Protocoles et technologies des réseaux Peer-to-Peer

Moarraf Yassine

(Ce travail fait partie d'un projet de fin d'année impliquant des recherches sur les réseaux peer-to-peer dans le développement d'une application de messagerie peer-to-peer.)

Les réseaux Peer-To-Peer reposent sur des concepts fondamentaux et des principes qui méritent une attention particulière. Il est essentiel de procéder à une analyse approfondie des principaux protocoles Peer-To-Peer. Cette exploration permettra d'examiner en détail les différentes approches adoptées par ces protocoles pour établir un réseau Peer-to-Peer.

1.Les responsabilités d’un protocol Peer-to-Peer:

Un protocole p2p a comme responsabilité principale d’assurer et de réaliser les principes du paradigme P2P qui sont L'auto-organisation, la symétrie des rôles, le partage des ressources, la scalabilité, l'autonomie des pairs et la résilience.

L'auto-organisation signifie que les pairs coopèrent dans la formation et la maintenance du réseau, chaque pair utilisant un état local et des informations partielles sur le réseau. Au contraire du modèle client/serveur.

La symétrie des rôles c’est que Les pairs ont des rôles symétriques. Contrairement au modèle client-serveur, où les rôles des participants sont asymétriques, les pairs sont fonctionnellement égaux. Chaque pair peut stocker des objets au nom d'autres pairs, prendre en charge des requêtes et routages des messages.

Du point de vue de la scalabilité, les réseaux pair-à-pair sont très évolutifs. Plusieurs applications P2P fonctionnent aujourd'hui avec des millions de participants. Une dimension importante de l'évolutivité est la capacité à faire fonctionner le réseau P2P lorsque sa taille est multipliée par 100 ou plus. D'un point de vue quantitatif, l'évolutivité signifie que le réseau et les ressources utilisées à chaque pair présentent un taux de croissance en fonction de la taille de la superposition qui n'est pas linéaire. Un autre aspect essentiel de l'évolutivité est la dégradation progressive. Lorsque les limites de performance de la superposition sont atteintes, la qualité de service diminue progressivement.

Les pairs sont autonomes. Chaque pair détermine ses capacités en fonction de ses propres ressources. Chaque pair détermine également le moment où il rejoint le réseau, les demandes qu'il adresse au réseau et quand il la quitte. L'autonomie des pairs entraîne l'imprévisibilité des services offerts par le réseau. Un pair qui cherche un objet et ne le trouve pas peut ne pas être en mesure de déterminer si l'objet n'existe pas dans le réseau ou si le pair qui stocke l'objet a quitté la superposition. Les pairs peuvent agir pour limiter leur contribution en ressources à la superposition, par exemple en se déconnectant de la superposition lorsqu'ils ne l'utilisent pas. La redondance et les incitations font partie des techniques a etre realisee par le protocol pour contrer l'imprévisibilité.

Un réseau P2P fournit un pool de ressources partagées. Les ressources qu'un pair contribue comprennent les cycles de calcul, le stockage sur disque et la bande passante réseau. Il existe un seuil minimum de contribution en ressources pour qu'un pair puisse rejoindre le réseau superposé P2P. Les ressources de chaque pair sont utilisées pour soutenir le fonctionnement du réseau et fournir des services d'application à d'autres pairs. La contribution des ressources doit être équitable. Un critère équitable de partage des ressources pourrait être que la contribution en ressources d'un pair ne dépasse jamais une certaine limite. Un autre critère pourrait être que la contribution moyenne en ressources de tout pair doit se situer dans une limite statistique de la moyenne globale du système P2P. De tels critères doivent être définis par le protocole en cours d'utilisation.

Les réseaux pair-à-pair doivent être résilients face à l'évolution dynamique des connexions et déconnexions des paires du réseau, phénomène appelé "churn". Un protocole P2P doit garantir cette résilience, c'est-à-dire maintenir le réseau face à ces changements.

Ces responsabilités ne sont généralement pas toutes satisfaites. Le but de les citer est de décrire un objectif quelque peu idéal qu'un protocole P2P peut atteindre, mais en réalité, les systèmes P2P peuvent relâcher une ou deux de ces contraintes. C'est le cas des réseaux P2P hybrides, par exemple. Certains systèmes utilisent des serveurs centraux pour authentifier les pairs ; une fois que les pairs sont authentifiés, le réseau lui-même fonctionne sans le serveur central. L'amélioration de la conception des protocoles pair-à-pair est un domaine de recherche actif.

2.Les propriétés et fonctionnalités des protocoles p2p:

Rappelant d'abord par la définition d'un protocol dans le context des réseau informatique:

"Un protocole définit la structure et l'ordre des messages échangés entre deux entités ou plus en communication, ainsi que les actions prises lors de la transmission et/ou de la réception d'un message ou d'un autre événement." (Kurose and Ross 2017, 9)

Donc un protocole d'une application pair-à-pair (P2P) se compose de différents types de messages et de leur sémantique, qui sont compris par tous les pairs. Au but d'établir le réseau p2p.

Souvent les protocoles p2p sont construits au niveau de la couche application de la pile de protocoles réseau ce qui crée une sorte d'un réseau superposé ce qui donne une sorte d'abstraction sur la complexité du réseau sous-jacent qui connecte les nœuds.

En plus, dans la plupart des conceptions, les pairs disposent d'un identifiant unique, qui est l'ID du pair ou l'adresse du pair. En plus, le protocole prend en charge une sorte de capacité de routage des messages. En utilisant plusieurs méthodes (flooding, random walk ….).

On trouve également que de nombreux types de messages définis dans divers protocoles P2P sont similaires. Pour citer des exemples de messages souvent implémentés dans les protocoles, on peut mentionner les messages de base utilisés par le protocole Gnutella v0.6.

Table 6.1 Basic Message Types for Gnutella v.0.6

Message Type	Meaning
`Ping`	Discover other hosts that are in the Gnutella network and basic information about connecting to them. If TTL=1 and Hops=0 or 1, treat the request as a direct probe of the receiving host. If TTL=2 and Hops=0, treat the request as a "crawler" ping that is collecting information about the neighbors of this host.
`Pong`	Reply to a ping. Provides the IP address and part number of the host and extensions supported by the peer. It may include pong responses cached from other peers. Cached entries are peers that are likely to be alive and are spread across the network; for example, by varied connections and hop-count values, these pongs are cached.
`Query`	Search for a file. Specifies the minimum transfer speed of the peer and the search criteria. The search criteria is text, such as a string of keywords. Search criteria of " " means return an index of all files shared by the peer. A peer forwards incoming queries to all its connected peers.
`QueryHit`	Response to a query. A peer returns query hit responses to previously forwarded queries back along the connection from which the query was received. Contains the number of hits in the result set and, for each hit, a list of [*file index, file size, file name, and list of extensions*].
`Push`	Download a request for firewalled peers.
`Bye`	Tell the remote host that the connection is being closed.

