Reti Peer-to-Peer

- Tipologia di sistemi distribuiti dove i servizi (file-sharing, backup) sono forniti grazie alla cooperazione di un gran numero di macchine tutte allo stesso livello.
  - Basati sulla condivisione di risorse
- Esempi file-sharing: Napster, eDonkey, con alcune funzionalità centralizzate, Gnutella,completamente decentralizzato, FastTrack, gerachico.
- Difetti: robustezza limitata, grossi volumi di traffico generati dal flooding, efficienza legata ai supernodi.

Distribuited backup

- Un servizio di backup distribuito su una rete decentralizzata non gerarchica è ottenuto tramite la condivisione, da parte dei suoi componenti, di una porzione di hard disk inutilizzata.
  - Giustificato dalla rapida crescita delle capacità dei dischi.

- E' simile al file-sharing, basato sulla condivisione di risorse.
  - File-sharing → risorsa = file. Backup → risorsa = quota disco.

Distribuited backup

- Nasce dall'esigenza di proteggere i dati da crash locali.
- A differenza dei sistemi di backup centralizzati, non richiede costi aggiuntivi.
- Richiesto a livello di protocollo un certo grado di sicurezza e disponibilità del servizio.
  - Dati personali risiedono su dei peer potenzialmente insicuri e devono essere protetti da letture, modifiche o divulgazioni indesiderate.
  - Fallimenti di un certo numero di nodi non devono compromettere il recupero dell'informazione.

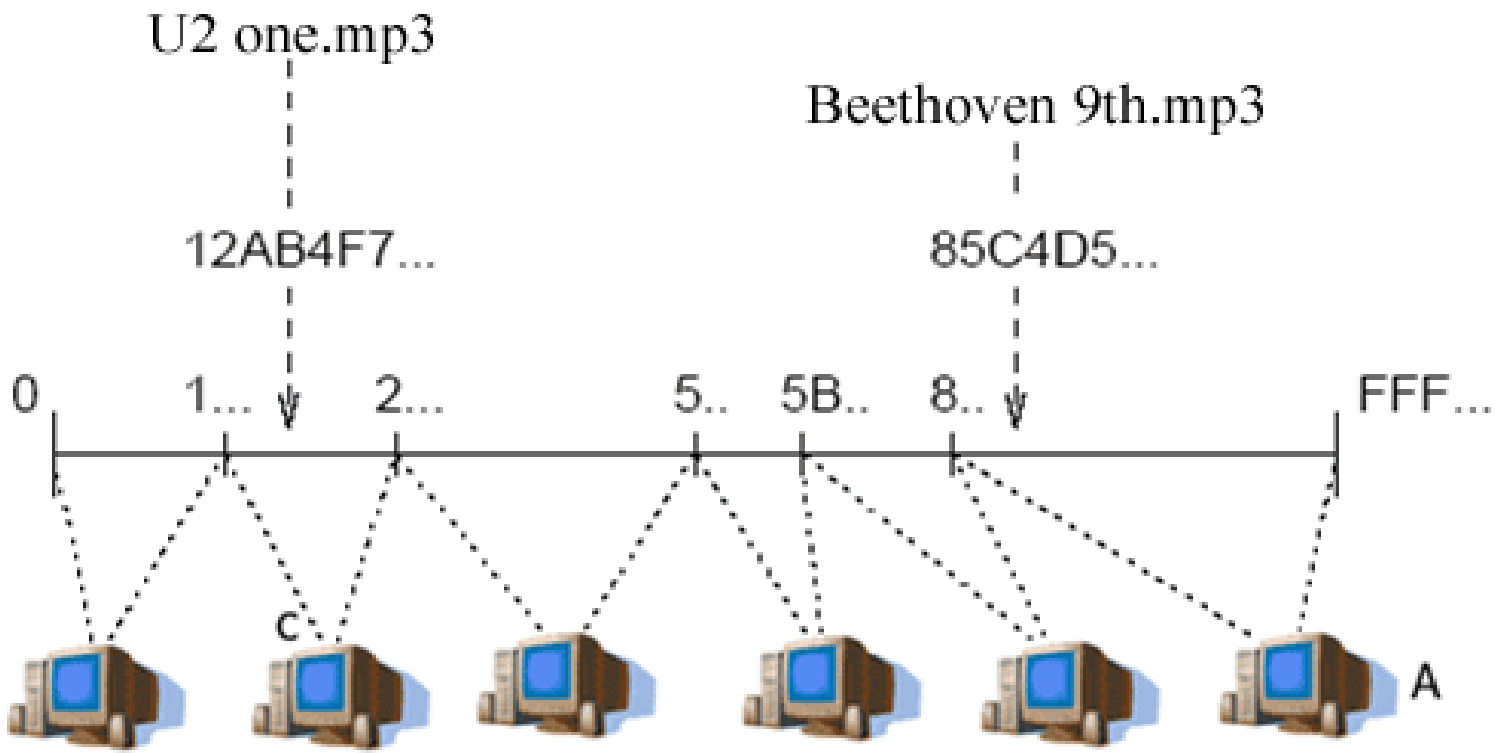
pStore

- Un servizio di backup distribuito su rete peer-to-peer.
- Si appoggia su Chord come location service.
- Un file in pStore è rappresentato da file block (FB) ed una file block list (FBL).
- FB ed FBL sono criptati con algoritmi a chiave privata. Autenticazione tramite firma digitale.
- I data chunks sono dentotati con C(i,p,s,d).
  - *i* è l'identificatore, dove *i*=H(H(*d*) ° *salt*). *salt* è usato per replicare i blocchi.
  - *p* sono i metadati pubblici.
  - *s* è la firma digitale sui metadati pubblici.
  - *d* sono i dati criptati. La chiave privata è l'hash degli stessi dati in chiaro.

