Partie 2 : Réalisation d'un serveur centralisé de gestion de plusieurs groupes de communication

I Conception

La réalisation de ce système se base sur deux types de nœuds Client dont le nombre est à fixer et Server instancié par une seule entité.
Les nœuds clients peuvent être hétérogènes en termes de caractéristiques hardware mais ayant un même profil unifié pendant tout le processus de communication.
La communication se fait seulement par échange de message entre les clients et le server.
La communication entre les clients et le server se réalise par échanges de messages d'une manière 1/ distribuée 2/ centralisée et 3/ en mode Client/Server tout en assurant 4/ un ordre consistant à la réception.

II Architecture

On a pensé à implémenter la solution sur un réseau filaire en mode infrastructures dans une architecture Client/Server.
La topologie construite est une topologie étoilée (en étoile). Cette dernière est générée automatiquement après avoir précisé, comme paramètre de simulation, le nombre de nœuds clients à utiliser.

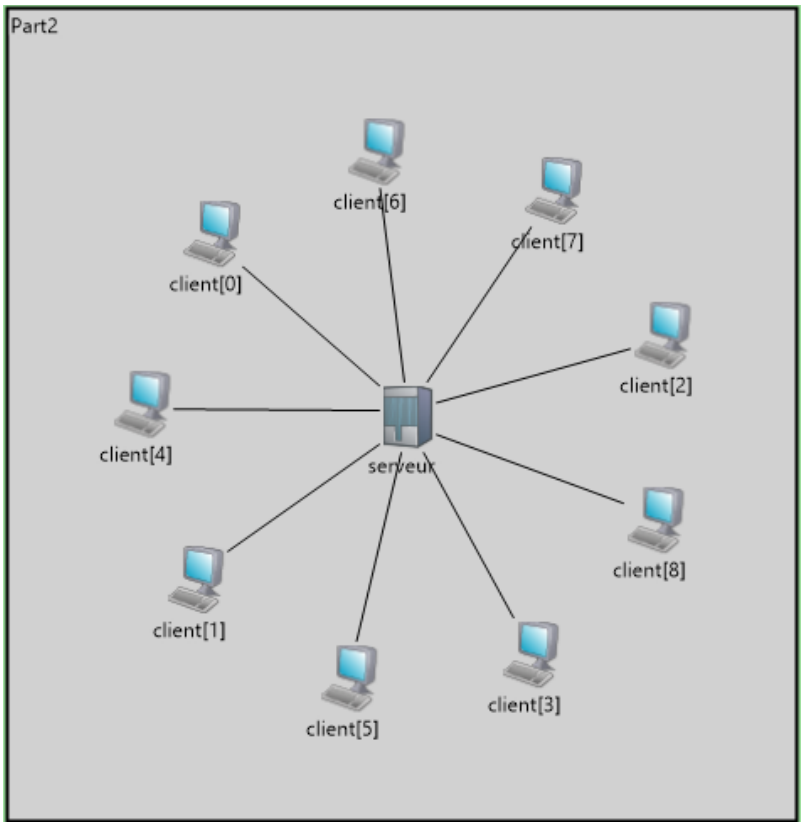

Figure 2 – Exemple de simulation de la topologie Client/Server en étoile avec 9 nœuds client

Chaque nœud client, possède une interface de type inout. Cette dernière assure une communication bidirectionnelle.

```
inout inoutport;
```

Le nœud server possède un groupe d'interface de type inout. Ces dernières assurent des communications bidirectionnelles avec les clients auxquelles elles sont liées.
Le nombre d'interface du nœud server est égal au nombre des nœuds clients.
Pour réaliser les liaisons entre le server et les clients, on a utilisé des liaisons en Full-Duplex comme suit.

```
client[i].inoutport <--> { delay = 500 ms; } <--> serveur.inoutport++;
```

