

Sistemi distribuiti peer-to-peer

Rocco Lo Russo

roc.lorusso@studenti.unina.it

Università di Napoli Federico II — 17/12/2025

Introduzione

In questo documento verranno approfonditi i sistemi peer-to-peer, seguendo il materiale didattico fornito dal professor Stefano Russo durante il corso di Sistemi distribuiti e IOT erogato nell'anno accademico 2025/26 presso l'università degli studi di Napoli Federico II.

1 Motivazioni e caratteristiche

1.1 Contesto storico

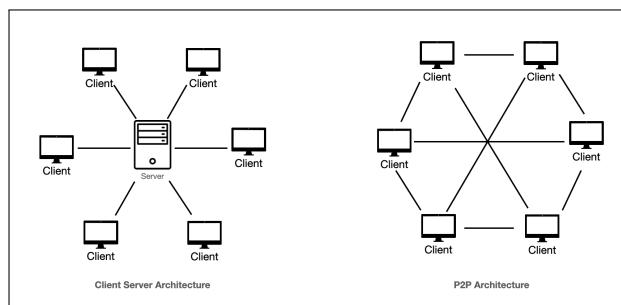
I sistemi P2P mirano a condividere risorse su vasta scala come scambio di informazioni , cicli di CPU e spazio su disco. Negli anni '90 e 2000, la potenza dei PC domestici (CPU e spazio disco) cresceva più velocemente della larghezza di banda centrale. La maggior parte di queste risorse rimaneva inutilizzata (idle). Invece di trattare i PC degli utenti come *clienti passivi* che richiedono solo dati, il P2P li trasforma in nodi attivi che contribuiscono. Un cambio di paradigma fondamentale è che non vi è centralizzazione di attività in specifici nodi: non c'è un'autorità centrale che deve gestire account, permessi o manutenzione hardware centralizzata. Il sistema si "auto-gestisce" ed è capace di auto-organizzarsi.

1.2 Caratteristiche dei sistemi P2P

Un sistema P2P è un sistema distribuito nel quale i nodi hanno simili responsabilità e capacità funzionali. Non vi è centralizzazione di attività in specifici nodi, e la cooperazione tra nodi avviene mediante interazioni tra pari. Questo modello trova principale applicazione nei sistemi di condivisione risorse, siano queste di tipo computazionale, informatico o multimediale. Nel modello P2P ogni nodo contribuisce alla gestione delle risorse globali, utilizzando servizi offerti dagli altri nodi ed offrendo, in cambio, servizi analoghi.

1.3 P2P vs Client-Server

Nelle architetture client-server, vi è una marcata asimmetria tra chi fornisce il servizio e chi lo richiede: i client sono iniziatori passivi di una richiesta, e non condividono le proprie risorse di rete, mentre il server è l'autorità centrale che gestisce dati e sicurezza. Per quanto riguarda le architetture distribuite di tipo peer



to peer, i nodi (chiamati peer) hanno "simili responsabilità e capacità funzionali". Non c'è distinzione fissa: ogni nodo agisce sia da client (quando scarica) che da server (quando invia); La cooperazione avviene direttamente tra pari. Ogni nodo contribuisce alla gestione delle risorse globali. La localizzazione e la gestione delle risorse risulta molto più semplice, di conseguenza, nell'approccio centralizzato, mentre per quanto riguarda l'approccio distribuito sono necessari algoritmi complessi sia per quanto concerne il lookup (localizzazione) che la gestione. Il limite più grande dell'architettura centralizzata è sicuramente la scalabilità: se il numero di client aumenta, il carico sul server centrale cresce linearmente fino a saturazione (questo rappresenta un collo di bottiglia), e ciò implica che l'aggiunta di nuovi client consuma risorse. L'approccio distribuito invece consente elevata scalabilità: l'aggiunta di un nuovo nodo porta nuove risorse (CPU, disco, banda) al sistema. Più utenti ci sono, più potente diventa il sistema. Infine, per quanto riguarda la tolleranza ai guasti, l'approccio client-server puro rappresenta un *single point of failure*: se il server centrale cade, il servizio si interrompe per tutti i client e dunque l'affidabilità è limitata dalla disponibilità del server; Nell'architettura P2P il sistema è progettato per gestire il *churn* (nodi che si connettono e disconnettono in modo imprevedibile), e l'affidabilità è ottenuta replicando percorsi e oggetti su più peer. Se un peer cade, altri possono fornire la stessa risorsa. Quanto discusso in questa sezione è riassunto in tabella 1.3.

