Lezione n.8
KADEMLIA
LA RETE KAD DI EMULE

Laura Ricci

20/3/2009

KADEMLIA: CONCETTI GENERALI

- Protocollo P2P proposto da P. Maymounkov e D. Mazières (University of New York).

- utilizza il protocollo UDP

- le associazioni tra i clients ed i files che essi condividono
  - non viene registrata su server particolari
  - viene gestita dai clients presenti sulla rete Emule. Ogni client ha infatti anche la funzione di server.

- ogni client agisce come server per l'associazione tra alcune chiavi e le rispettive fonti

- obiettivo della ricerca: trovare quei client che hanno la responsabilità relativa alla chiave della ricerca in corso

KADEMLIA: CONCETTI GENERALI

- Sistema distribuito per la memorizzazione e la ricerca di coppie <key, value>
- Assegna mediante SHA un identificatore di 160 bits ai nodi ed alle chiavi
- Anche se presenta molte similitudini con altri sistemi (Chord, Pastry), presenta un insieme di caratteristiche non offerte simultaneamente da nessun sistema esistente
- Ogni coppia <key,value> viene assegnata al nodo il cui identificatore è più vicino alla chiave, secondo una metrica definita sull'OR ESCLUSIVO
- Ad esempio la distanza tra l'identificatore ID1 = 0100 e l'identificatore ID2=0111 è la seguente

  D(0100,0111) = 0100 XOR 0111 = 0011 = 3

- Kademlia appartiene alla famiglia delle DHT basate su prefix-matching (a cui appartengono anche Pastry e Tapestry)
  - Ad ogni hop la chiave viene inoltrata da un nodo N_1 verso un nodo N_2 il cui identificatore è più vicino alla chiave (secondo la metrica definita)

PREFIX MATCH DHT

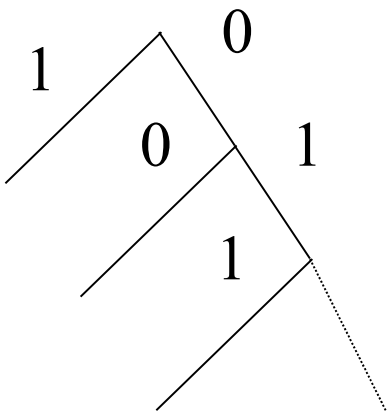
- Key lookup: supponiamo che il nodo N1 ricerchi la chiave K
- N1 ricerca all'interno della propria routing table un'entrata che punti ad un nodo N2 identificato da un ID la cui configurazione binaria coincide almeno nei b bits più significativi con la configurazione di k
- La chiave viene inoltrata verso N2
- N2 ricerca all'interno della propria routing table un' entrata che punti ad un nodo N2 identificato da un ID la cui configurazione binaria coincide almeno nei 2b bits più significativi con la configurazione di k
- Il look up prosegue ad ogni passo un 'match' di lunghezza sempre maggiore tra la chiave e l' identificatore del nodi
- Il look up termina non appena non è possibile individuare all'interno della tabella di routing un match di lunghezza maggiore. Il nodo più vicino alla chiave è stato raggiunto

PREFIX MATCH DHT

- b = numero di bits corretti ad ogni passo = symbol size
- Kademlia utilizza b=1, ad ogni passo di look up incrementa di 1 bit il match tra la chiave e l'identificatore del nodo
- Routing Kademlia $O(\log(n))$
- Pastry utilizza di solito b=4, ad ogni passo incrementa di 4 bits il match tra la chiave e l'identificatore del nodo
- Routing Kademlia $O(\log_2{}^b(n))$
- La dimensione delle routing table di un nodo e il numero di look-up hops dipendono da b
- Osservazione: anche alcuni protocolli a livello IP possono essere considerati prefix matching, il matching viene calcolato sull'indirizzo IP dei nodi

PREFIX MATCH DHT: ROUTING TABLES

- Le Prefix Match DHT sono caratterizzate da tabelle di routing strutturate ad albero
- In Kademlia la tabella di routing di ogni nodo può essere rappresentata come un albero binario non bilanciato
- Ogni foglia dell'albero binario
  - corrisponde ad un prefisso binario
  - contiene un insieme di puntatori a nodi caratterizzati da quel prefisso



Nodo 010....

PREFIX MATCH DHT:TABELLEDI ROUTING

- Le righe della tabella di routing corrispondono alle foglie dell'albero binario
- K- buckets:
  - lista di riferimenti a nodi (contatti) memorizzati in ogni riga della tabella di routing
  - ad ogni passo di lookup il nodo ha la possibilità di scegliere tra k diversi contatti
  - K = 1 in Pastry
  - K ≅ 20 in Kademlia
- Un valore di K > 1 garantisce
  - una maggiore robustezza e tolleranza ai guasti del routing
  - la possibilità di scegliere percorsi di routing alternativi

KADEMLIA: ROUTING

- Parallel Routing: una chiave K ricevuta da un nodo viene inoltrata in parallelo ad α nodi prelevati da un k-bucket
- Strategie di routing
  - Routing Iterativo:
    - il nodo che invia una richiesta di look up coordina l'intero processo di ricerca
    - ad ogni passo, un nodo invia una richiesta di look up ed attende una risposta
    - la risposta ricevuta indica quale è il successivo passo di routing
  - Routing Ricorsivo: la richiseta di look up viene inoltrata automaticamente da un peer al successivo
- Kademlia utilizza un routing iterativo

KADEMLIA: CARATTERISTICHE GENERALI

Caratteristiche generali della rete Kademlia

- Prefix Matching Routing
- K-Buckets: ridondanza nelle tabelle di routing
- Parallel Routing
- Iterative Routing

Nessuna DHT riunisce tutte queste caratteristiche

Prefix Match Routing caratteristico anche di Pastry, Tapestry ed altre DH

Le altre caratteristiche potrebbero essere introdotte anche in altre DHT.

KADEMLIA:CONCETTI GENERALI

- In Kademlia ogni nodo ed ogni chiave ha un identificatore di 160 bits
- Distanza tra due nodi (non geografica, ma sull'overlay) definita mediante l' OR esclusivo calcolato bit a bit degli identificatori dei 2 nodi.
- Alcune proprietà dell'OR esclusivo utilizzate in Kademlia:

$d(x, y) > 0$ se $x \neq y$

Simmetria

$\forall x, y : d(x, y) = d(y, x)$

Disuguaglianza Triangolare

$d(x, y) + d(y, z) \geq d(x, z)$

Unidirezionalità: dato un x ed una distanza Δ, esiste un solo y tale che

$d(x, y) = \Delta$

KADEMLIA: LA METRICA

- XOR : definisce, a differenza di Chord, una metrica simmetrica
- Dato un nodo n, ogni riga della tabella di routing di Kademlia, contiene riferimenti ad altri nodi che si trovano entro una certa fascia di distanza, rispetto ad n
- La metrica definita fa sì che quando un nodo n riceve una query da un nodo k può utilizzare l'identificatore di k per inserirlo nella propria routing table, nella riga relativa alla fascia di distanza appropriata per k
- Ogni nodo può arricchire la propria routing table mediante le informazioni contenute nelle query
- Al contrario, in Chord:
  - se il nodo a riceve una query dal nodo b, questo implica che b possiede nella propria finger table un riferimento ad a
  - l'informazione contenuta nella query non può, in generale, essere utilizzata da a per arricchire la propria tabella di routing, perchè è possibile che nessun finger di a punti a b

KADEMLIA: LA METRICA

- Maggiore è il prefisso che caratterizza due nodi, minore è la loro distanza calcolata mediante l'XOR

- Esempio:

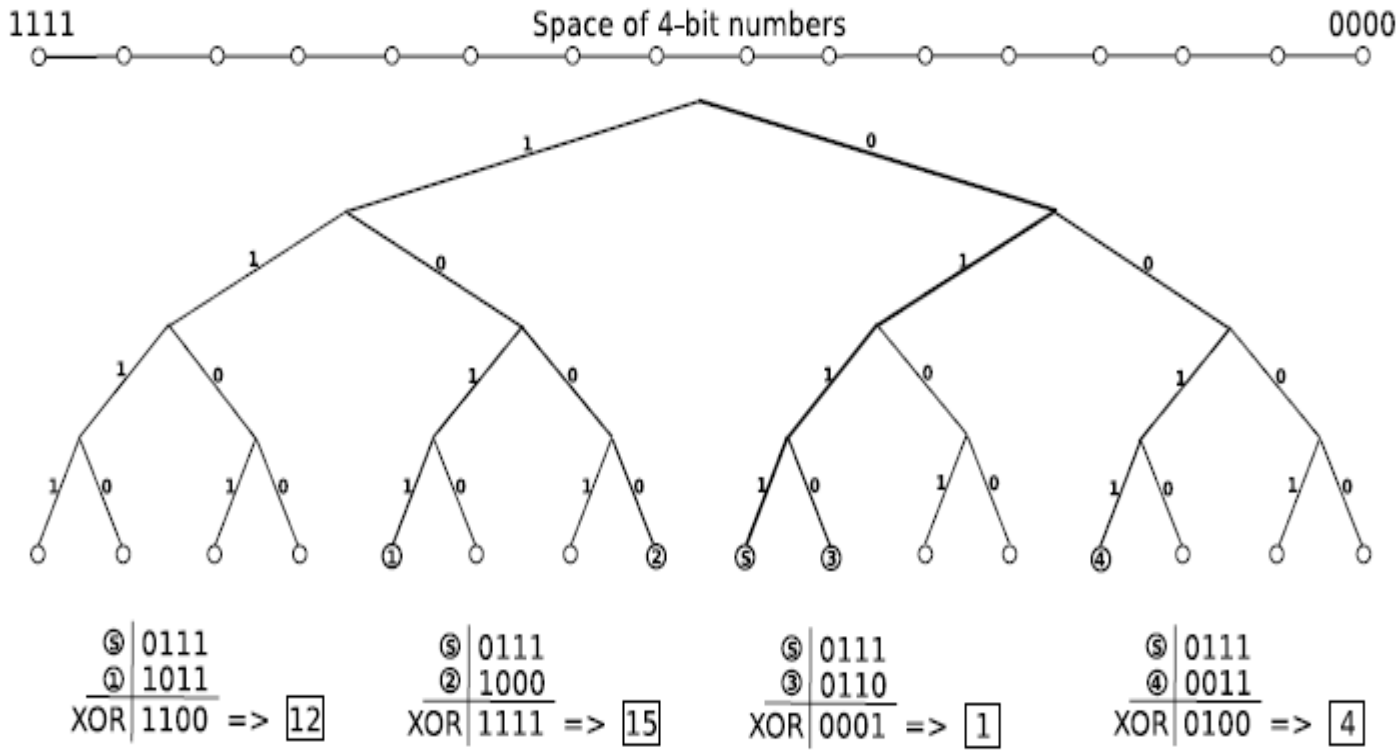
  000100 XOR 000101 = 000001 Prefisso a comune 00010, distanza 1

  010000 XOR 000001 = 010001 Prefisso comune 0, distanza 17

- Nodi 'vicini' sono caratterizzati da un lungo prefisso comune, secondo la metrica definita