Distribuited Hash Table

- Introdotte nei sistemi decentralizzati per migliorare scalabilità, robustezza, efficienza nel reperimento delle risorse.
- Idea base: tabella Hash, suddivisa tra i peer.
- Ai nodi della rete ed alle risorse sono assegnati identificatori (unici) da un determinato insieme. (es. interi da 0 a 2^{160}-1 tramite funzioni hash crittografiche tipo SHA-1).
- Ogni nodo tiene delle mappature ID→valore. Il valore mantenuto è dipendente dalla applicazione.

Distribuited Hash Table



Chord

- Gli Identificatori sono sistemati, logicamente, in uno spazio circolare, modulo 2^m.
- Gli identificatori delle risorse, o chiavi, sono mappati dal nodo successivo, procedendo in senso orario.(con ID ≥)
- Le chiavi di un nodo sono replicate nei successivi *r* nodi, dove *r* è un parametro di sistema, pubblicamente noto.
- Ogni nodo mantiene una tabella di routing con *m* entry chiamata *finger table. L'i*-esima (*1 ≤ i ≤ m)* entry per un nodo *n* contiene l'indirizzo IP del primo nodo che succede *n,* a distanza almeno 2^{i-1} (*i*-esimo finger di *n*).
  - Sono mantenuti nodi a distanze esponenziali.

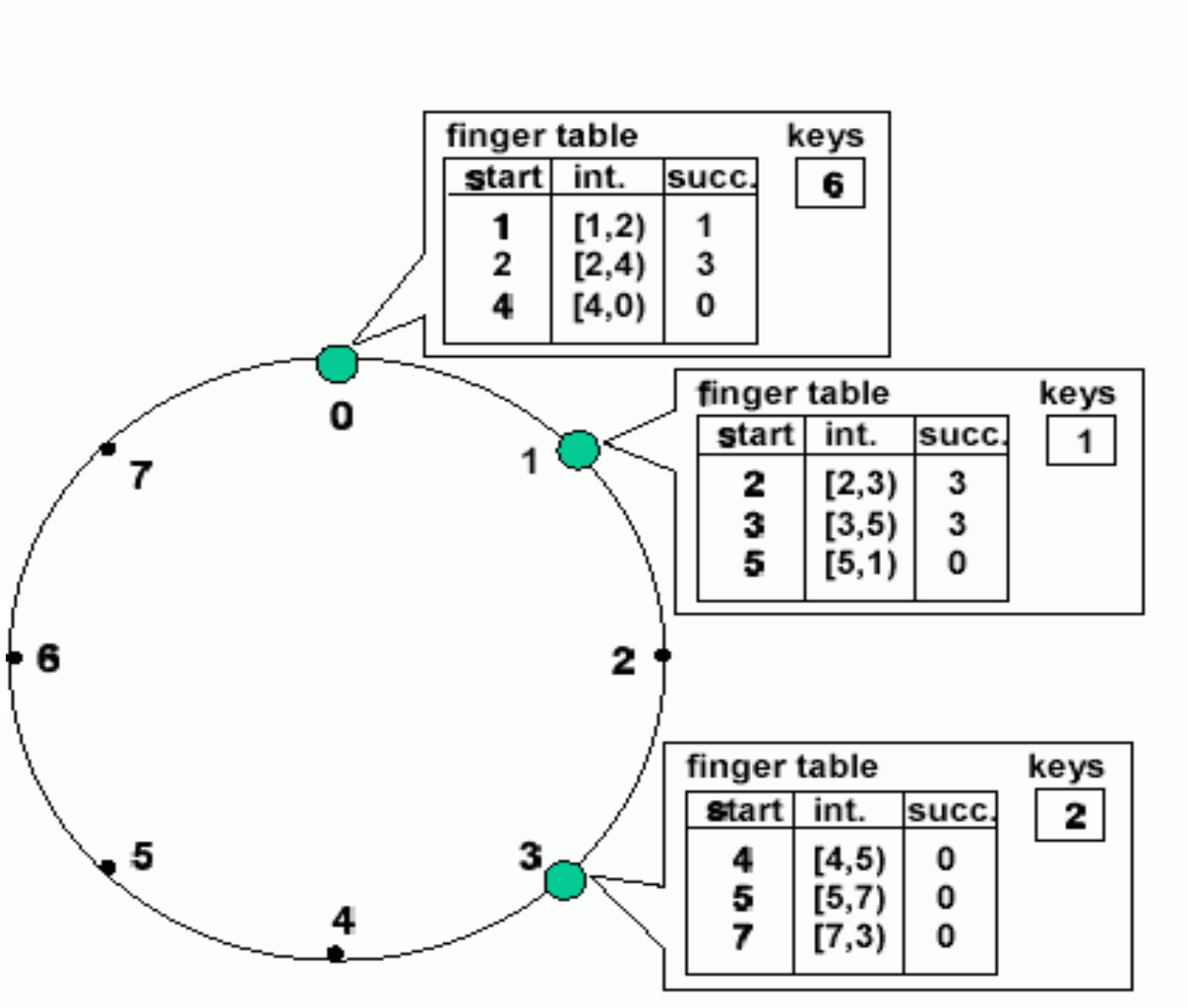
Chord

Esempio di un anello di dimensione 2^3.

