

RAPPORT PROJET TP

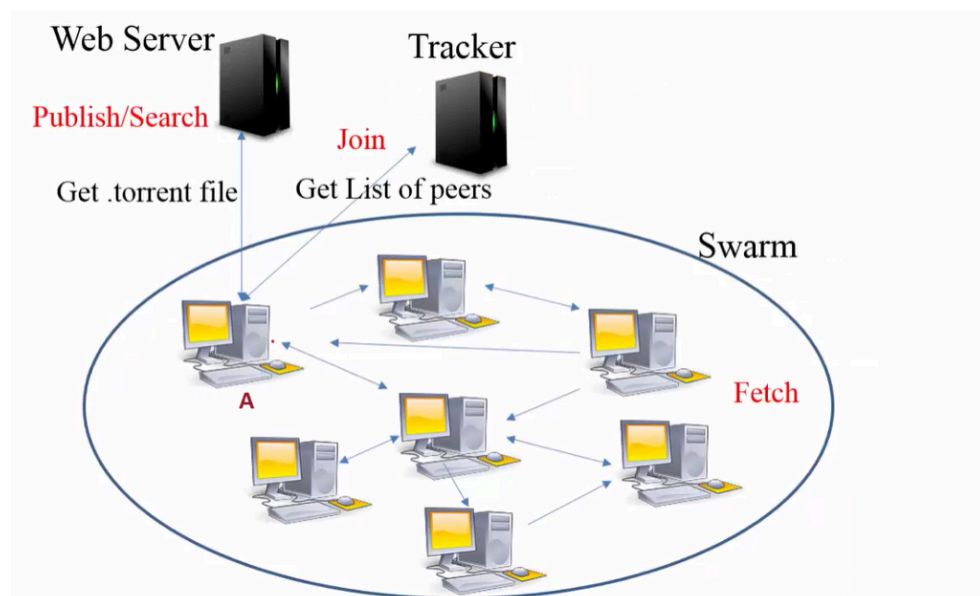
(Conception d'un système réparti pour échanger des fichiers)

Réponses au question :

1- Donner le principe de fonctionnement lors de la recherche d'une ressource et comment se fait le transfert (faite une recherche sur internet pour se documenter sur ces logiciels).

Torrent

Torrent est une méthode de partage de fichiers sur Internet utilisant le protocole BitTorrent, qui permet une distribution décentralisée et en pair à pair (P2P). Il est efficace pour le téléchargement des gros fichiers exemple : image ISO,...



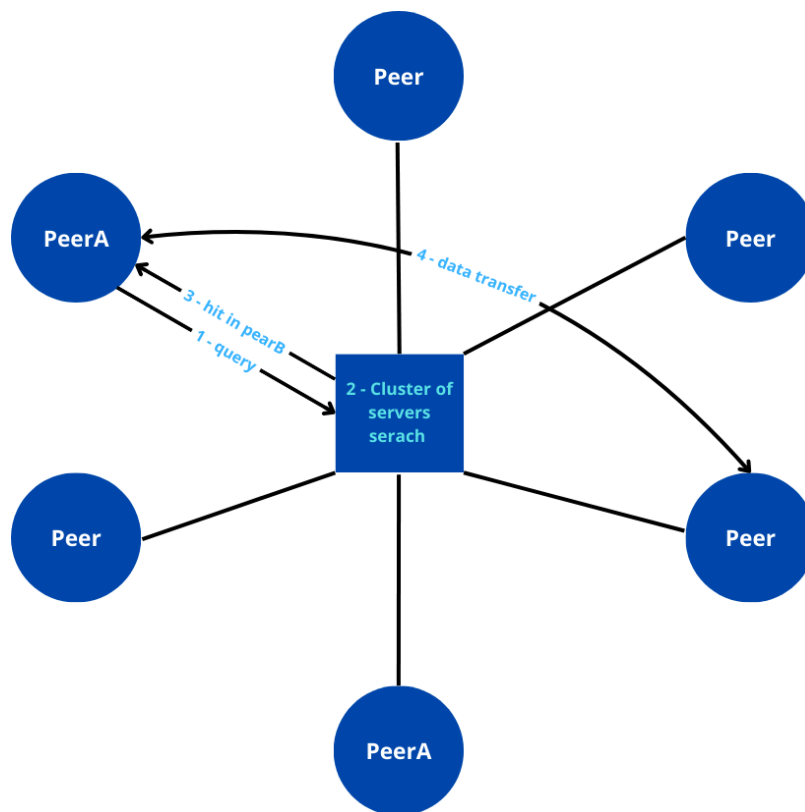
Fonctionnement :