III Structure de données et message utilisés

Chaque nœud client i possède un vecteur adress_c[i][4], ce dernier est composé de 4 cases qui contiennent 1/l'ID du réseau (10.0.0.0/16) 2/l'ID du sous réseau(10.0.x.0/24) 3/ et l'ID de la machine respectivement.
Le serveur contient une matrice adress_s[253][3],le nombre de ligne total = 253 = nombre maximum de nœud dans la topologie.
Chaque ligne contient 3 cases qui reflète 1/L'ID du nœud 2/l'ID du sous réseau auquel le nœud appartient 3/L'ID machine de l''adresse IPv4 de ce même nœud.
Chaque nœud i possède un buffer queue[i] lui servant à stocker tous les messages reçus. Cela permet de faire un processus de tri basé sur des numéros de séquence avant la consommation pour assurer un ordre consistant à la consommation.
La solution repose sur un seul type de message nommée Segment, composé de 5 champs comme suit :

```
packet Segment {
    int source; //id du nœud émetteur
    int destination;//id du groupe de destination
    int num_seq; //numéro d'ordre de consommation du message
    int msg_dhcp; //si = 1 donc c'est un msg d'attribution d'adresse IP
envoyé par le server vers le client, m'adresse en question et définie dans le champ
adresse
    int address[4]; }
```

IV Principe de fonctionnement

Tout d'abord, le serveur attribue des adresses IPv4 à ses clients, ces adresses IPv4 attribuées appartiennent toutes à la même adresses 10.0.0.0/16, mais elles se différent dans l'ID du sous réseau x 10.0.x.0/24. L'ID du sous réseau est entre 0 et 2 (donc il y'aura 3 groupes), et le serveur choisit cette variable aléatoirement. Donc on aura 3 sous réseau qui regroupe l'ensemble des nœuds.
Le nœud ayant l'ID = 0 envoie au serveur 3 messages au 1er groupe (le groupe ayant l'ID du sous réseau = 0), puis désactive sa pile protocolaire.
A la réception de chaque message, le server affecte un numéro de séquence selon l'ordre de réception, puis le diffuse à tous les nœuds membres du groupe indiqué par le client source. A la fin de la réception des 3 messages, le serveur supprime le nœud 0 de sa liste d'adresses.
Les clients membres du groupe indiqué par le client source reçoivent les 3 messages, et les consommes selon leurs numéros de séquence.

V Performance de la simulation

Le seul message de control utilisé est celui d'attribution d'adresses IPv4 par le serveur à ses clients.
Les messages échangés sont d'un seul type : Segment. Aucun processus d'identification du type du message reçu par un nœud i n'est nécessaire (traitement monocritère).
La totalité des messages circulant dans le réseau est = Message de calcul (user data) + n (où n = nombre total de nœud dans la topologie)

VI Critique de la solution proposée

Avantages

1/ Simplicité de conception et de déploiement.
2/ L'unicité du server reflète l'absence totale des problèmes de cohérence (aucun message de contrôle inter-serveurs, ni un processus de mise-à-jour ne sont nécessaires pour assurer cette cohérence).
3/ Accès et transmission rapide des messages (le délai dépend seulement des caractéristiques des liaisons, et non pas des chemins pris par ces messages).
4/ Aucun protocole de routage n'est nécessaire.

Inconvénients

1/ D'éventuels problèmes de disponibilité vu l'unicité du serveur, et pour cette même raison, la solution n'est pas tolérante aux pannes qui affectent le serveur.
2/ La solution proposée ne passe pas à l'échelle car elle dépend de la capacité du server, du nombre max de groupes et des nombres de clients dans chaque groupe (253 clients maximum pour tous les groupes), cela implique des problèmes d'évolutivité (No Scalability).
3/ Possibilité de saturation des services offerts par le serveur (service DHCP et service de diffusion des messages) si le nombre des messages des clients, ou les clients eux-mêmes, augmentent (Deny Of Service).
4/ Taille plus ou moins grande du message Segment (vu qu'il sert à assurer les messages des clients et les messages de control (DHCP) en même temps, ça sera plus léger d'utiliser deux type de messages pour chaque type de donnée (données de control / données de calcul).
5/ Absence d'un protocole pour assurer la fiabilité de transmission des messages. Tout message perdu pendant le processus de transmission ne sera par renvoyé.