Caratteristica	Client-Server	Peer-to-Peer
Struttura	Centralizzata	Distribuita
Posizione Risorse	Risiedono nei Server centrali	Condivise tra i nodi (peer)
Responsabilità	Asimmetrica: il Server gestisce, il Client utilizza	Simmetrica: simili per tutti i nodi (servono e richiedono)
Scalabilità	Limitata: il server diventa un collo di bottiglia all'aumentare dei client	Alta: le risorse del sistema crescono organicamente con l'aggiunta di nuovi nodi
Ricerca (Lookup)	Semplice: l'indirizzo del server è noto	Complessa: richiede algoritmi distribuiti e Overlay Network
Costo Gestione	Alto: richiede infrastruttura dedicata e manutenzione server	Basso: sfrutta infrastruttura esistente (edge resources)
Tolleranza Guasti	Bassa: Single Point of Failure (se cade il server, cade il servizio)	Alta: resiliente grazie alla repli-cazione e auto-organizzazione



Info: L'**Affidabilità** (Reliability) misura la continuità del servizio corretto. È definita come la probabilità che un sistema svolga correttamente la sua funzione prevista, senza guasti, per un determinato periodo di tempo e sotto condizioni specifiche. Le metriche chiave sono MTTF (Mean Time To Failure), che misura quanto tempo un componente funziona correttamente prima di rompersi, e si usa principalmente per componenti **non riparabili** (es. una lampadina, un hard disk che si sostituisce) o per indicare solo la fase di *uptime* in un sistema riparabile, o MTBF (Mean Time Between Failures), che misura il tempo medio che intercorre tra l'inizio di un guasto e l'inizio del guasto successivo e si usa per sistemi riparabili (es. un server, un software che viene riavviato).

Se definiamo $\lambda = \frac{1}{MTTF}$, ovvero tasso di guasto (assunto come costante nel tempo), è possibile definire l'affidabilità mediante la funzione $R(t) = e^{-\lambda t}$, ovvero la probabilità che il sistema sia ancora funzionante al tempo t .

La **Disponibilità** (Availability) è definita come la probabilità che un sistema sia operativo e accessibile in un qualsiasi momento arbitrario in cui ne viene richiesto l'uso. La metrica chiave è la *percentuale* di uptime, e si calcola come $A = \frac{MTBF}{MTBF+MTTR}$, dove con MTTR intendiamo il tempo medio necessario per riparare il guasto.

2 Requisiti



Warning: I **requisiti funzionali** specificano *cosa* il sistema deve fare, in termini di azioni, mentre i **requisiti non funzionali** descrivono *come* il sistema deve essere, in termini di qualità. I requisiti non funzionali determinano anche i vincoli di progettazione.

2.1 Requisiti funzionali

I requisiti funzionali di un sistema P2P sono:

- Semplificare la realizzazione di servizi distribuiti;
- I client devono poter individuare e comunicare con i fornitori del servizio;
- Deve essere possibile aggiungere e rimuovere servizi;
- Deve essere possibile aggiungere e rimuovere host;
- Trasparenza del tipo di risorsa utilizzata.

Osserviamo che con *Trasparenza del tipo di risorsa utilizzata* intendiamo che il sistema P2P deve essere in grado di gestire diverse tipologie di risorse trattandole in modo uniforme. Il protocollo di base (le operazioni di ricerca, instradamento e connessione) non deve dipendere dalla natura specifica della risorsa. Per il sistema, una risorsa è un oggetto generico da localizzare e scambiare. L'applicazione che usa la rete P2P deve poter richiedere una risorsa senza doversi preoccupare di come quella specifica tipologia di dato è memorizzata o gestita fisicamente dall'altro pari.

2.2 Requisiti non funzionali

I requisiti non funzionali di un sistema P2P sono:

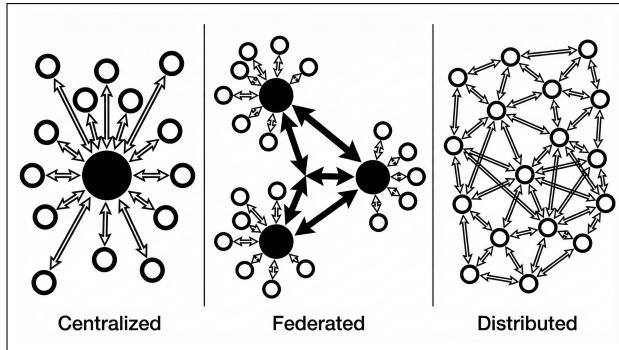
- Scalabilità (sfruttare tutti gli host);
- Bilanciamento del carico (da cui dipendono le prestazioni);
- Ottimizzazione delle interazioni tra pari;
- Disponibilità;
- Sicurezza (privacy e integrità delle informazioni);
- Anonimità e sollevamento dalle responsabilità di chi possiede e condivide dati.