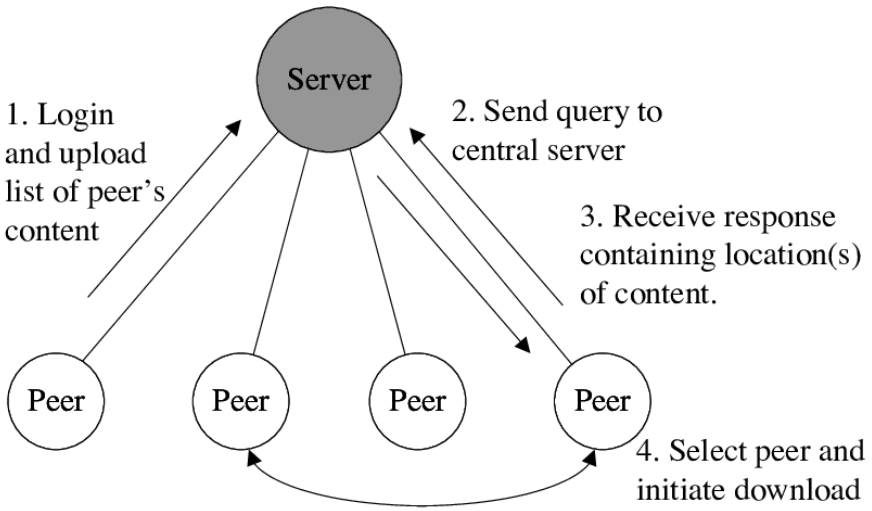
(Koegel Buford et al., 2009, 141)

Dans ce qui suit, nous allons aborder notre analyse des protocoles existants, en expliquant les mécanismes de recherche, de découverte de pairs, de transfert de fichiers et de routage utilisés dans ces protocoles, selon le plan suivant : nous commencerons par les protocoles utilisés dans les réseaux P2P non structurés, puis nous nous concentrerons sur les réseaux P2P structurés. Ensuite, nous nous focaliserons sur l'aspect du partage des données et les défis qui y sont associés.

3.Vue sur les protocoles P2P Actuels:

3.1 Napster:

Créé en 1999 comme une plateforme de partage de fichiers musicaux, Napster a connu une notoriété phénoménale. Malgré l'utilisation d'une architecture hybride. Bien qu'il n'adopte pas une approche P2P pure, Napster a été largement reconnu comme le premier système de partage de fichiers de son genre et aussi en un laps de temps relativement court.
Dans le système Napster, les utilisateurs partagent des fichiers MP3 stockés localement sur leurs disques durs. Les informations textuelles des fichiers, telles que les titres des chansons, étaient générées, indexées et stockées par le serveur Napster. Chaque utilisateur du réseau Napster utilisait le logiciel client Napster pour se connecter au serveur centralisé. Les utilisateurs connectés au serveur Napster pouvaient effectuer des recherches basées sur des mots-clés pour trouver des fichiers audio spécifiques. Le serveur renvoie alors une liste de fichiers correspondants, accompagnée de leur description et de leur emplacement, à l'utilisateur. Ce dernier tentait ensuite de se connecter à l'utilisateur possédant le fichier audio souhaité et de transférer le contenu cible de manière pair-à-pair (P2P).



(An Empirical Analysis of Network Externalities in Peer-To-Peer Music Sharing Networks)

` 3.2 Gnutella:

Gnutella a été le premier protocol qui a utilisé une architecture pair-à-pair (P2P) à 100% et est resté l'un des systèmes les plus populaires à ce jour. Les premières versions du protocole Gnutella, jusqu'à la version 0.4, utilisaient une approche non structurée avec diffusion des requêtes (query flooding). Cependant, après que la scalabilité soit devenue un problème de performance évident, la version la plus récente du protocole Gnutella (version 0.6) a adopté une architecture de super-pair dans laquelle les pairs à haute capacité sont des super-pairs et toutes les requêtes sont routées, en utilisant un mécanisme de diffusion, entre les super-pairs.

L'un des type importante des message qu'un protocol doit release c'est le recherche (query) d'informations spécifiques., l'approche utilise dans la version 0.4 de Gnutella était la diffusion (flooding), dans ce qui suit on vas essayer d'expliquer cette technique

Flooding:

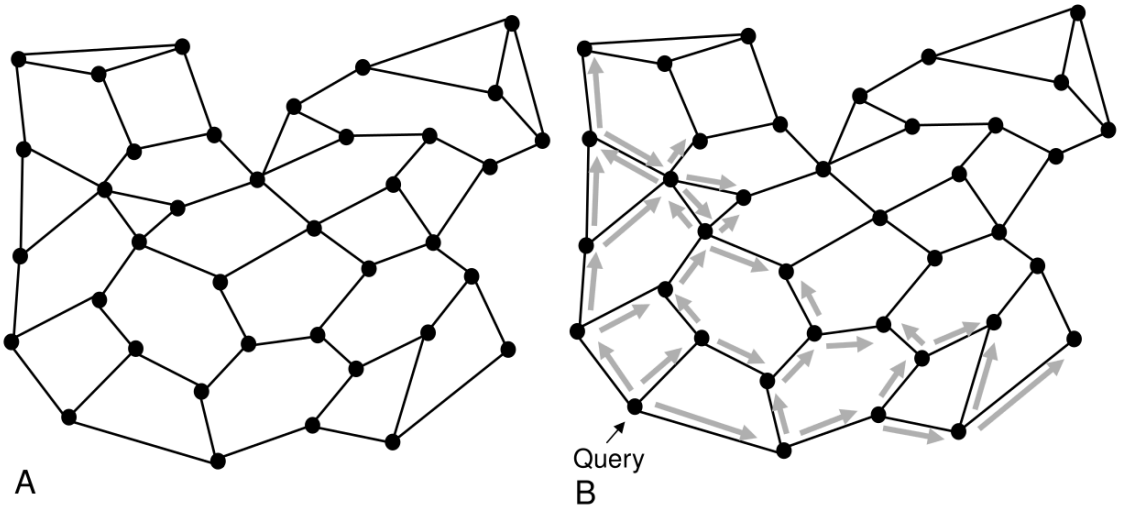


FIGURE 3.1 (A) Unstructured topology showing connections between peers and (B) query flooding to four hops.

(Koegel Buford et al. 47)

Dans une topologie non structurée, considérons un nœud B dont la visibilité dans le réseau P2P se limite uniquement à ses voisins (les nœuds directement connectés à B). À présent, B souhaite obtenir des informations identifiées par une clé K, telles que ces informations existent dans un autre nœud du réseau dont B n'a pas connaissance. L'approche du flooding au but de savoir quelle noeuds possède l'information, consiste à d'essayer d'envoyer une requête à chaque voisin que B connaît. Si les pairs voisins n'ont pas les informations, ils peuvent à leur tour transmettre la demande à leurs voisins, et ainsi de suite.

Avec ce mécanisme on peut voir la nécessité d'utiliser aussi des techniques pour éviter que les messages ne circulent indéfiniment dans le réseau,Tout d'abord, en cas de boucle de messages ou de sa réception par plusieurs chemins, chaque pair peut maintenir une liste des identifiants des messages déjà reçus. S'il reçoit à nouveau le même message, il le rejette simplement comme étant un doublon. Deuxièmement, afin d'éviter que les pairs ne conservent les messages indéfiniment, ce qui pourrait entraîner une surcharge de stockage, chaque message est assorti d'une valeur de durée de vie (TTL) qui limite sa persistance. La valeur TTL d'un message est définie par son expéditeur et est réduite de 1 à chaque étape de transmission. Lorsque la valeur TTL d'un message atteint 0, il cesse d'être transmis

Tout cela se traduit en pseudo-code par :

```
FloodForward(Query q, Source p)
// have we seen this query before?
```

```
if(q.id in oldIdsQ) return // yes, drop it
oldIdsQ = oldIdsQ union q.id // remember this query
// expiration time reached?
q.TTL = q.TTL - 1
if q.TTL <= 0 then return // yes, drop it
// no, forward it to remaining neighbors
foreach(s in Neighbors) if(s != p) send(s,q).
```

La requête est satisfaite quand le nœud qui reçoit l'information la reçoit, dans ce cas cette nœud utilise l'adresse d'émetteur original du requete pour lui transmettre l'information.

Notant que même si la requête est satisfaite, la propagation du message peut encore continuer dans le réseau jusqu'à que la valeur de TTL de tous ces messages devient 0. Ce qui présente une utilisation un peu excessive des ressources du réseau. Une façon d'éviter ça, est de commencer la recherche avec une petite valeur de TTL, En cas de succès, la recherche s'arrête. Dans le cas contraire, la valeur TTL est augmentée d'une petite valeur et la requête est réémise. Cette variante du Flooding est nommée 'iterative deepening' ou 'expanding ring'.

