Conseguenze Algoritmiche del fenomeno small world

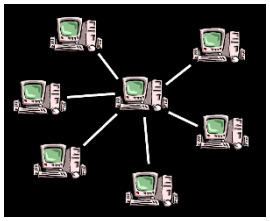
Peer-to-Peer (P2P)

Il termine Peer-to-Peer (P2P) si riferisce ad un'architettura logica di rete in cui i nodi non sono gerarchizzati sotto forma di client o server, ma sotto forma di *nodi equivalenti* o *peer* che possono cioè fungere sia da cliente che da server verso gli altri host della rete, ogni nodo ha quindi identiche capacità e responsabilità e tutte le comunicazioni sono potenzialmente simmetriche.

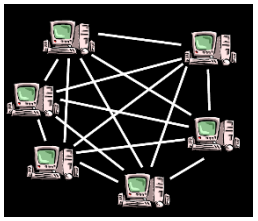
Grazie al modello P2P, i computer possono comunicare e condividere i file e altre risorse, invece di passare attraverso un server centralizzato. Ciascun computer (nodo) è responsabile del passaggio dei dati alle altre macchine.

Le connessioni non nascono spontaneamente, ma devono essere richieste da una delle parti in causa.
Attraverso un client scaricato gli utenti si connettono alla rete. Dopo la connessione, ha inizio la comunicazione tra i nodi che avviene tramite uno scambio di messaggi.
I messaggi servono a:

- segnalare la propria presenza sulla rete
- chiedere una o piu' risorse risorsa
- servire la richiesta di una o piu' risorse
- trasferire le risorse

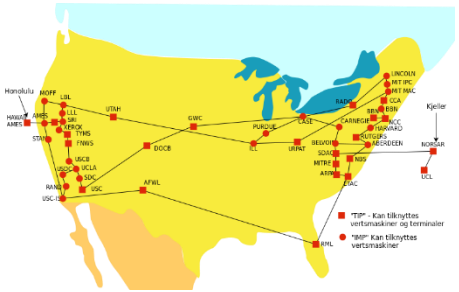


Client-server



Peer-to-peer

Quello di nodi paritari non è un concetto recente, infatti ARPAnet consisteva di nodi paritari



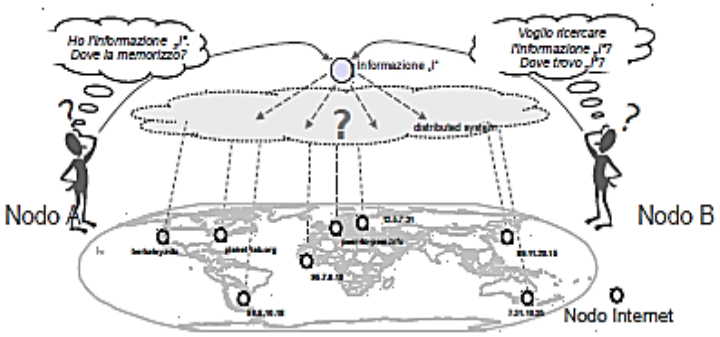
L'introduzione del Web e la grande differenza, in termini di prestazioni, fra macchine "Client" e "Server" ha spostato l'attenzione verso i sistemi Client\Server;

Le reti peer-to eer sono state sviluppate nel corso degli anni '90 e sono state utilizzate principalmente all'interno delle singole aziende per condividere informazioni tra ricercatori.

L'interesse verso questo tipo di protocolli è aumentato con la nascita dei primi sistemi per file-sharing (Napster (1999), Gnutella(2000));

- Nel 2000, 50 milioni di utenti hanno scaricato il Client di Napster;
- Napster ha avuto un picco di traffico di circa 7 TB in un giorno;

Come si trovano i file nelle reti peer-to-peer?



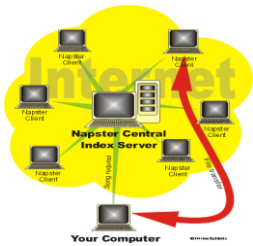
Lookup. L'operazione di lookup è utilizzata per individuare i dati che sono distribuiti tra una collezione di macchine.