- Soit **A** le premier utilisateur qui va télécharger un fichier torrent à partir d'un serveur web.
- Un fichier torrent contient des métadonnées sur les fichiers à partager (noms, tailles, etc.) et l'adresse du tracker.

- Ensuite, **A** va contacter le tracker avec son adresse ip (Join) ce qui va lui permet de recevoir la liste des pairs ayant le fichier/des parties de fichier souhaité, ce qui va impliqué aussi sa jointure dans ce réseau de pairs (swarm)
- **A** commence à télécharger des morceaux du fichier depuis plusieurs paires simultanément. Pendant ce temps, il partage aussi les parties qu'il a déjà téléchargées avec d'autres pairs.
- **A** deviendra ensuite lui-même un fournisseur pour ce fichier.

Napster

Voici une figure qui représente le fonctionnement du logiciel **Napster** ⇒ C'est une architecture semi-centralisée



Le système Napster se constituait de :

- Un cluster de serveurs qui maintiennent les adresses des clients ainsi que les fichiers qui ont partagé

Il se fonctionnait ainsi :

- Les peers se connectent au serveur, par exemple (PeerA), il envoie une requête contenant les mots clés/ le titre d'une musique(1.)
- Les serveurs du cluster coopèrent pour rechercher la musique (2.) et envoient la liste des musiques correspondantes à PeerA (3.).

- Ensuite, quand l'utilisateur de PeerA choisit la musique à télécharger, une connexion peer-to-peer entre PeerA et PeerB se fait pour transférer les données (4.)

2- Ecrire un algorithme distribué qui simule ces réseaux.

Algorithme qui simule le fonctionnement de Napster

Processus peeri

// Variables

peerIDi : int

adressIPIndex : string

// Initialisation

peerIDi = <lire Fichier Config>

adressIPIndex = <lire Fichier Config>

si(peerIDi == 0){ // c le serveur

 TC = crée table clients

 TR = crée table ressources

 démarrer une connexion TCP et se mettre en écoute

}sinon{ // c le client

 choisir action

 action 0 : envoyer rejoindre() à adressIPIndex

 action 1 : envoyer ajouter ressource(nom) a adressIPIndex

 action 2 : envoyer rechercher ressource(nom) a adressIPIndex

 }

}

// a la réception du message rejoindre à partir de peerIDj

ajouterClient(adressIPj) dans TC

// a la réception du message ajouter ressource à partir de peerIDj

ajouterRessource(nom, peerj) dans TR

// a la réception du message rechercher ressource à partir de peerIDj

owner = rechercherRessource(nom) dans TR

envoyer recevoirOwner(owner) a peerIDj

// a la réception du message recevoir owner a partir de index

envoyer demanderContenu(nom) a owner

// a la réception du message demander ressource à partir de peerIDj

envoyer contenuRessource(ressource) a peerIDj

// a la réception du contenu du fichier à partir de peerIDj

enregistrerRessource()

- L'initiateur c'est le serveur
- Le serveur doit toujours être active, alors l'algorithme se termine chez lui quand il le fait arrêter explicitement
- Les peers aussi doivent être à l'écoute, l'algorithme chez eux se termine lorsqu' il ferme leurs sessions explicitements