Con:

- N = {0,1,3}
- K = {1,2,6}.

N l'insieme dei nodi, K è l'insieme delle chiavi.



Chord

- Quando si ricerca il valore relativo ad una chiave *k* bisogna contattare il nodo 'responsabile' per *k*, cioè il suo successore (ha l'ID più vicino a *k*).
- Ogni nodo non ha una conoscenza completa della rete.
  - Si conoscono solo *m* nodi dalla *finger table*.
- Se un nodo non conosce il successore di *k,* cerca nella sua *finger table* il nodo più vicino che precede *k,* ed inoltra a questo la query.
- Un nodo che riceve una query la inoltra alla stessa maniera.

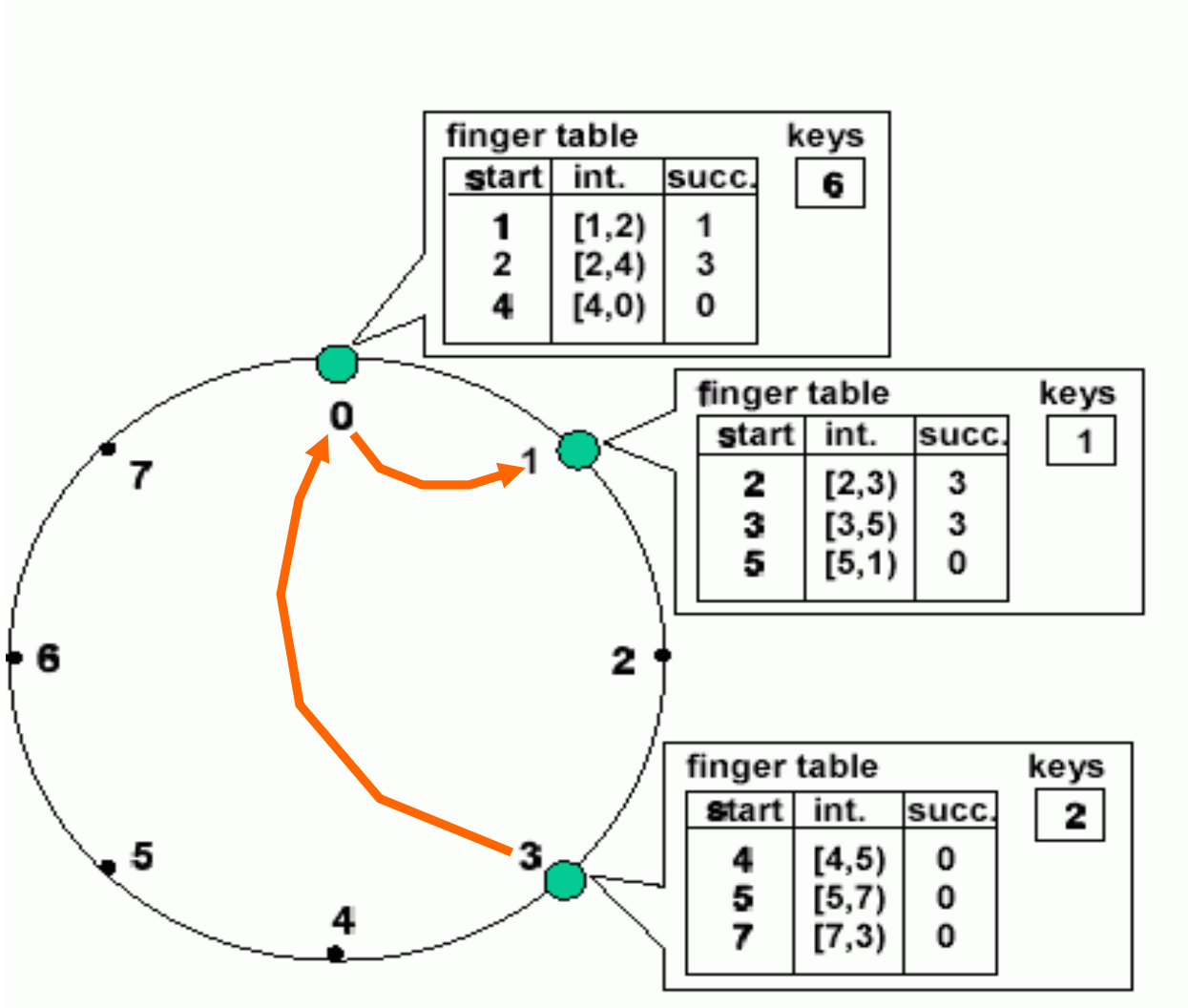
Chord

Esempio di un anello di dimensione 2^3.

Con:

- N = {0,1,3}
- K = {1,2,6}.

N l'insieme dei nodi, K è l'insieme delle chiavi.



Chord

Problemi :

- Alto traffico di mantenimento della rete.
  - Un nodo verifica continuamente la presenza dei vicini e dei finger.
- Meccanismo di replicazione inadeguato per alcuni servizi.
  - Se i valori fossero dei file sarebbe sprecato troppo spazio.
- Inconsistenza delle informazioni di routing su *join* e *leave* (o fallimenti di nodi) concorrenti.
- Procedure di *leave* costose in termini di nodi da aggiornare e chiavi da trasferire.