È interessante osservare che la disponibilità non è un requisito banale da ottenere, in quanto il sistema P2P prevede che i nodi possano connettersi e soprattutto disconnettersi in qualsiasi momento, in maniera non controllata né predicable.

3 Caratteristiche dei sistemi P2P

3.1 Ciclo di vita

Un nodo (applicazione) che utilizza un sistema P2P svolge il seguente ciclo di vita: **Boot**, ovvero la fase in cui un'applicazione appena avviata intende far parte del sistema ottenendo informazioni per connettersi agli membri del sistema; **Lookup**, ovvero la fase in cui l'applicazione intende acquisire una risorsa, e quindi ricerca quali dei peer del sistema la detengono; **Resource sharing**, ovvero la fase in cui l'applicazione contatta un peer identificato nella fase precedente per acquisire una risorsa, ed effettua lo scambio dati.



3.2 Soluzioni architetturali

Il ciclo di vita presentato può essere coadiuvato da diverse scelte architetturali, più o meno complesse e in linea con i requisiti. Presentiamo dunque diverse scelte architetturali per la realizzazione di un sistema P2P.

Centralizzata	Esiste un server centrale a cui ogni peer chiede informazioni di <i>boot</i> e <i>lookup</i> .
Federata	Esiste un server per ogni gruppo di peer, e questi server sono interconnessi per garantire consistenza dei dati gestiti.
Distribuita	Non esiste un server dedicato, ma i dati sono distribuiti tra i peer e sono ottenuti tramite algoritmi distribuiti.

3.3 Tassonomia sistemi P2P

In base alle scelte architetturali, è possibile classificare i sistemi P2P come segue:

P2P Puro	Le fasi di boot, lookup e sharing sono P2P;
P2P	Le fasi di lookup e sharing sono P2P, mentre la fase di boot utilizza server centralizzati o federati;
P2P ibrido	Sharing in P2P, boot tramite server (centralizzati o federati), lookup tramite peer particolari (server federati).

In base alla topologia della rete i sistemi P2P sono classificati in **strutturati** e **non strutturati**.

Nei sistemi non strutturati, i peer si collegano tra loro in modo sostanzialmente casuale, senza seguire una regola geometrica o logica predefinita per la formazione della topologia. In questa configurazione, la rete si presenta come un grafo casuale in cui ogni nodo mantiene semplicemente una lista di vicini, e non esiste alcuna correlazione tra l'identificativo di un nodo e il contenuto che esso possiede o gestisce. La conseguenza diretta di questa assenza di struttura è che le operazioni di ricerca (lookup) risultano poco efficienti, poiché un nodo non sapendo "dove" si trovi un dato deve interrogare la rete *alla cieca*. Sebbene questo approccio soffra di scarsa scalabilità e non offra garanzie sui tempi di risposta, presenta il vantaggio di essere molto robusto e capace di auto-organizzarsi facilmente anche in presenza di frequenti connessioni e disconnessioni dei nodi. Al contrario, i sistemi strutturati impongono una rigida organizzazione logica alla rete attraverso una policy globale che definisce esattamente come i nodi devono collegarsi tra loro e dove devono essere posizionati i dati. Questa struttura, spesso definita overlay network, è costruita sopra il protocollo IP e utilizza algoritmi specifici (come le Distributed Hash Table o DHT) per associare in modo deterministico ogni risorsa a un preciso nodo della rete. Grazie a questa organizzazione, la ricerca diventa estremamente efficiente perché è possibile calcolare matematicamente quale nodo è responsabile di una certa risorsa, permettendo di raggiungerla con un numero di passaggi molto ridotto, tipicamente logaritmico rispetto al numero dei nodi. Il "prezzo" da pagare per questa efficienza è una maggiore complessità nel mantenere la struttura coerente quando i nodi entrano o escono dalla rete.