3.3 FastTrack:

Un autre protocol P2P utilisé dans les topology non structuré, c'est le FastTrack, apparue au même temps que Gnutella a été utilisé par un nombre des programmes de partage des fichiers.
FastTrack est un protocole propriétaire qui inclut des mécanisme de cryptage, mais Un projet nommé giFT qui essaye a analysé ce protocole en 2003 en utilisant un client modifier avec un outil de décryptage du protocole et un packet sniffer. Cette étude a démontré que FastTrack utilise une architecture de super-pairs dans laquelle les pairs à haute capacité sont des super-nœuds (SN) et les pairs à basse capacité sont des nœuds ordinaires (ON: ordinary node).

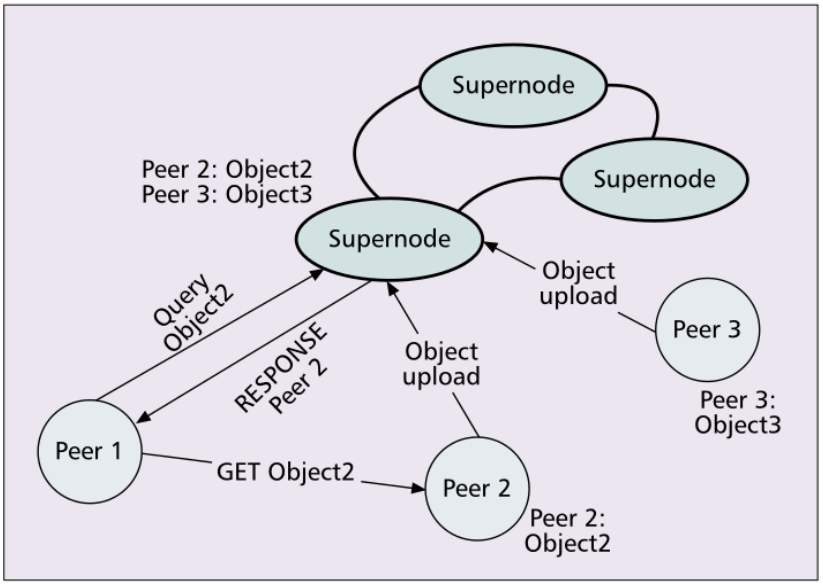


Figure 9. *FastTrack peers connect to Superpeers whereby the search is routed through the Superpeers and downloads are done from the peer peers.*

Dans un réseau comptant environ 3 millions de nœuds, le nombre de nœuds (SN) se situe entre 25 000 à 40 000. Chaque nœud (ON) maintient une connexion avec un nœud (SN). Le super-nœuds fournit à son client (ON) une liste d'autres super-nœuds, que l'ON met en cache. Après avoir émis une requête au SN et reçu ses réponses, l'ON se déconnecte du SN actuel et se reconnecte à un nouveau SN de sa liste. Lors des reconnexions à de nouveaux SN, il reçoit également une nouvelle liste de SN qu'il fusionne avec la liste existante.
Un nœud (SN) maintient environ 40 à 50 connexions avec d'autres SN. Un SN sur une connexion résidentielle à large bande maintient des connexions avec environ 50 à 80 ON. Les SN disposant d'une capacité réseau plus importante maintiennent des connexions avec environ 100 à 160 ON à tout moment donné.
La durée de vie moyenne d'une connexion SN-SN est de 34 minutes et celle d'une connexion SN-ON est de 11 minutes. Environ 33 % des connexions durent moins de 30 secondes.

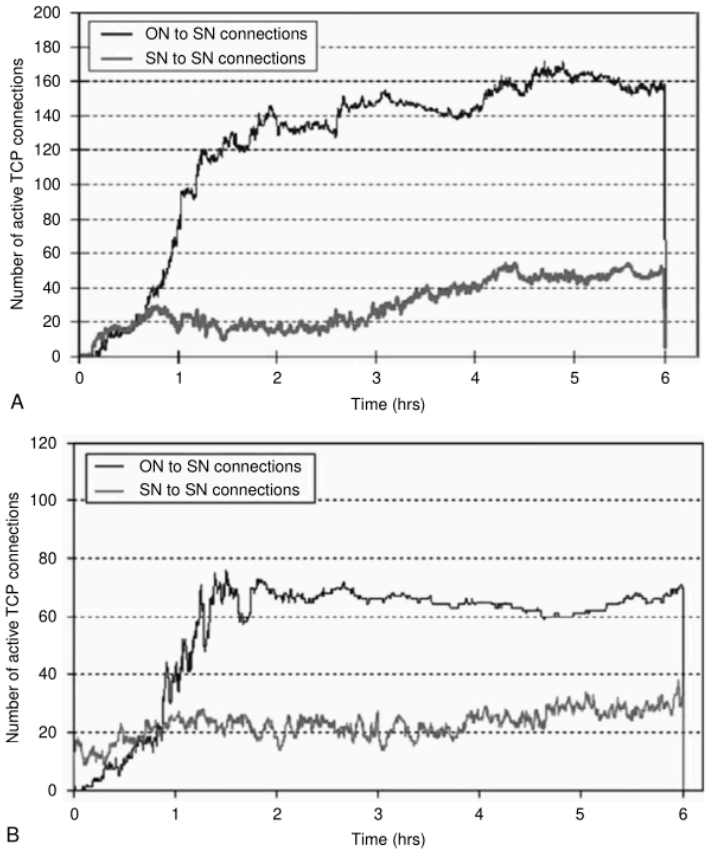


FIGURE 3.7 FastTrack supernode (SN) and ordinary node (CN) connectivity: (A) an SN located on a university campus network and (B) an SN located on a residential broadband connection.

Les dynamiques des connexions SN-SN et SN-ON semblent avoir plusieurs objectifs, notamment la répartition de la charge (load distribution) entre les SN, l'amélioration de la localité des connexions et le mélange des connexions par les ON, afin d'augmenter la couverture des recherches sur le réseau. Une entropie de connexion élevée rend également le suivi des transferts entre pairs plus difficile.

3.4 Freenet (Actuellement Hyphanet):

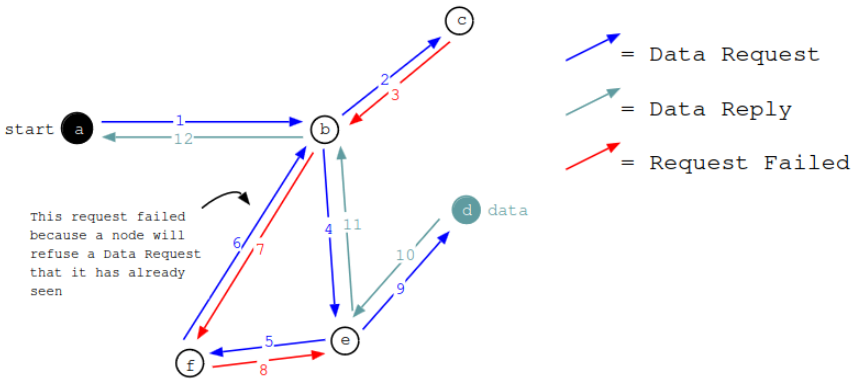
Freenet a été proposé par Ian Clarke en 1999 comme un mécanisme de partage de fichiers pair à pair distribué offrant sécurité, anonymat et réfutabilité. Chaque fichier est nommé par des clés indépendantes de son emplacement. Chaque nœud maintient sa propre DataStore locale qu'il rend disponible au réseau pour la lecture et l'écriture, ainsi qu'une table de routage dynamique contenant les adresses d'autres nœuds et les clés qu'ils sont censés détenir. Il est prévu que la plupart des utilisateurs du système exécutent des nœuds, à la fois pour fournir des garanties de sécurité contre l'utilisation involontaire d'un nœud étranger hostile et pour augmenter la capacité de stockage disponible pour l'ensemble du réseau. Les nœuds et les fichiers partagent le même espace de clés, qui les identifie. Cette clé est générée à partir du hachage en utilisant l'algorithme SHA-1. Les clés qui identifient les pairs sont appelées "clés de routage".

