RFC (V2) - UGEGreed

Projet Programmation Réseau 2023

Cesaire Agonsè - Nicolas Atrax

1.Introduction :

1.1 Objectif :

Le but de notre protocole est de permettre la connexion de plusieurs applications organisées en réseau afin de résoudre des conjectures proposées par l'utilisateur en répartissant efficacement les tâches entre celles-ci.
Pour se faire, ce protocole dépend de serveurs TCP et d'échanges de différents paquets dont la composition est détaillée dans la suite de ce document.
Ces paquets devront s'échanger dans un certain ordre et selon un certain contexte détaillé également ci-dessous.

1.2 Glossaire:

- **nœud :** une application du réseau qui peut être un serveur , un client, ou les 2 à la fois.
- **père :** Le père d'un nœud est l'unique serveur du nœud en question.
- **fils :** Le fils d'un nœud est l'un des clients du nœud en question.
- **source :** la source est le nœud originel qui demande un calcul. C'est lui qui, lorsque que toutes les conjectures seront calculées, enverra le résultat final d'un calcul.
- **l'envoyeur :** l'envoyeur est le nœud qui envoie une conjecture à un autre nœud. Il est celui qui envoie un paquet à un autre mais pas forcément la source de la conjecture.
- **routeur** : C'est le carnet d'adresse pour envoyer un paquet à un nœud du réseau.

1.3 Différentes opérations d'une application :

Une application peut avoir plusieurs comportements. A tout moment elle pourra signaler à son réseau qu'elle veut :

- se connecter à une application du réseau
- se déconnecter à une application du réseau
- demander un calcul
- envoyer un résultat
- rediriger un résultat

1.3.a Se connecter à une application du réseau:

A ← B

"A" est un serveur et "B" est un client qui s'y connecte.

“A” sera le père de “B” et “B” sera le fils de “A”.

“B” à la connexion envoie à “A” une requête indiquant son adresse d'identification. “A” regarde s' il est en mesure de pouvoir accepter une nouvelle connexion.

- si c’est le cas. A change d'états et refuse toute nouvelle connexion et envoie une requête réponse à “B” contenant l’adresse d'identification de “A” et la liste des nœuds dans le réseau. “B” va pouvoir créer son routeur et relier tous les nœuds du réseau à “A”. Enfin “A” rajoute à son routeur l’adresse d’identification de “B” puis change d’état pour accepter à nouveau une nouvelle connexion.
- sinon il envoie une requête réponse qui refuse la connexion explicitement et “B” se déconnecte sur le champ.

“A” envoie la déclaration de “B” à tout autres noeuds “X” qui contiendra l’adresse d'identification de “B” et l’adresse d'identification de l’envoyeur de la requête (ici “A”) pour mettre à jour le routeur de “X”. Et ceci récursivement.

Après l’opération, si “A” est en train d’effectuer un nouveau calcul alors.

- “A” devra envoyer à “B” une requête contenant la moitié des plages de calcul de “A” et le résultat sera donc à envoyer à l’adresse “A”.

1.3.b Se déconnecter d’une application du réseau:

A ← B

“A” est un serveur et “B” est un client qui se déconnecte spontanément .

- “B” devra envoyer à “A” une information de déconnexion à “A” avec la liste des fils attachés à lui Y.
  Si B a des calculs en cours alors l’action sera interdite et donc annulée.
- “A” est donc au courant de la future déconnexion.
- Si “A” n’est pas en mesure de récupérer de nouvelles connexions alors il enverra une requête de refus et les interactions s'arrêtent ici sinon une requête d'acceptation sera envoyée.
- “B” se déconnecte à “A” et envoie à tous ses fils une requête qui dit qu’ils doivent se connecter à “A”.
- Tous les fils de “B” se déconnectent de “B” et se connectent à “A”.
- “A” attend la connexion de tous les clients de la liste Y qui seront les nouveaux fils de “A”.
- Enfin “A” alerte à tous ses autres nœuds via une requête que “B” doit être supprimé et remplacé de leurs routeurs par “A”.
- Tous les noeuds “X” recevant ces requêtes font de même, récursivement, avec les autres noeuds en remplaçant “A” par “X”

NB: si le nœud “B” est en mode ROOT la déconnexion est interdite par le protocole. La liste Y est une suite de requêtes distinctes.