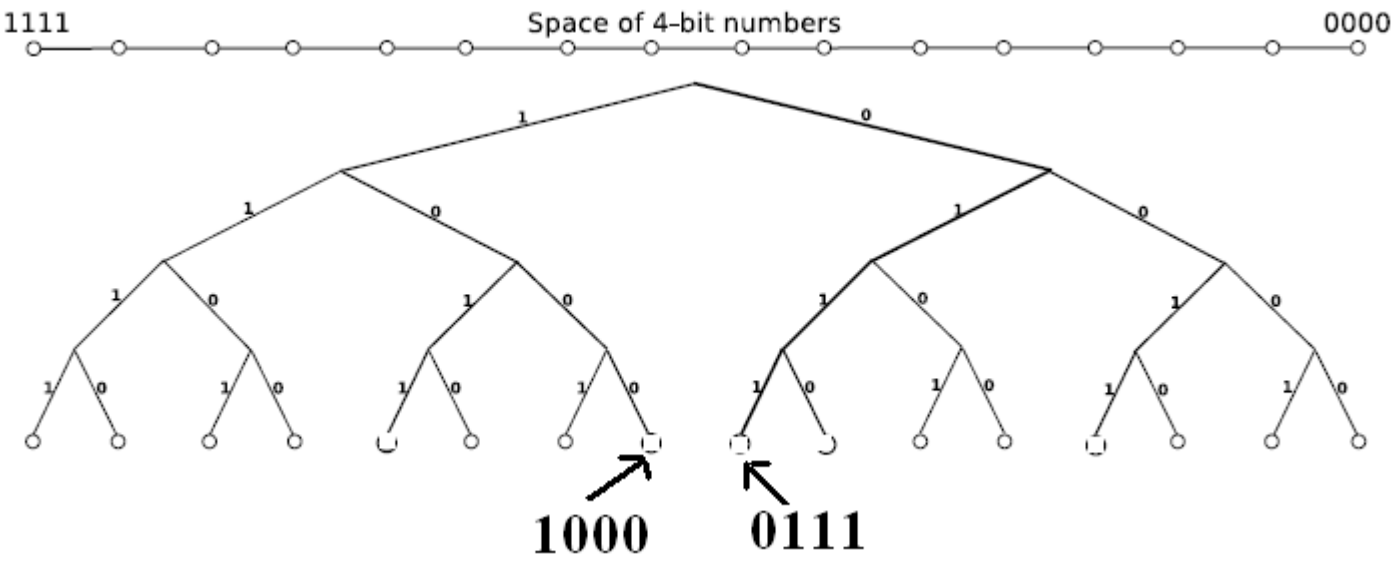
- Lo spazio degli identificatori può essere rappresentato mediante un albero binario, in cui le foglie corripondono agli identificatori

KADEMLIA:LO SPAZIO DEGLI IDENTIFICATORI



- ogni nodo n dell'albero è identificato da ID(n) = stringa di bit ottenuta concatenando le cifre binarie presenti sul cammino dalla radice ad n
- solo un sottoinsieme degli identificatori è, in ogni istante, assegnato ai peer

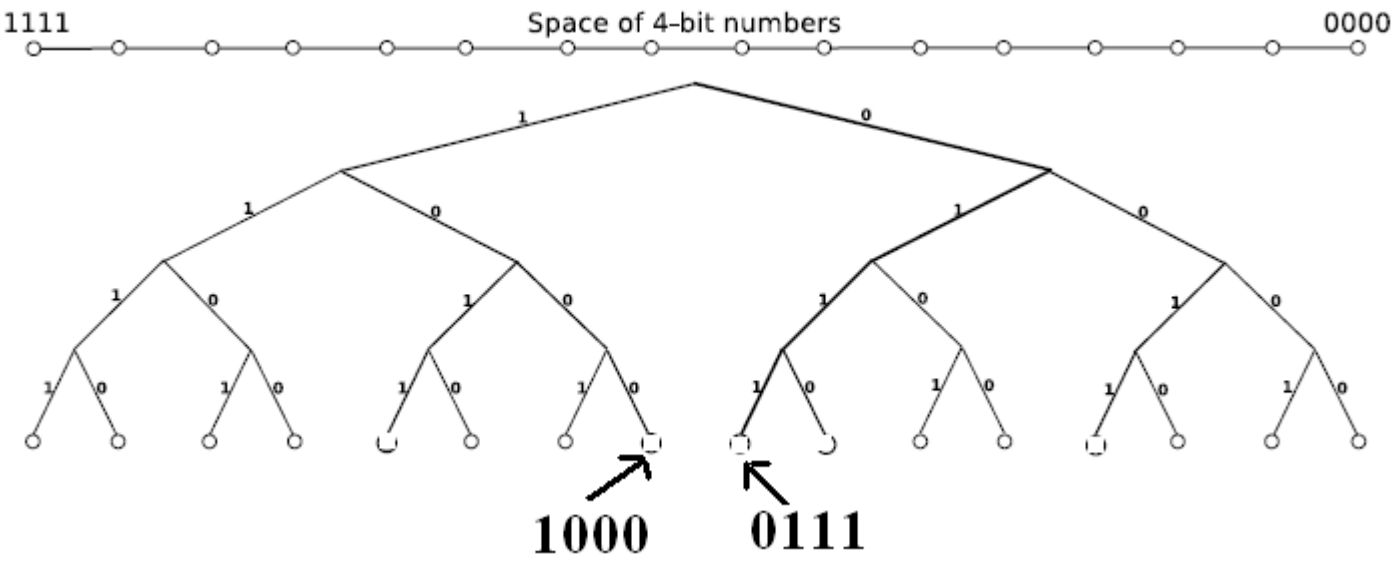
KADEMLIA:LO SPAZIO DEGLI IDENTIFICATORI



- due foglie vicine hanno identificatori numericamente vicini, ma distanti secondo la metrica dell'OR esclusivo

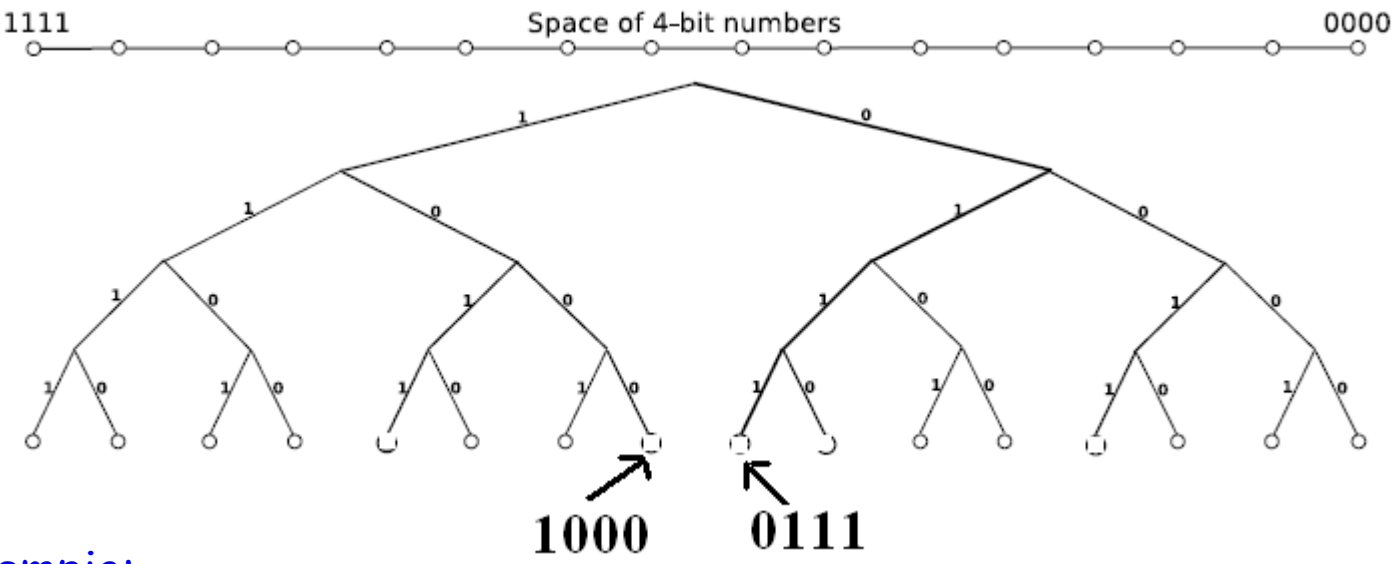
Esempio: 1000 XOR 0111 = 1111, distanza = 15

KADEMLIA:LO SPAZIO DEGLI IDENTIFICATORI



- consideriamo un identificatore n, ad esempio l'identificatore 0111
- consideriamo il cammino dalla radice al nodo n = 0111
- lungo tale cammino si incontra una sequenza T di sottoalberi che non contengono n, $T=T_1,....T_k$
- in generale la distanza d tra il nodo n ed un qualsiasi nodo appartenente ad un T_i di altezza h è compresa nel seguente intervallo: $2^{h-1} \leq d < 2^h$

KADEMLIA:LO SPAZIO DEGLI IDENTIFICATORI



Ad esempio:

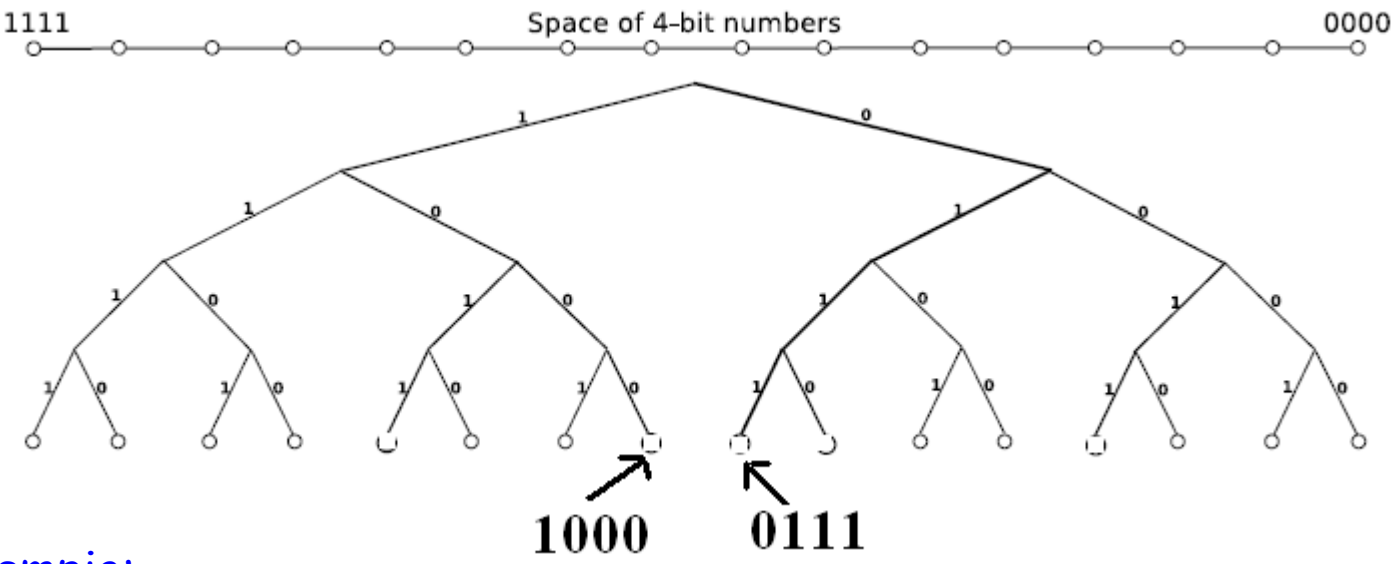
calcolare la distanza tra il nodo n=0111 ed una qualsiasi foglia appartente al sottoalbero sinistro della radice, di altezza 4