Pour les opérations de recherche et récupérer des fichier, Freenet cette fois ci un routage entre les nœuds qui est basé sur la distance entre les clés. Les requêtes de ces opérations

sont transmises aux pairs dont la clé de routage correspond et la clé qui identifie le fichier son les plus proche. Si une requête échoue, le pair essaiera la clé de routage suivante la plus proche au clé du fichier d'après sa table de routage. En plus pour combattre ce cas (requête échoue) Freenet intègre des mécanisme pour faire du cache des fichiers le long du chemin du retoure du requete. Pour les celles soit d'insertion ou récupération en utilisant un cache LRU.

Ici la distance entre les deux hash est une distance lexicographique et par conséquent il n'existe pas de notion de proximité sémantique lorsqu'il s'agit de la proximité des clés. donc, il n'y aura aucune corrélation entre la proximité des clés et la popularité similaire des données,

Un exemple de séquencement d'une requêtes



3.5 Gia:

Gia est un protocole peer to peer a buté de construire un réseau non structurée, Il a amélioré la scalabilité des conceptions utilisant la diffusion (flooding) ou la marche aléatoire(random walk), leurs créateur la considère comme Gnutella mais en gardant à l'esprit la scalabilité. Un aspect clé de Gia est sa capacité à distinguer les nœuds en fonction de leur capacité et à distribuer la charge à travers le réseau en fonction de la capacité de chaque nœud. En particulier,le degré de connexion et le taux de requêtes autorisées sont régulés selon la capacité disponible du nœud. La capacité à reconnaître et à s'adapter à l'hétérogénéité des nœuds est un problème pratique car les grands réseaux p2p auront une quantité significative de bande passante des nœuds, de la CPU et de la capacité de stockage. Cela signifie également que les nœuds de faible capacité ne seront pas surchargés.

Les améliorations de conception de Gia peuvent être divisées en quatre composants suivants :

- Adaptation dynamique de la topologie: Gia forme un réseau dans lequel les nœuds centraux, ceux ayant un degré de connectivité élevé avec d'autres nœuds, sont les

nœuds de haute capacité. Le mécanisme est adaptatif en ce sens que les nœuds recherchent activement plus de voisins pour satisfaire leur propre niveau de capacité.

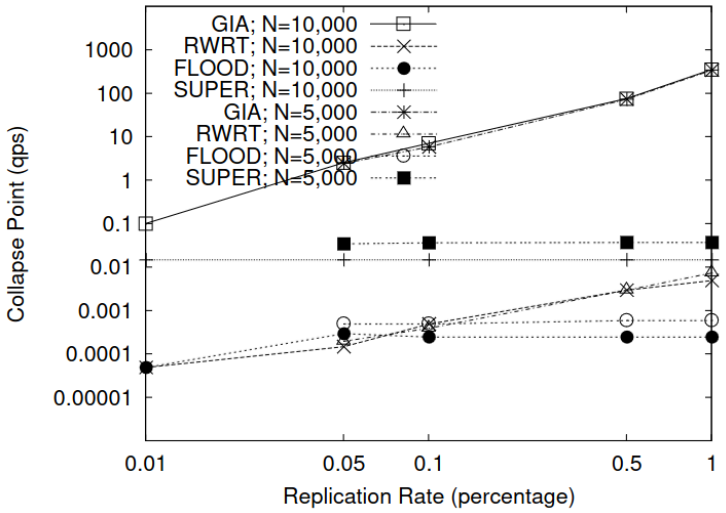
- Contrôle actif du flux: Pour éviter que les nœuds ne soient surchargés de messages, les nœuds fournissent des jetons à leurs voisins pour réguler le débit des messages, donc, un nœud ne peut envoyer un message à son voisin que lorsqu'il a reçu un jeton de ce voisin. Les jetons sont attribués aux voisins proportionnellement à leur capacité.
- One-hop index réplication: Les nœuds envoient une copie de la table des objets qu'ils stockent à leurs voisins. Lorsqu'un nœud reçoit un message de requête, il recherche à la fois dans son index local et dans les copies fournies par les voisins pour voir s'il y a une correspondance.
- Biased random walk search protocol: En conséquence des trois améliorations précédentes, les nœuds de haute capacité ont probablement un degré de connectivité élevé et possèdent les informations d'index pour le plus grand nombre de pairs. De plus, ces nœuds sont capables de gérer davantage de trafic de requêtes. Un pair qui transmet une requête la transfère de préférence à un voisin de haute capacité.
-

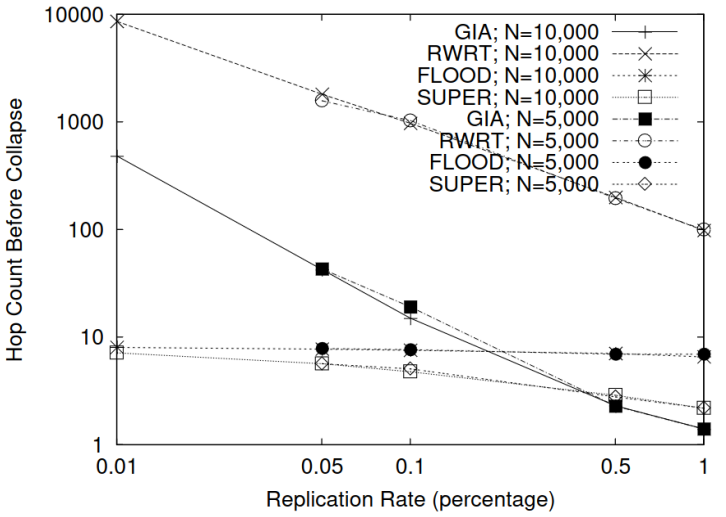
-Comparaison de GIA avec RWRT, SUPER et FLOOD:

L'article ou GIA est présente compare son comportement avec la marche aléatoire sur une topologie aléatoire (RWRT), le flooding (FLOOD) et une architecture de superpair avec des facteurs de réplication variables et différentes tailles de système jusqu'à 10 000 nœuds. On note que dans l'architecture de superpair, les pairs de haute capacité forment le "backbone" du réseau et les autres pairs ne se connectent qu'aux superpairs. Une requête d'un pair est envoyée à son superpair, qui propage la requête à d'autres superpairs en utilisant le flooding.

Pour cette comparaison, les métriques suivantes sont définies :

- Collapse Point (CP) : Le point au-delà duquel le taux de succès passe en dessous de 90%. Cette métrique reflète la capacité totale du système.
- Hop-count before collapse (CP-HC) : le nombre moyen de sauts avant Collapse Point.
- Taux de succès : mesuré comme la fraction de requêtes émises qui parviennent à localiser les fichiers désirés.
- Nombre de sauts : mesuré comme le nombre de sauts nécessaires pour localiser les fichiers demandés.





Le CP et le CP-HC sont mesurés en augmentant les facteurs de réplication. Dans les graphiques ci-dessus, nous avons un graphique des résultats pour des systèmes avec 5 000 et 10 000 nœuds. Pour un système de 10 000 nœuds, nous avons une simulation jusqu'à une réplication de 0,01 %, car cela correspond à une seule réponse correspondante dans l'ensemble du système pour n'importe quelle requête.

La comparaison montre que GIA a un point d'effondrement beaucoup plus élevé que les autres techniques, ce qui signifie qu'il a une plus grande capacité réseau. En ce qui concerne le nombre de sauts, GIA nous conduit à avoir moins de sauts avant l'effondrement (Collapse Point) que RWRT et FLOOD. Cependant, SUPER a le nombre de sauts le plus faible parmi eux. Il présente un nombre de sauts comparable à GIA lorsque le taux de réplication dépasse 0,5 %.