Limitiamo il problema al compito comune di cercare dati che sono associati con una chiave di ricerca univoca (non, ad esempio trovare tutti gli elementi il cui contenuto ha alcune proprietà). La chiave unica ci deve far sapere su quale nodo (potenzialmente uno tra migliaia di nodi) si trovano i dati cercati.

La sfida è trovare un modo per individuare un nodo che memorizza i dati associati alla key in modo distribuito e *scalabile*.

Scalabile: Il lavoro richiesto ad un nodo nel sistema non deve crescere (o almeno cresce lentamente) in funzione del numero di nodi nel sistema.

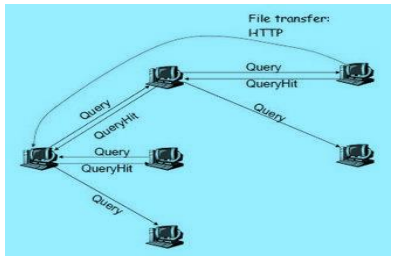
Lookup. Ci sono tre approcci di base che possono essere adottati per localizzare i dati:



Coordinatore centrale. Questo metodo utilizza un server che si occupa di individuare risorse che sono distribuite tra un insieme di server.

Napster è un classico esempio di questo.

Il Google File System (GFS) è un altro.



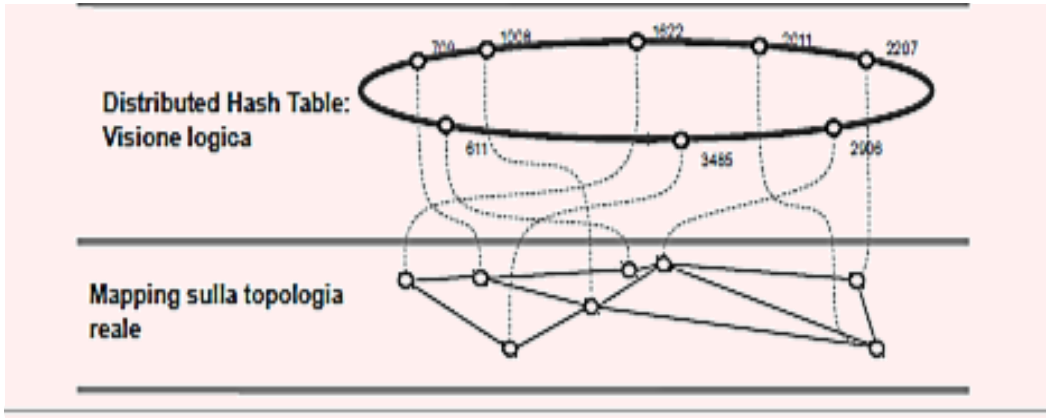
Flooding (Allagamento) . Questo metodo si basa sull'invio di query ad un grande insieme di macchine (potenzialmente tutte le macchine della rete), al fine di trovare il nodo che ha i dati di cui abbiamo bisogno.

Gnutella è un esempio di questo per la condivisione di file peer-to-peer.

Distributed Hash Tables *(DHT-* tabelle hash distribuite). Questa tecnica si basa sull'uso di tecniche di hashing della chiave per individuare il nodo che memorizza i dati associati. Ci sono molti esempi di questo, tra cui Chord, Amazon Dynamo, CAN, e Tapestry.

In una DHT, ad ogni risorsa e ad ogni nodo è associata una chiave (Key), tale chiave viene creata facendo l'hash del nome della risorsa o dell'IP del nodo.

Ogni nodo del sistema è responsabile di un insieme di risorse/chiavi



Chord

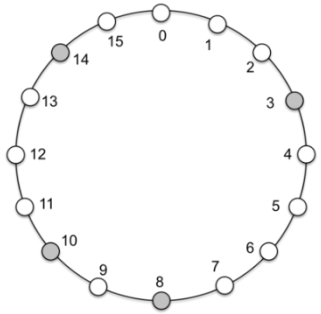
Il protocollo *Chord* mappa una key (che coincide con il nome del file) in un nodo:

- Le chiavi sono i file che stiamo cercando.
  Il computer che mantiene la chiave può poi puntare alla vera posizione del file.