Generazione	Operazioni	Topologia	Esempi
Prima	P2P ibrido	P2P non strutturato	Napster
	P2P		Gnutella 0.4
Seconda	P2P ibrido		Gnutella 0.6, BitTorrent
Terza	P2P puro	P2P strutturato	Chord, Pastry, Freenet

4 P2P di prima generazione

I sistemi P2P di prima generazioni fanno fasi di boot e lookup centralizzate (o federate), mentre le fasi di sharing sono realizzate in modalità peer to peer. La topologia di rete è non strutturata.

4.1 Napster

Napster è un sistema per lo sharing di file musicali P2P ibrido con topologia di rete non strutturata, lanciato nel Giugno 1999. Il servizio fu disattivato nel 2001 per infrazione sulla legge per il diritto d'autore. Questa soluzione prevedeva lookup centralizzato, secondo la logica che il nodo che svolgerà la funzione di server per un client è quello a meno *hop* di distanza. In questo modo si evitava anche che il primo nodo a rendere disponibile un file diventasse *hot spot* per lo stesso file.

i **Info:** In un sistema distribuito, un *hot spot* si verifica quando un singolo nodo viene sovraccaricato da un numero eccessivo di richieste, diventando un collo di bottiglia che rallenta o blocca il servizio. Se non ci fosse alcun meccanismo di controllo, il primo nodo che condivide un file molto popolare (ad esempio una nuova canzone appena uscita) rischierebbe di essere contattato simultaneamente da migliaia di utenti, saturando la sua banda e rendendo il file indisponibile per tutti.

Quando un client cerca un file, il server centrale restituisce una lista di peer che detengono quella risorsa. Il sistema favorisce il download dal nodo che si trova a meno *hop* di distanza dal richiedente, sfruttando così la *località della rete*. Man mano che il file viene scaricato dai primi utenti, esso diventa disponibile su più nodi (replicazione). I futuri richiedenti non andranno tutti a bussare alla porta del "primo nodo", ma verranno indirizzati verso le nuove copie del file se queste si trovano su nodi a loro più vicini. I limiti di Napster erano che non c'era nessuna garanzia di disponibilità dei file e l'indice centralizzato. Infatti, sebbene il trasferimento dei file avvenisse direttamente tra utenti (P2P), la ricerca dipendeva interamente dai server centrali di Napster. Questo crea un Single Point of Failure: se i server dell'indice vengono spenti (come accaduto per via legale) o si sovraccaricano, l'intera rete smette di funzionare perché nessuno sa più "chi ha cosa". Inoltre, mantenere un indice globale di tutti i file posseduti da milioni di utenti richiede risorse server massicce. Un limite tollerabile (in quanto Napster serviva allo sharing di file musicali) era la consistenza debole: se un utente cancellava un file o andava offline, l'indice centrale poteva non aggiornarsi istantaneamente. L'utente poteva serenamente riprovare con un altro peer.

5 P2P di seconda generazione

I sistemi P2P di seconda generazioni hanno fase di boot centralizzata (o federata) mentre fase di lookup e sharing distribuite. Questi sistemi non presentano topologia di rete strutturata (overlay network senza organizzazione).

5.1 Gnutella 0.4

Sistema che raccoglie l'eredità di Napster, sul quale si sono basate applicazioni molto note. In questo sistema, la fase di boot è centralizzata (gnutellahost.com). La fase di lookup è distribuita, le richieste vengono propagate mediante tecniche di **flooding**. In fase di boot, i peer ricevono una lista degli identificativi dei vicini. Il lookup avviene senza un server, quindi un nodo invia una query a tutti i vicini, i quali inoltrano la richiesta ai propri vicini se non dispongono dell'informazione cercata. Il messaggio di query dunque si

propaga all'interno del sistema fino a raggiungere, se esiste, il nodo che detiene l'informazione. Questa tecnica di *inondazione* presenta l'inconveniente che i messaggi possono circolare nel sistema indefinitamente, ripassando per nodi già visitati. Le soluzioni possibili sono:

- Ogni nodo mantiene una lista di query già valutate, e qualora si ripresenti la stessa query (source, risorsa), viene semplicemente scartata;
- Ogni query ha un TTL, che viene decrementato ad ogni ricezione, cosicché quando il suo valore è nullo la query viene automaticamente scartata.

Sussiste ancora l'inconveniente che, siccome diverse repliche della stessa query sono in transito nel sistema, il mittente della richiesta potrebbe ricevere l'informazione più di una volta se nel sistema più di un nodo la detiene.