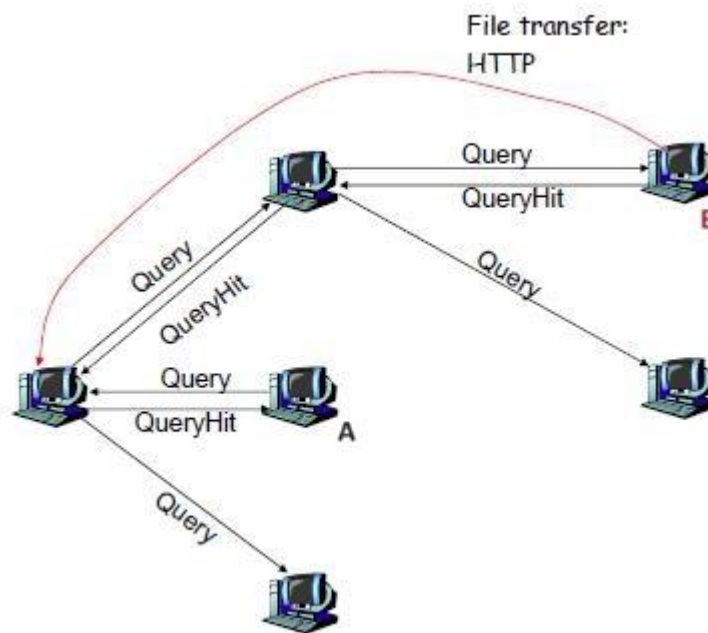
3- Quels sont les avantages et les inconvénients de cette organisation ?

AVANTAGES	INCONVÉNIENTS
efficacité du traitement des requêtes et faible surcharge	besoin d'entretien du serveur
pas de stockages des ressources dans le serveur	vulnérabilité (comme DDos attack)
	le serveur est un maillon faible, il a une seul entrée si ce point tombe en panne tout le réseau tombe
	problèmes de scalabilité

4- Donner une autre organisation des nœuds qui résout certains des problèmes rencontrés dans l'approche précédente.

Puisque les peers des architectures précédentes ont des fonctionnalités égales, alors on peut éliminer le serveur, et rendre le processus de recherche et délivrance de requêtes **complètement distribuée en peer-to-peer**

Il existe un protocole appelé **Gnutella** qui fonctionne avec ce principe, je vais l'expliquer comme exemple



Principe de fonctionnement : Exploration d'un réseau (flooding) avec sauvegarde du chemin

- Supposant que A recherche une ressource, il envoie une requête vers tous ses voisins
- Ensuite, si la ressource ne se trouve pas chez eux, il diffuse encore la requête vers leurs voisins et ainsi de suite jusqu'à arriver vers un nœud qui dispose de la ressource
- Supposant que B est celui qui dispose de la ressource, il envoie un message query hit qui contient (son @ip, port, speed) vers celui qui lui a envoyé la requête et ainsi de suite query hit va prendre la même route de la requête initiale jusqu'à arriver à A
- A va finalement contacter B directement par les informations encapsulées dans query hit, c'est-à-dire par son @ip:port
- B va lui répondre en lui envoyant le contenu de la ressource souhaitée
- Le paramètre speed est utilisé lorsque l'exploration de réseau indique qu'il y a plusieurs nœuds qui disposent de la même ressource, alors A va se trouver avec plusieurs choix, il va tout simplement choisir celui qui a le moindre speed

5- Écrire l'algorithme correspondant.

// Variables

idPeeri : int

voisinsi : liste des IDs des voisins de peer i

nbrVoisins : int

speed : coût du chemin

perei : id processus père de pi qui lui a envoyé le message rechercher ressource

cpt : compteur de nombre de peeri dans le chemin, il sert dans le retour pour savoir le processus initiateur

// Initialisation

voisinsi = <lire fichier de configuration>

nbrVoisins = |voisinsi|

speed = 0

cpt = 0

perei = 0

démarrer une
connexion TCP et
se mettre en
écoute

choisir option :