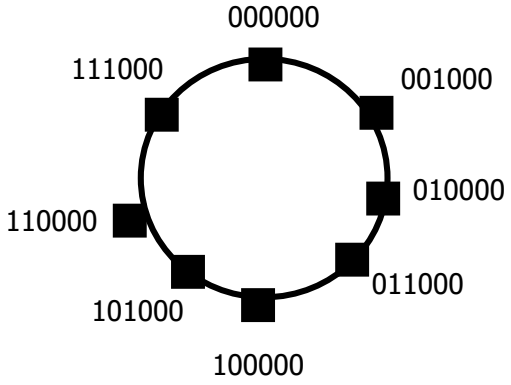
- Chiavi e nodi hanno ID di m bit loro assegnati :
  - ID del nodo è un hash-code dell'indirizzo IP (32 bit)
  - ID della chiave è un hash-code della chiave che identifica il file

  NOTA: gli ID dei nodi e gli ID delle chiavi sono nello stesso spazio.

Chord su ring. Le ID sono ordinate lungo un anello. Poiché in uno spazio di ID di *m* bit, ci sono 2^m identificatori, le ID sono quindi ordinate lungo l'anello modulo 2^m. L'anello delle ID è chiamato *anello Chord.*

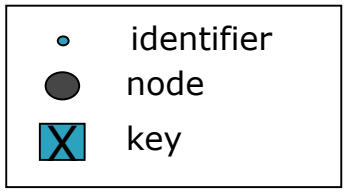


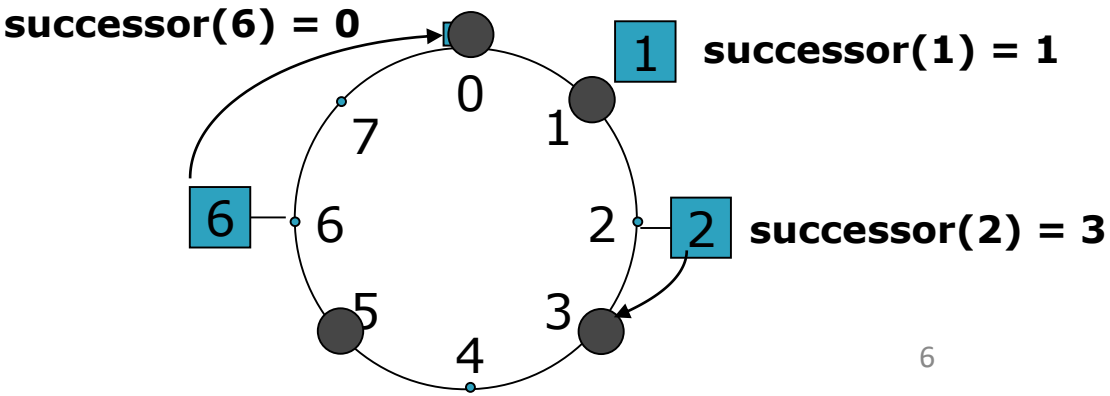
Un anello Chord con m=4. Abbiamo quindi 16 identificatori da 0 a 15 disposti lungo l'anello; solo a 4 di tali ID (i nodi pieni 3, 8, 10 e 14) corrispondono a 4 macchine effettivamente presenti nella rete e mappate su tali nodi.



Un anello Chord con m=6. Abbiamo quindi 64 identificatori da 0 a 63 disposti lungo l'anello; di cui 8 (i nodi con ID 000000, 001000,…) sono nodi effettivamente presenti nella rete.