1000 XOR 0111 = 1111 =15

1111 XOR 0111 = 1000 = 8

$8 \leq d < 15$

KADEMLIA:LO SPAZIO DEGLI IDENTIFICATORI



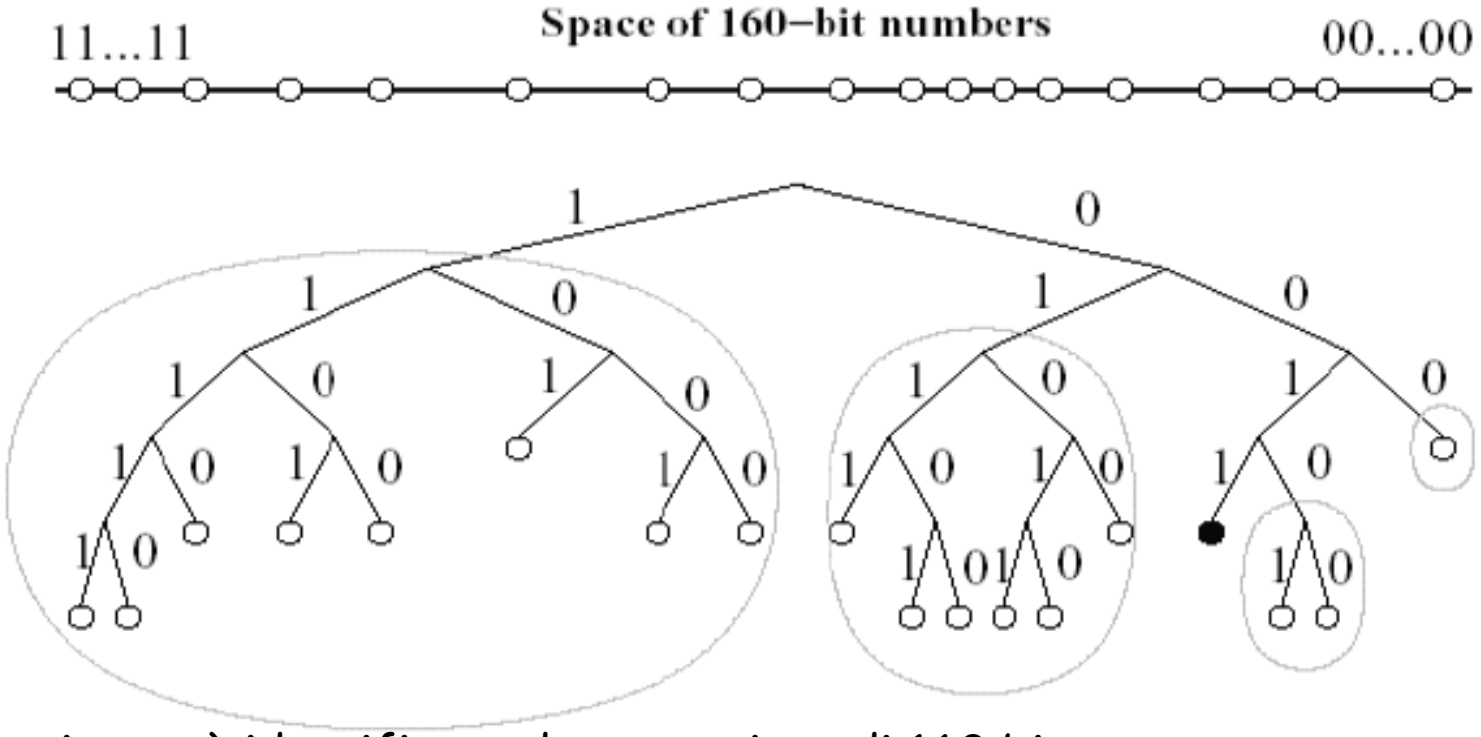
Ad esempio:

calcolare la distanza tra il nodo n=0111 ed la foglia 0110 appartente al sottoalbero adiacente a quello contenente n

0110 XOR 0111 = 0001 = 1

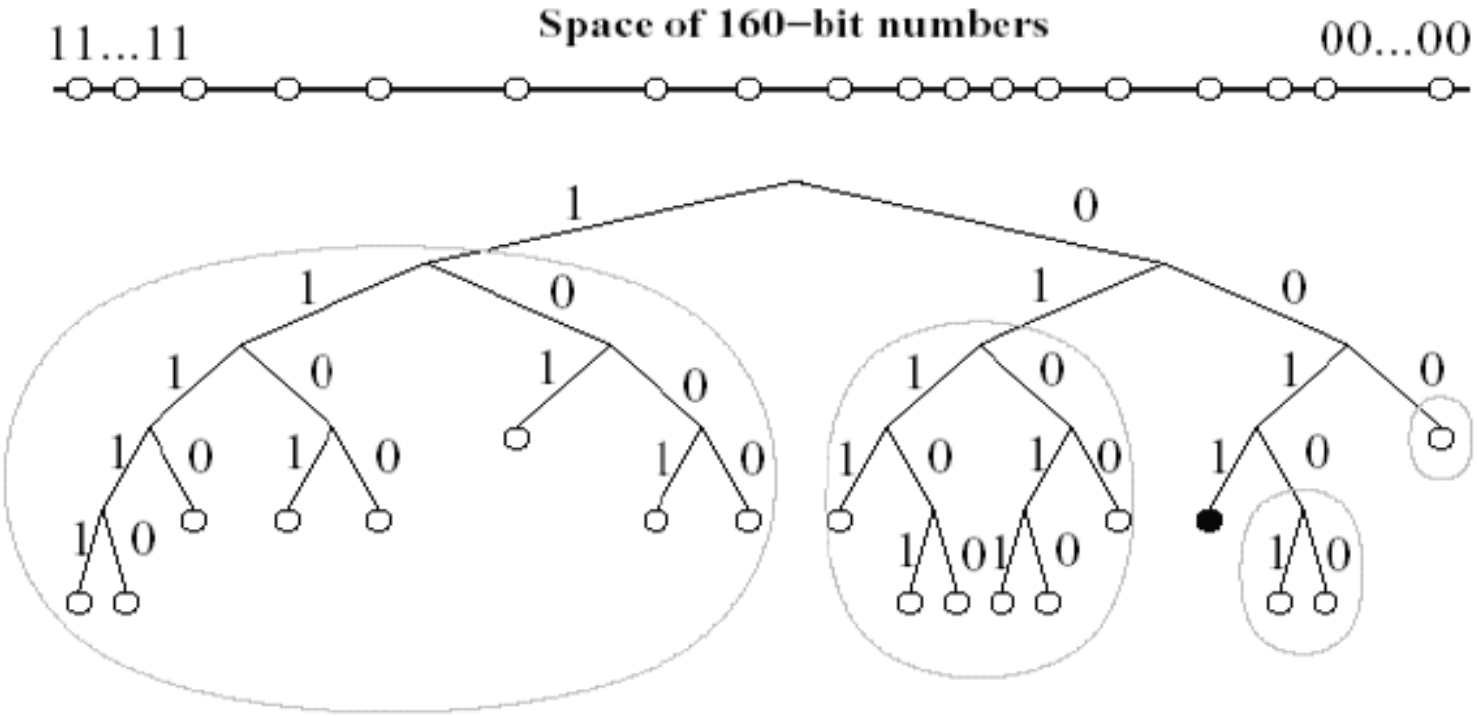
$2^0 \leq d < 2^1$

KADEMLIA:L'ALBERO DEI PEER



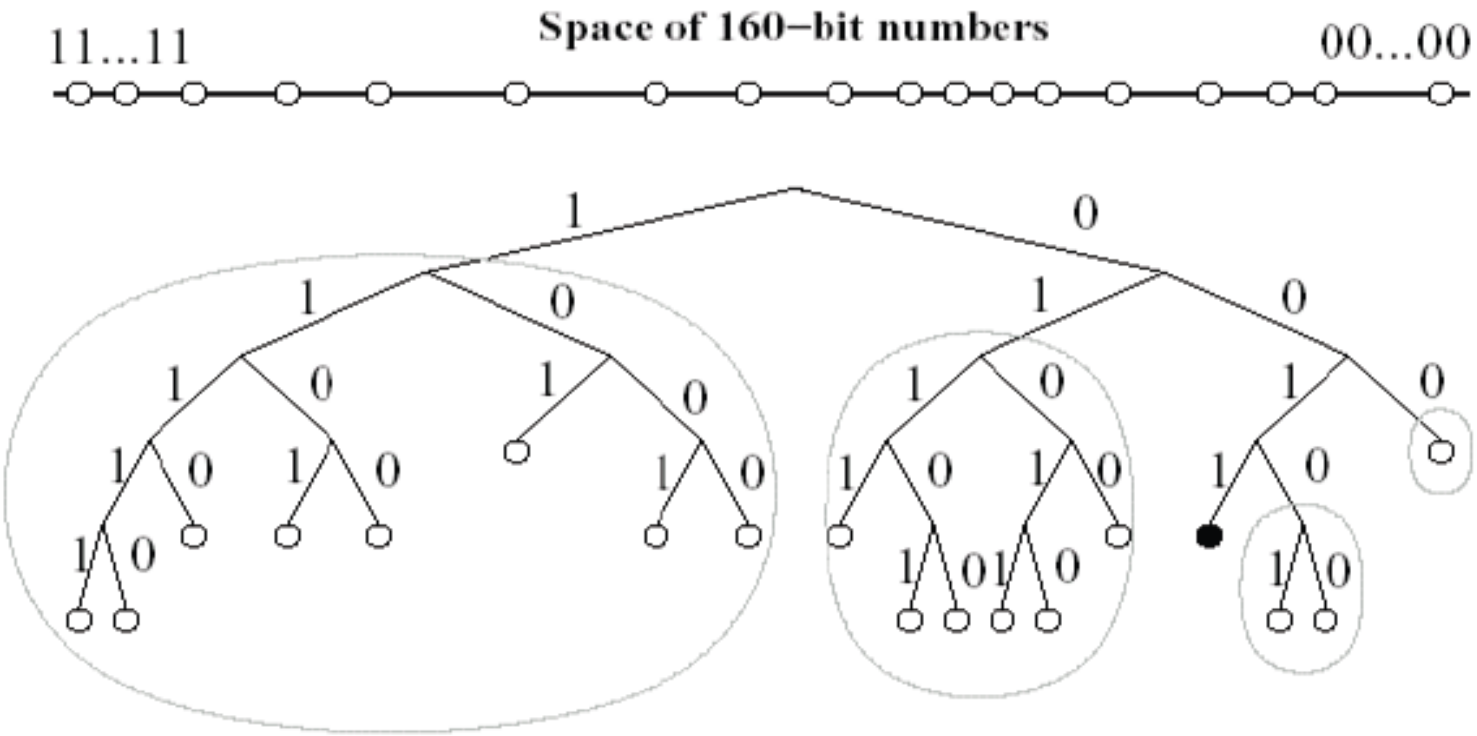
- ogni peer è identificato da una stringa di 160 bits
- costruiamo un albero binario non bilanciato che metta in evidenza solo gli identificatori assegnati a peer presenti sulla rete
- ogni nodo dell'albero corrisponde ad un peer presente sulla rete
- l'identificatore del nodo rappresenta, in generale, un prefisso dell'identificatore del peer