1.3.c Demander un calcul :

Quelque soit la hiérarchie des nœuds voici un réseau banal.

D
↕
E ↔ A ↔ C ↔B

Un nœud “C” demande 100 calculs au réseau et il devient alors la source d’un calcul.

- Il divise le nombre de conjectures par le nombre de fils + son père + lui-même.
- Il envoie ensuite à tous ses fils et à son père les plages dédiées de calcul.
- Une requête contiendra l’id correspondant au type de paquet, l’id de conjecture, les informations correspondants au jar utilisé, la plage des valeurs.

D
↕
E ↔ A ← C → B

De manière arbitraire “A” et “B” auront 33 conjectures à gérer et “C” 34. (100/3)

- Comme une application est capable de retenir en mémoire de qui provient chaque requête, chaque nœud recevant cette requête recalculera à son tour une répartition des calculs avec ses autres nœuds excepté celui qui l’a envoyé.

D
↑
E ← A ↔ C ↔ B

De manière arbitraire “D”, “E” et “A” auront 11 conjectures à gérer. (33/3)

- Les requêtes s’arrêteront donc aux feuilles de l'arbre (ici E, D et B).

Une fois la requête envoyée, l’application attend de recevoir la validation ou le refus de sa demande afin de savoir si elle doit envoyer la requête à une autre application.

Si, après la répartition des calculs entre ses fils (et son père), l’application ne peut toujours pas traiter une certaine plage de valeur, celle-ci envoie alors un refus de la demande à l’envoyeur en précisant la plage de valeur qui n’a pas pu être traitée.

1.3.d Envoyer un résultat :

Toujours sur le même exemple de demander un calcul, une fois une conjecture calculée pour une valeur donnée, un nœud enverra une réponse au nœud qui a envoyé la demande de calcul.

D
↓
E → A ↔ C ← B

Cette requête contiendra l’id correspondant au type de paquet, l’id de la conjecture, la valeur traitée et une chaîne de caractères contenant les informations liées au calcul de conjecture (valeur traitée, validité de la conjecture, résultat).

D
↕
E ↔ A → C ↔ B

Le nœud qui reçoit la requête regarde s' il est lui-même la source sinon il renvoie cette requête au nœud qui a envoyé la demande de calcul car celui-ci a gardé en mémoire la demande.

Enfin la source, après avoir reçu tous les résultats d'une conjecture, peut renvoyer leur concaténation.

1.3.e Rediriger un calcul :

D
↕
E ↔ A ↔ C ↔B

A tout moment selon diverses situations vu ci-dessus, en se déconnectant, un nœud X peut décider de rediriger un calcul vers un nœud Y en envoyant une requête contenant l'id d'un calcul avec l'adresse du nœud qui devra recevoir le résultat (qui sera considéré alors comme l'envoyeur aux yeux du noeud qui reçoit cette requête).
Pour cela on considère qu'il n'y a pas d'erreur de la part d'une application qui envoie cette requête et le nœud Y sera en mesure de pouvoir envoyer le résultat au nœud désiré.

2. Paramètres du protocole :

2.1 Les différents types de paquets :

Chaque paquet échangé au sein du réseau se compose d'abord d'un int (en Big Endian) représentant l'id correspondant au type de paquet en question. Voici la liste des différents id de paquet possibles:

- 0 = requête de calcul (voir le point a)
- 1 = réponse à la requête de calcul (voir le point a)
- 2 = paquet de résultat du calcul (voir le point b)
- 3 = paquet de déconnexion à un serveur (voir le point c)
- 4 = paquet de redirection de connexion (voir le point c)
- 6 = paquet de déclaration de nouveau noeud dans le réseau (voir le point d)
- 7 = paquet réponse à une déclaration de nouveau noeud le réseau (voir le point d)
- 8 = paquet de suppression de noeud dans le réseau (voir le point c)

2.1.a Demande de calcul :

Une fois que l'utilisateur demande le test d'une conjecture, l'application doit être en mesure d'envoyer aux autres membres du réseau une demande de calcul correspondant à la plage de valeur de la conjecture à tester.
Un nœud a alors fini de conjecturer son calcul lorsque toutes ses plages sont calculées.