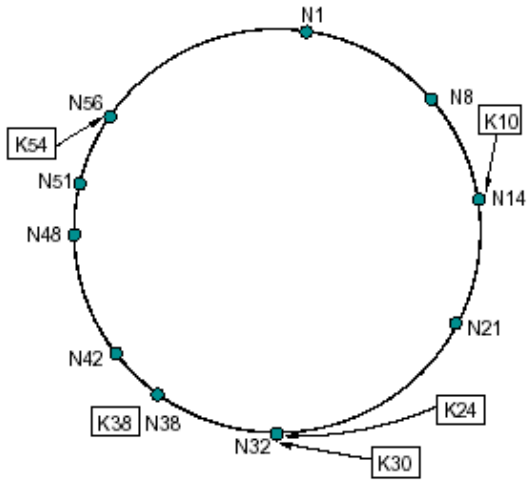
Assegnazione delle chiavi. Il file k è assegnato ad un nodo con ID *almeno* k (in modulo)
Questo nodo è il nodo successore della chiave k ed è indicato come *successor(k).*



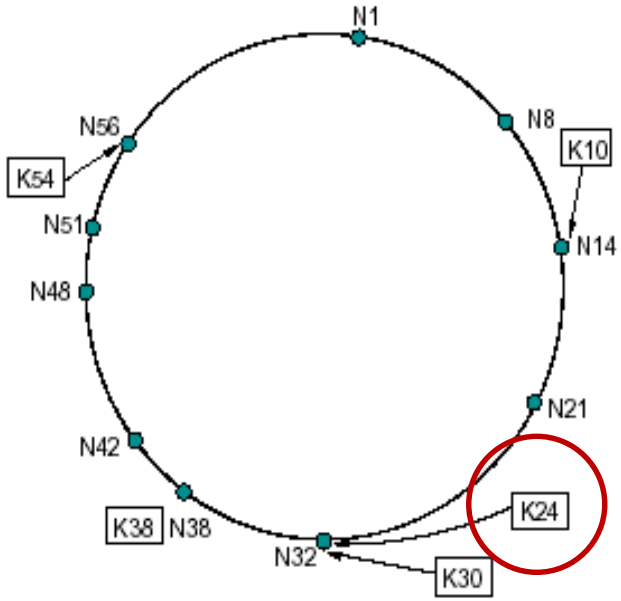


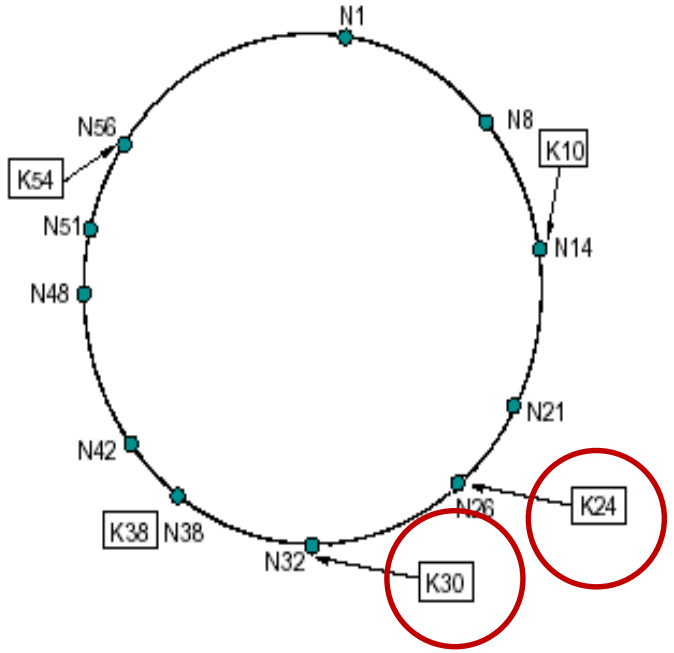
La figura mostra un anello di 8 ID, con 4 nodi (corrispondenti alle ID 0, 1, 3, 5) e 3 chiavi (1,2 e 6). Poichè il nodo 1 è presente nel ring, allora *successor(1)=1.* La chiave 2 ha *successor(2)=3*, poiché 2 non è un nodo presente nel ring ma il nodo immediatamente maggiore lo è. Infine abbiamo *successor(6)=0* poichè nella rete non vi sono nodi corrispondenti ad ID 6 e 7, quindi il primo nodo presente con ID almeno 6 è il nodo 0.