KADEMLIA:L'ALBERO DEI PEER



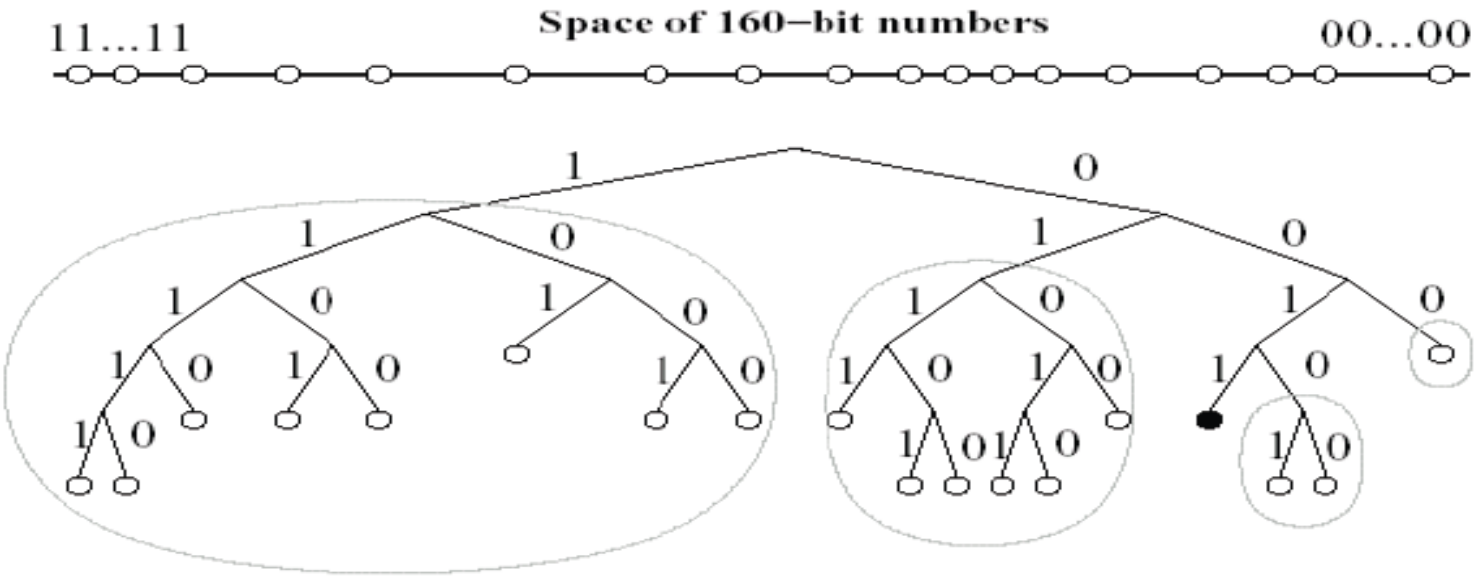
- ogni peer è rappresentato da una foglia f dell'albero tale che ID(f) è un prefisso dell'identificatore del peer ed ID(f) identifica univocamente il peer all'interno dell'overlay
- Esempio: la foglia identificata dall'ID 0111 rappresenta un peer il cui identificatore è rappresentato dal prefisso 0111.
- Nessun altro peer è identificato da un identificatore con lo stesso prefisso

KADEMLIA:L'ALBERO DEI PEER



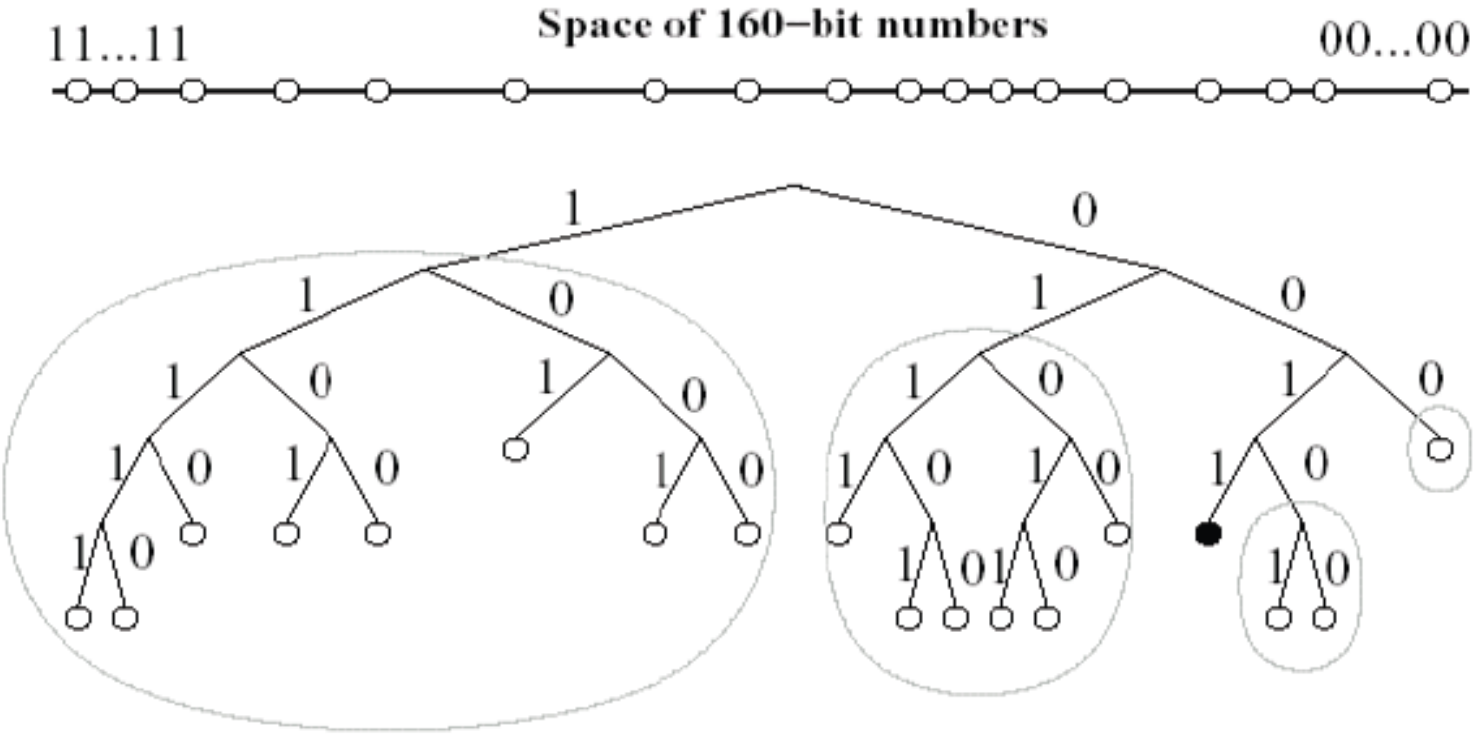
- ogni peer è rappresentato dal nodo n tale che ID(n) è un prefisso dell'identificatore del peer ed ID(n) identifica univocamente il peer all'interno dell'overlay

KADEMLIA: METRICA ED ALBERO BINARIO



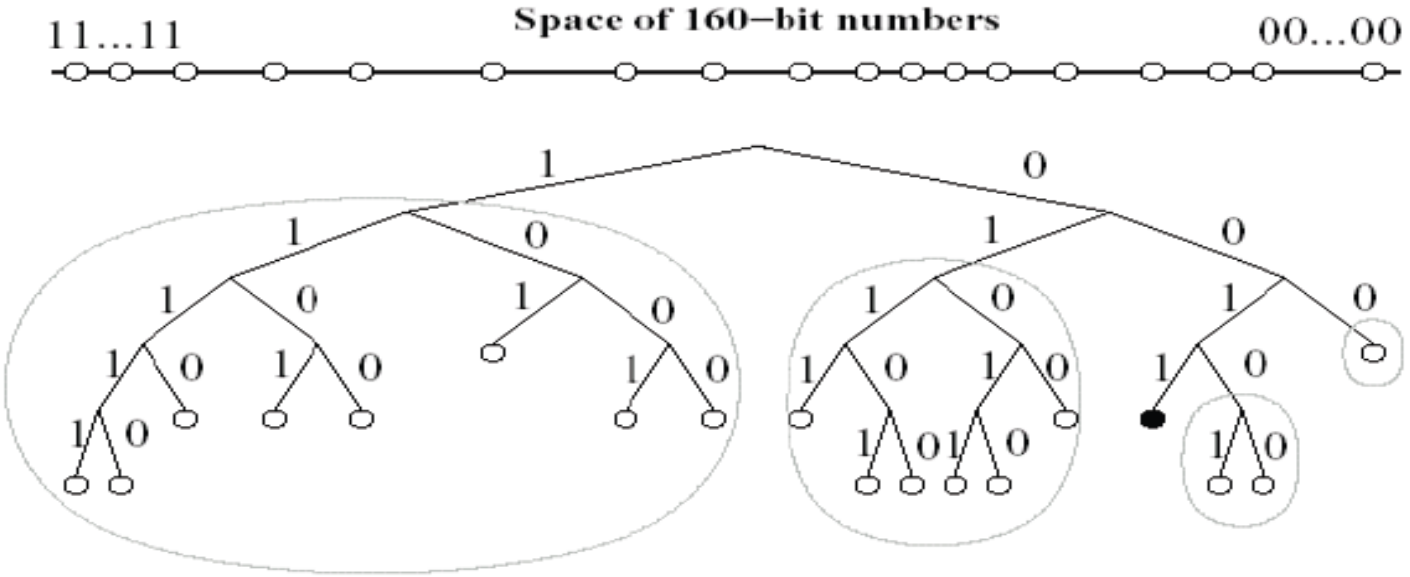
- la foglia più vicina ad un nodo di identificatore x è quella che condivide il prefisso più lungo con il nodo x,
- ad esempio la foglia più vicina al nodo 1000….è 1001…, qualsiasi altro nodo differirà per un bit più significativo e la distanza sarà maggiore

KADEMLIA: LE TABELLE DI ROUTING



- l'albero binario viene suddiviso, a partire dalla radice, in una successione di sottoalberi sempre più piccoli che non contengono n
- il primo sottoalbero contiene metà dell'albero binario che non contiene n
- Il secondo sottoalbero contiene la metà della rimanente parte dell'albero che non contiene n e così via

