Corso di **Tecnologie per il cloud**

6. Sistemi P2P

Introduzione

I sistemi **Peer-to-Peer (P2P)** sono stati i primi veri sistemi distribuiti focalizzati sulla scalabilità in base al numero di nodi. Vengono impiegati in svariati modelli di cloud computing e si suddividono in sistemi P2P **strutturati** e **non strutturati:**

- In una rete peer-to-peer (P2P) non strutturata, i nodi non sono organizzati in alcun ordine particolare, il che significa che la comunicazione nodo a nodo avviene in modo “casuale”. Ciò rende le reti P2P non strutturate adatte a casi d'uso ad alta attività, come le piattaforme social dove gli utenti possono unirsi o lasciare la rete regolarmente.

- Le reti peer-to-peer (P2P) strutturate differiscono dalle reti P2P non strutturate perché i nodi sono organizzati in modo strutturato e seguono un'architettura ben organizzata che consente loro di interagire in modo più efficiente. Questo tipo di organizzazione permette agli utenti di trovare e utilizzare i file in modo più efficace rispetto alla ricerca casuale. Nelle reti P2P strutturate, le funzioni di hash sono comunemente utilizzate per le ricerche nel database.

Peer to Peer e Grid Applications:

Il paradigma P2P (Peer-to-Peer) è una forma di architettura decentralizzata per la condivisione di risorse e l'elaborazione distribuita di informazioni. In una rete P2P, ogni nodo partecipa in modo attivo alla condivisione di risorse e informazioni con gli altri nodi, senza la necessità di un server centrale.

Anche se il P2P e la Grid Computing sono entrambe tecnologie per l'elaborazione distribuita, non sono la stessa cosa. Le Grid Applications o Grid Computing, infatti, sono un tipo di infrastruttura di elaborazione distribuita che prevede l'utilizzo di risorse di calcolo e di memoria di una rete di computer interconnessi per elaborare grandi quantità di dati. Al contrario, il P2P è un paradigma per la condivisione di risorse e l'elaborazione distribuita, che non richiede l'utilizzo di una rete dedicata di computer interconnessi.

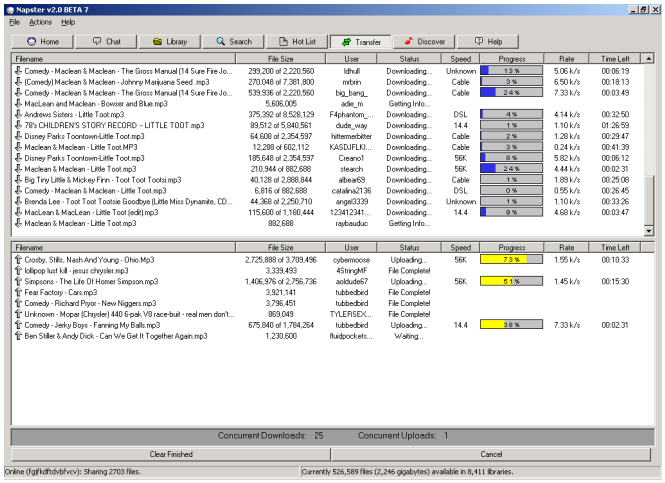
In sintesi, il P2P e le Grid Applications sono due tecnologie distinte per l'elaborazione distribuita, anche se entrambe condividono l'obiettivo di sfruttare la potenza di elaborazione distribuita di una rete di computer per gestire grandi quantità di dati.

IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Napster

Napster non è un vero e proprio **protocollo Peer-to-Peer** infatti si appoggia sulla presenza di server centralizzati che fanno da coordinatori; parliamo di Napster perché è stato il punto di partenza dal quale poi si sono evoluti tutti gli altri sistemi P2P che oggi conosciamo.

Il client Napster era un'applicazione software che gli utenti installavano sui propri computer per accedere alla rete Napster. Il client consentiva agli utenti di cercare, scaricare e condividere file musicali digitali con altri utenti. Inoltre, il client Napster offriva funzionalità come la gestione delle librerie musicali e la chat tra utenti.



Dalla sua interfaccia, bastava semplicemente cercare il nome dell'artista, così come il nome dell'album o della canzone, per visualizzare una serie di risultati, trovati in computer sparsi in giro per il mondo. La velocità di download era chiaramente dipendente dalla banda in upload dei nodi in grado di inviare il file.

Ciò che ci interessa in particolar modo di un sistema P2P è comprendere come avviene la ricerca di questi contenuti e sapere dove reperirli nel momento in cui il client effettua una query.

Struttura di Napster

Ciascun nodo che esegue Napster è detto **peer** e si connette ai server di *napster.com*, che appunto permettono la comunicazione con gli altri peer che eseguono il medesimo client.

Ipotizzando che uno dei peer abbia il file della canzone **Penny Lane** dei Beatles: nel momento in cui lo si carica sul client Napster, non viene condiviso su nessun server, ma rimane in loco nel peer che lo propone.

IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

I server memorizzano solo le informazioni che riguardano il **puntamento** ai peer e alle loro directory. Nello specifico, hanno una tabella strutturata come di seguito:

Filename	Info about
...	...
PennyLane.mp3	Beatles @ 128.84.92.23:1006
...	...

Il **client** dunque:

1. **Si connette** ai server di Napster;
2. **Fornisce** una lista dei file musicali che era intenzionato a condividere con altri peer;

Il **server mantiene** una tabella contenente: ***nome del file, indirizzo ip*** e ***numero di porta***. L'informazione è quindi legata a **dove** reperire il file sul client che lo fornisce.

Ricerca in Napster

Nel momento in cui un altro client è intenzionato a cercare un file nella rete di Napster:

1. Procede con l'**inviare** le parole chiave immesse dall'utente al server;
2. Il server **cerca** con le parole chiave fornite, non il file (che logicamente non ha), ma dove andare a reperire quest'ultimo, attraverso la tabella memorizzata. Dopodichè, ritorna la lista degli host che contengono il file con tali parole chiave, tipicamente una **tupla**: ***<indirizzoIP, numeroPorta>***;
3. Il client **pinga** ciascun host nella lista per determinare quello con la miglior velocità di trasferimento;
4. Sempre il client **scarica** il file dal miglior host.

Tutte le comunicazioni avvengono mediante il protocollo affidabile **TCP**.

Problemi di Napster

Nonostante quella di Napster fosse un'introduzione funzionale dei sistemi P2P, soffriva di alcuni problemi:

- La presenza di server centralizzati costituiva fonte di **congestione**. Essendo il coordinamento della rete a carico di essi, era facile l'intasamento;
- La presenza di server centralizzati costituiva fonte di **intolleranza ai guasti**;
- Nessuna implementazione sicura per le password e i messaggi.

Poco tempo dopo, Napster venne accusato di violazione del copyright ed è stato soppresso come servizio P2P. Oggi Napster è tornato in vita come servizio musicale a pagamento.

Gnutella

Gnutella è un **protocollo di rete** peer-to-peer non strutturato utilizzato per la condivisione di file su Internet. Quindi, tecnicamente parlando, **Gnutella è un protocollo di rete, non un software o un client.** Tuttavia, poiché il protocollo Gnutella viene utilizzato principalmente attraverso software client che implementano il protocollo stesso, spesso si fa riferimento ai programmi software che utilizzano il protocollo Gnutella come "client Gnutella".

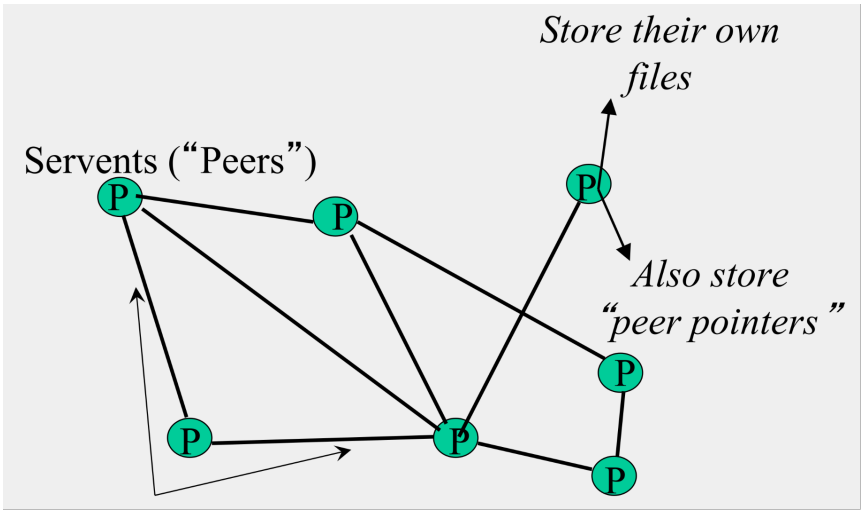
*Nonostante contenga la parola GNU, non fa parte del progetto GNU. La **Nutella** era uno degli alimenti preferiti dagli sviluppatori.*

Pregi di Gnutella

Gnutella risolveva uno dei problemi più gravi che affliggeva Napster, ossia la presenza di server soggetti a congestione e intolleranza ai guasti. Infatti:

- I **server** sono stati **eliminati** completamente;
- Ciascun **client interagisce direttamente con altri client**;
- Ciascun **client funge anche da server**, ed è detto **servent**.