Per m = 6, il numero di di identificatori è 64. L'anello Chord ha 10 nodi (1,8,14,21,32,38,42,48,51,56) e mantiene 5 chiavi (10,24,30,38,54).
Il successore della chiave 10 è il nodo 14.
Le chiavi 24 e 30 sono entrambe assegnate al nodo 32, poiché non sono presenti nodi con ID compresa tra 24 e 31.



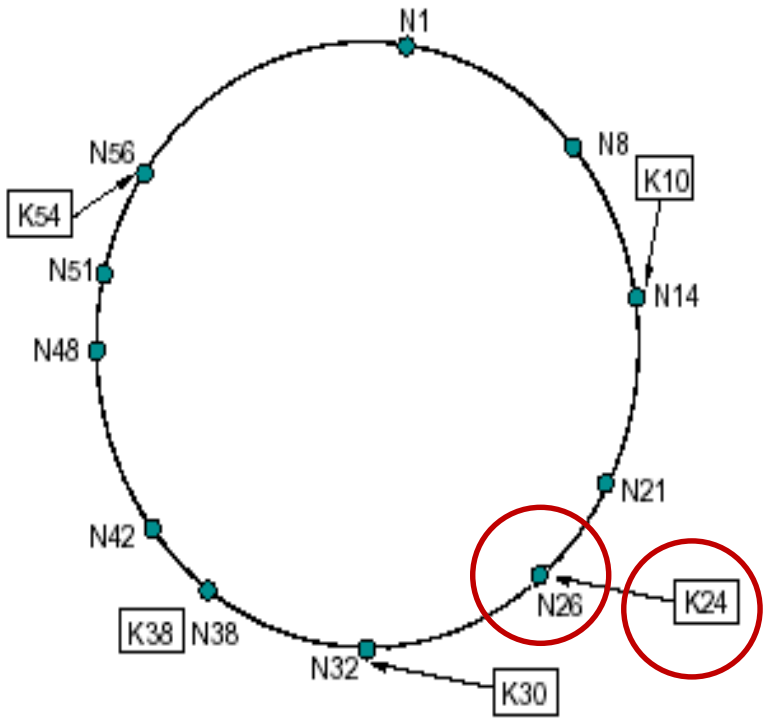
Join di un nodo. Quando un nodo *n* si unisce alla rete, alcune chiavi precedentemente assegnate al successore di n sono riassegnate a n. Più precisamente ad n sono riassegnate le chiavi con valore al più n.

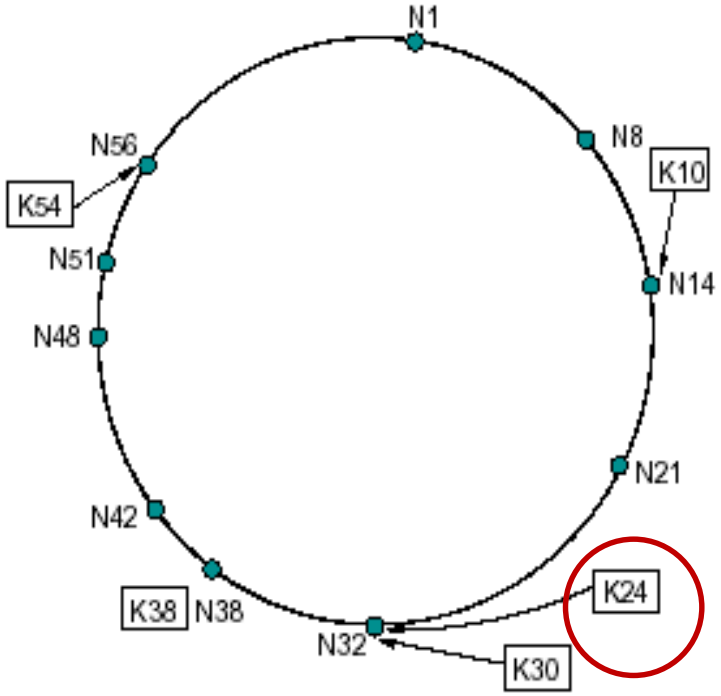




Nel momento in cui il nodo 26 si unisce alla rete, la chiave 24 precedentemente assegnata al nodo 30 (il successore di 26) ha ora come successore 26 ed è quindi riassegnata al nodo 26.