KADEMLIA: METRICA ED ALBERO BINARIO



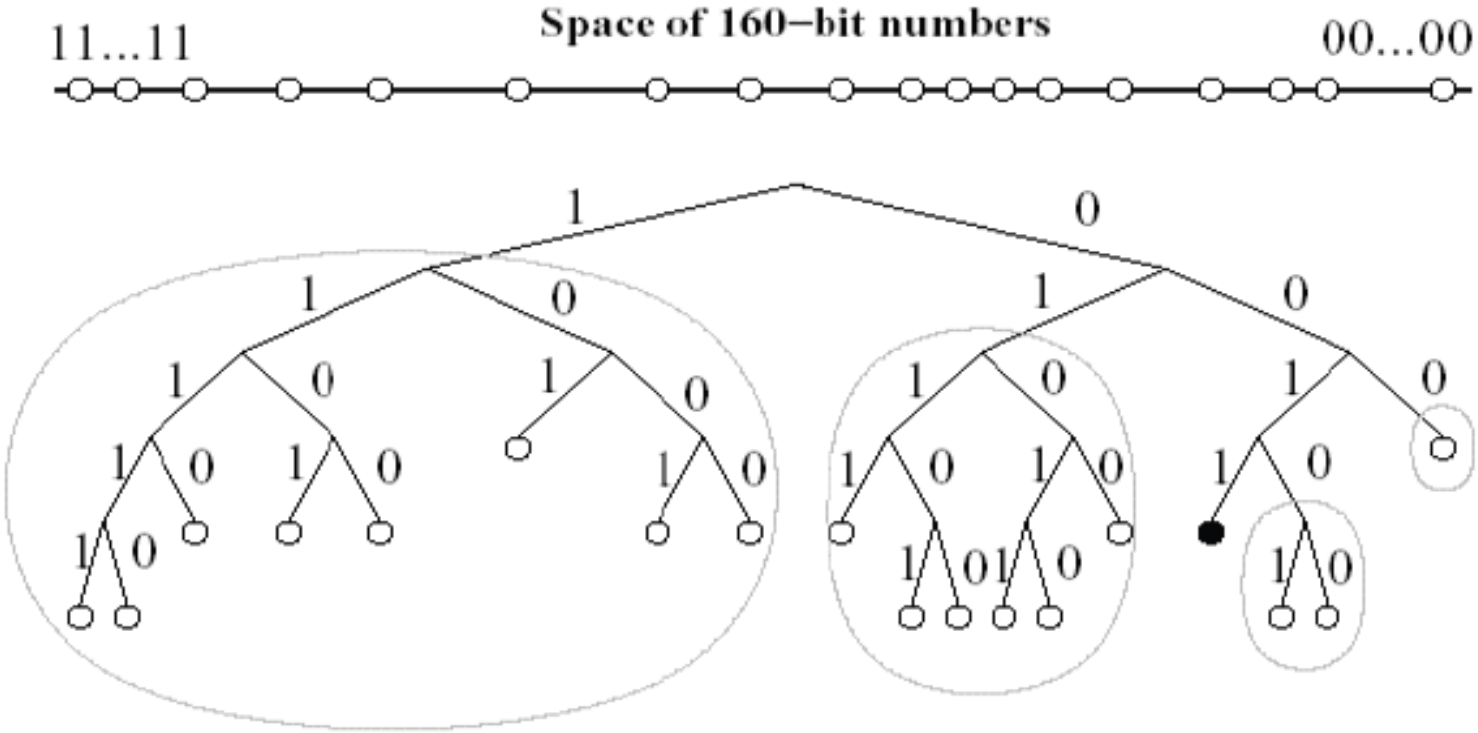
- consideriamo il peer nero n, ogni peer nel sottoalbero sinistro della radice ha un id che differisce da quello di n nella cifra più significativa
- la distanza d tra n ed un qualsiasi peer nel sottoalbero sinistro è tale che

$$2^{159} \leq d < 2^{160}$$

- in generale, vale la formula vista in precedenza

$$2^{i-1} \leq d < 2^{i}$$

KADEMLIA: LE TABELLE DI ROUTING



- Le tabelle di routing di Kademlia assicurano che ogni nodo conosca alcuni nodi in ogni sottoalbero così individuato
- Ogni query viene inviata ad uno dei nodi conosciuti in un sottoalbero
- Il nodo contattato propaga la query verso sottoalberi via via più vicini al target

KADEMLIA: TABELLE DI ROUTING

- Tabelle di routing basate sulla costruzione di K-Buckets

- K-Bucket: Lista contenente (al massimo) K (K in genere = 20) contatti. Ogni contatto è una tripla del tipo

  <IP address, UDP port, Node ID>

- La tabella di routing contiene B k-buckets, B =160 (lunghezza degli ids)

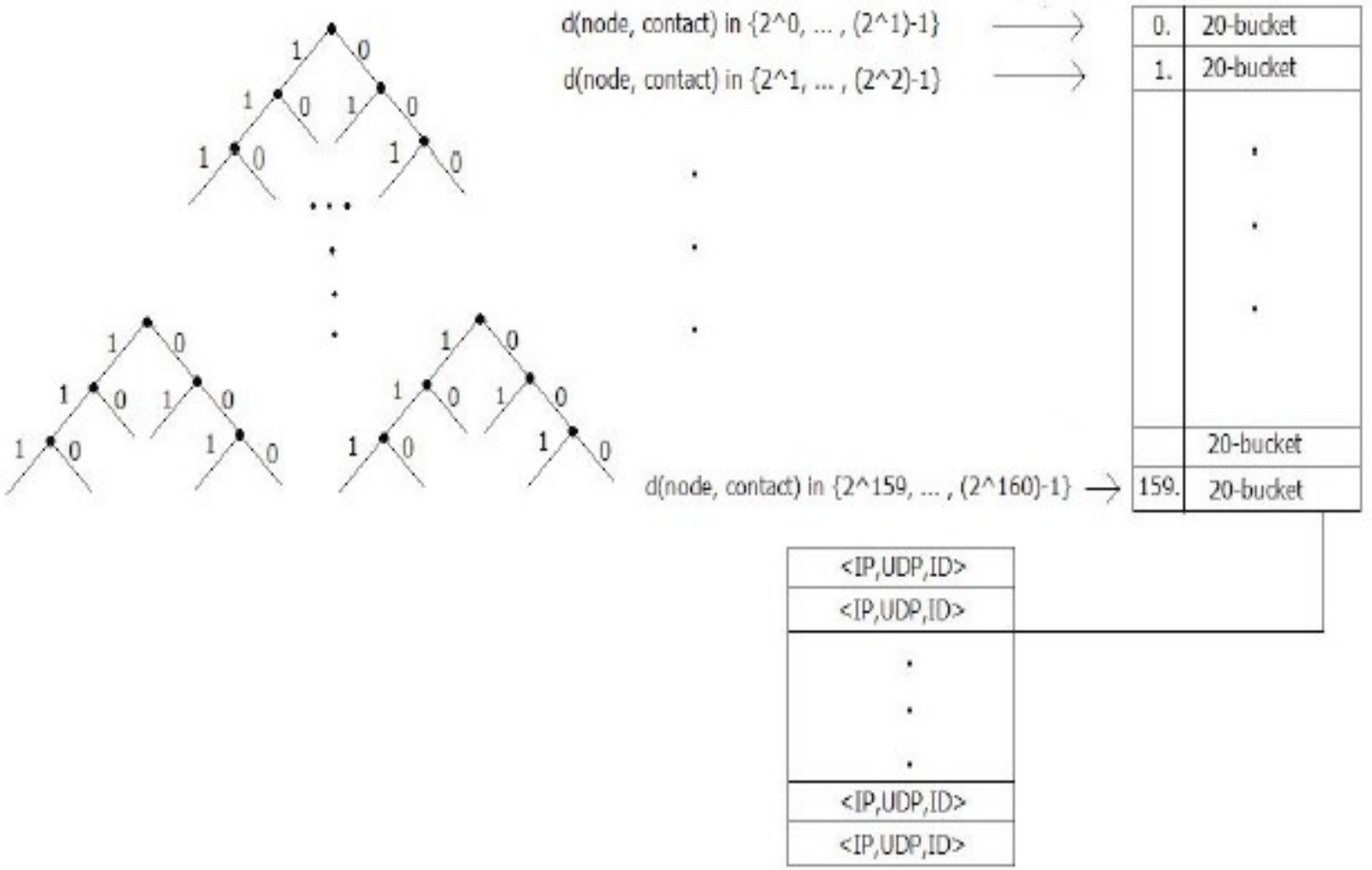
- Il k-bucket di indice i del nodo n, per ogni $0 \leq i \leq 160$, contiene riferimenti a nodi che si trovano ad una distanza compresa tra 2^i e 2^{i+1} da n

-

- Dato il bucket di indice j presente nalla tabella di routing del nodo node e considerato un contatto contact in quel bucket, vale che

$$\forall\, 0 \leq j < B : 2^j \leq d(node, contact) < 2^{j+1}$$

KADEMLIA: LE TABELLE DI ROUTING

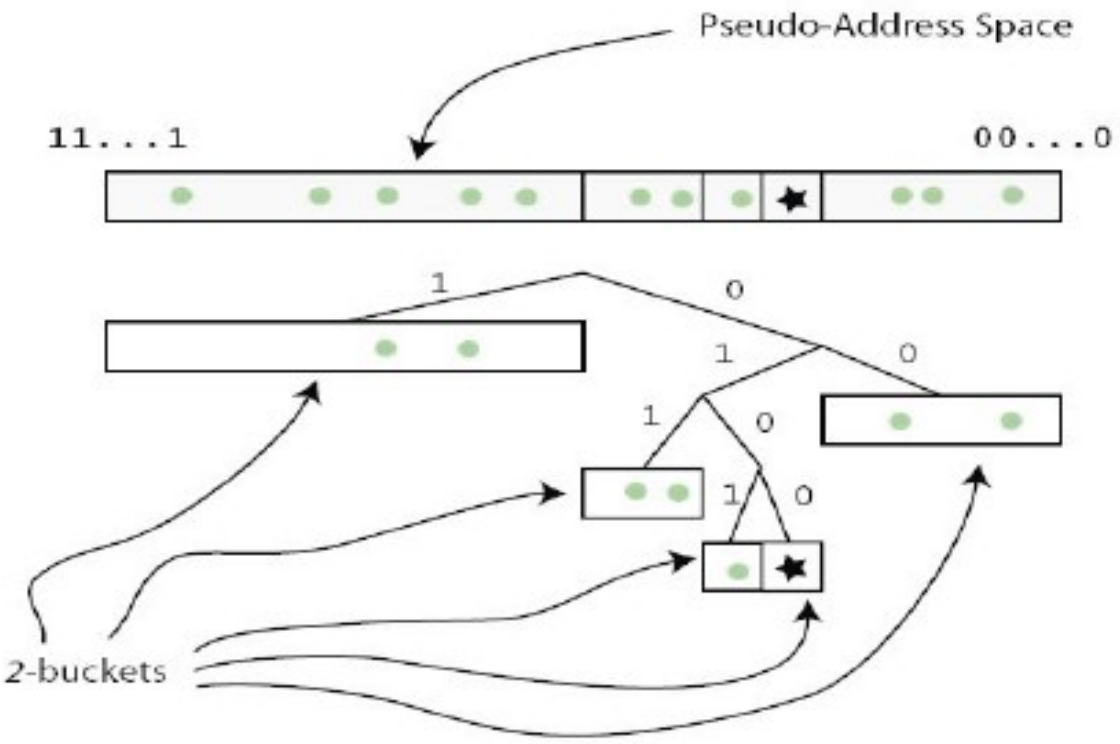


KADEMLIA: LA TABELLA DI ROUTING

- Ogni K bucket copre un sottospazio dello spazio degli identificatori
- L'insieme tutti i k-buckets copre l'intero spazio degli identificatori
- alcune entrate della tabella di routing contengono poche entrate
- altri buckets contengono un maggior numero di contatti, ma mai più di K
- Ogni K bucket contiene i nodi che presentano un certo prefisso comune con il nodo dato
- Il valore di K viene stabilito in modo che la probabilità che si verifichi un crash d più di K nodi del sistema sia un evento altamente improbabile