5.2 Gnutella 0.6

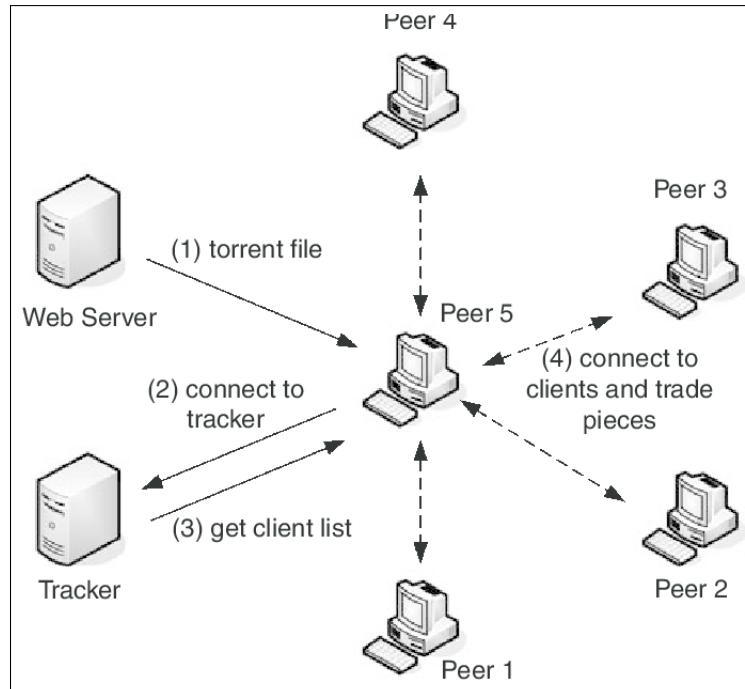
In questa versione di Gnutella, i peer sono distinti in *foglie* e *ultrapeer* (nodi con risorse aggiuntive). Ogni foglia è connessa ad alcuni ultrapeer (massimo 3), mentre gli ultrapeer sono connessi tra loro (almeno 32 ultrapeer in connessione). Il lookup avviene sfruttando gli ultrapeer come in tutti i sistemi p2p ibrido, sfruttando il **Query Routing Protocol**: Ciascun nodo produce una Query Routing Table (QRT), ovvero una tabella dove sono conservate le tuple (hash_value, risorsa_possesta). La QRT viene inviata agli ultrapeer associati, e ognuno di questi genera la propria QRT relativa ai propri file e unendo le QRT ricevute, e aggiungendo l'identificativo del peer che detiene la risorsa. Un nodo foglia invia una richiesta ad un ultrapeer per volta ed attende risposta per un certo tempo, prima di contattare un altro ultrapeer. Ogni query contiene l'indirizzo di rete del primo ultrapeer contattato dal nodo foglia, in modo che chi ha il file contatta direttamente quest'ultimo evitando di ripercorrere a ritroso tutto il percorso.

5.3 BitTorrent

Soluzione P2P di seconda generazione per il file sharing. Un server mantiene file *.torrent*, che contengono meta-information sui file da scaricare. Un *tracker* tiene traccia di tutti i peer che detengono un determinato file. Un *downloader* contatta periodicamente il tracker per aggiornare le sue informazioni e ricevere la lista di peer che posseggono il file in download, così da contattarli. Il protocollo può essere riassunto osservando il ciclo di vita di un client:

- **Fase di boot:** Il client si connette ad un server dove recupera un file *.torrent*, file contenente l'indirizzo del *tracker*;
- **Fase di lookup:** Il client contatta un *tracker*, dal quale ottiene una lista di tutti i peer che stanno condividendo il file in quel momento;
- **Fase di sharing:** P2P puro, il client contatta direttamente i peer (aggiornandoli periodicamente mediante *tracker*) per scambiare frammenti di file. Lo scambio segue algoritmi precisi per massimizzare l'efficienza.

Lo scambio di frammenti avviene secondo questo criterio: inizialmente, un peer chiede frammenti scelti a caso. Quando ne ha scaricati almeno 4, passa ad eseguire il *Rarest first algorithm*. Ogni peer mantiene una lista (*id_frammento, numero_peer*) in cui conserva per ogni frammento il numero di peer che lo detengono. I frammenti più rari sono memorizzati in una lista apposita, e da quella sono scaricati frammenti casuali. Quando un blocco di un frammento è stato scaricato, verranno richiesti anche gli altri con altissima priorità in modo da avere buona possibilità di scaricare tutto il frammento (Strict Priority Policy).