Requête de demande de calcul:

Ce paquet n'est pas borné et se décompose comme suit :

- Un int (en Big Endian) représentant l'id de la requête.(0)
- Un int (en Big Endian) représentant l'id du calcul.
- L'inetSocketAddress(*) du nœud à l'origine de ce calcul.
- Un int (en Big Endian) représentant la taille (en octets) de l'url du Jar encodé en UTF-8
- Les octets correspondants à l'url du Jar encodé en UTF-8
- Un int (en Big Endian) représentant la taille (en octets) du nom qualifié de la classe contenue dans le Jar encodé en UTF-8
- Les octets correspondants au nom qualifié de la classe contenue dans le Jar encodé en UTF-8
- Un int (en Big Endian) représentant la borne inférieur de la plage de valeurs à tester
- Un int (en Big Endian) représentant la borne supérieur de la plage de valeurs à tester

Réponse:

Une fois cette requête reçue, le membre l'ayant reçu doit indiquer s'il accepte ou refuse la demande.

Ce paquet n'est pas borné et se décompose comme suit :
- Un int (en Big Endian) représentant l'id du paquet.(1)
- Un int (en Big Endian) représentant l'id du calcul.
- L'inetSocketAddress(*) du nœud à l'origine de ce calcul.
- Un int(en Big Endian) représentant l'acceptation(1) ou le refus(0) de la demande

Dans le cas d'un refus, l'application envoie la plage de valeur qui n'a pas été traitée afin d'indiquer à l'envoyeur quelle nouvelle demande il devra effectuer auprès des autres applications ainsi que le jar et le nom de la classe.
Le paquet contient alors les informations suivantes :
- Un int (en Big Endian) représentant la taille (en octets) de l'url du Jar encodé en UTF-8
- Les octets correspondants à l'url du Jar encodé en UTF-8
- Un int (en Big Endian) représentant la taille (en octets) du nom qualifié de la classe contenue dans le Jar encodé en UTF-8
- Les octets correspondants au nom qualifié de la classe contenue dans le Jar encodé en UTF-8
- Un int (en Big Endian) représentant la borne inférieur de la plage de valeurs qui n'a pas pu être traitée
- Un int (en Big Endian) représentant la borne supérieur de la plage de valeurs qui n'a pas pu être traitée

2.1.b Envoie du résultat du calcul

Une fois le calcul effectué, le membre doit envoyer le résultat à l'application lui en ayant fait la demande.

Ce paquet n'est pas borné et se décompose comme suit :
- Un int (en Big Endian) représentant l'id du paquet.(2)
- Un int (en Big Endian) représentant l'id du calcul.
- L'inetSocketAddress(*) du nœud à l'origine de ce calcul.
- Un int (en Big Endian) représentant la valeur traité.
- Un int (en Big Endian) représentant la taille (en octets) de la chaîne de caractères encodé en UTF-8 contenant le résultat de la conjecture.
- Les octets correspondants à la chaîne de caractères encodé en UTF-8 contenant le résultat de la conjecture.

2.1.c Déconnexion au serveur

Lors de la déconnexion d'un client à un serveur, celui-ci envoie une requête indiquant la liste des fils qui lui sont rattachés via différentes requêtes distinctes pour chaque client et calculs à traiter.

Requête de déconnexion :

Ce paquet ne dépassent pas 2048 octets et se décompose comme suit :
- Un int (en Big Endian) représentant l'id de la requête.(3)
- Un byte (boolean) indiquant si c'est une réponse ou non(0).
- L'inetSocketAddress(*) du nœud qui se déconnecte.
- Un long (en Big Endian) représentant le nombre de fils qui vont devoir se connecter au serveur directement.(int)
- L'inetSocketAddress(*) de chaque nœud qui se connectent au père.

Le père envoie directement une réponse si il est en capacité de recevoir de nouvelles connexions.

Requête réponse d'une déconnexion :

- Un int (en Big Endian) représentant l'id de la requête.(3)
- Un byte (boolean) indiquant si c'est une réponse ou non(1).
- Un byte (boolean) indiquant le refus ou non.

Si il n'y a pas de refus on passe à la suite sinon il ne se passe rien.
Le père n'accepte plus de nouvelles connexions et attend uniquement les nœuds qui doivent se connecter.

Requête de redirection:

Le client envoie alors une requête à tous ses fils.