Il collegamento tra i vari peer viene effettuato per indirizzo IP e porta, e si utilizza il protocollo TCP. Il collegamento in figura è detto **overlay graph** (una sorta di rete logica parallela a quella “internet” esistente.



IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Formato little-endian

Tutti i campi, ad eccezione dell'indirizzo IP, sono nel formato **little-endian**; è una convenzione utilizzata nell'architettura dei computer per l'organizzazione dei dati in memoria. In questo formato, i byte meno significativi (il byte di minor peso) vengono memorizzati nella posizione di memoria più bassa, mentre i byte più significativi (il byte di maggior peso) vengono memorizzati in posizioni di memoria più alte.

In altre parole, nel formato little-endian, i byte vengono organizzati in modo inverso rispetto alla loro rappresentazione naturale nella notazione decimale o binaria. Questa scelta può sembrare controintuitiva, ma è una decisione di progettazione ampiamente utilizzata in molte architetture di computer.

Supponendo di avere, ad esempio, il valore esadecimale **0x12345678**, l'ordine del little-endian prevede di suddividere il numero in gruppi da **1 byte** (8 bit) ciascuno. Nel caso specifico si avrebbero 4 byte, cioè una DWORD (32 bit). La memorizzazione inizia dal byte meno significativo e finisce col byte più significativo:

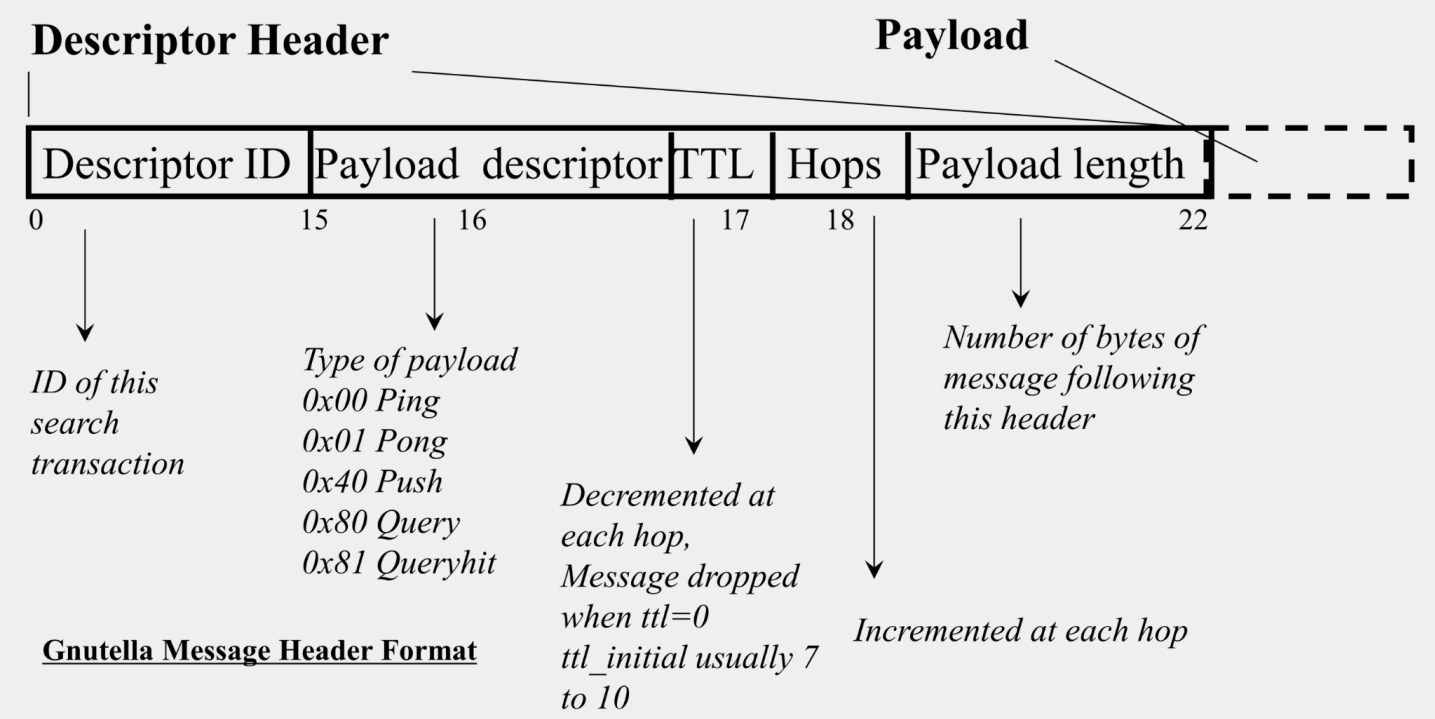
Byte 3	**Byte 2**	**Byte 1**	**Byte 0**
0x78	0x56	0x34	0x12

In un ordine di tipo **big-endian**, avremmo invece la memorizzazione che inizia dal byte più significativo e che finisce col byte meno significativo.

Byte 3	**Byte 2**	**Byte 1**	**Byte 0**
0x12	0x34	0x56	0x78

IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO

Header di un messaggio Gnutella



L'header di un messaggio Gnutella è costituito come in figura.

- Ciascun messaggio contiene un **descriptor** che è l'ID della transazione di ricerca;
- Il **payload descriptor** definisce il tipo di messaggio;
- Il **TTL** è il time-to-live che viene decrementato ad ogni salto. Inizialmente è posto pari a 7 o 10, in base ai vicini, quando **ttl=0**, il messaggio non viene più inviato a nessuno;
- Gli **Hops** sono appunto il numero di salti all'interno di una rete, ovvero, il numero di volte che il messaggio viene inviato ai peer più vicini;
- Il **payload length** è la lunghezza del messaggio che segue l'header.
- **Payload**

Payload di Gnutella

Gnutella prevede cinque payload (messaggi) differenti, che sono rispettivamente:

1. **Ping:** messaggio per chiedere l'aggiornamento della lista dei peer più vicini rispetto al peer corrente;
2. **Pong:** messaggio di risposta al ping
3. **Query**: messaggio di ricerca con le parole chiave
4. **QueryHit**: messaggio di risposta alla query
5. **Push:** messaggio di inizializzazione del trasferimento di un file

Messaggio Ping-Pong

I messaggi ping pong sono una parte essenziale del protocollo di rete Gnutella e vengono utilizzati per la scoperta di nodi e la connessione tra di essi. La comunicazione tra nodi tramite messaggi ping pong serve a creare una mappa della rete Gnutella, consentendo ai nodi di sapere quali altri nodi sono disponibili e quali file possono essere scaricati da essi. Inoltre, i messaggi ping pong vengono utilizzati per mantenere la connessione tra i nodi. Quando un nodo riceve un messaggio di pong, sa che la connessione con l'altro nodo è attiva e può continuare a scambiare messaggi con esso.

Quando un peer riceve un messaggio Ping, esso risponde con un **Pong** message, il quale contiene:

- L'**indirizzo IP** e la **porta**;
- Il **numero di file condivisi**;
- Lo **spazio** condiviso (in kB).