Chord

Esempio:

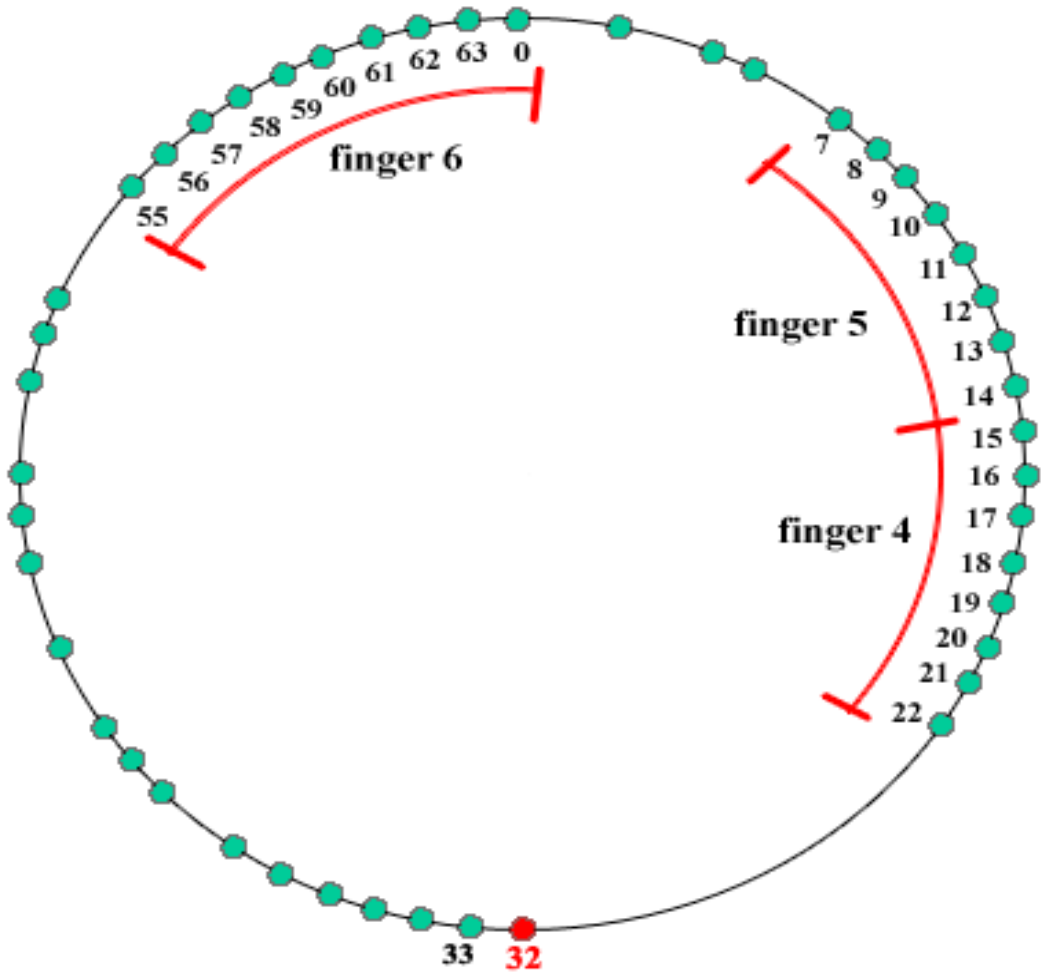
Anello con 2^6 ID totali (finger di 6 entry), 46 nodi attivi.

'Buco' da 23 a 31 a seguito si uscite o fallimenti di nodi

Se esce 32, 26 su nodi su 46 dovranno aggiornare le loro finger table.

Tutte le chiavi di 32 saranno trasferite ad un altro nodo(es. 36 se $r = 3$)

Situazioni peggiori se aumenta la dimensione dello spazio degli identificatori.



Nuova soluzione per il location service

- Basato su una modifica al modello di Chord che permette di 'muoversi' nell'anello di dimensione 2^m in senso orario e antiorario.
  - Si ha una conoscenza simmetrica dell'anello.

- Ogni nodo mantiene una *finger table* con *2m -1* riferimenti. *L'i*-esima (*1 ≤ i ≤ m)* entry per un nodo *n* contiene l'indirizzo IP del nodo che succede *n* a distanza 2^{i-1} (nodo n+2^{i-1}) e l'indirizzo IP del nodo che precede n a distanza 2^{i-1} (nodo n-2^{i-1}).
  - La *m*-esima entry è uguale per sia per i successori sia per i predecessori.

- Se l'indirizzo IP di un nodo cosi determinato non è disponibile, per qualunque ragione, il riferimento nella tabella viene marcato come non valido.