Leave di un nodo. Quando il nodo n lascia la rete, tutte e chiavi assegnate a n vengono vengono riassegnate al successore di n.



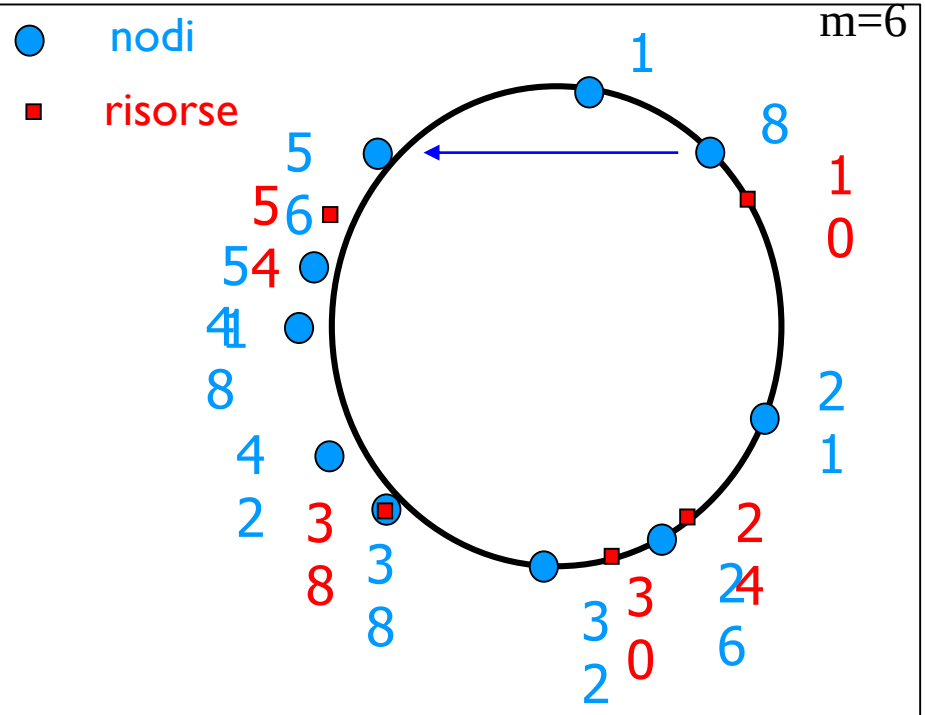


Nel momento in cui il nodo 26 lascia la rete, la chiave 24 ad esso assegnata, viene riassegnata al nodo 30, il successore di 26..

Lookup. Supponiamo di avere N nodi e K chiavi (file) presenti nella rete. Dato un valore k vogliamo determinare a quale nodo è assegnata la chiave k.

Supponiamo che il nodo 8 sia interessato alla chiave 54:

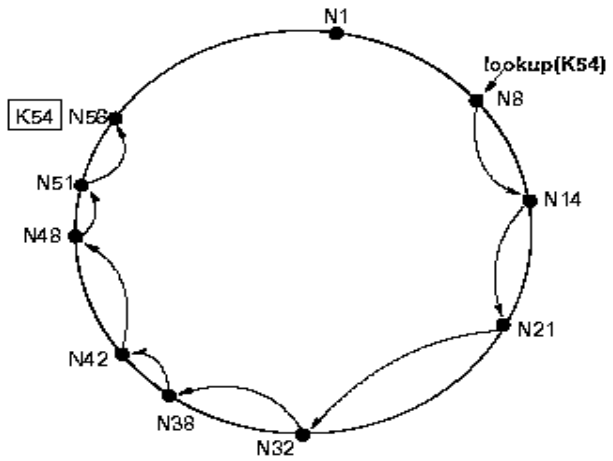
Se il nodo 8 conosce tutti i nodi, è in grado di sapere senza fare nessuna richiesta che la chiave 54 è gestita dal nodo 56