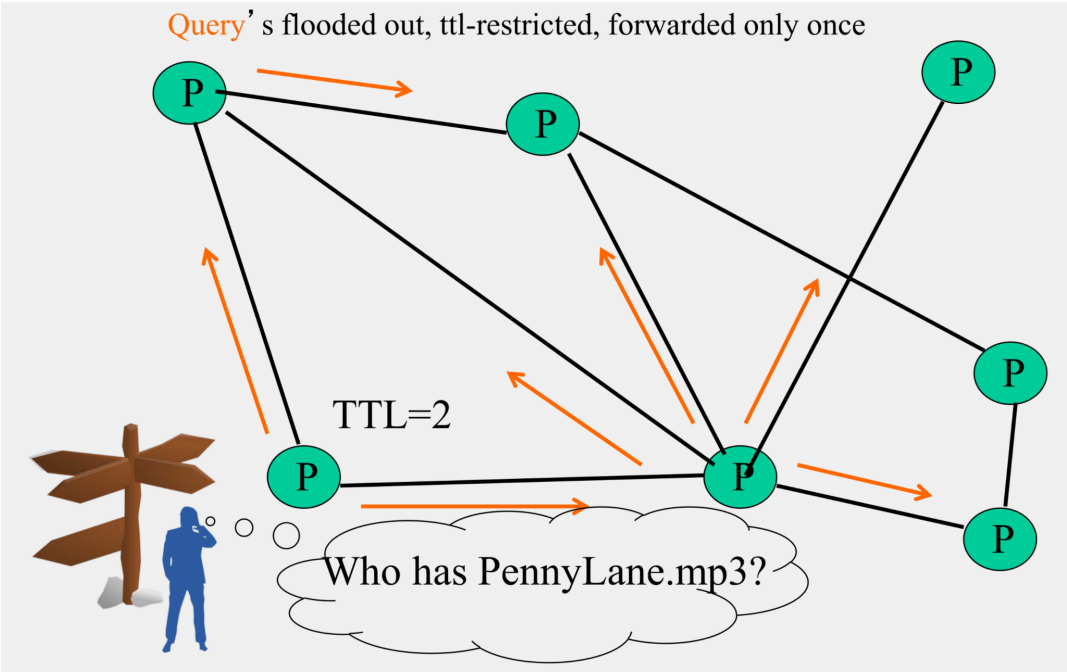
Gli ultimi due campi sono utilizzati dal peer richiedente per preferire quei vicini che possiedono un numero maggiore di file.

Messaggio Query

Un messaggio query contiene:

- La **velocità minima** richiesta dal peer che ha effettuato la ricerca. Per velocità minima si intende proprio la banda di collegamento, ad es. 100 Mbps;

- Le **parole chiave** della ricerca.



Nella figura sovrastante, supponiamo che il TTL del messaggio generato dal peer in basso a sinistra sia pari a 2 (in realtà è un numero compreso tra 7 e 10 ma per fare capire come funziona il TTL si è preferito scegliere un numero più piccolo)

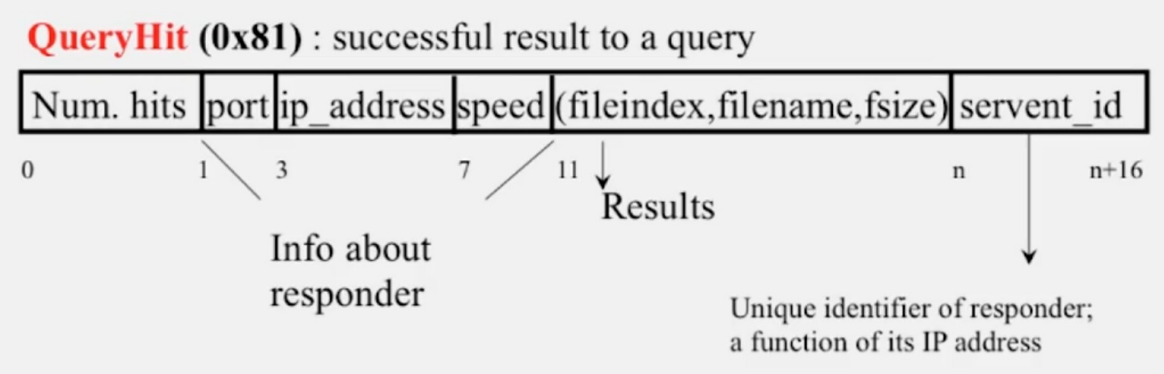
Ciascun vicino:

- Quando riceverà la query, inizierà a cercare all'interno dei suoi file locali il file richiesto dalla query, sulla base delle keywords;

- Procederà poi con l'inoltro della query stessa ai suoi vicini, ad eccezione di quello da cui l'ha appena ricevuto. Il TTL di ciascun messaggio sarà pari a 1 (decrementato dal TTL precedente). I peer che riceveranno l'inoltro non lo inoltreranno nuovamente, in quanto il TTL sarà pari a zero.

Messaggio QueryHit

Una volta che viene inviata la query, come vengono ritornati i vari risultati di ricerca? Si sfruttano i messaggi **QueryHit**.



- **Num hits** è il numero di file corrispondenti;
- La **porta** e l'**indirizzo IP**, così come la **velocità** del peer che risponde;
- Per ciascun file corrispondente si ha un **indice**, un **nome** e una **dimensione**;
- Il **servent ID** è un identificativo univoco del nodo che risponde alla ricerca.

Sulla base di quanto visto finora:

1. Per evitare trasmissioni **duplicate**, ciascun peer mantiene una lista dei messaggi ricevuti di recente;
2. Un messaggio **Query** ricevuto viene inoltrato a tutti i vicini, ad eccezione del peer che per primo l'ha inviato;
3. Ciascuna Query è identificata da un **DescriptorID**, e il suo inoltro avviene una singola volta;
4. I messaggi QueryHit vengono rimandati solo ai peer per i quali la query ricevuta ha il **medesimo** DescriptorID;

Dopo che si riceve il messaggio **QueryHit**:

1. Il **richiedente** sceglie il peer migliore che ha fornito la QueryHit e inizializza con esso una richiesta HTTP mediante **indirizzo IP** e **porta**. Nella richiesta è compreso un campo **range**, al fine di supportare il trasferimento parziale dei file.

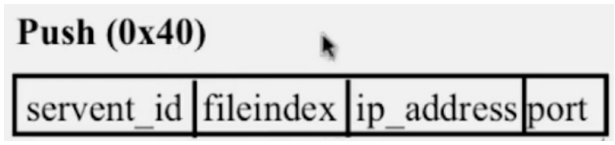
2. Il destinatario **risponde** con i pacchetti del file e inizia il download.

Le comunicazioni avvengono tramite il protocollo **HTTP** in quanto ampiamente usato e ben collaudato.

Presenza di un firewall

Cosa succede se il richiedente (A) , del file, si rende conto che il sender (B) si trova dietro un firewall o una rete sotto NAT? Vediamo tutti i passaggi:

- Supponiamo che B sia sotto NAT e che non possa accettare connessioni dall'esterno
- Punto fondamentale: B apre comunque un insieme di connessioni verso l'esterno per inserirsi nell'overlay
- B quindi può ricevere la **Query mediante le connessioni dell'overlay** network che esso stesso ha aperto verso altri peer. Può inviare i messaggi di risposta a query (query hit) mediante le queste connessioni
- A invia una Query e B ha il contenuto che la soddisfa. B invia il query hit mediante le connessioni dell'overlay ed inserisce nel query hit
  - il proprio ServentID
  - setta il firewalled flag
- A riceve il QueryHit e invia un messaggio di Push a B.
- Il messaggio di PUSH contiene:



  - il Servent_Id di B
  - un riferimento al file che A vuole scaricare
  - indirizzo IP e porta di A
- Il messaggio di Push segue a **ritroso lo stesso cammino percorso dal messaggio di QueryHit.**
- I Servent_Id viene utilizzato per implementare il backward routing del messaggio di Push
- Quando B riceve il messaggio di Push, apre una connessione verso A ed invia il file richiesto

Problemi di Gnutella

Come Napster, non è perfetto. Vediamo quali sono alcuni problemi:

- I messaggi Ping-Pong costituiscono il 50% del traffico. Una soluzione sarebbe quella di adoperare una cache per ridurre la frequenza di ping/pong, detta anche **Multiplex**;
- Persistono ricerche ripetute con le medesime parole chiave;
- La banda di connessione dei peer non è ottimale;
- Il 70% degli utenti Gnutella nel 2000 scaricavano senza mai caricare;
- Sensibilità al **flooding**.

FastTrack

FastTrack è un **protocollo peer-to-peer non strutturato** utilizzato per la condivisione di file su Internet ed è un protocollo utilizzato da diversi client P2P come Kazaa, Morpheus e Grokster.
Tuttavia, dopo una serie di azioni legali contro i client che utilizzavano FastTrack, il protocollo è diventato meno popolare nel corso degli anni. Oggi, ci sono solo pochi client P2P che utilizzano ancora il protocollo FastTrack.