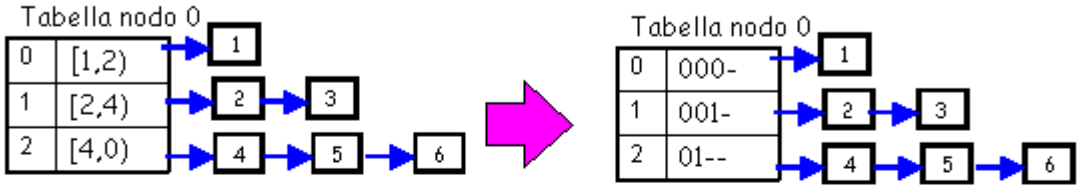
KADEMLIA: LE TABELLE DI ROUTING

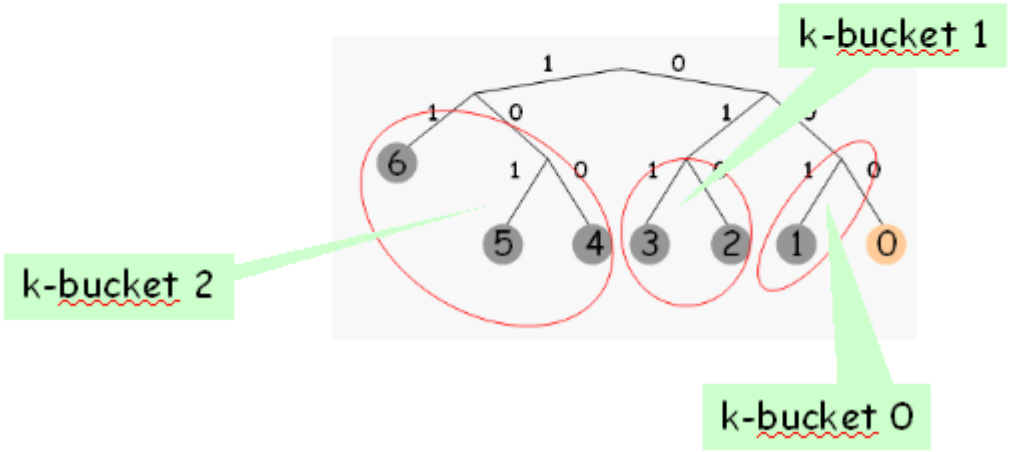
- La tabella di routing è un albero binario, dove le foglie sono i k buckets



KADEMLIA: LA TABELLA DI ROUTING

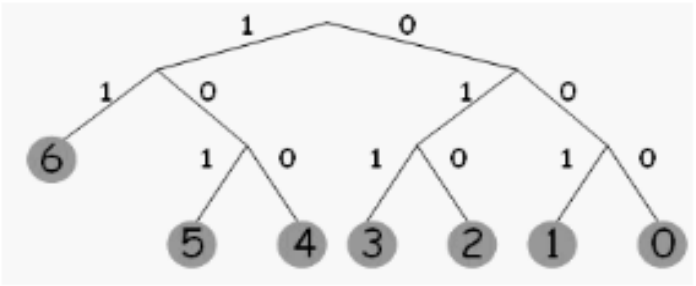
Kademlia: k-buckets (5/5)





KADEMLIA: LA TABELLA DI ROUTING

Esempio: tabella di routing continente un insieme di 2-buckets per ogni riga



m=3 #bit chiave

Tabella nodo 0

0	[1,2)
1	[2,4)
2	[4,0)

nodo1

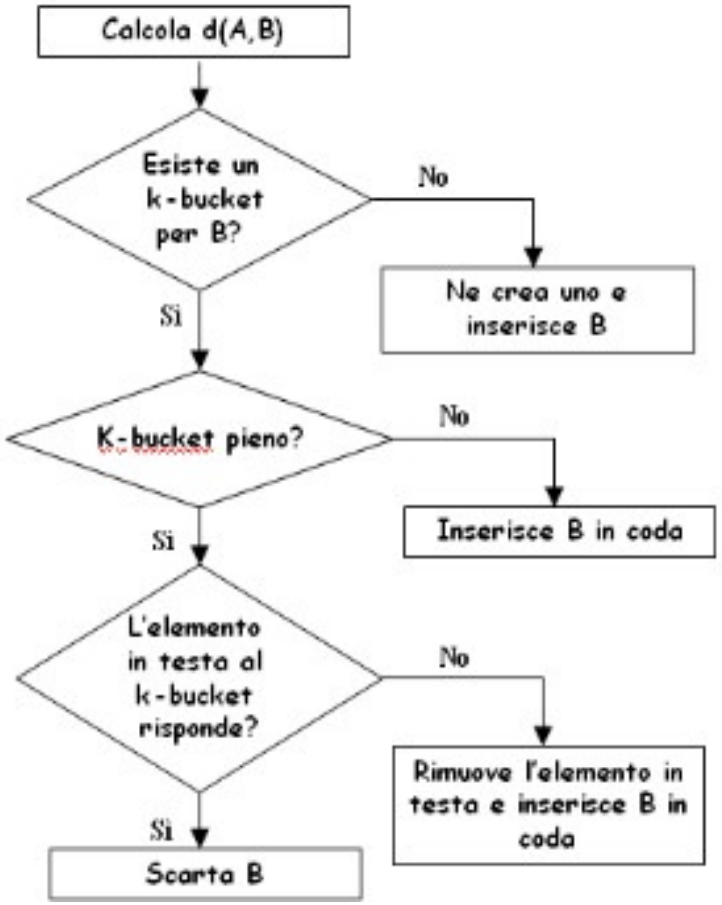
nodo2 → nodo3 } K-bucket

nodo4 → nodo5

K = numero massimo di elementi in un k-bucket:
parametro di sistema fisso (qui k=2)

KADEMLIA: GESTIONE DEI K-BUCKETS

Supponiamo che il nodo A riceva un messaggio da un nodo B



KADEMLIA: GESTIONE DEI K-BUCKETS

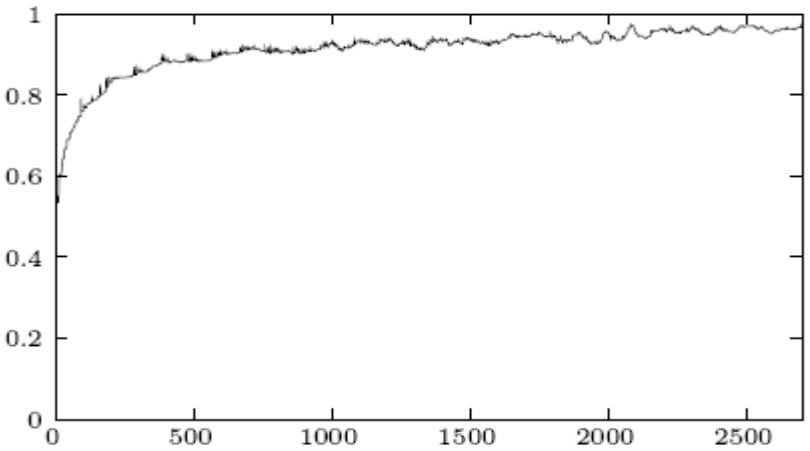
- Quando un nodo N Kademlia riceve un messaggio da un nodo M, può utilizzare l'ID di M per aggiornare il k-bucket B relativo

- Se B non contiene M ed B non è pieno, M viene aggiunto a B

- Se B non contiene M e B è pieno, N invia un ping al nodo P con cui ha avuto contatti meno recentemente
  - se P non risponde entro un certo intervallo di tempo, N elimina dal bucket P e vi inserisce M
  - Se P risponde, M viene scartato

- Politica per la gestione dei k-buckets: least recently seen evinction,
  - i nodi eliminati sono quelli in testa alla lista del k-bucket, cioè quelli contattati meno di recente
  - un nodo che fa ancora parte della rete non viene mai rimosso

KADEMLIA: GESTIONE DEI K BUCKETS



- Uptime= tempo totale di permanenza di un peer sull'overlay
- Il grafico mostra la percentuale di nodi di una rete Gnutella che rimangono online nell'ora successiva in funzione del loro uptime

  L'immagine mostra come più un nodo rimane online più è alta la probabilità che vi rimanga ancora per un lungo intervallo di tempo
- Questa considerazione è alla base della politica di Kademlia per la gestione dei contatti nei k-buckets: si preferisce sempre mantenere nel bucket i contatti che sono presenti sulla rete da una maggior quantità di tempo

KADEMLIA: GESTIONE DEI K-BUCKETS

- **Esempio:**

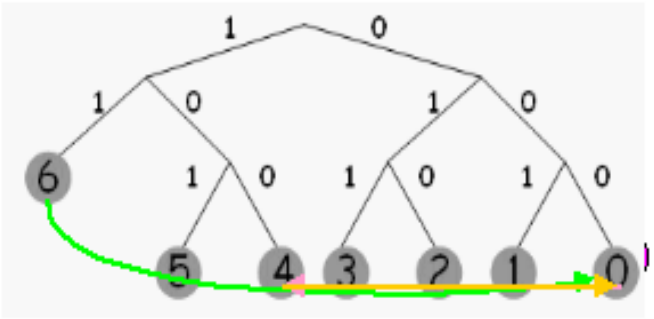


Tabella nodo 0

0	[1,2) → nodo1
1	[2,4) → nodo2 → nodo3
2	[4,0) → nodo4 → nodo5 → nodo4

m=3; k=2

1. 6 manda un messaggio a 0, ma il k-bucket è pieno (k=2)
   1. 0 interroga 4
   1. 4 risponde e viene spostato al fondo (6 è fuori)

KADEMLIA: GESTIONE DEI K-BUCKETS

- Esempio:

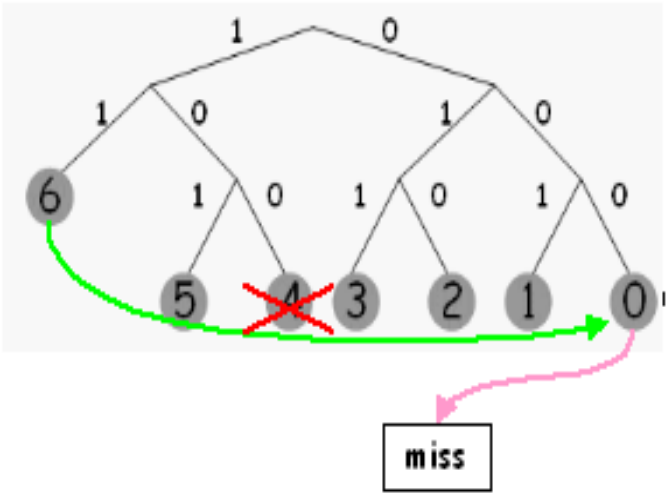


Tabella nodo 0

0	[1,2)
1	[2,4)
2	[4,0)

nodo1
nodo2
nodo3
nodo4
nodo5
nodo6

m=3; k=2

1. 6 manda un messaggio a 0, ma il k-bucket è pieno (k=2)
1. 0 interroga 4 (nodo in testa)
1. 4 non risponde: 6 è messo in fondo al k-bucket