Se ogni nodo avesse informazioni su ogni altro nodo presente nella rete, la ricerca di una chiave avrebbe costo O(1) msg. Per fare ciò servono però informazioni globali, quindi una tabella di routing di dimensione pari al numero N di nodi presenti nella rete. Ciò comporta costi di manutenzione delle tabelle proibitivi.

Lookup scalabile. Se ciascun nodo sa come contattare il suo successore sul ring degli identificatori, tutti i nodi possono essere visitati in tempo lineare mediante una *ricerca sequenziale:* una query per un dato identificativo potrebbe essere passata intorno al ring attraverso i puntatori al successore fino a quando non si incontra il nodo che contiene la chiave cercata.

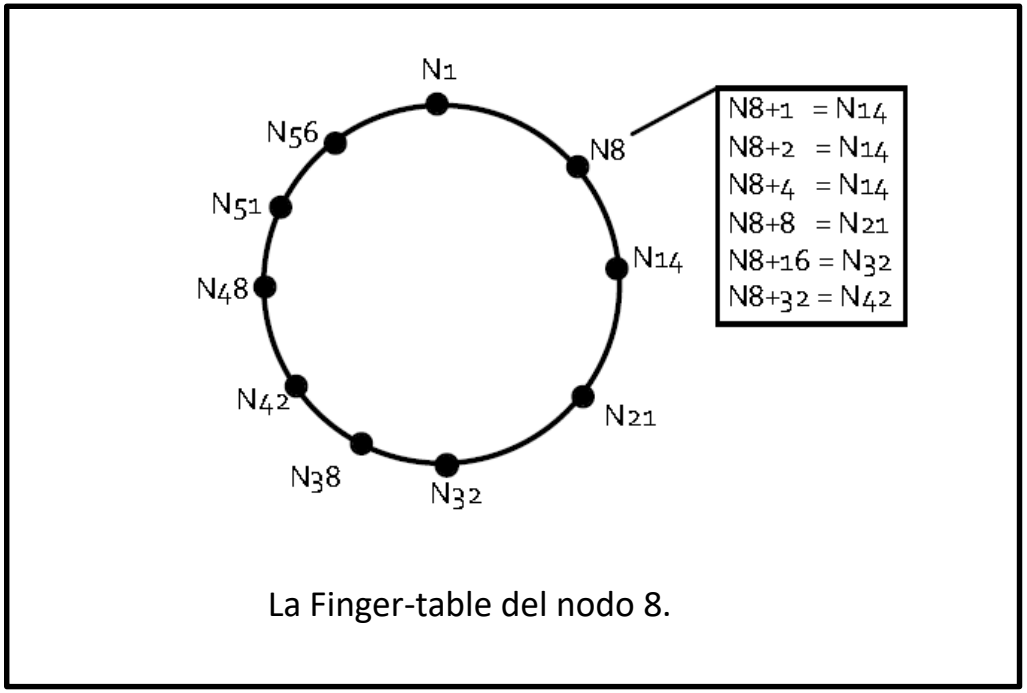
Tuttavia questo non è sufficiente per garantire prestazioni accettabili, infatti si esegue una ricerca lineare ed il costo è di O(N) msgs.



Lookup da N8 per la chiave con ID 54. Il nodo 8 invia una query al suo successore 14, il nodo 14 contatta il proprio successore 22 che contatta 38, il nodo 38 a sua volta invia la query a 42 che la invia al suo successore 51; il nodo 51 ha come successore il nodo 56 che gestisce la chiave 54.