Come funziona:

- Condivisione dei file: gli utenti di FastTrack installano un software client sul proprio computer e selezionano i file che desiderano condividere con gli altri utenti nella rete. Il software del client crea quindi un indice di questi file e li mette a disposizione degli altri utenti nella rete FastTrack.

- Ricerca dei file: gli utenti di FastTrack possono cercare i file desiderati utilizzando il software client. La ricerca viene effettuata attraverso una query inviata a tutti gli altri nodi nella rete FastTrack.

- Connessione tra i nodi: quando un utente trova il file desiderato, il software client FastTrack avvia una connessione tra il nodo dell'utente che cerca e il nodo che possiede il file. **La connessione viene stabilita attraverso un sistema di peer intermedi che facilitano la comunicazione tra i nodi.**
  Questi nodi intermedi sono chiamati **"super nodi" o "super peer"** e servono per facilitare la ricerca dei file e il trasferimento dei dati. Il ruolo dei Super Peer è importante perché consente di ridurre il carico sulla rete, aumentare la velocità di ricerca dei file e migliorare l'affidabilità del sistema. Inoltre, i Super Peer possono gestire le connessioni tra i peer e facilitare il routing dei dati attraverso la rete.

  Un super nodo non è detto che lo debba rimanere per sempre, può succedere che esso cambi nel tempo e che un altro nodo diventi il nuovo super nodo. Il premio di super peer viene affidato a quei peer che mantengono alta la loro **reputazione**:

  1. Inizialmente il livello di reputazione di un utente è pari a 10;
  2. Sulla base del periodo di connettività e del numero totale di upload, può incrementare fino a 1000.

- Download dei file: una volta stabilita la connessione tra i nodi, il file viene trasferito dal nodo che lo possiede al nodo che lo ha cercato. Il trasferimento avviene in modo decentralizzato.

BitTorrent

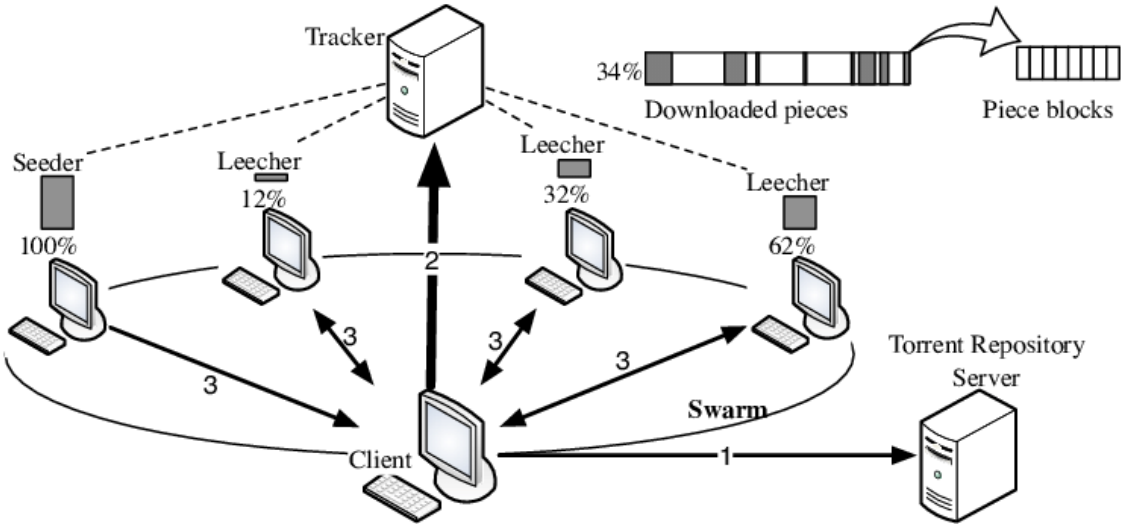
BitTorrent è un **protocollo peer-to-peer ibrido** per la distribuzione efficiente di file su Internet.

Quindi, BitTorrent è un **protocollo e non un software specifico (Anche se BitTorrent ha sviluppato l'omonimo client)**. Tuttavia, ci sono molti client BitTorrent disponibili, come uTorrent, Vuze, Deluge, qBittorrent, Transmission e molti altri, che utilizzano il protocollo BitTorrent per la condivisione di file.

BitTorrent è un esempio di **protocollo P2P ibrido** che utilizza sia **elementi** di un sistema **P2P strutturato** (come la DHT) che elementi di un sistema **P2P non strutturato** (come l'uso del tracker)

Come funziona:

- Quando un peer apre col client un file *.torrent*, si mette in contatto col server **tracker**. Il tracker conosce l'indirizzo di ogni peer partecipante allo swarm (insieme di peers che collaborano alla distribuzione di uno stesso file e sono coordinati da uno stesso tracker) e consente quindi ad ogni peer di individuare altri peer nello stesso swarm. Tipicamente ad ogni file condiviso vi è associato un torrent, ovvero la lista dei peer che posseggono il file richiesto;



- Un peer può essere di due tipi:
  a. **Seeder :** contiene il file per intero;
  b. **Leecher :** contiene solo alcuni blocchi del file.

  Quando un peer entra a far parte della lista, è anch'esso un **leecher**, in quanto non contiene ancora il file per intero;

- Un file è suddiviso in blocchi che variano tra i 32 kB e i 256 kB, dunque un peer può selezionare il blocco (chunk) che desidera scaricare. Viene favorita una policy di tipo **Rarest First**, ossia, vengono privilegiati i blocchi più rari nel contesto della diffusione del file; i blocchi vengono inviati ai vicini che hanno fornito ad esso la miglior banda in download. Questo incentiva i nodi, quindi, a fornire un collegamento più veloce possibile. Idem per i seed.

- Meccanismo di Chocking:
  Ogni peer apre una **connessione TCP** (connessione bidirezionale) verso un insieme di altri peer i cui indirizzi sono stati ricevuti dal tracker; ogni **connessione** si può trovare, in uno dei due seguenti stati:
  - unchocked: la connessione è utilizzata per il trasferimento dei dati
  - chocked: la connessione viene 'soffocata' cioè non viene utilizzata per il trasferimento dei dati, anche se viene mantenuta aperta

L'algoritmo di chocking varia in relazione allo stato del peer, ovvero se è un **seeder** o **leecher.**

Algoritmo di chocking nel caso in cui si è un **leecher (ovvero quando mancano parti di file)**: ciascun peer mantiene un numero limitato di vicini unchoked (Al massimo vi sono 4 unchocked peers, in ogni istante) e periodicamente valuta, per ogni vicino, la velocità di download nel round precedente e decide quali vicini 'strozzare', ovvero, applica il Chocking, ossia un rifiuto (temporaneo) a trasferire file (può essere revocato) le connessioni vengono comunque lasciate aperte, per non pagare più di una volta il costo dell'apertura della connessione TCP.

Vi sono due tipi di unchocking:

- **Regular unchoking:** è una operazione che viene effettuata ogni 10 secondi (in termini di connessione TCP sono un tempo relativamente lungo per permettere il pieno sfruttamento del canale di comunicazione), vengono ordinati i peer remoti che hanno richiesto qualche pezzo, secondo la loro velocità di download nell'ultimo periodo e viene effettuato l'unchoke dei **primi tre peer** (quindi dei 3 peer più veloci)
- **Optimistic unchoking:** ogni 30 secondi, si sceglie un ulteriore peer, in modo casuale e si effettua l'unchocking, si fa poi l'upload per determinare se il nuovo peer può migliorare la velocità di download.
  Ciò consente di **valutare** la capacità di download di **nuovi peer** e **permette** il **bootstrap di nuovi peer** sulla rete che non hanno alcun pezzo da offrire (per questo è importante all'inizio chiedere un pezzo non raro al maggior numero di peer possibile)

Algoritmo di chocking nel caso in cui il peer è un **seeder:**
una volta scaricato tutto il file il nodo diventa un seeder, il suo comportamento quindi si ribalta ovvero, se prima l'interesse era quello di scaricare il file dai nodi più veloci, adesso si cerca di diffondere il file ai nodi leecher con un'alta velocità di download, indipendentemente dal loro contributo allo swarm

DHT (Distributed Hash Table)

Una Distributed Hash Table (DHT) è una struttura dati distribuita utilizzata nei sistemi P2P per memorizzare e recuperare informazioni in modo distribuito. Ogni nodo nella rete P2P memorizza solo una porzione della tabella hash, anziché l'intera tabella, in modo da evitare il singolo punto di fallimento. La DHT è spesso utilizzata nei sistemi P2P per la condivisione di file, come BitTorrent, per trovare altri peer che condividono il file e scaricarne parti da loro.