option 0 : envoyer requête rechercher ressource(nom,speed) à tous
les voisinsi

// a la réception du message rechercher ressource à partir de peerj

si(rechercher ressource != vrai){

perei = peerj

cpt = cpt + 1

speed = speed + 1

envoyer requête rechercher ressource (nom,speed,cpt) a tous voisinsi

}sinon{

speed = speed + 1

envoyer queryHit(@ip, port,speed,cpt) a peerj

}

// a la réception du message queryHit à partir de peerj

cpt = cpt - 1

si (cpt == 0) {

envoyer demande ressource au peer @ip:port qui a le moindre speed

}sinon{

forwarder queryHit(@ip, port,speed,cpt) a peerj

}

// a la réception du message demander ressource à partir de peerj

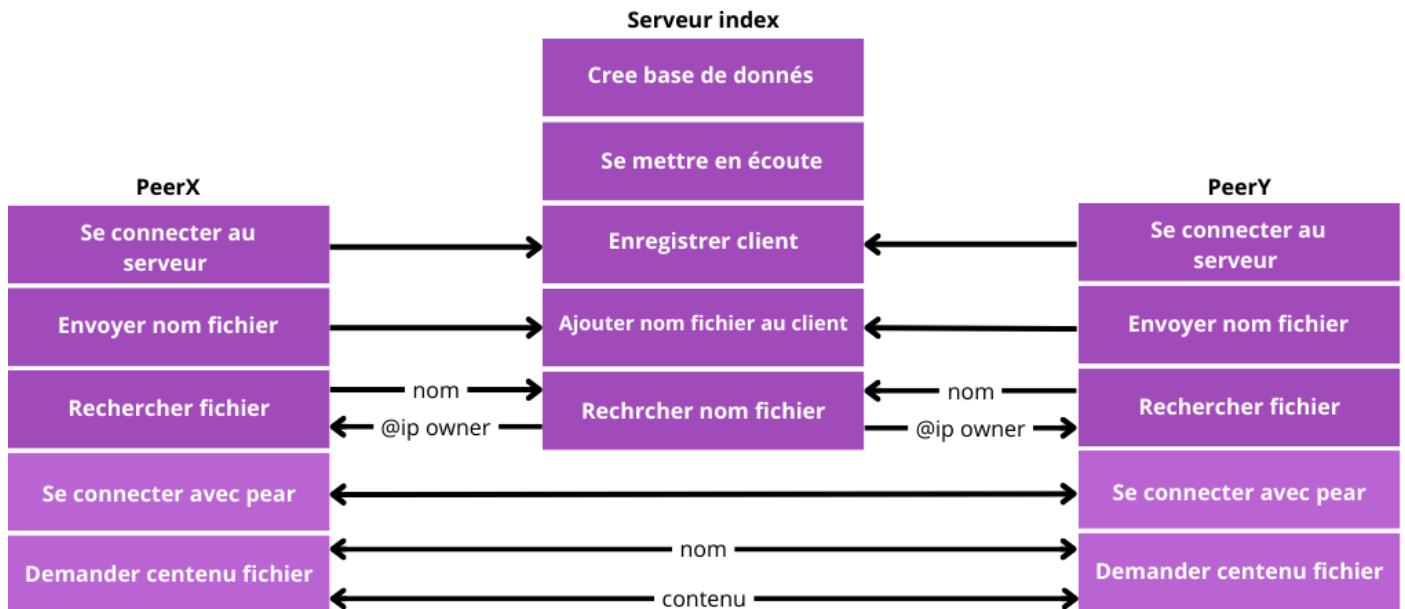
envoyer la ressource a peerj

// a la réception du contenu du fichier à partir de peerj

enregistrerRessource()

Implementation :

Architecture semi-centralisée : comme Napster



La **communication** se fait par envoi de messages

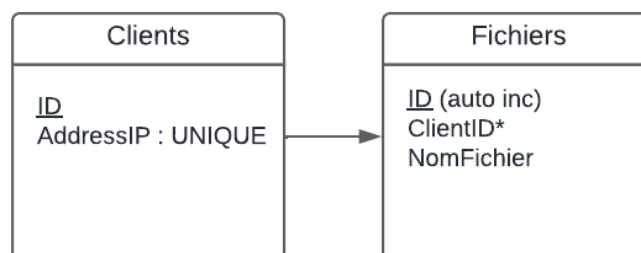
Serveur index:

Les trois services implémentée dans le serveur comme goroutine sont :