Nuova soluzione per il location service

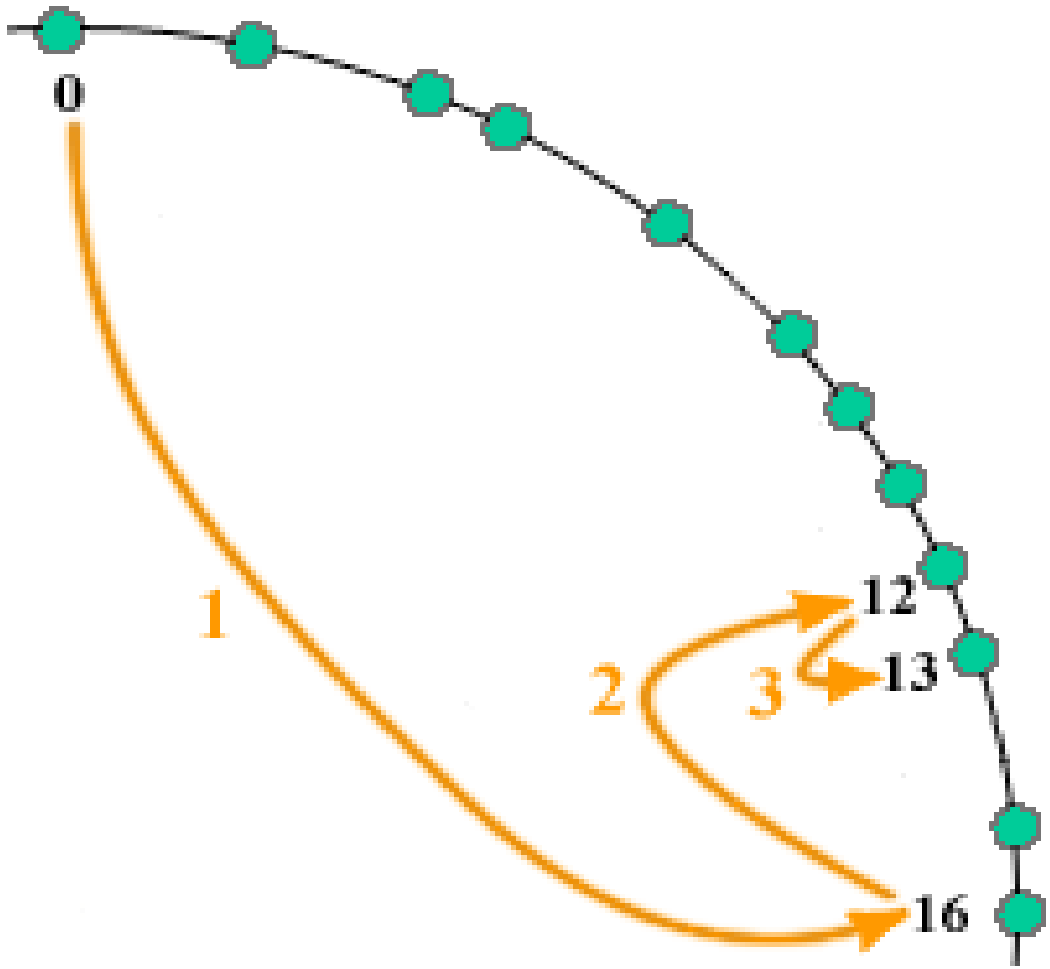
Finger table nodo 0 (2^6 ID totali, 11 riferimenti)				
distanza	ID nodo precedente	Indirizzo IP	ID nodo successore	Indirizzo IP
2^0	63	10.0.0.63	1	10.0.0.1
2^1	62	Non valido	2	10.0.0.2
2^2	60	10.0.0.60	4	10.0.0.3
2^3	56	Non valido	8	Non valido
2^4	48	10.0.0.48	16	10.0.0.16
2^5	32	10.0.0.32	32	

Nuova soluzione per il location service

- Se un nodo non conosce l'indirizzo IP di un certo ID, cerca nella sua *finger table* il riferimento valido più vicino al quale 'delegare' la richiesta di location.
  - Se esistono due nodi alla stessa distanza può essere scelto uno qualunque dei due

- Una location per un certo ID può concludersi in tre modi:
  1. Si reperisce l'indirizzo IP.
  2. Si reperisce un nodo che possiede il riferimento, ma questo è non è valido.
  3. Non si arriva a nessuna delle due precedenti informazioni.
     - La location è stata abortita per aver superato il numero massimo di salti consentiti.