- DHT - BitTorrent: dato che l'utilizzo di un tracker costituisce un punto di centralizzazione che può diventare un **collo di bottiglia** o un **single point of failure**, sono usate tecniche trackerless basate su DHT (Kademlia sviluppata inizialemnte per e-Mule e successivamente adottata da BitTorrent) oppure Multi Trackers; per quanto riguarda DHT:
  - Ogni peer esegue anche le funzionalità di tracker
  - Ad ogni peer viene associato un identificatore di 160 bit e nello spazio degli identificatori vengono mappati i descrittori dei files (InfoHash)
  - Per ogni file:
    - Chiave = InfoHash del file
    - Informazione = Lista di peers appartenenti allo swarm

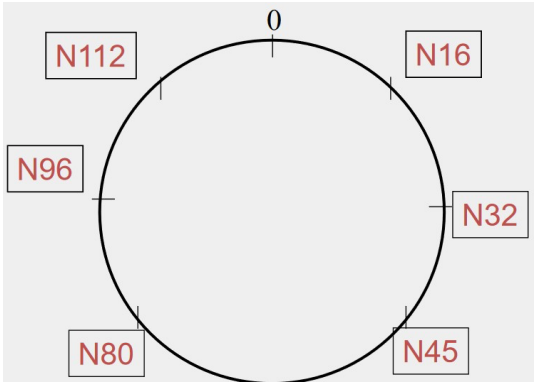
Chord

Chord è un **protocollo P2P strutturato** che associa per ogni dato una chiave e conserva la tupla chiave-valore nel nodo a cui la chiave è assegnata. I nodi possono inoltre entrare (join) o uscire (leave) dalla rete e rispondere continuamente alle richieste anche se il sistema è in continuo cambiamento.

Consistent hashing

Analogamente a Gnutella, un peer Chord seleziona i nodi più vicini a se stesso in maniera intelligente. Mentre Gnutella seleziona i vicini sulla base di una matrice che classifica i nodi col maggior numero di KB posseduti, Chord sfrutta il cosiddetto **consistent hashing** che prevede l'uso dell'**hashing function SHA-1** per assegnare ID :

- **Chiavi** tramite la combinazione chiave-valore, la memorizzazione di queste chiavi avviene all'interno di una **DHT**
  - Questo meccanismo (DHT) consente di:
    - **Bilanciare** il carico;
    - Avere elevata **tolleranza ai guasti**;
    - Avere dei meccanismi di **lookup** e **insert** efficienti;
    - **Localizzare** i dati.



- Per ciascun **nodo** si prende l'**indirizzo IP** e la **porta** si fa lo SHA-1 che restituisce una stringa a 160 bit. Questo output viene troncato, al fine di ottenere un identificativo per quel peer chiamato **Peer ID** (numero compreso tra 0 e 2^m-1). Tale identificativo **NON** è univoco,motivo per il quale possono sussistere conflitti all'interno della rete, tuttavia in **casi** veramente **molto rari**.

Si può a tal proposito disegnare un cerchio che consiste di 2^m punti logici, che va da 0 a 2^m-1

Il parametro "m" viene utilizzato per determinare la dimensione dello spazio degli identificatori e il numero di finger mantenuti da ogni nodo. La scelta del parametro "m" influisce sull'efficienza, la scalabilità e la tolleranza ai guasti del network Chord.
Il valore di "m" determina la dimensione dello spazio degli identificatori nel network Chord.

Lo spazio degli identificatori è tipicamente una potenza di 2, quindi "m" viene di solito scelto in modo che lo spazio degli identificatori contenga 2^m nodi. Ad esempio, se "m" viene impostato su 8, lo spazio degli identificatori potrebbe ospitare 256 nodi.
Il numero di finger mantenuti da ogni nodo viene tipicamente impostato su 2^m, dove ogni finger rappresenta una connessione a un nodo specifico nello spazio degli identificatori. I finger vengono utilizzati per mantenere informazioni di routing, consentendo una ricerca efficiente e il routing dei messaggi all'interno del network.
. Ad esempio, posto *m* = 7, si hanno poi 6 nodi. Il primo nodo in alto a destra ha come id 16, il secondo 32, ecc.

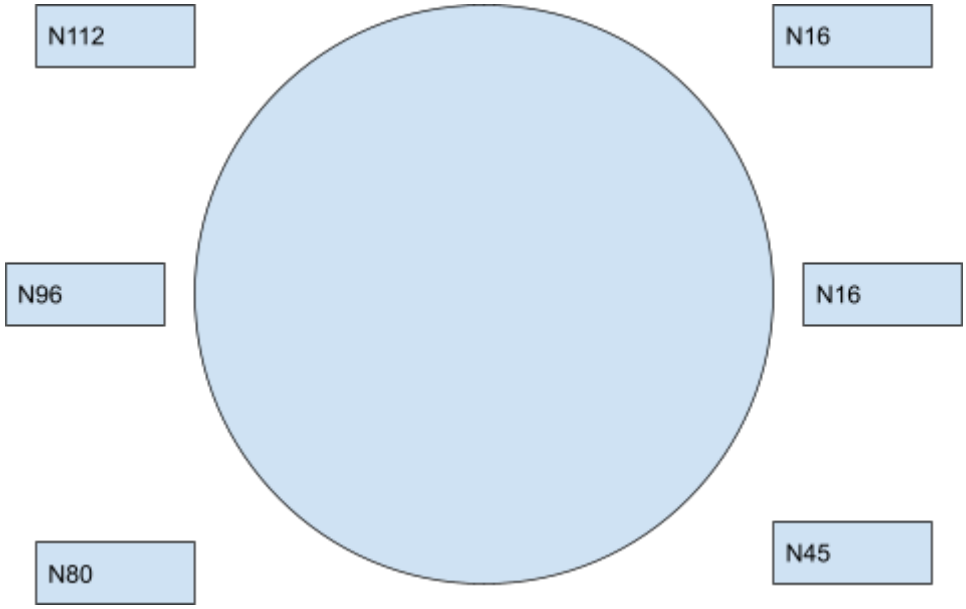
Ciascun nodo conosce il suo predecessore e successore. Inoltre, grazie a una **finger table** conosce alcuni successori in senso orario.

Finger table

Ciascun nodo, al fine di instradare le query in maniera molto veloce, è dotato di una tabella detta **finger table**, costituita essenzialmente come nel seguente esempio. SI SUPPONE SIA LA F.T. DEL NODO 80:

i **(indice)**	$ft[i]$ **(primo posto pieno successivo all'indice)**
0	96
1	96
2	96
3	96
4	96
5	112
6	16

- i rappresenta il valore dell'esponente della $(n + 2^i)(mod\ 2^m)$ posizione, dove n è il nodo corrente;
- $ft[i]$ rappresenta il nodo immediatamente successivo rispetto alla *i*-esima posizione.



Esempio

Con riferimento alla tabella sopra citata, supponendo che appartenga al nodo **N80**, la posizione $(80 + 2^0)(mod\ 2^7)$ corrisponde alla posizione 81. Tuttavia, il nodo più vicino è **N96**, per cui tutti i valori i da 0 a 4 ($80 + 2^4 = 80 + 16 = 96$) condurranno sempre a N96.

Posizionare i file nel ring

In Chord, i file vengono inseriti in dei nodi adatti allo scopo e sempre sottoposti al meccanismo di **consistent hashing:**

1. Si prende il nome del file e si applica una funzione di **hashing**;
2. La funzione di hashing produce un output di tipo SHA-1 a 160 bit;
3. L'output viene troncato a 2^m bit e memorizzato nel primo peer in possesso dell'ID maggiore o uguale alla sua chiave.

Esempio
Il file dal nome *cnn.com/index.html* viene mappato dall'algoritmo di hashing come **K42**. Viene quindi memorizzato nel peer con ID maggiore o uguale a N42 più vicino, ad esempio **N45**, visto che N42 non esiste.

Ricerca nei nodi

Supponiamo a questo punto che il nodo N80 debba cercare il file con hash 42 in N45, come fa a reperirlo, pur non conoscendolo nella sua finger table?