1. **Enregistrer_Client()** : permet de rajouter l@ip du client dans la table clients
2. **Ajouter_Nom_Fichier(NomFichier)** : permet d'ajouter le nom du fichier dans la table fichiers, en lui donnant l'ID du client qui l'a envoyé
3. **Rechercher_Nom_Fichier(NomFichier)** : permet de rechercher le nom du fichier dans la table fichiers et envoyer l@ip de son propriétaire si trouvé

Schéma de la bdd :

Un client peut être propriétaire d'un ou plusieurs fichiers



Peer (server|client):

Les peers vont se connecter au serveur index pour trois raisons :

1. S'enregistrer : ce qui va déclencher le service **Enregistrer_Client()**
2. Envoyer nom fichier : ce qui va déclencher le service **Ajouter_Nom_Fichier(NomFichier)**
3. Rechercher nom d'un fichier : ce qui va déclencher le service **Rechercher_Nom_Fichier(NomFichier)**

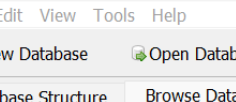
Les peers se connectent entre eux pour :

1. **Demander contenu d'un fichier** : il envoie le nom du fichier, il reçoit son contenu

Captures :

Joindre l'index

```
C:\Users\DELL\Desktop\M1\Algo Reparti\Projet\project - index\index>go run .
Serveur TCP démarré, en attente de connexions...
Client enregistré : ::1:8081
```



The screenshot shows the 'DB Browser for SQLite' application window. The title bar reads 'DB Browser for SQLite - C:\Users\...'. The menu bar includes 'File', 'Edit', 'View', 'Tools', and 'Help'. Below the menu bar, there are two buttons: 'New Database' and 'Open Database'. The main interface has two tabs: 'Database Structure' (selected) and 'Browse Data'. Under 'Database Structure', the 'Table:' dropdown is set to 'clients'. Below this, the table structure is displayed with two columns: 'id' and 'address'. Each column has a 'Filter' button. The data table shows one row with 'id' 1 and 'address' 15 ::1:8081.

	id	address
1	15	::1:8081

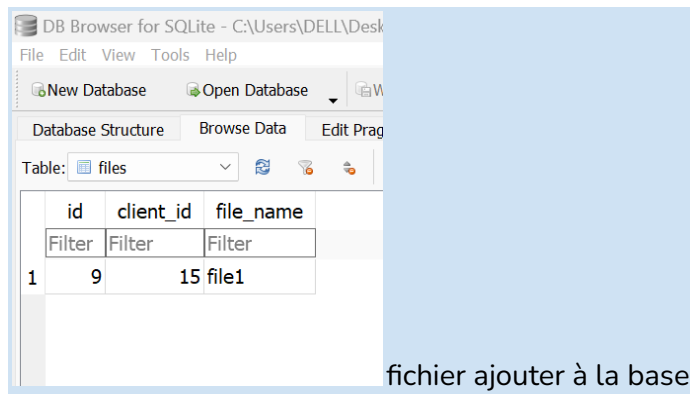
Publier un fichier

```
C:\Windows\System32\cmd.exe
C:\Users\DELL\Desktop\M1\Algo Reparti\Projet\project - index\index>go run .
Serveur TCP démarré, en attente de connexions...
Client enregistré : ::1:8081
File enregistré : file1

Windows PowerShell
PS C:\Users\DELL\Desktop\M1\Algo Reparti\Projet\project - peer1\peer1>go run .
Choose an option: [join | publish | search]
Peer server is listening on port 8081...
join
Server response: Client enregistré avec succès

Choose an option: [join | publish | search]
publish
Enter the file name to publish:
file1
Server response: File enregistré avec succès

Choose an option: [join | publish | search]
```

Chercher un fichier et recevoir son contenu

```
PS C:\Users\DELL\Desktop\M1\Algo Reparti\Projet\projet - peer2\p
eer> go run .
Choose an option: [join | publish | search]
Peer server is listening on port 8080...
join
Server response: Client enregistré avec succès

Choose an option: [join | publish | search]
search
Enter the file name to search:
file1
File found at: ::1:8081

File received successfully.
Choose an option: [join | publish | search]
```