Nuova soluzione per il location service



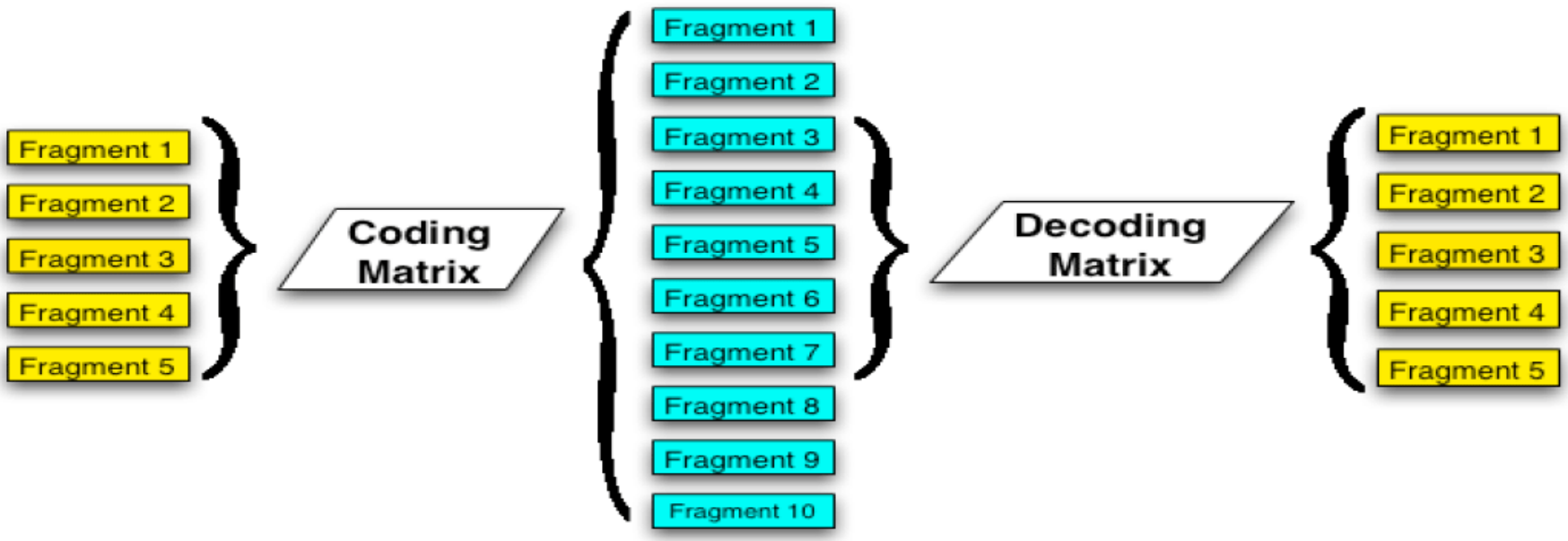
Information Dispersal Algorithm

- Proposto da Rabin nel 1989, disperde l'informazione iniziale in *n* frammenti, di questi soltanto *m,* con *m*<*n* sono necessari per la sua ricostruzione.

- Funzionamento:
  1. L'input viene inizialmente suddiviso in B blocchi della stessa dimensione.
  2. Ciascuno di questi B blocchi viene ulteriormente suddiviso in F frammenti di dimensione S
     - *(F*S è la dimensione di ciascun blocco)*
  3. IDA disperde gli F frammenti di input in N frammenti di output. Dove N ≥ F ed N/F è il fattore di ridondanza desiderato
  4. Un qualunque sottoinsieme di F frammenti degli N frammenti di output permette la ricostruzione del blocco iniziale.

Information Dispersal Algorithm

- I frammenti di output sono ottenuti come una combinazione lineare dei frammenti di input tramite una matrice di codifica di dimensione N * F.
- La matrice di decodifica ottenuta da quella di codifica



Information Dispersal Algorithm

- Semplice esempio con interi.

$$(3 \quad 1 \quad 2) * \begin{pmatrix} 3 & 2 & 1 & 1 & 3 \\ 13 & 3 & 15 & 1 & 17 \\ 1 & 5 & 7 & 11 & 6 \end{pmatrix} = (24 \quad 19 \quad 32 \quad 26 \quad 38)$$

$$(19 \quad 32 \quad 38) * \begin{pmatrix} 2 & 1 & 3 \\ 3 & 15 & 17 \\ 5 & 7 & 6 \end{pmatrix}^{-1} = (3 \quad 1 \quad 2)$$

$$(24 \quad 19 \quad 26) * \begin{pmatrix} 3 & 2 & 1 \\ 13 & 3 & 1 \\ 1 & 5 & 11 \end{pmatrix}^{-1} = (3 \quad 1 \quad 2)$$

Information Dispersal Algorithm

- Nessun sottoinsieme di nodi sotto un una certa soglia può ricostruire il backup.
- Nessun nodo conosce il numero di frammenti necessari per ricostruire il backup ed il fattore di ridondanza.
- Solo il possessore della matrice di codifica può ricostruire il backup.
  - Maggiore sicurezza: utilizzo di una matrice di codifica differente per ogni blocco.

Integrazione IDA + location service

- Il nostro nuovo servizio di backup distribuito nasce dall'integrazione di IDA con l'algoritmo di location descritto.

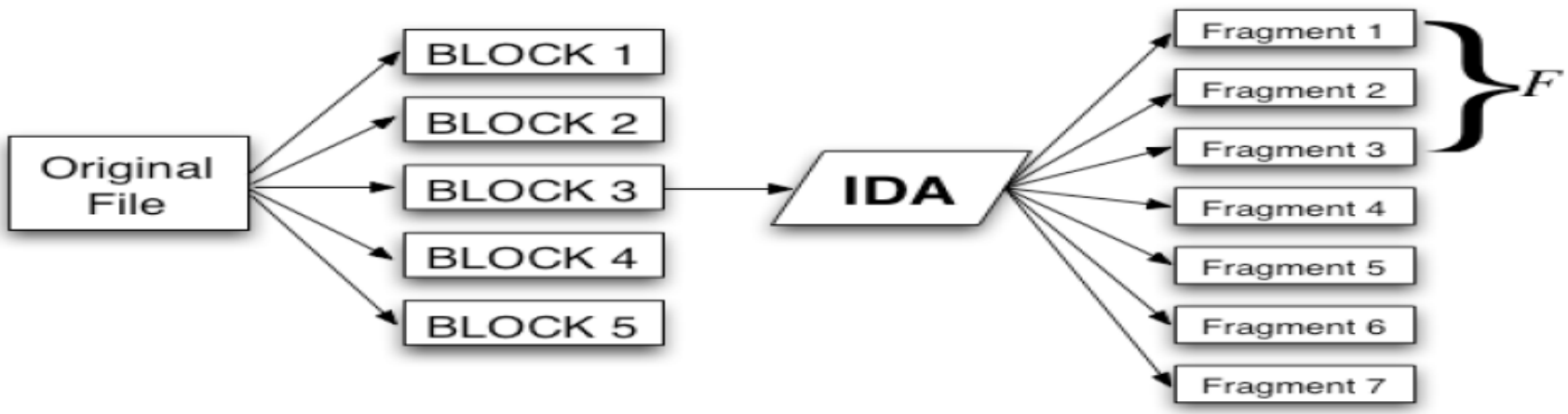
- Le procedure definite nel nostro protocollo sono:
  - *Splitting* → Suddivisione del backup in blocchi e frammenti.
  - *Identification* → Costruire gli identificatori dai frammenti.
  - *Dispersal* → Disperdere i frammenti in rete.
  - *Retrieval* → Ricostruzione del backup originale.
  - *Join* → Entrata di un nuovo nodo in rete.