1. La query viene mandata al più grande successore della finger table che è **<=k**, dove k è 45. Nel caso specifico, chiederà a **N16**;
2. A questo punto **N16** farà la stessa cosa, chiedendo a **N32**, poiché stiamo supponendo che nella sua

finger table venga prima di **N45**, e che quest'ultimo non sia presente nella finger table di **N16**;
3. La finger table di **N32** conosce **N45**, quindi **N32** dirà a **N45** che **N80** vuole il file;
4. **N45** risponderà a **N80**.

La ricerca nei nodi impiega un tempo nell'ordine di $O(log(N))$, in quanto ad ogni step la distanza tra la query e il peer col file si riduce di un fattore di almeno 2. Dopo una serie di inoltri dell'ordine di $log(N)$, la distanza dalla chiave è all'incirca pari a $2^m/2^{\,log(N)} = 2^m/N$.

Il numero di identificatori dei nodi nel range $2^m/N$, è con elevata probabilità dell'ordine di $O(log(N))$.

- 2^m erano appunto i posti a sedere disponibili nel ring;
- N il numero di nodi effettivamente presenti.

Gestione dei malfunzionamenti in Chord

Quando un nodo in Chord fallisce, esce logicamente dalla rete e questo non può che portare a una visione distorta della ring table, che dunque va sistemata.

Problema 1

Per esempio, supponendo di avere tre nodi: N16, N32 e N45. Nell'ipotesi che ad andare giù sia N32, ciò implica che N16 non può in alcun modo contattare N45 per chiedergli dei file.

Una soluzione a questo problema potrebbe essere quella di mantenere una serie r di successori, e in caso di fallimento del nodo immediatamente successivo, si possono utilizzare i successori.

Ma quanto deve essere grande r? Si potrebbe pensare di scegliere un valore pari a $r = 2log(N)$.
Supponendo che il 50% dei nodi vadano giù, la probabilità di un dato nodo per la quale almeno **un** successore rimanga in vita è pari a:

$$P(E) = 1 - \left(\frac{1}{2}\right)^{2\log_2 N} = 1 - \frac{1}{N^2}$$

Ovvero, 1 meno la probabilità che tutti i successori si siano guastati. Poiché si suppone il 50%, si mette ½.
La probabilità che quanto detto sia **vero** per tutti i nodi in vita è data da:

$$\left(1 - \frac{1}{N^2}\right)^{N/2} = e^{-\frac{1}{2N}} \simeq 1$$

Problema 2

Un altro problema potrebbe essere legato al fatto che ad andare giù sia proprio il nodo provvisto del file da fornire. In tale caso, una soluzione a questo problema potrebbe essere quella di duplicare il file su più nodi, in particolar modo a r successori e predecessori.

Aggiornamento della finger table

Quando un peer entra all'interno della rete, si mette in contatto con un server DNS che gli fornisce un paio di indirizzi IP legati rispettivamente al successore e al predecessore.

Sulla base del numero assegnato, deve presentarsi all'immediato predecessore e successore.

Esempio
Supponendo che il nuovo nodo sia **N40**, l'introducer presenta quest'ultimo a **N45** e **N32**. In tal contesto avviene che:

1. **N32** aggiorna il suo successore a **N40**, mentre in precedenza era **N45**;
2. **N40** inizializza come suo successore **N45** e inizializza la sua finger table da esso;
3. Sempre **N40**, comunicherà periodicamente con nodi vicini per aggiornare la sua finger table.

La comunicazione periodica con i nodi vicini, al fine di aggiornare la finger table è offerta mediante un determinato **protocollo di stabilizzazione**. Tale protocollo viene fatto girare su tutti i nodi e non solo su quello che si sta presentando al momento.

Anche i file dei successori devono essere copiati. A tale scopo, **N40** dovrà copiare dei file da **N45**, il quale contiene i duplicati di **N32** e ora di **N40**.

Il nuovo peer che fa il join avrà un impatto di discussioni con altri peer pari a $log(N)$, poiché deve aggiornare le proprie finger table, le quali sono sempre nell'ordine di $log(N)$, mentre il numero di messaggi per ogni peer che esegue è il join nella rete è dell'ordine di $O(log(N) * log(N))$.

La stabilità forte viene ottenuta con round nell'ordine di $O(N^2)$, poiché ciascuno dei nodi dovrà dare un'informazione su ciascuno degli altri nodi presenti nella rete. Si osservi che, a differenza del logaritmo, N^2 è una condizione **deterministica**.

Churn Rate

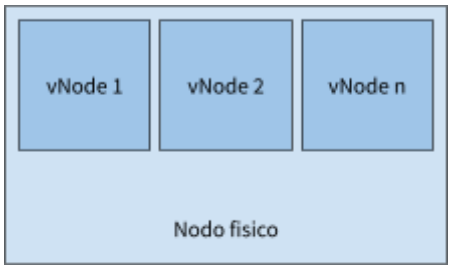
È opportuno effettuare un'operazione di failure detection qualora la rete debba aggiornarsi circa l'abbandono di uno o più peer da essa. Il **tasso di abbandono** viene definito **churn** rate; nei sistemi P2P il tasso di abbandono è molto alto.

Load balancing

Il load balancing si riferisce al processo di distribuzione del carico di lavoro dello spazio delle chiavi in modo uniforme tra i nodi del sistema. Ciò è importante perché se lo spazio delle chiavi non è bilanciato, alcuni nodi possono diventare sovraccarichi con una quantità sproporzionata di responsabilità, portando ad un aumento della latenza e potenziali fallimenti del sistema.

Nodi virtuali

In pratica, ogni nodo fisico in Chord viene mappato a un insieme di nodi virtuali, ognuno responsabile di una porzione di intervallo di chiavi, quindi tra loro sono considerati indipendenti.

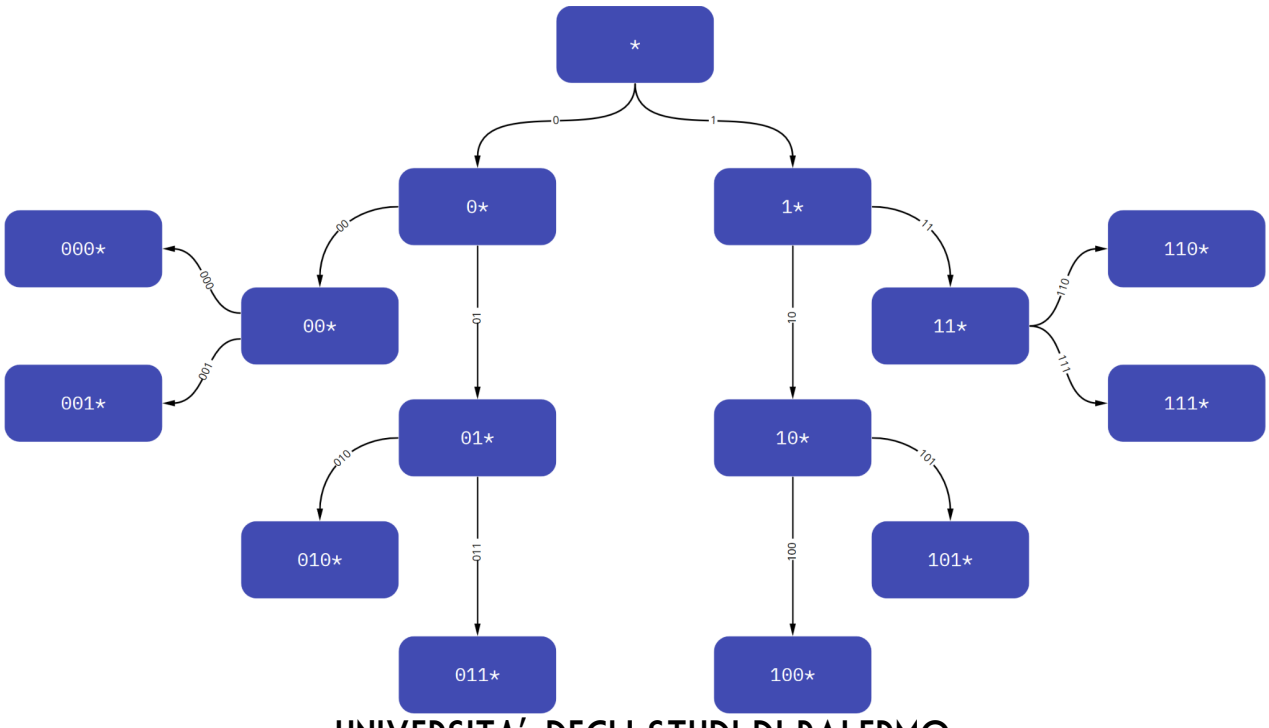


Confronto delle prestazioni:

Protocollo	Memoria	Lookup Latency	# Messaggi per lookup
Napster	O(1) (O(N) per server)	O(1)	O(1)
Gnutella	O(N)	O(N)	O(N)
Chord	O(log(N))	O(log(N))	O(log(N))

Pastry

Realizzato da Microsoft e dalla Rice University, è un **protocollo peer-to-peer strutturato** molto simile a Chord. Infatti, Pastry assegna dei ID ai nodi proprio come fa anche Chord, sfruttando una logica strutturata ad anello. Anche su Pastry ogni nodo puó avere piú di 1 successore ma, a differenza di Chord, Pastry utilizza il principio del **Leaf-Set** che permette ai nodi di tenere conto sia dei loro successori ma anche dei precessori in maniera diversa rispetto a come fa Chord. La differenza principale consiste quindi nella gestione della tabella di routing. Il routing si basa sul prefix matching implementato seguendo una logica strutturata a ipercubo.