Entità	Ruolo e Funzionalità
Server (.torrent)	Mantiene i file con estensione .torrent, i quali contengono le meta-informationi necessarie sui file da scaricare, ma non il contenuto vero e proprio.
Tracker	È l'elemento di coordinamento che tiene traccia di tutti i peer che detengono un dato file (o parti di esso). Monitora i membri dello <i>swarm</i> e sa quali frammenti possiedono. Viene contattato periodicamente dai downloader per aggiornamenti.
Downloader (Peer)	Contatta il Tracker per ricevere la lista dei peer. Riceve e diffonde (upload) frammenti del file simultaneamente da/verso molteplici peer. Decide quale frammento richiedere usando algoritmi specifici.
Swarm	Rappresenta l'insieme collettivo di tutti i peer (sia chi ha il file completo che chi lo sta scaricando) che stanno condividendo un determinato file. Il Tracker tiene traccia dei membri dello swarm.

6 P2P di terza generazione

I sistemi P2P di terza generazioni hanno tutte le fasi completamente distribuite, ma con overlay strutturata secondo il paradigma DHT.

6.1 Overlay Network

Per migliorare l'efficienza delle operazioni di lookup è possibile considerare una rete logica costruita al di sopra del protocollo di rete IP che interconnette i vari nodi del sistema distribuito. Viene introdotta specificamente per migliorare l'efficienza delle operazioni di lookup, permettendo di utilizzare protocolli di routing specifici per l'applicazione invece del semplice routing IP. Si ottiene una policy globale, il che significa che la rete non si forma in modo casuale (come in Gnutella), ma segue un disegno preciso che stabilisce:

- **Topologia:** Come i nodi devono collegarsi tra loro (ad esempio formando un anello logico);
- **Placement degli oggetti:** Su quali nodi devono essere collocati i dati;
- **Routing:** Come devono essere instradati i messaggi per raggiungere la destinazione in modo efficiente.

Caratteristica	IP Routing	Overlay Routing
Scala	2^{32} (IPv4) o 2^{128} (IPv6) indirizzi. Indirizzi gerarchici e buona parte pre-allocata.	Maggiore spazio di indirizzamento [$> 2^{128}$] e assenza di vincoli.
Load Balancing	Il carico dei router dipende dalla topologia e dal relativo traffico.	Il carico è indipendente dalla topologia (oggetti disposti a caso).
Dinamiche di rete	Tabelle di routing aggiornate asincronamente con ritardi di circa 1 ora.	Tabelle di routing aggiornate in maniera sincrona o asincrona in frazioni di secondo.
Fault Tolerance	Ridondanza realizzata dagli ISP ed amministratori di rete costosa.	Replicazione di percorsi ed oggetti.
Identificazione destinazione	Ogni indirizzo IP identifica un singolo nodo.	Messaggi indirizzati verso la replica più vicina di un oggetto.
Sicurezza e anonimità	Sicurezza ottenuta solo se tutti i nodi sono fidati. Nessuna garanzia di anonimia.	Sicurezza ottenibile anche su reti non pienamente fidate. Si può ottenere un certo grado di anonimia.



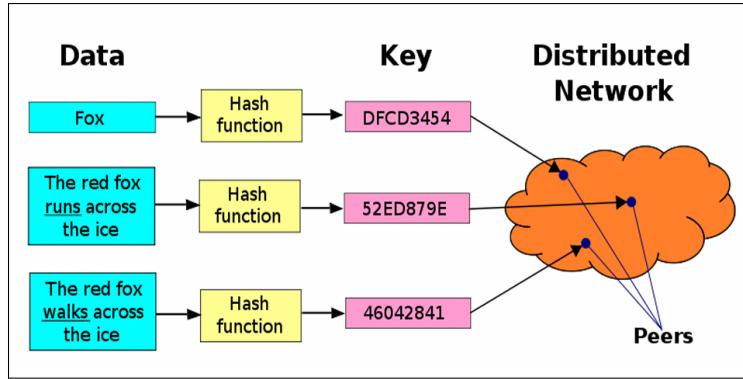
Warning: I protocolli di prima e seconda generazione non sono scalabili, in quanto le prestazioni (come ad esempio la ricerca di un file) peggiorano linearmente con l'aumentare del numero di nodi. Nella trattazione che segue, assumiamo che la scalabilità sia direttamente legata all'efficienza dell'algoritmo usato per il lookup.