- Non è necessario alcun protocollo di uscita.

Splitting

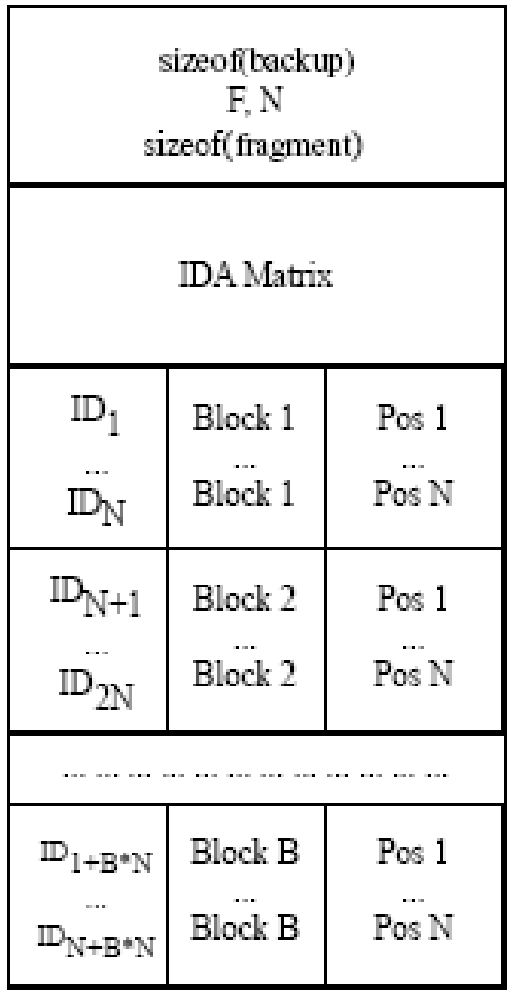
- File diviso in *B* blocchi suddivisi ulteriormente in *F* frammenti ed *N* sono generati da IDA.

- *N* ha lo stesso ordine della dimensione della rete, *F* è scelto in modo da ottenere una certo fattore di disponibilità, *R* = *N/F* è il fattore di ridondanza desiderato.
  - Se 0.9 è la probabilità che 10 nodi su 120 sono vivi scegliamo *F*=10 e *N*=120, ed avremo una disponibilità del 0.9 con *R*=12.



Identification

- L'ID di ciascun frammento è computato effettuando l'hash del suo contenuto.
- Il file dei metadati tiene tutte le informazioni necessarie alla ricostruzione del backup.
- Solo chi possiede questo file è in grado di ricostruire il backup originale.



Dispersal

- ID dei frammenti divisi logicamente in due parti. La più significativa *IDN* è usata per indirizzare il nodo, la meno significativa *IDF* è usata per indirizzare il frammento all'interno del nodo.
  - Il numero di bit dell'IDN è assegnato in base alle dimensioni della rete ed una volta definito è mantenuto costante.

- Ogni nodo usa l'algoritmo di location per reperire gli indirizzi dei nodi con ID uguali agli IDN dei frammenti.
  - Se l'indirizzo reperito è relativo ad un nodo attivo si invia ad esso il frammento.
  - Se l'indirizzo è di un nodo spento o l'ID è un riferimento non valido, viene computato un insieme di possibili sostituti da un'opportuna funzione *successor(IDN_i , j)* pubblicamente nota.(su *s* successori)
  - Se nessuna la location fallisce la dispersione del particolare frammento è abortita.

Retrieval

- Per ciascun blocco bisogna reperire una soglia di almeno F indirizzi di nodi accesi che contengono i frammenti.

- Per ogni blocco iteriamo i seguenti passi:
  1. Cerchiamo di reperire almeno F nodi accesi usando l'algoritmo di location.
     - Se vengono completate meno di F location, il reperimento fallisce.
  2. Se dopo la prima fase il numero dei nodi accesi è minore di F ma i riferimenti reperiti sono F' ≥ F, usiamo la funzione *successor(IDN_i , j)* sui riferimenti non validi o a nodi spenti.