UNIVERSITA' DEGLI STUDI DI PALERMO

Figura 1 Leaf-Set

Pastry è progettato per consentire la comunicazione e la ricerca efficiente di risorse e per gestire un gran numero di nodi all'interno di una rete P2P. **Utilizza un indirizzamento strutturato basato su un insieme di ID a lunghezza fissa rappresentato come una stringa di bit, i dati sono salvati sfruttando una DHT sottoforma di coppia chiave valore.**

La ricerca in Pastry avviene attraverso un meccanismo chiamato **proximity routing**. Quando un nodo desidera inviare un messaggio a un altro nodo con un determinato ID, utilizza la tabella di routing locale per instradare il messaggio verso il nodo che si avvicina di più all'identificatore di destinazione. Per far ciò, ogni nodo in Pastry dispone di una tabella di routing che gli permette di instradare i messaggi verso il nodo che si avvicina di più all'identificatore di destinazione, questa tecnica è chiamata **prefix matching** e il suo ruolo è quello di cercare il prefisso comune tra l'identificatore di destinazione e gli identificatori dei nodi nella tabella di routing. Il messaggio viene quindi passato in avanti da nodo a nodo fino a raggiungere il nodo di destinazione.

Per eseguire un prefix matching sugli indirizzi binari, si cerca un prefisso comune tra l'indirizzo cercato e gli indirizzi nell'insieme. Il prefisso comune può essere di lunghezza variabile e corrisponde alle prime cifre dell'indirizzo. Ad esempio, se stiamo cercando gli indirizzi binari che abbiano un prefisso di 010, il prefix matching restituirebbe gli indirizzi 010110, 010101 e tutti gli altri indirizzi che iniziano con 010. Gli indirizzi come 011001 e 001100 non sarebbero considerati in quanto non iniziano con il prefisso cercato.

La ricerca anziché avvenire per mezzo dell'approccio $(N + 2^i)(mod\ 2^m)$ visto per Chord, si ha un ordine di $O(log(N))$.

Per riuscire a trovare il percorso più veloce viene utilizzata un'altra tecnica chiamata **distanza di Hamming** utilizzata in Pastry come una misura per determinare la prossimità tra gli identificatori dei nodi. I nodi con identificatori che hanno una distanza di Hamming inferiore sono considerati più vicini tra loro nella rete Pastry. Per calcolare la distanza di Hamming, si confrontano le due stringhe posizione per posizione e si contano il numero di posizioni in cui i simboli sono diversi. Ad esempio, se si hanno due stringhe di nuemri bionari come quelle riportate sotto, la distanza di Hamming tra di loro sarà 2, perché le due stringhe differiscono in due posizioni.

"01**0**10**1**"

"01**1**10**0**"

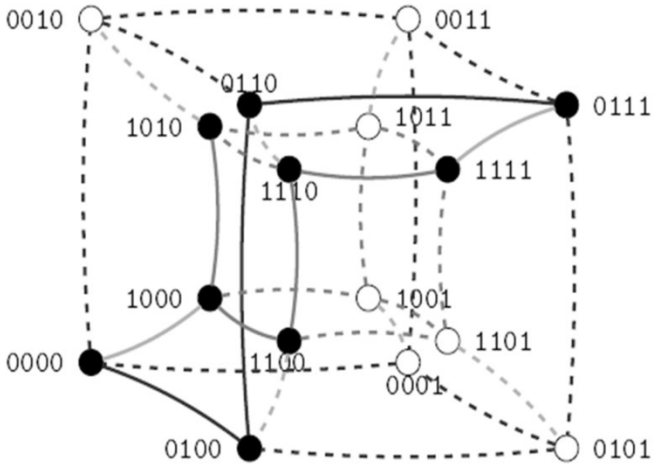


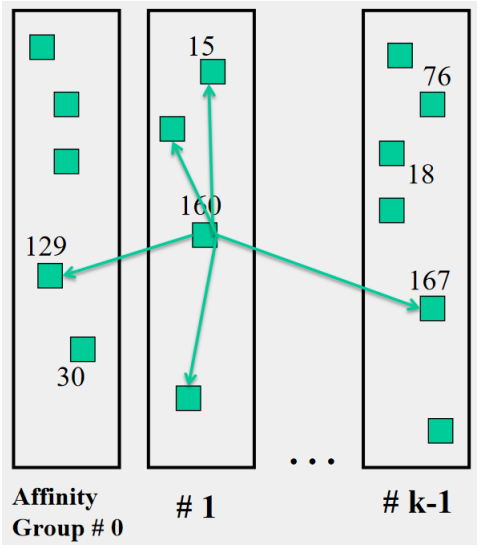
Figura 2 Ipercubo

Quindi il prefix matching viene impiegato per capire in che area bisogna spostarsi per avvicinarsi al nodo di destinazione, mentre la distanza di Hamming per trovare il percorso più breve.

Pastry gestisce in modo efficiente i collegamenti e il routing anche quando i nodi entrano o escono dalla rete, mantenendo l'efficienza del routing con un numero limitato di informazioni di routing da memorizzare in ogni nodo.

Il protocollo Pastry offre una serie di proprietà desiderabili per le reti P2P, come l'affidabilità, l'efficienza del routing, la scalabilità e la tolleranza ai guasti. È stato utilizzato come base per la progettazione di vari sistemi distribuiti, compresi servizi di archiviazione distribuita, reti di condivisione di file e applicazioni di trasmissione di streaming.

Kelips



Kelips, è un altro **protocollo peer-to-peer ibrido** così come BitTorrent, dove si usano i cosiddetti **affinity groups**. Questi consistono in k gruppi **virtuali**, e tale valore è pari a:

$$k \sim \sqrt{N}$$

Dove N è il numero di nodi che fanno parte della rete. Per determinare come i nodi vadano suddivisi nei vari gruppi, si calcola l'**hash** mediante l'algoritmo **SHA-1** di ciascuno di essi e si effettua l'operazione modulo per

$$hash \mod \sqrt{N}$$

il valore di k che è l'arrotondamento al valore intero più vicino della radice quadrata di N. Ciascun peer conosce **tutti** gli altri che fanno parte del medesimo affinity group. Nel caso della figura sottostante, il peer con ID 160 conosce gli altri tre peer che fanno parte dell'affinity group #1, ad esempio 15.

Approfondimento ricevimento

Quando in Kelips un nuovo nodo vuole essere aggiunto, viene fatto su di esso (probabilmente su ip e porta) un hash di tipo SHA-1, questo viene poi moltiplicato per un parametro k che è un numero intero (dunque approssimato) ottenuto dalla radice quadrata del numero totale di nodi che fanno parte della rete, il numero che uscirà fuori è proprio il valore k esimo del gruppo nel quale il nodo verrà collocato

Inoltre, in ciascun affinity group esiste un cosiddetto **contact member peer**, che permette di introdurre l'affinity group ad altri peer che non fanno parte del medesimo.

Con riferimento alla figura a destra, nel caso di #0, il contact member sarà il peer 129, nel caso di #1 il peer 160, nel caso di # k-1 il peer 167.

Approfondimento ricevimento

Se il **contact member** va offline, la rete si predispone all'elezione di un nuovo peer al quale bisognerà fornire le più aggiornate liste dei file presenti all'interno del suo affinity group ma anche tutte le informazioni per poter rintracciare i contact member degli altri gruppi

Memorizzazione dei file

A differenza di Chord e Pastry, l'approccio di memorizzazione dei file in Kelips è simile a quanto visto in Gnutella e Napster. Gli affinity groups conservano solo ed esclusivamente i metadati correlati ad un dato file, mentre il file vero e proprio viene memorizzato in quel nodo che per primo lo mette a disposizione o in nodi che contengono il file replicato.