Ce paquet ne dépassent pas 2048 octets et se décompose comme suit :

- Un int (en Big Endian) représentant l'id de la requête.(4)
- L'inetSocketAddress(*) du nœud de redirection.

Requête de suppression d'un noeud dans le réseau :

- Un int (en Big Endian) représentant l'id de la requête.(8)
- L'inetSocketAddress(*) du père du nœud supprimé.
- L'inetSocketAddress(*) du nœud supprimé.

Enfin le père peut se remettre dans un état normal et accepter de nouvelles connexions.

2.1.d déclaration de noeud

Lorsqu'une application reçoit un fils, l'application fils envoie son identité.

Requête de déclaration de noeud (not for all) :

- Un int (en Big Endian) représentant l'id de la requête.(6)
- Un byte (boolean) qui indique si la requête est valable pour tout le réseau ou non (0).
- L'inetSocketAddress(*) de l'application fils.

L'application envoie une réponse si elle est capable d'accepter une nouvelle node ou non.

Si non :

Requête d'acceptation de noeud:

- Un int (en Big Endian) représentant l'id de la requête.(7)
- Un byte (boolean) qui indique le refus ou non (1) du nouveau nœud.

L'application fils se déconnecte alors proprement et les interactions s'arrêtent ici.

Si oui :

Requête d'acceptation de noeud:

- Un int (en Big Endian) représentant l'id de la requête.(7)
- Un Byte (boolean) qui indique le refus ou non (0) du nouveau nœud.
- L'inetSocketAddress(*) de l'application père
- Un int (en Big Endian) représentant la taille de la liste des nœuds dans le réseau (autre que le nœud fils et père).
- pour chaque nœud de la liste d'inetSocketAddress(*) de celui-ci.

L'application père déclare le nouveau nœud à tous ses autres nœuds. De même, récursivement, chaque nœud qui reçoit cette requête, l'enverront à ses autres nœuds.

Requête de déclaration de noeud (for all) :

- Un int (en Big Endian) représentant l'id de la requête.(6)
- Un byte (boolean) qui indique si la requête est valable pour tout le réseau ou non (0).
- (origine) L'inetSocketAddress(*) du nœud qui est nouveau dans le réseau.
- (destination) L'inetSocketAddress(*) du nœud le plus proche rattaché au nœud origine.

* Encodage d'une inetsocketAddress :

Une inetSocketAddress est toujours encodé de cette manière:

- Un int (en Big Endian) de représentant la taille (en octets) du string de l'host encodé en UTF-8
- Le string de l'host encodé en UTF-8 du nœud.
- Un int (en Big Endian) représentant son numéro de port.

2.2 Choix et difficultés rencontrés :

A l'origine nous voulions partager de manière équitable tous les calculs sur les différents nœuds du réseau. Alors nous avons d'abord pensé à envoyer une requête qui signale, lorsqu'une nouvelle application est sur le réseau, le nombre de noeuds présent sur le réseau et qui se propage récursivement sur chaque branche d'un noeud pour pouvoir anticiper la charge de calcul à partager selon les fils et les pères de chaque noeuds. Mais nous nous sommes rendu compte qu'il était très difficile de gérer les chemins et les redirections de calcul. Alors nous avons choisi une autre solution plus simple et fonctionnelle, qui divise la charge de travail par le nombre de nœuds connectés sur chaque nœud. L'efficacité des calculs d'un réseau est alors dépendante de la disposition de celui-ci. Ainsi plus un réseau à de branche par nœud, plus il est efficace. En revanche, à partir de l'application ROOT plus le réseau est profond avec des étages avec peu de branche moins il sera efficace.

De plus, nous avons choisi de faire le moins de requêtes différentes possibles, pour avoir un protocole le plus simple possible, même si cela implique de la sécurité et des confirmations manquantes. Nous considérons que les applications d'un même réseau respecterons le protocole à la lettre pour ne pas avoir d'erreur.

Enfin au début du développement de l'application, nous voulions gérer la répartition des calculs qu'effectue une node lors d'une déconnexion avec une requête faite pour (id 5). Mais malheureusement, vers la fin du développement, nous nous sommes rendu compte que l'implémentation provoque beaucoup trop de scénarios différents pour au final une petite fonctionnalité. Alors nous avons décidé de régler ce problème en interdisant à une node d'effectuer un calcul lors d'une déconnexion. Ainsi il n'existe pas de requête 5.