Quello che si cerca di fare con l'introduzione di una rete logica strutturata è minimizzare il numero di messaggi necessari per fare lookup e minimizzare per ogni nodo le informazioni relative agli altri nodi.

6.2 Distributed Hash Table

I sistemi P2P strutturati fanno uso della DHT per associare agli identificatori globali unici dei file (GUID), generati con funzioni hash, la loro locazione.

Ad ogni file ed ad ogni nodo è associata una chiave, creata facendo l'hash del nome del file o dell'IP del nodo. Ogni nodo del sistema è responsabile di un insieme di file, ovvero un insieme di chiavi. Sia nodi



che risorse sono rintracciabili mediante chiave. Quando un nodo vuole scaricare un file, calcola l'hash del nome del file (la chiave). Non sapendo dov'è il file, interroga la rete chiedendo: "Chi è il responsabile di questa chiave?". Grazie alla struttura organizzata, la richiesta viene instradata in modo efficiente verso il nodo giusto. Per quanto presentato finora, non sappiamo come identificare il nodo responsabile di una determinata chiave. Esistono a questo proposito diversi protocolli.

6.3 Chord

Si basa su una struttura di rete logica ad anello con $N = 2^m$ GUID, dove m è il numero di bit con cui codificare la chiave. Il nodo responsabile di una determinata chiave K è quello con il primo identificativo W tale che W succeda K in senso orario. Ogni nodo conserva:

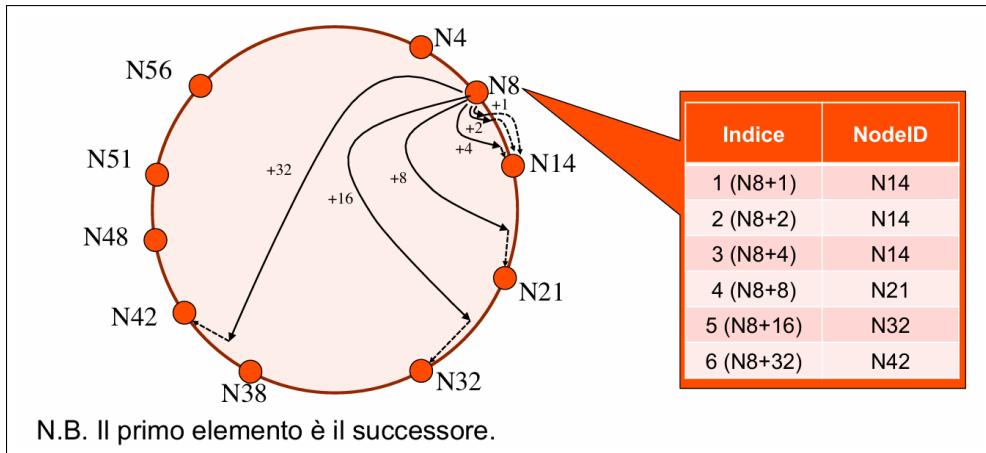
- Id del predecessore;
- Tabella dei successori;
- Tabella dei *fingers*.

La **Successor table** è utilizzata per ricerche nelle immediate vicinanze, e contiene gli id degli m successori e del predecessore. La **Finger table** è utilizzata per ricerche in nodi lontani, e contiene gli id degli m peer a distanza 2^i , con $0 \leq i \leq m - 1$. Ogni nodo che vuole trovare il responsabile della chiave K , chiama la procedura `find_successor(id)`.

```

1 n.find_successor(id){
2     // se id e' tra n ed il suo successor (primo elemento finger table)
3     if (id in (n, successor)){
4         return successor;
5     }
6     else{
7         n_1 = closest_preceding_node(id);
8         return n_1.find_successor(id);
9     }
10 }
11
12 n.closest_preceding_node(id){
13     for(int i = m; i>0; i--){
14         if (finger[i] in (n, id)) return finger[i];
15     }
16     return n;
17 }
```

Se l'identificatore è compreso tra la chiave del nodo e la chiave del successore (ovvero il primo elemento della finger table), allora ritorna il suo successore. Altrimenti si cerca il nodo la cui chiave si avvicina di più all'id desiderato senza però superarlo. Nella procedura `closest_preceding_node(id)`, ci si interroga "il finger in questione ha una chiave compresa tra la mia e la risorsa che sto cercando?"; In caso positivo, si itera la ricerca del successore a partire dal nodo trovato. La correttezza dell'algoritmo si basa sul fatto che ogni peer conosca il proprio successor, cosa non banale considerando che un sistema distribuito P2P



un nodo può disconnettersi in maniera non sistematica o non prevedibile. Per mantenere la correttezza dei dati di routing, sono eseguiti periodicamente degli appositi algoritmi di stabilizzazione. Se un nodo riceve una richiesta per una chiave K che è numericamente molto vicina al proprio ID (poco più avanti nell'anello), invece di calcolare la Finger Table, controlla direttamente la sua lista dei successori.