Ricerca scalabile veloce. Se lo spazio delle ID è di 2^m valori, ogni nodo mantiene una tabella di m elementi, detta *Finger Table.*

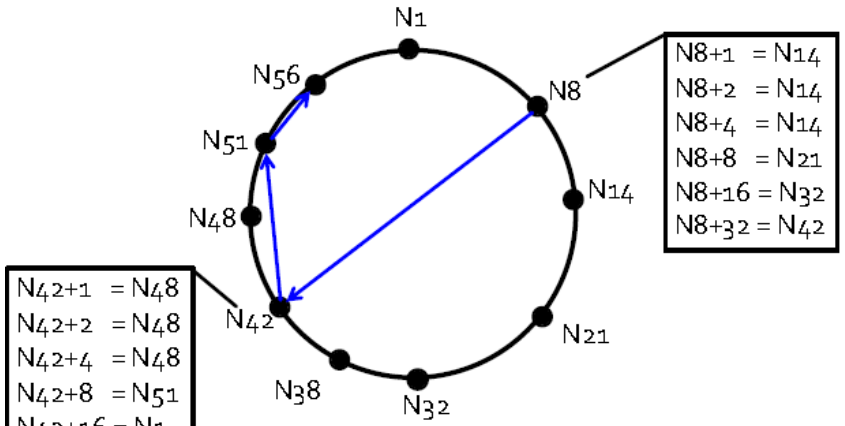
L'elemento i-esimo della finger table per un nodo n contiene l'indirizzo del 2^i *-mo* vicino di n in senso orario; cioè punta al *primo* nodo con $ID \geq n + 2^i$ (equivalentemente al successore di $n + 2^i$)



La Finger-table del nodo 8.

Nota: quando un nodo si unisce alla rete vengono violati i puntatori a lungo raggio di tutti gli altri nodi. Ad esempio se arriva il nodo 40, l'ultimo elemento della tavola di 8 sarà N40 e non più N42. Questo problema è gestito dai protocollo di Join e Leave di Chord.

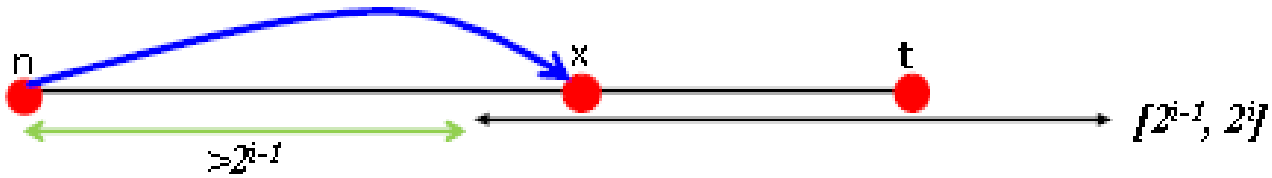
Algoritmo di ricerca: Ad ogni passo si segue il collegamento più lungo, presente nella finger table del nodo corrente, che non supera il target.



Da N8 trova la chiave con ID 54. Il nodo 8 invia la query al nodo 42 (che non supera 54), il nodo 42 guardando nella sua tabella invia la query al nodo più lontano presente nella tavola che non supera il target, …, il nodo 51 invia al suo successore che è anche il nodo target (*successor(54)*).

Fatto: La ricerca di una chiave in una rete di *N* nodi visita *O (log N)* nodi.
Dim. Supponiamo che il nodo *n* effettua una query per la chiave *k* e che la chiave *k* risiede al nodo target *t* (cioè *t=successor(k)*).
Quanti passi servono per raggiungere *t*?

La ricerca parte al nodo *n.*
Sia *i* un intero tale che *t* è contenuto nell'intervallo $(n + 2^{i-1}, n + 2^{i}]$
La finger table di *n* contiene un puntatore al nodo *x* che è il primo nodo *presente* nella rete con id maggiore di $n + 2^{i-1}$.



x e t sono entrambi in $(n + 2^{i-1}, n + 2^{i}]$. Ne segue che la distanza di t da x è al più 2^{i-1} .
Quindi in un passo abbiamo dimezzato la distanza da t.
Possiamo farlo al più $\log_2$ N volte. Ne consegue che troviamo t in $O(\log_2 N)$ passi.