3.6 Chord:

Chord est un protocole (DHT: Distributed hash table) qui utilise le hachage cohérent (consistent hashing) pour attribuer des clés à ses pairs .La fonction de hachage cohérent attribuent aux pairs et aux clés de données un identifiant de m bits en utilisant SHA-1

comme fonction de hachage de base. L'identifiant d'un pair est choisi en hachant l'adresse IP du pair, tandis qu'un identifiant de clé est produit en hachant la clé de données. Donc les pairs et les données partagent le même espace d'adressage.
Le hachage cohérent garantit que les pairs peuvent entrer et sortir du réseau avec une perturbation minimale, ce qui contribue à la nature décentralisée du système. Cette conception vise à équilibrer la charge sur le système, chaque pair gérant approximativement le même nombre de clés, et il y a peu de mouvements de clés lorsque les pairs rejoignent ou quittent le réseau.

Le protocole Chord établit des règles et des procédures visant à attribuer chaque clé à un pair, ce que nous discuterons par la suite.

L'ensemble des identifiants des pairs est ordonné sur un cercle identité en modulo m, appelé "Chord Ring" (voir la figure). Chord impose qu'une pièce de données de clé K est attribuée au premier pair dont l'identifiant est égal ou suit K dans l'espace des identifiants. Ce pair est désigné par "successor(K)". Le pair qui est "successor(K)" est le pair qui suit immédiatement K sur le Chord ring dans le sens horaire.
Par exemple dans la figure: L'objet de clé 10 est attribué au neux avec identifiant 14 donc: N14 = successor(10)
Pour maintenir la cohérence des hachages (consistente hashing) lorsqu'un **pair n** rejoint le réseau, certaines clés précédemment attribuées au successeur de **n** doivent maintenant être réattribuées à **n**. Lorsque le pair **n** quitte le système Chord, toutes ses clés attribuées sont réassignées au successeur de n.
Chaque pair dans Chord ring doit savoir comment contacter son pair successeur actuel sur le cercle des identifiants. Les requêtes de recherche utilisent la correspondance de la clé et de l'identifiant du nœud. Pour un identifiant donné, il peut être transmis autour du cercle via ces pointeurs de successeur jusqu'à ce qu'il rencontre une paire de pairs qui inclut l'identifiant souhaité ce dernier de la paire est le pair auquel la requête est associée. Un exemple d'une opération de recherche est dans la figure.
Avec m est le nombre de bits d'un identifiant, chaque pair n maintient une table de routage avec jusqu'à m entrées, appelée "The finger table". La ième entrée dans la table du pair n contient l'identité du premier pair s qui succède à n d'au moins 2^i - 1.
En autre terme la ligne i du table de routage contient l'identité de s = successor(n + 2^(i-1))
Toujours 1 <= i <= m car le nombre des lignes dans la table de routage ne dépasse par m.
Chaque ligne du table de routage contient le couple identifiant, adresse IP, un exemple de table de routage (finger table) est présente dans la figure.
Cette structure de routage admet deux caractéristique importantes
Premièrement, chaque nœud stocke des informations sur seulement un petit nombre d'autres nœuds et en sait plus sur les nœuds qui le suivent de près sur le cercle des identifiants que sur les nœuds plus éloignés. La deuxième ce que la table de routage d'un nœud ne contient généralement pas suffisamment d'informations pour déterminer directement le successeur d'une clé arbitraire K. Par exemple, le nœud 8 dans la Figure ne peut pas déterminer le successeur de la clé 34 par lui-même, car ce successeur n'apparaît pas dans la table des doigts du nœud 8.
Dans un état stable avec N pairs dans le système, chaque pair maintient des informations d'état de routage pour environ O(log N) autres pairs. Cela signifie que chaque pair n'a besoin de suivre qu'un nombre logarithmique d'autres pairs pour un routage efficace. Bien que cela puisse être efficace, le protocole garantit également que les performances se

dégradent de manière gracieuse même lorsque les informations d'état de routage deviennent obsolètes. La dégradation gracieuse des performances dans Chord signifie que même si les informations d'état de routage ne sont pas entièrement à jour, le système peut encore fonctionner raisonnablement bien. Cette propriété est essentielle pour la robustesse dans des environnements dynamiques où les pairs rejoignent ou quittent fréquemment le réseau.

