation est un peu plus complexe. Notez d'abord que le réseau `net` construit par la méthode `setUp` correspond à celui de la figure 2.

— Quel bug ce test met-il en évidence ?

— Comment ce bug limite-t-il la topologie des réseaux qui peuvent fonctionner ?

— Qu'est-ce que cela vous dit sur les tests qui passaient au départ, en particulier `testScenarioLprDisconnected` ?

— Voyez-vous une solution *simple et correcte* qui ne nécessite pas une nouvelle classe ?

T 11 La solution réelle nécessite de modéliser les équipements de routage en tant que tels.

— En quoi un hub se comporte-t-il différemment d'une machine normale connectée au réseau ?

— Comment cette différence se traduit-elle par rapport à la classe Node ?

T 12 Modélisation d'un hub :

— Créez le package Java `a43.lan.nodes`, et ajoutez-y une nouvelle classe Hub, étendant Node.

— Faites une copie `HubNetworkTest` de `BasicNetworkTest` et utilisez Hub pour le nœud central.

Gardez `BasicNetworkTest` inchangée pour le reste du projet.

— Faites les redéfinitions et/ou extensions nécessaires dans Hub pour faire passer les tests de `HubNetworkTest`.

Note : un hub n'ayant pas d'information de routage, il propage chaque paquet qu'il reçoit sur tous ses liens, sauf le lien par lequel le paquet est arrivé.

— Simplifiez l'implémentation de Node, sachant que désormais ses instances n'ont plus besoin de faire de routage.

Autres nœuds spécialisés

T13 Implémentez et incorporez dans les tests quelques autres types de nœuds spécialisés, par exemple :

Cette tâche est rapide et, selon la topologie du réseau, faisable indépendamment de Hub.

— Les nœuds `Workstation` incrémentent un compteur interne à chaque paquet qu'ils reçoivent, et fournissent un accesseur en lecture à la valeur du compteur.

— Les nœuds `Printer` ont un stock de papier initial (quantité définie à la construction du nœud), et un bac de sortie représenté par une liste de chaînes. À chaque message reçu, le `payload` est « imprimé », ajoutant ajoutant une chaîte au bac de sortie et consommant une feuille de papier.

— Les nœuds `Server` se comportent comme un serveur *daytime* : ils traitent chaque paquet reçu comme une requête, et répondent à l'émetteur en leur envoyant un autre paquet portant la date et l'heure.

Routage

T14 Créez une nouvelle classe `RoutingNetworkTest` sur le modèle de `BasicNetworkTest`, mais pour tester le réseau en figure 3 que vous avez construit précédemment : adaptez `buildNetwork` ainsi que les tests, et essayez d'acheminer un paquet (par exemple entre a1 et c1).

T15 Si vous utilisez Hub pour les nœuds A, B, C de `RoutingNetworkTest`, que se passe-t-il ?

T16 Implémentez une classe `Router` disposant d'une table de routage. La table de routage peut être représentée par une collection de type `Map<String,Link>` associant le lien de retransmission idéal à chaque nom de nœud destinataire possible. Configurez les tables de routage manuellement, à la construction du réseau.

T17 Modifiez `RoutingNetworkTest` pour faire des trois nœuds centraux A, B, C des instances de `Router` avec les tables de routage adéquates. Testez et validez le fonctionnement.

Partie 3 — Spécialisation des paquets

Expiration

Normalement les paquets expirent au bout d'un certain temps : c'est le TTL ou *time to live*. Pour modéliser cela, vous allez définir un nouveau type de paquet, qui expirera après avoir traversé un certain nombre de liens, ce nombre étant spécifié à la construction du paquet.

Pour cette partie, essayez le développement dirigé par les tests.

T 18 Créez une classe de tests JUnit `a43.tests.TtlPacketTest`. Supposez qu'une classe `a43.lan.packets.TtlPacket` existe ; imaginez comment vous l'utiliseriez et écrivez des tests pour différentes valeurs du TTL (nul, trop court, exact, plus long que nécessaire) et différents chemins à travers un réseau construit selon vos besoins.

«New…», «JUnit Test Case»

T 19 Eclipse vous proposera de créer le package `a43.lan.packets`, la classe `TtlPacket`, les méthodes que vous aurez utilisé dans vos tests. Laissez-vous guider, faites les définitions et corrections nécessaires dans `TtlPacket`, `TtlPacketTest`, et le reste du code (voir la tâche suivante).

T 20 Lorsqu'un paquet est transmis à travers un lien, cela doit décrémenter son TTL. Pour cela il y a plusieurs approches possibles :

— Ajouter une sous-classe `TtlLink` ayant connaissance explicite de `TtlPacket`, par exemple en ignorant les paquets dont le TTL est négatif ou nul.

— Faire que la méthode `transmit` de `Link` donne la possibilité aux paquets d'intervenir lors de leur transmission, et éventuellement d'y faire *veto*.

Quelle solution vous semble la plus simple ? La plus flexible ?

T 21 Ajouter une méthode dans `Packet` :

```
boolean vetoTransitVia(Link l) { return false; }
```

Dans `Link`, corriger la méthode `transmit` pour qu'elle respecte le veto du paquet. Vérifiez que cette modification n'a pas d'impact sur les tests des parties précédentes, puisque les instances de `Packet` n'utilisent jamais leur droit de veto.

T 22 Redéfinissez `vetoTransitVia` dans `TtlPacket` pour faire veto si le paquet a expiré. Vérifiez que vos tests définis au début de la partie passent.

Autres spécialisations de `Packet` possibles pour aller plus loin :

— Paquets avec différents protocoles applicatifs (par ex. ICMP / ping) et traitements associés par les nœuds ;

— Paquets à plusieurs destinataires : broadcast, multicast…

Partie 4 — Spécialisation des liens

Fiabilité

On introduit une sous-classe de `Link` : `a43.lan.links.Wifi`. Les transmissions radio étant moins fiables, les liens wifi peuvent perdre des paquets lors de la transmission.

T23 En suivant la méthodologie de développement dirigé par les tests, implémentez `Wifi` avec des coupures de connexion déclenchées arbitrairement depuis le code de test, pour un nombre de paquets donnés. Une fois que le lien a perdu suffisemment de paquets, la connexion se rétablit spontanément.

T24 Pour simuler des liens non fiables perdant des paquets selon une certaine probabilité, il faudrait utiliser un générateur de nombres pseudo-aléatoires ; comment pouvez-vous contrôler cela pour toujours avoir des tests déterministes ?

Voir la classe `java.util.Random`.

T25 Étendez `Wifi` pour avoir un taux de perte configurable à la construction de l'objet, en plus du mécanisme des coupures de connexion arbitraires.

T26 Écrivez un programme vérifiant expérimentalement le taux de perte de paquets entre deux points non contigus d'un réseau non fiable, pour différentes combinaisons de liens fiables/non fiables. Vous pouvez réutiliser la classe `Workstation` si vous l'avez implémentée (page 5).

Engorgement

T27 Dans un vrai réseau, chaque lien peut transmette un certain nombre de paquets par seconde, et le fait indépendemment des autres liens et paquets en transit. Le modèle actuel de l'application est-il réaliste vis-à-vis de l'acheminement des paquets dans le temps ? Comment l'étendriez-vous pour pouvoir faire des simulations d'engorgement du réseau ?