KADEMLIA: REFRESH PERIODICO DEI K-BUCKETS

- I k-buckets sono rinfrescati periodicamente dal passaggio di richieste attraverso i diversi nodi.
  - Anche se qualche nodo abbandona la rete, le nuove informazioni ricevute dai nodi a cui inoltro le query 'rinfrescano' la k-bucket list
- Può tuttavia accadere che un k-bucket non sia rinfrescato per un certo periodo a causa della mancanza di richieste sui nodi del range coperto dal k-bucket
- Per questa ragione un refresh viene effettuato una volta ogni ora
  - si sceglie casualmente un ID all'interno del range coperto dal bucket e si effettua una ricerca per quell'ID

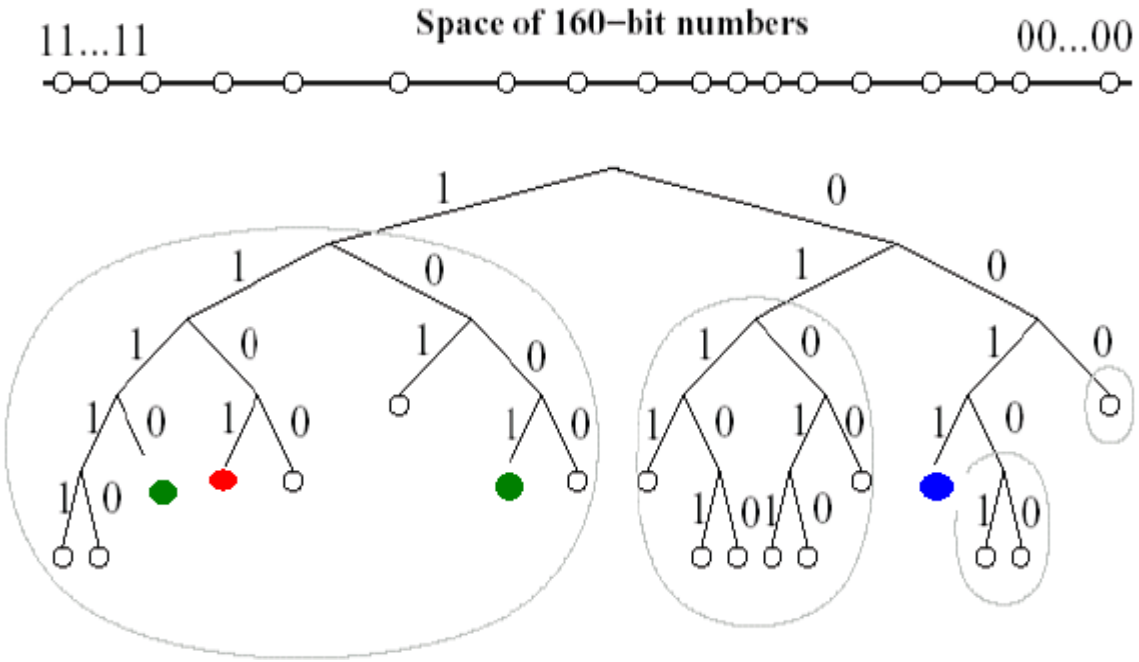
KADEMLIA: IL ROUTING

- Al momento del join(),
  - il nuovo nodo inserisce il nodo di bootstrap nel k-bucket relativo
  - individua alcuni nodi vicini e li inserisce nei bucket
  - il k-bucket viene poi arrichito attraverso l'informazione contenuta nelle query che passano attraverso il nodo
- Consideriamo un nodo x e altri due nodi y e z, con

  $$dist(y,x) < dist(z,x).$$

  è possibile che z conosca x, mentre y non conosce x. Questo è dovuto alla strategia con cui vengono creati i k-buckets, che prevede di costruire incrementalmente i bucket con l'informazione di cui si viene dinamicamente a conoscenza
- Non sempre l'invio della query al nodo più vicino al target porta al cammino più breve verso il target
- Routing: si invia la query agli α nodi più vicini al target. La proprietà della unidirezionalità garantisce che tutti i cammini convergano verso il target

KADEMLIA: IL ROUTING



Supponiamo che il nodo blu 0011 ricerchi il nodo rosso 1101.
Possiede un riferimento ai nodi verdi 1001 e 1110, per cui vale

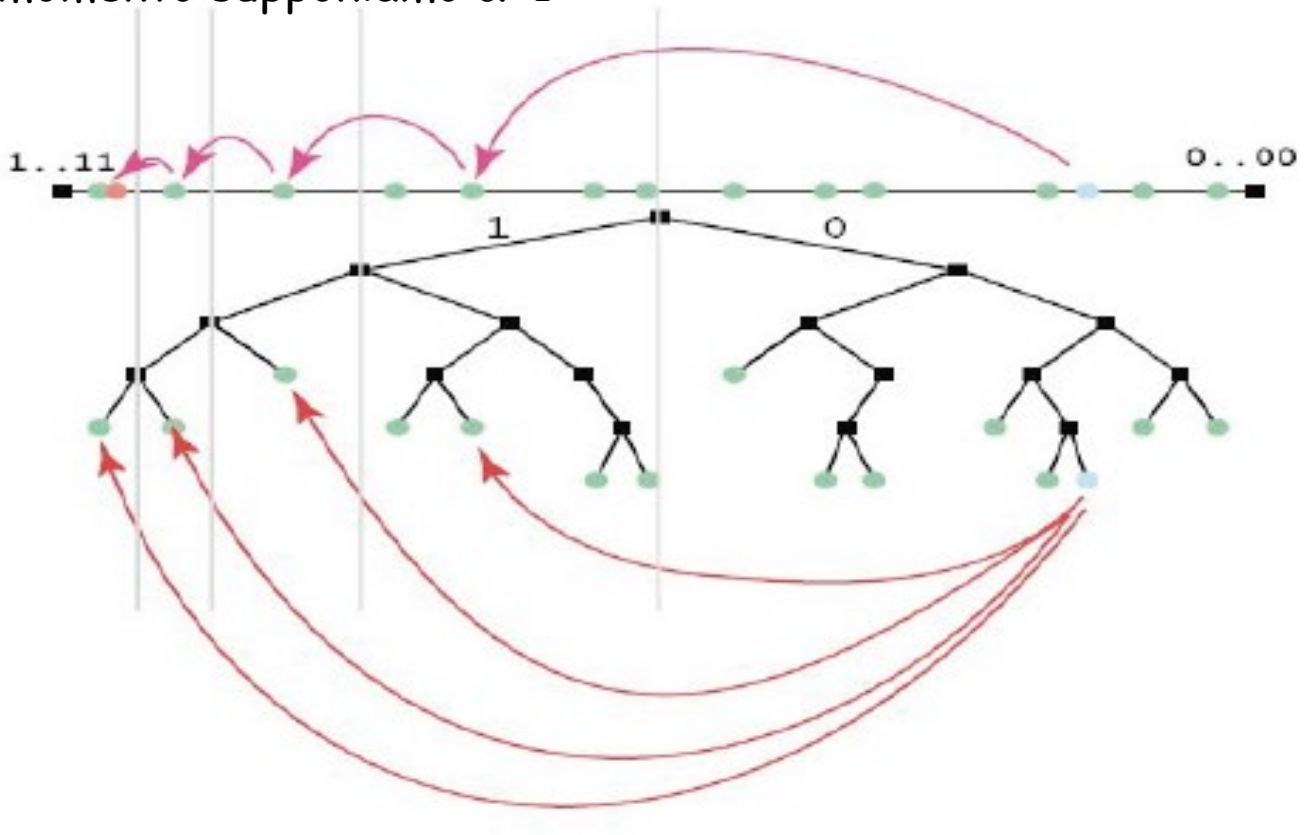
dist (1101, 1001) = 4

dist (1101, 1110) = 3

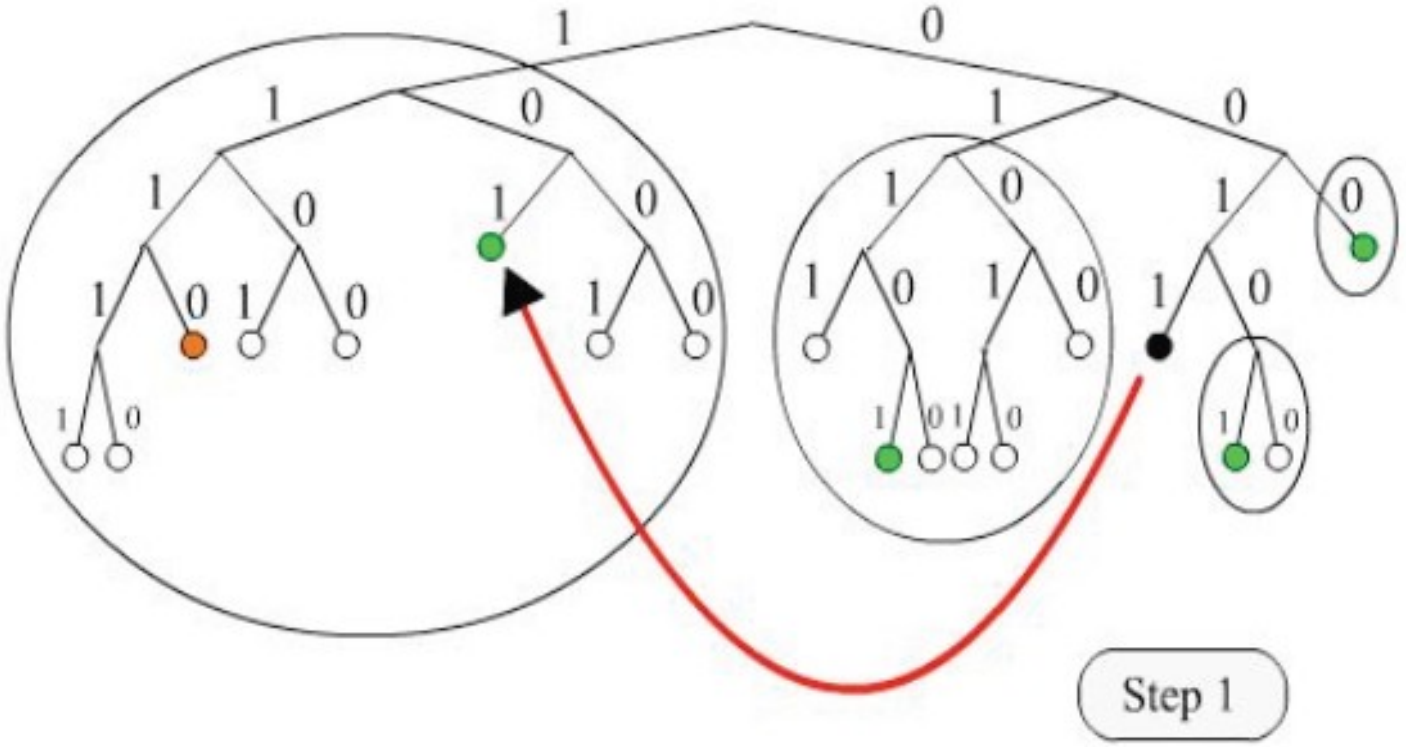
è possibile che il nodo più distante dal target, 1001, possieda un riferimento al target, mentre il nodo più vicino 1110 non lo possiede
Il nodo blu invia la richiesta ad entrambe i nodi