Pour trouver le successeur d'une clé arbitraire K Chord utilise la procédure récursive suivante:

```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```

Fig. 5. Scalable key lookup using the finger table.

Si l'identifiant tombe entre n et son successeur *id ∈]n, successor],* la recherche du successeur est terminée et le nœud n renvoie son successeur. Sinon, on cherche dans sa table de routage le nœud' dont l'identifiant précède immédiatement *id*, puis invoque la recherche du successeur de n'. Le choix de n' repose sur le fait que plus n' est proche de *id*, plus il en saura sur le cercle des identifiants dans la région de *id*.

À titre d'exemple, considérons le cercle Chord dans la Figure, et supposons que le nœud 8 souhaite trouver le successeur de la clé 54. Étant donné que le plus grand identifiant dans la table de routage du nœud 8 qui précède 54 est le nœud 42, le nœud 8 demandera au nœud 42 de résoudre la requête. À son tour, le nœud 42 déterminera le plus grand le plus grand identifiant dans sa table de routage qui précède 54, c'est-à-dire le nœud 51. Enfin, le nœud 51 découvrira que son propre successeur, le nœud 56, succède à la clé 54, et renverra donc le nœud 56 au nœud 8.

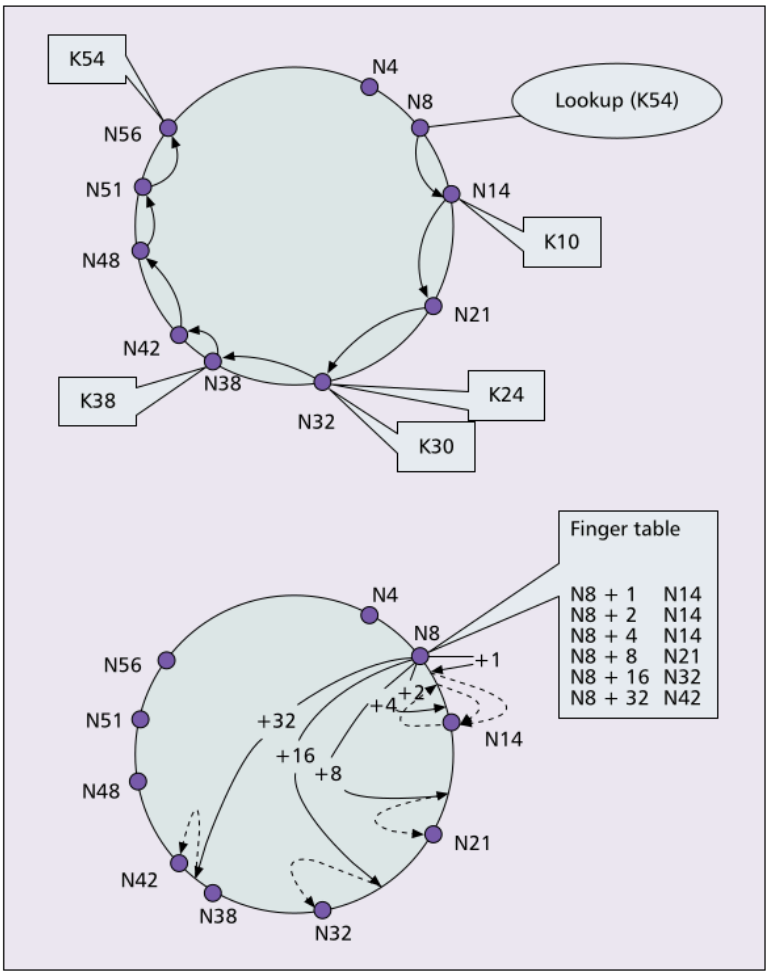


Figure 4. *Chord ring with identifier circle consisting of ten peers and five data keys. It shows the path followed by a query originated at Peer 8 for the lookup of key 54. Finger table entries for Peer 8.*

Le protocole Chord utilise un protocole de stabilisation qui s'exécute périodiquement en arrière-plan pour mettre à jour les pointeurs de successeur et les entrées dans la table des routages. La correction du protocole Chord repose sur le fait que chaque pair est conscient de ses successeurs. Lorsque des pairs échouent, il est possible qu'un pair ne connaisse pas son nouveau successeur et qu'il n'ait aucune chance de l'apprendre. Pour éviter cette situation, les pairs maintiennent une liste de successeurs de taille r, qui contient les r premiers successeurs du pair. Lorsque le pair successeur ne répond pas, le pair contacte simplement le pair suivant sur sa liste de successeurs. En supposant que les défaillances de pair surviennent avec une probabilité p, la probabilité que chaque pair de la liste de successeurs échoue est p^r. Augmenter rend le système plus robuste. En ajustant ce paramètre, n'importe quel degré de robustesse avec une bonne fiabilité et une résilience aux pannes peut être atteint.

3.7 Kademlia:

Kademlia est un protocole de table de hachage distribuée (DHT) pair-à-pair. Kademlia présente plusieurs caractéristiques désirables qui ne sont pas offertes simultanément par les autres DHT.

- Il minimise le nombre de messages de configuration que les nœuds doivent échanger pour apprendre les uns des autres.
- Les informations de configuration se propagent automatiquement comme un effet secondaire des opérations de recherche de clés.
- Les nœuds disposent d'un niveau de connaissance et de flexibilité suffisant pour acheminer les requêtes par des chemins à faible latence.
- Kademlia utilise des requêtes parallèles et asynchrones afin d'éviter les retards d'expiration causés par les défaillances de nœuds.
- L'algorithme par lequel les nœuds enregistrent l'existence des autres résiste à certaines attaques de déni de service de base.

Dans Kademlia l’espace des identifiants est l’espace des valeurs en 160-bit, chaque nœud qui participe dans le réseau possède un identifiant dans cette espace, de même, les clés des paires (clé, valeur) appartiennent au même espace.
Suivant la même approche des autres protocoles DHT, une clé est stockée dans le nœud avec ID proche de l' identifiant du clé. La notion de distance entre identifiants dans Kademlia est basée sur l'opération XOR.

D'un point de vue global sur un réseau p2p géré par Kademlia, les nœuds identifiés par leur ID de 160-bit sont représentés sur une arbre binaire.

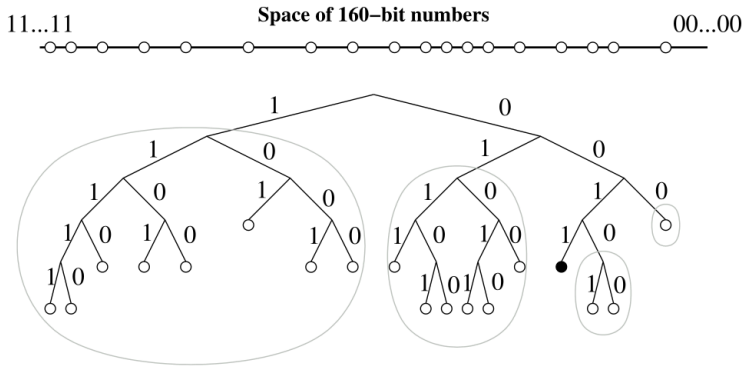


Fig. 1: Kademlia binary tree. The black dot shows the location of node $0011\cdots$ in the tree. Gray ovals show subtrees in which node $0011\cdots$ must have a contact.

Pour un noeuds donne cette arbre binaire peut être divisé en sous arbre de la façon suivante:
nous divisons l'arbre binaire en une série de sous-arbres successivement plus petits qui ne contiennent pas le nœud. Le sous-arbre le plus élevé se compose de la moitié de l'arbre binaire ne contenant pas le nœud. Le sous-arbre suivant se compose de la moitié de l'arbre restant ne contenant pas le nœud, et ainsi de suite. Dans l'exemple du nœud 0011, les sous-arbres sont entourés et se composent de tous les nœuds avec les préfixes 1, 01, 000 et 0010 respectivement.

Le protocole Kademlia garantit que chaque nœud connaît au moins un nœud dans chacun de ses sous-arbres, si ce sous-arbre contient un nœud. Avec cette garantie, n'importe quel nœud peut localiser n'importe quel autre nœud par son identifiant.

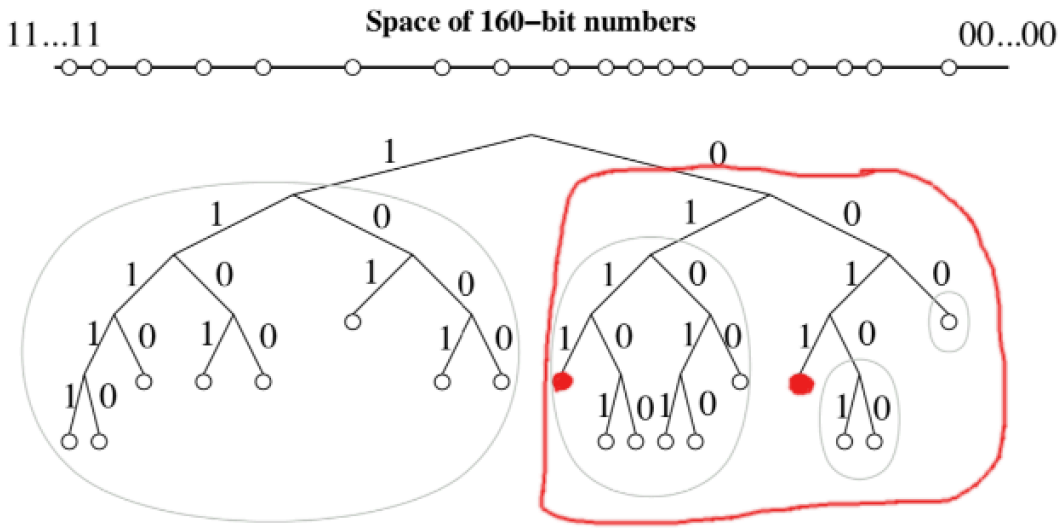
La métrique XOR:

Pour attribuer des paires <clé,valeur> à des nœuds particuliers, Kademlia s'appuie sur une notion de distance entre deux identifiants. Étant donnés deux identifiants de 160 bits, x et y, Kademlia définit la distance entre eux comme leur ou exclusif (XOR) bit à bit, interprété comme un entier, $d(x, y) = x \oplus y$.
Nous remarquons que la métrique XOR présente les propriétés suivantes :
C'est une métrique valide, bien que non euclidienne, avec :

- $d(x, x) = 0$
- $d(x, y) > 0$ si $x \neq y$
- $d(x, y) = d(y, x)$ (symétrie)
- Elle satisfait l'inégalité triangulaire :
  $d(x, y) + d(y, z) \geq d(x, z)$

Une point important sur le choix du XOR comme métrique c'est qu' elle capture une distance implicite dans la représentation des nœuds en arbre binaire. Tels que :
Dans un arbre binaire complètement peuplé d'identifiants (ID) de 160 bits, la magnitude de la distance entre deux ID est égale à la hauteur du plus petit sous-arbre les contenant tous

les deux. Autrement dit, plus deux ID sont proches dans l'arbre binaire, plus leur distance XOR sera faible. La distance XOR reflète donc directement la proximité des ID dans la structure de l'arbre.
Cette propriété est essentielle pour le fonctionnement du protocole Kademlia, car elle permet de router efficacement les requêtes vers les nœuds les plus pertinents en se basant sur la distance XOR entre les identifiants.
Par exemple:



La distance des deux nœuds marque en rouge est: 0111 XOR 0011 = 0100, le premier 1 dans 0100 est en 3ème position, donc hauteur du plus petit sous-arbre contenant les deux nœuds est également 3.

Dans Kademlia chaque nœud possède un table de routage constitué de 160 listes d'autres nœuds, chaque élément d'une liste contient: (IP address, UDP port, Node ID).