Per quanto riguarda la **complessità**, le informazioni che ogni nodo deve mantenere sugli altri nodi sono $2m+1$, ovvero $O(\log N)$. L'operazione di lookup richiede m messaggi ovvero $O(\log N)$, e il costo per la riconfigurazione della rete è $O((\log N)^2)$.

6.4 Pastry

Anche Pastry utilizza una overlay network ad anello, ma sfrutta il meccanismo del *prefix matching*. Lo spazio delle GUID è circolare (predecessore di 0 è $2^{128} - 1$). Ad ogni nodo è associato un GUID a 128 bit, ottenuto applicando una funzione di hash alla chiave pubblica. Ad ogni file è associato un GUID a 128 bit, ottenuto applicando una funzione hash al nome del file stesso. I GUID sono distribuiti in maniera casuale tra 0 e $2^{128} - 1$. Per determinare i nodi vicini, pastry utilizza metriche di località (Round trip latency o hop count), in modo da sfruttare la vicinanza numerica dei nodi. Ogni query di un dato con chiave K viene instradata verso il vicino la cui GUID è numericamente più vicina a K. Per fare ciò, ogni nodo dispone delle seguenti strutture:

Leaf Set (L) Contiene (GUID, IP) dei peer più vicini: l con GUID maggiore ed l con GUID minore.

Routing Table (R)
Strutturata ad albero, contiene (GUID, IP).

La routing table è organizzata come una matrice: Le righe indicano la lunghezza del prefisso in comune tra il nodo corrente e i nodi elencati in quella riga, mentre le colonne sono tante quanto la base numerica, e ogni colonna rappresenta il valore della prossima cifra. In figura vediamo un esempio di routing table con prefisso 65A1.

0	0	1	2	3	4	5	6	7	...	E	F
	GUID, IP		GUID, IP	GUID, IP	GUID, IP	GUID, IP					
1	60	61	62	63	64	65	66		...	6E	6F
	GUID, IP		GUID, IP								
2	650	651	652		...		65A	65B	...	65E	65F
3	65A0	65A1	65A2							65AE	65AF

L'algoritmo di routing funziona così: supponiamo che un nodo A riceva un messaggio riguardante la chiave D. Innanzitutto, controlla se la chiave D cade nell'intervallo coperto dal suo Leaf set, e se la risposta è positiva semplicemente consegna il messaggio al nodo numericamente più vicino a D; altrimenti il nodo

calcola quante cifre sono condivise tra la chiave del nodo stesso e D; supponendo condividano l cifre, controlla la riga l della routing table, alla $l+1$ -esima colonna, e se c'è un nodo inoltra il messaggio a quel nodo (guadagnando una cifra in più di corrispondenza); se invece la entry è vuota, inoltra il messaggio ad un nodo qualsiasi che condivide un prefisso lungo almeno quanto il suo, e che sia numericamente più vicino a D. Pastry utilizza un meccanismo di **HeartBeat** per rilevare fallimenti e riparare i leaf set: i peer si scambiano periodicamente messaggi di ping, e il ping che non risponde entro un certo timeout è considerato fallito. Il nodo che ha scoperto il fallimento, contatta il peer numericamente successivo a quello fallito, chiedendo una copia del suo leaf set; il leaf set ricevuto avrà delle sovrapposizioni con quello in possesso dal peer che ha iniziato la procedura, ma avrà peer validi per sostituire uello fallito. I peer vicini vengono informati del fallimento ed eseguono la stessa procedura. Per quanto riguarda la **complessità**, le entry che ogni nodo deve mantenere per quanto riguarda la tabella di routing sono $16 \log_1 6N \rightarrow \log N$. Analogamente per il leaf set. L'operazione di lookup richiede $O(\log N)$, e il costo per la riconfigurazione della rete è $O(\log N)$ messaggi.