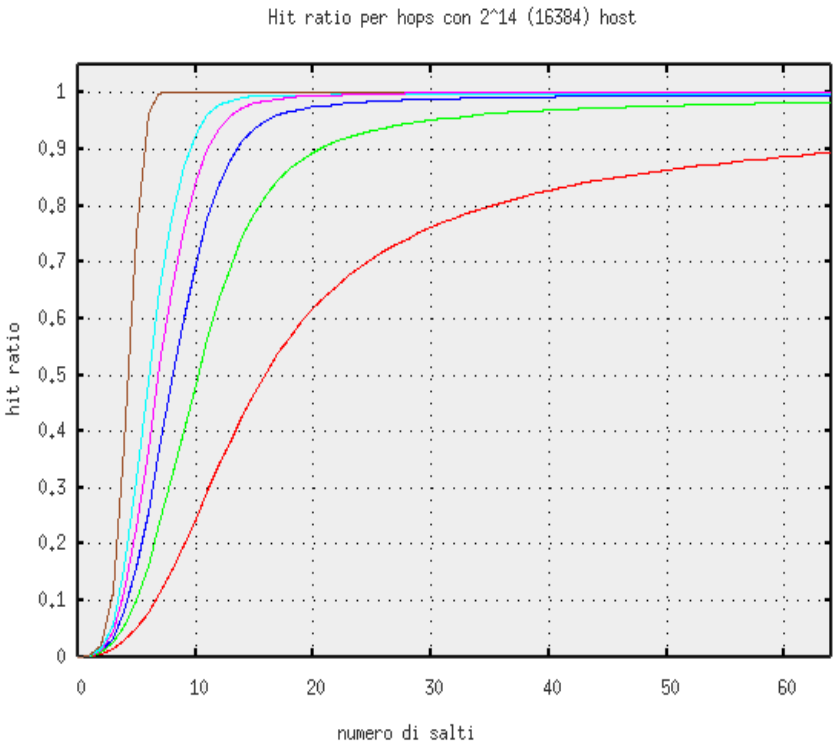
Join

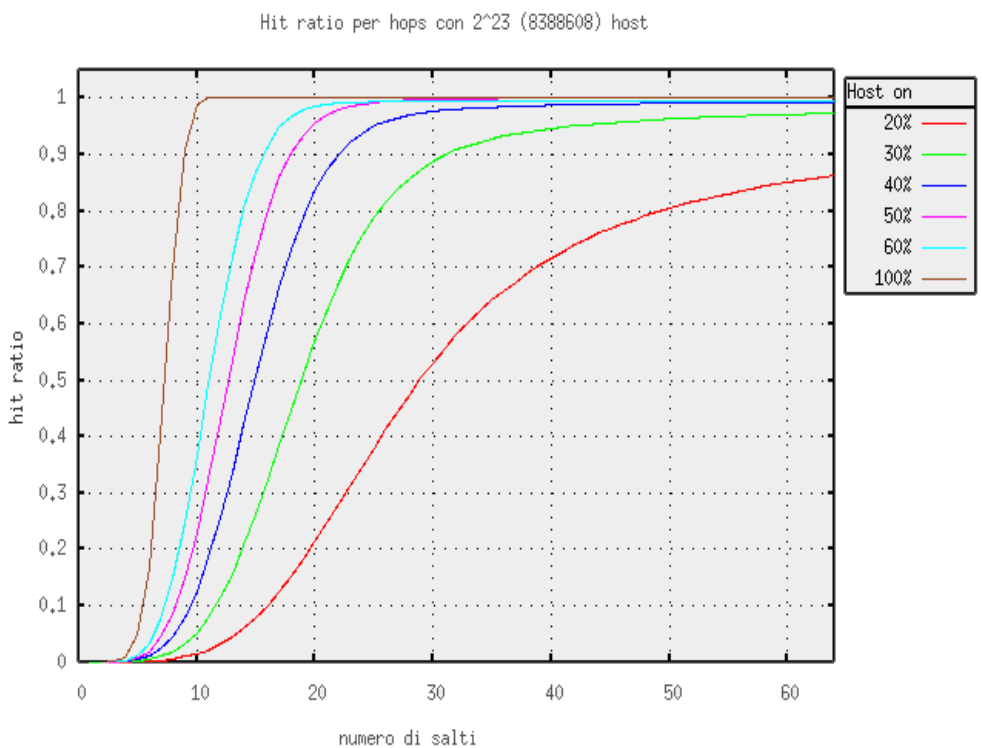
- Un nodo per entrare nella rete logica deve conoscere l'indirizzo di almeno un altro nodo già in rete (entry point).

- Un nodo che vuole entrare in rete:
  1. Ottiene un ID unico.
  2. Inizializza la sua finger table delegando al suo entry point il compito di trovare i riferimenti corrispondenti.(ID a distanza esp.)
  3. Notifica la sua presenza ai nodi accesi della sua finger.
     - Ogni nodo compare nella tabella dei finger dei nodi presenti nella sua tabella.
  4. Recupera dai successori i frammenti con IDN uguale al proprio ID (computati tramite *successor(IDN_i , j)*).

Risultati Sperimentali

Fattore di ricerche su host accesi andate a buon fine, al variare del numero di salti (max 64).
E' rappresentata una curva per ogni diversa percentuale di host accesi.
Il grafico di sinistra e su uno spazio di 2^{14} mentre quello di destra 2^{23}.





Risultati Sperimentali

- La lunghezza media dei cammini di routing con densità di host accesi al di sopra del 20% è *O (log n)*
  - Nel grafico in figura si ha, su $2*10^5$ prove casuali in uno spazio di dimensione 2^{14}, il numero di occorrenze di un cammino al variare del numero di salti.Nel caso del 20% la lunghezza media si attesta intorno al logaritmo di *n* (14) mentre nel caso del 30% scende a circa 0.65 * log n (9,10).