Pour chaque $i \in [0,\ 160[$, l'i^{eme} liste d'un nœud contient les information sur les nœuds avec une distance entre 2^i et 2^{i+1} de lui même. Dans la littérature, chaque liste est nommée k-bucket.
Chaque k-bucket est maintenu trié par ordre de dernière apparition, le nœud vu le moins récemment se trouve en tête, et le plus récemment vu se trouve en queue.
Pour les petites valeurs de i, les k-buckets seront généralement vides (car il n'y aura pas de nœuds appropriés). Pour les grandes valeurs de i, les listes peuvent atteindre une taille maximale de k, où k est un paramètre du système. La valeur de k est choisie de manière à ce que k nœuds donnés aient une très faible probabilité de tomber en panne en l'espace d'une heure (par exemple, k = 20). Cette organisation des k-buckets permet à Kademlia de maintenir une connaissance efficace du réseau, en conservant les informations sur les nœuds les plus récemment vus.

Lorsqu'un nœud Kademlia reçoit un message (requête ou réponse) d'un autre nœud, il met à jour le k-bucket approprié pour l'identifiant (ID) du nœud expéditeur.

- Si le nœud expéditeur existe déjà dans le k-bucket du destinataire, ce dernier le déplace à la fin de la liste.
- Si le nœud n'est pas déjà dans le k-bucket approprié et que le bucket à moins de k entrées, le destinataire insère simplement le nouvel expéditeur à la fin de la liste.

- Cependant, si le k-bucket approprié est plein, le destinataire fait un ping du nœud vu le moins récemment pour décider de la suite.

- Si ce nœud ne répond pas, il est évincé du k-bucket et le nouvel expéditeur est inséré à la fin. Sinon, s'il répond, il est déplacé à la fin de la liste, et le contact du nouvel expéditeur est rejeté.

On note que cette organisation offre une résistance à certaines attaques DOS, car de nouveaux nœuds ne peuvent pas remplacer les anciens dans les k-buckets.

Le protocole Kademlia se compose de quatre messages : ping, store, find_node et find_value.

- Le message ping sert à vérifier si un nœud est en ligne.
- Le message store instruit un nœud à stocker une paire <clé, valeur> pour une récupération ultérieure.

- Le message find_node prend un identifiant (ID) de 160 bits en argument. Le destinataire de ce message renvoie des triplets <adresse IP, port UDP, ID du nœud> pour les k nœuds qu'il connaît les plus proches de l'ID cible. Ces triplets peuvent provenir d'un seul k-bucket ou de plusieurs k-buckets si le k-bucket le plus proche n'est pas plein. Dans tous les cas, le destinataire du message doit renvoyer k éléments (sauf s'il y a moins de k nœuds dans tous ses k-buckets réunis, auquel cas il renvoie tous les nœuds qu'il connaît).

- Le message find value se comporte comme find node, en renvoyant des triplets <adresse IP, port UDP, ID du nœud>, avec une exception. Si le destinataire du message a reçu un message store pour la clé, il renvoie simplement la valeur stockée.

La procédure la plus importante qu'un participant Kademlia doit effectuer est de localiser les k nœuds les plus proches d'un identifiant (ID) de nœud donné. Nous appelons cette procédure une recherche de nœud (node lookup).
Kademlia emploie un algorithme récursif pour les recherches de nœuds. L'initiateur de la recherche commence par sélectionner α nœuds dans son k-bucket le plus proche non vide (ou, si ce bucket a moins de α entrées, il prend simplement les α nœuds les plus proches qu'il connaît). L'initiateur envoie ensuite en parallèle et de manière asynchrone des messages find_node à ces α nœuds choisis. α est un paramètre de concurrence du système, comme par exemple 3.
À l'étape récursive, l'initiateur renvoie le message find_node aux nœuds dont il a appris l'existence lors des messages précédents. (Cette récursion peut commencer avant que tous les α messages précédents n'aient renvoyé de réponse). Parmi les k nœuds les plus proches de la cible dont l'initiateur a entendu parler, il en choisit α qu'il n'a pas encore interrogés et leur renvoie le message find node.
Les nœuds qui ne répondent pas rapidement sont retirés de la considération jusqu'à ce qu'ils répondent. Si un tour de message find_node ne parvient pas à trouver un nœud plus proche que le plus proche déjà vu, l'initiateur renvoie le message find node à tous les k nœuds les plus proches qu'il n'a pas encore interrogés. La recherche se termine lorsque l'initiateur a interrogé et reçu des réponses des k nœuds les plus proches qu'il a vus.

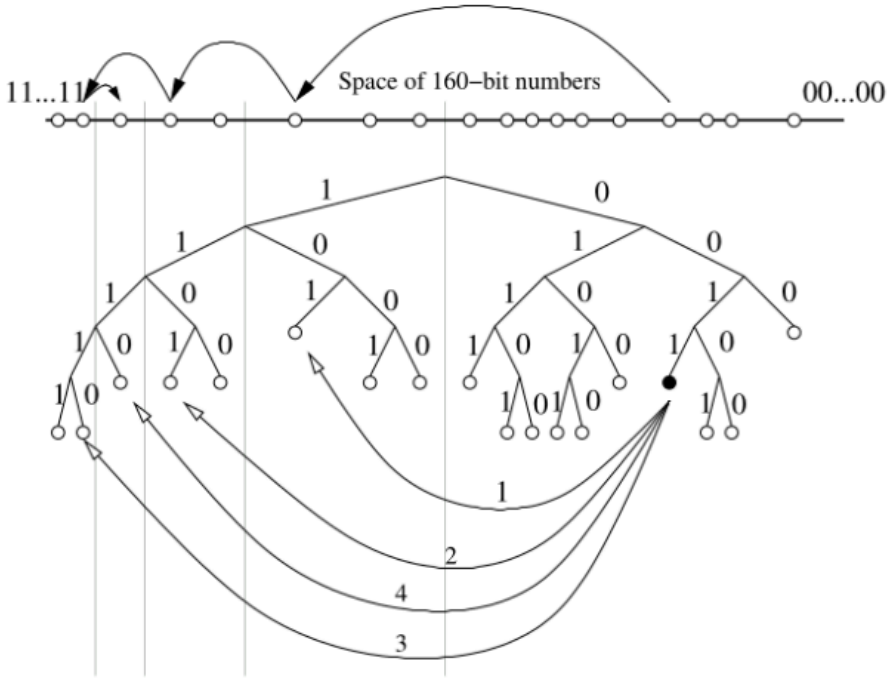


Fig. 2: Locating a node by its ID. Here the node with prefix 0011 finds the node with prefix 1110 by successively learning of and querying closer and closer nodes. The line segment on top represents the space of 160-bit IDs, and shows how the lookups converge to the target node. Below we illustrate RPC messages made by 1110. The first RPC is to node 101, already known to 1110. Subsequent RPCs are to nodes returned by the previous RPC.

La plupart des opérations sont mises en œuvre en utilisant la procédure de recherche de nœud décrite précédemment.

Pour stocker une paire <clé, valeur>, un participant localise les k nœuds les plus proches de la clé et leur envoie des message store. De plus, chaque nœud republie les paires <clé, valeur> au besoin pour les maintenir en vie, Cela assure la persistance de la paire <clé, valeur> avec une très haute probabilité.

Pour trouver une paire <clé, valeur>, un nœud commence par effectuer une recherche pour trouver les k nœuds dont les ID sont les plus proches de la clé. Cependant, les recherches de valeurs utilisent des messages find_value plutôt que find node. De plus, la procédure s'arrête immédiatement dès qu'un nœud renvoie la valeur. À des fins de mise en cache, une fois qu'une recherche a réussi, le nœud requérant stocke la paire <clé, valeur> sur le nœud le plus proche de la clé qu'il a observé et qui n'a pas renvoyé la valeur.

Pour rejoindre le réseau, un nœud u doit avoir un contact avec un nœud w déjà participant. u insère w dans le k-bucket approprié. u effectue ensuite une recherche de nœud pour son propre ID de nœud. Enfin, u rafraîchit tous les k-buckets plus éloignés que son voisin le plus proche. Lors de ces rafraîchissements, u replie ses propres k-buckets et s'insère dans ceux d'autres nœuds si nécessaire.

3.8 Tox:

Tox est un protocole peer-to-peer, distribué et libre. En utilisant des technologies existantes telles que le réseau dispersé et la cryptographie forte, Tox peut offrir une expérience de messagerie instantanée. Les fichiers peuvent être partagés aussi rapidement que le permet la connexion Internet de vous et de votre interlocuteur, les appels audio sont instantanés, et il n'y a pas de limites arbitraires au nombre de personnes que vous pouvez avoir dans une conversation de groupe.