Per ciascun filename viene quindi calcolato l'hash, associato a un gruppo e ciascun peer del gruppo replica i metadati, costituiti da: <*filename, file location*>. **Gli affinity groups NON memorizzano i file**.

Lookup

Il lookup di un file funziona nel seguente modo:

1. Si calcola l'hash del filename da ricercare e si individua a quale affinity group potrebbe appartenere;
2. Ci si rivolge al contact member dell'affinity group associato;
3. Il contact group **sa poi come fornire il peer contenente quel dato file;**
4. Se tale lookup fallisce, bisogna rivolgersi dai peer vicini per cercare un nuovo contatto.

Il costo del lookup è di **un round trip time**. La tabella di lookup è nell'ordine dei MB, quindi è facilmente contenibile nella memoria RAM di molti laptop o workstation moderne. Bisogna però sottolineare che in presenza di un numero elevato di peer, quindi N molto grande, è opportuno disporre di grandi quantità di memoria.

Aggiornamento vicini e file

L'aggiornamento della membership list avviene per mezzo di un meccanismo **gossip-based**, come quello visto nei capitoli precedenti, per ciascun affinity group e tra affinity group (contact member).

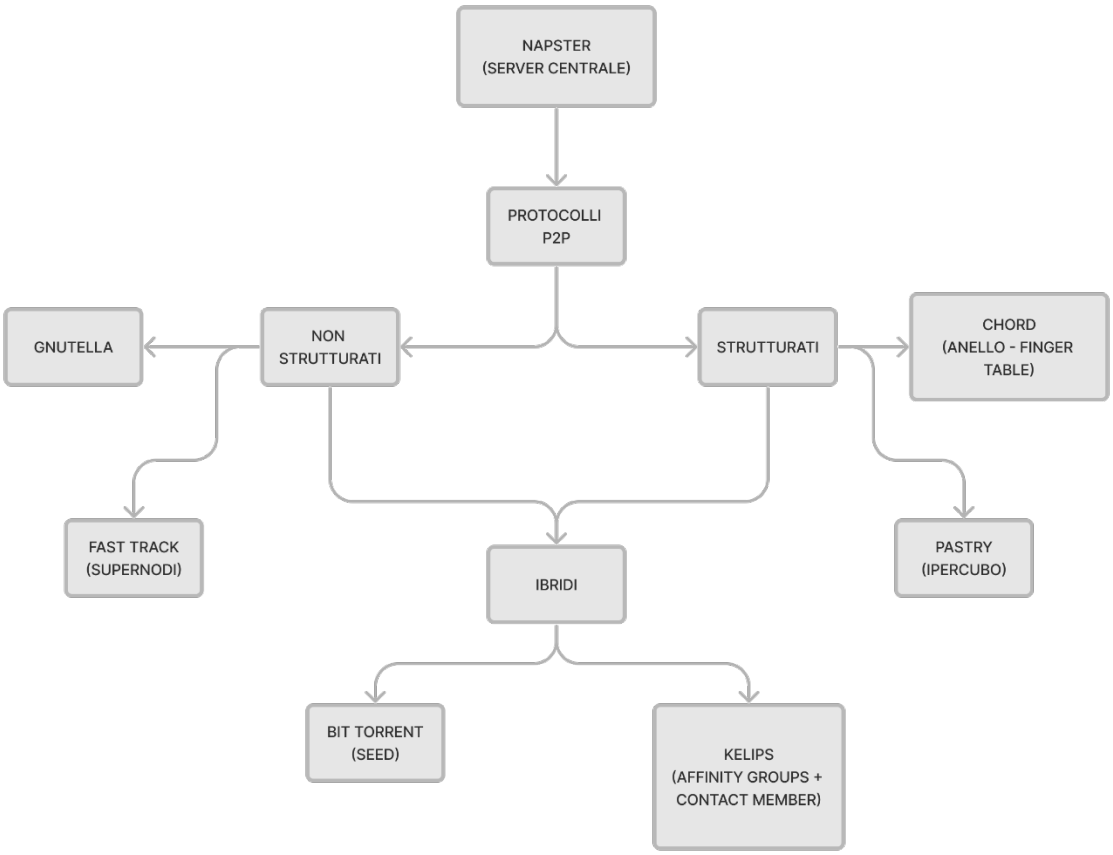
Il **tempo di disseminazione**, in particolare quando un nodo fallisce, è nell'ordine di $O(log(N))$.

Anche i metadati dei file devono essere aggiornati periodicamente, in quanto prima o poi scadono. **Quando si vuole eliminare un file dalla rete, basta rimuoverlo dalla lista dei file presenti nell'affinity group** che ciclicamente viene inoltrata dal contact member peer agli altri contact member degli altri gruppi. Erroneamente viene detto che i file è come se avessero degli heartbeat counter, ma ciò non è corretto.

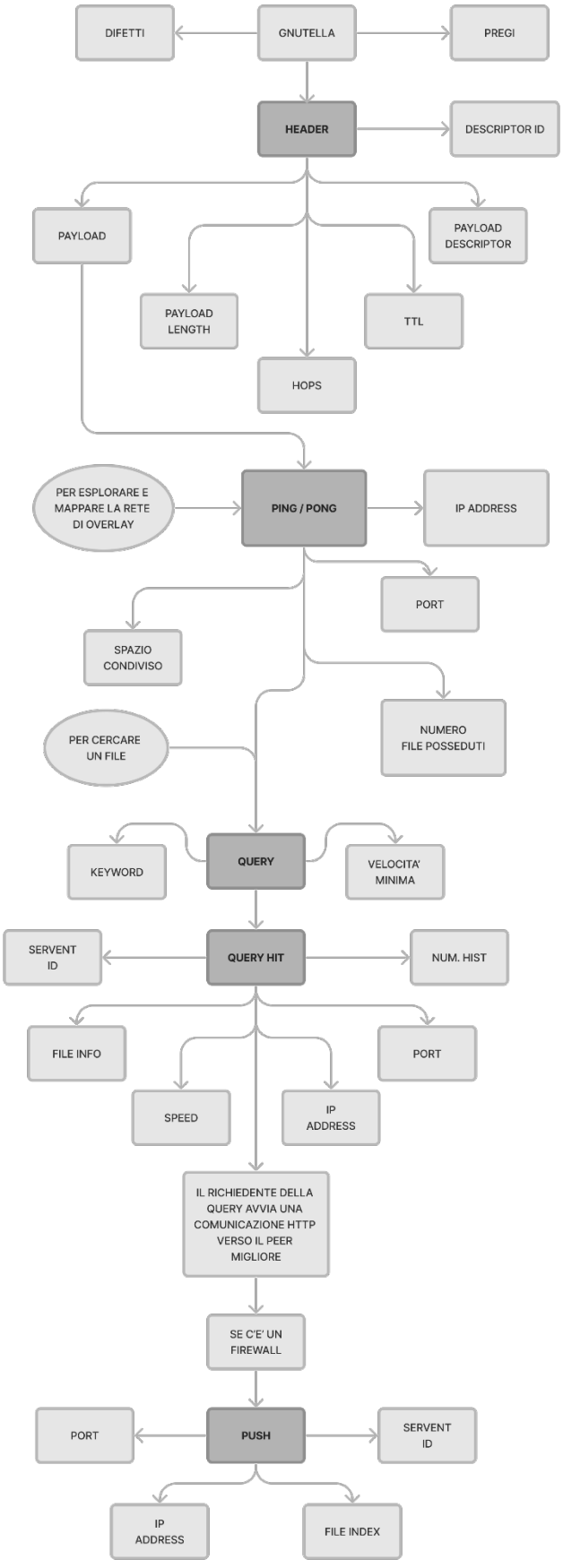
Confronto con Pastry e Chord

Kelips utilizza molta più memoria di Pastry e Chord, molta più banda, nonostante i costi di lookup siano di gran lunga inferiori ($O(1)$ per lookup e $O(\sqrt{N})$ contro $O(log(N))$).

Mappa Capitolo



Mappa Gnutella



IVAN SOLLAZZO - ANDREA PISCOPO - NICOLAS BRUNO
PROFESSORE PIERLUIGI GALLO - INGEGNERIA CIBERNETICA
UNIVERSITA' DEGLI STUDI DI PALERMO