L'intérêt principale de Tox est de mettre la messagerie sécurisée à la disposition de tous, cela implique les objectif suivantes;

- Authentification : Tox vise à fournir une communication authentifiée. Cela signifie que pendant une session de communication, les deux parties peuvent être sûres de l'identité de l'autre partie. Les utilisateurs sont identifiés par leur clé publique. Notant que si la clé secrète est compromise, l'identité de l'utilisateur est compromise et un attaquant peut se faire passer pour cet utilisateur. Lorsque cela se produit, l'utilisateur doit créer une nouvelle identité avec une nouvelle clé publique.

- Chiffrement de bout en bout : Le protocole Tox établit des liens de communication chiffrés de bout en bout. Les clés partagées sont dérivées de manière déterministe à l'aide d'une méthode similaire à Diffie-Hellman, de sorte que les clés ne sont jamais transférées sur le réseau.

- Secrets éphémères : Les clés de session sont renégociées lorsque la connexion entre les pairs est établie.

- Confidentialité : Lorsque Tox établit un lien de communication, il vise à ne pas divulguer à un tiers les identités des parties impliquées (c'est-à-dire leurs clés publiques).

- Résilience :

    - Indépendance de l'infrastructure : Tox évite de s'appuyer sur des serveurs autant que possible. Les communications ne sont pas transmises via des serveurs centraux ni stockées sur ces derniers. Rejoindre un réseau Tox nécessite de se connecter à un nœud bien connu appelé nœud d'amorçage. N'importe qui peut exécuter un nœud d'amorçage, et les utilisateurs n'ont pas besoin de leur faire confiance.

    - Tox tente d'établir des chemins de communication dans des situations réseau difficiles. Cela comprend la connexion à des pairs derrière un NAT ou un pare-feu. Diverses techniques aident à y parvenir, telles que le "hole punching" UDP, UPnP, NAT-PMP, d'autres nœuds non fiables agissant comme relais, et les tunnels DNS.

    - Résistance aux attaques de déni de service basiques : des délais courts rendent le réseau dynamique et résilience contre les tentatives d'empoisonnement.

- Configuration minimale : Tox vise à être presque sans configuration. La convivialité est un aspect important de la sécurité. Tox vise à rendre la sécurité facile à atteindre pour les utilisateurs moyens.

Une point important a mentionné que l'anonymat n'est pas inclus dans le protocole Tox lui-même, mais il offre un moyen facile de s'intégrer avec des logiciels fournissant l'anonymat, tels que Tor. Par défaut, Tox tente d'établir des connexions directes entre les pairs ; par conséquent, chacun est conscient de l'adresse IP de l'autre, et des tiers peuvent être en mesure de déterminer qu'une connexion a été établie entre ces adresses IP. Une des raisons de l'établissement de connexions directes est que le relais de conversations multimédias en temps réel sur des réseaux d'anonymat n'est pas réalisable avec l'infrastructure réseau actuelle.

-Le fonctionnement du protocole Tox:

Avant de commencer la communication avec quelqu'un, le nœud doit d'abord être connecté au DHT qui contient les informations sur les utilisateurs dans le réseau.
Donc les utilisateurs de Tox sont connectés par une version modifiée de DHT Kademlia , et des nœuds d'amorçage sont utilisés pour aider les connexions au pool. Pour ce connecter a ce DHT nous avons besoin de connaître l'adresse IP et le port d'un nœud DHT, ainsi que sa clé publique, pour pouvoir protéger la communication. Dans Tox, chaque pair qui a les connexions UDP activées est un nœud DHT. Cependant, contrairement aux pairs Tox réguliers, les utilisateurs utilisant Tox pour communiquer avec leurs amis, les nœuds DHT peuvent être dédiés, c'est-à-dire avoir une adresse IP plus permanente et être en ligne la plupart du temps, ce qui signifie que connaître l'adresse IP, le port et la clé publique de seulement quelques nœuds est généralement suffisant pour pouvoir se connecter toujours au réseau DHT.

Une fois connecté eu DHT, la méthode pour se connecter et communiquer à quelqu'un est de connaître son adresse IP et son port, Dans le modèle de serveur centralisé, l'adresse IP et le port peuvent être stockés sur le serveur central et associés à un identifiant unique, tel que le nom d'utilisateur ou l'e-mail, de sorte que vous puissiez interroger le serveur central de l'adresse IP et du port de l'utilisateur sous le nom d'utilisateur X et le serveur vous donnerait l'adresse IP et le port les plus récents pour l'utilisateur X. Cependant, dans notre cas, il n'y a pas de serveur central où de telles informations pourraient être stockées, car la centralisation va à l'encontre de la philosophie de Tox. Heureusement, il existe une solution qui permet de stocker l'adresse IP et le port d'un pair associé à l'identifiant unique du pair de manière décentralisée - DHT, DHT est essentiellement une table de paires clé-valeur qui est distribuée parmi tous les pairs du réseau. Ainsi, pour pouvoir se connecter aux utilisateurs Tox avec qui nous voulons communiquer, nous devons d'abord nous connecter au réseau DHT, puis rechercher les adresses IP et les ports de ces utilisateurs Tox en utilisant leurs identifiants uniques.

Tox utilise un nombre de 32 octets comme identifiant unique de l'utilisateur, c'est la clé publique de la paire de clés asymétriques permanente de l'utilisateur. La paire de clés est générée à l'aide d'une fonction aléatoire cryptographique, de sorte que la probabilité que deux personnes génèrent le nombre aléatoire est extrêmement faible, ce qui en fait un bon identifiant unique. La clé publique de 32 octets est représentée textuellement sous forme de 64 caractères hexadécimaux en majuscules, chaque paire de caractères encodant un seul octet, par exemple F404ABAA1C99A9D37D61AB54898F56793E1DEF8BD46B1038B9D822E8460FAB67. Une nouvelle paire de clés est générée chaque fois que l'instance est réinitialisée.

Plusieurs implémentations du protocole Tox existent, l'une d'entre elles étant c-toxcore, et c'est celle-ci qui servira de base à notre implémentation de l'application de chat dans la prochaine partie de ce rapport.

Bibliographie:

- Chawathe, Yatin, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. "Making Gnutella-like P2P Systems Scalable." https://courses.cs.washington.edu/courses/cse522/05au/GIA.pdf.
- Koegel Buford, John F., Heather Yu, and Eng K. Lua. 2009. *P2P Networking and Applications*. Elsevier Science.
- Kurose, James F., and Keith W. Ross. 2017. *Computer Networking: A Top-down Approach*. : Pearson.
- Petar Maymounkov, and David Mazieres. "Kademlia: A Peer-to-peer Information System Based on the XOR Metric." https://www.ic.unicamp.br/~celio/peer2peer/kademlia/maymounkov02kademlia.pdf.
- "A SURVEY AND COMPARISON OF PEER-TO-PEER OVERLAY NETWORK SCHEMES." I*EEE Communications Surveys & Tutorials*.