KADEMLIA: IL ROUTING

- α= numero di nodi a cui viene propagata la query, ad ogni passo
- per il momento supponiamo α=1



KADEMLIA: IL ROUTING(α=1)

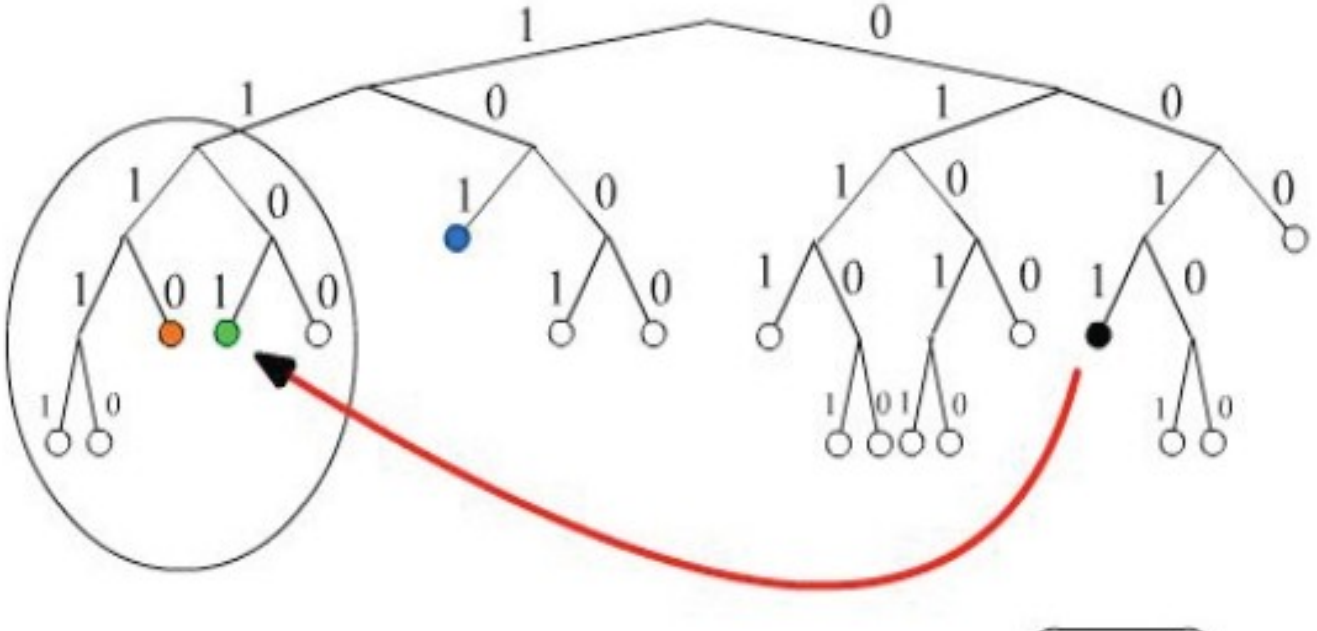


Nodo nero :sorgente della query (0011)

Nodo arancio : target della query (1110)

Nodi verde : nodi conosciuti dalla sorgente negli altri sottoalberi

KADMELIA: IL ROUTING(α=1)



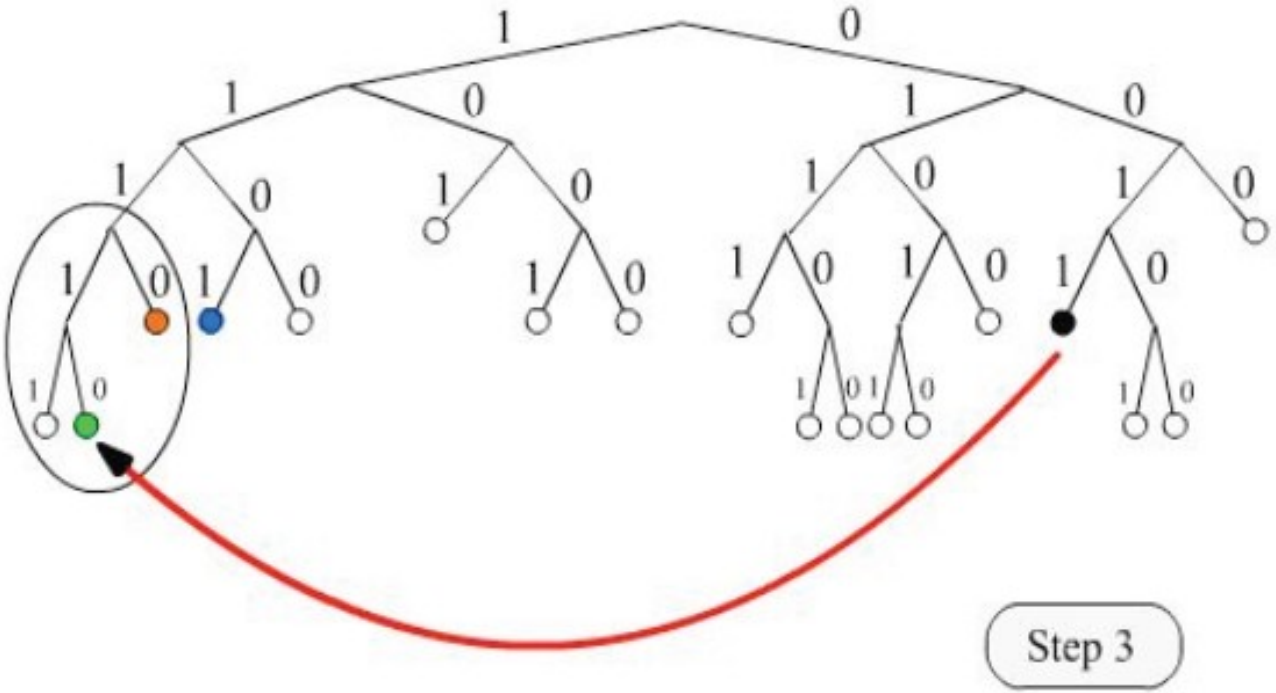
Nodo nero : sorgente della query (0011)

Nodo arancio : target della query (1110)

Nodi verde : nodi conosciuti dalla sorgente negli altri sottoalberi

Nodo Blu : restituisce alla sorgente un ulteriore nodo da contattare (iterative routing)

KADMELIA: IL ROUTING(α=1)



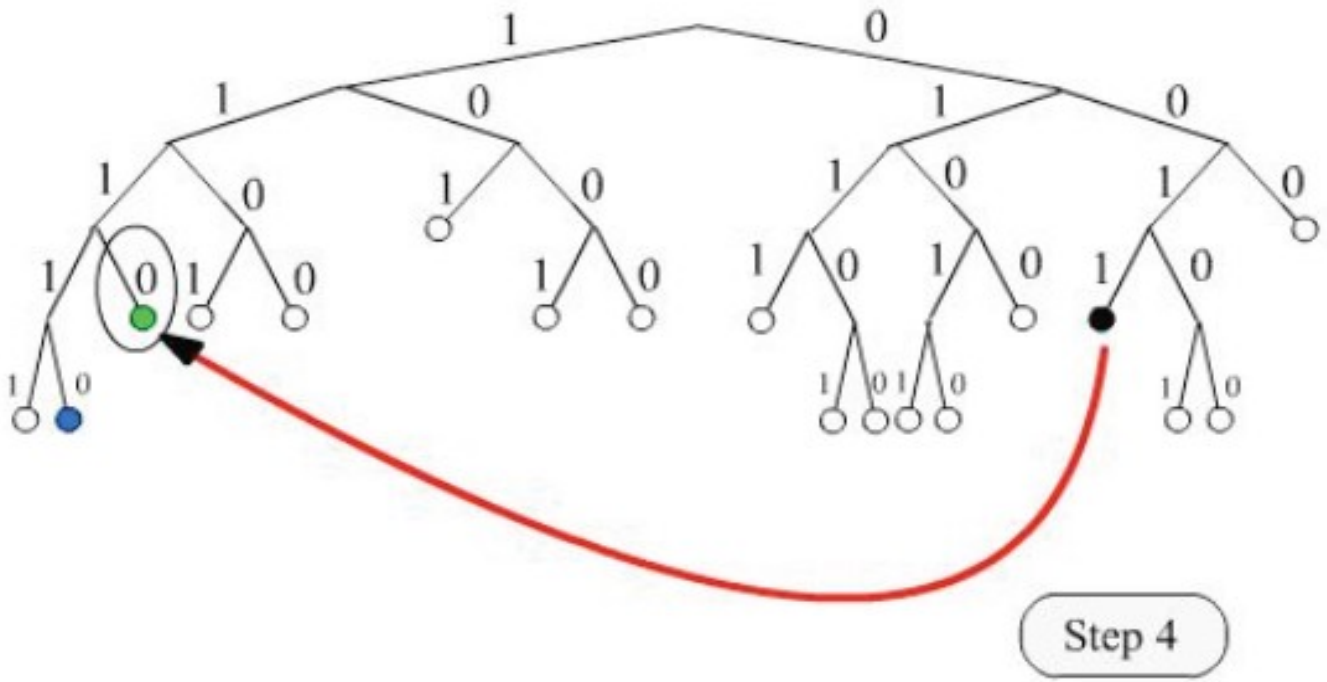
Nodo nero : sorgente della query (0011)

Nodo arancio : target della query(1110)

Nodi verde : nodi conosciuti dalla sorgente negli altri sottoalberi

Nodo Blu : restituisce alla sorgente un ulteriore nodo da contattare

KADMELIA: IL ROUTING(α=1)



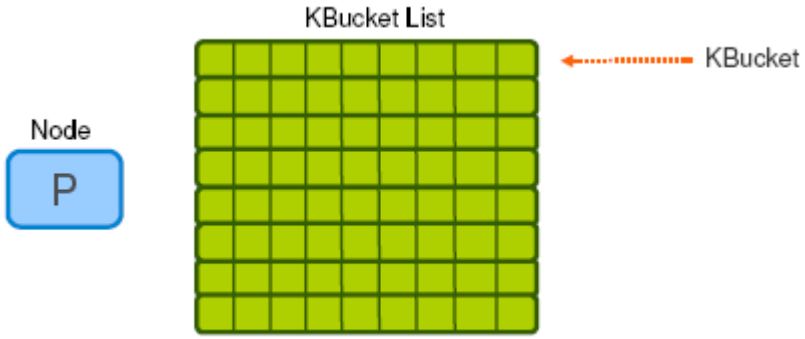
Nodo nero : sorgente della query (0011)

Nodo arancio : target della query (1110)

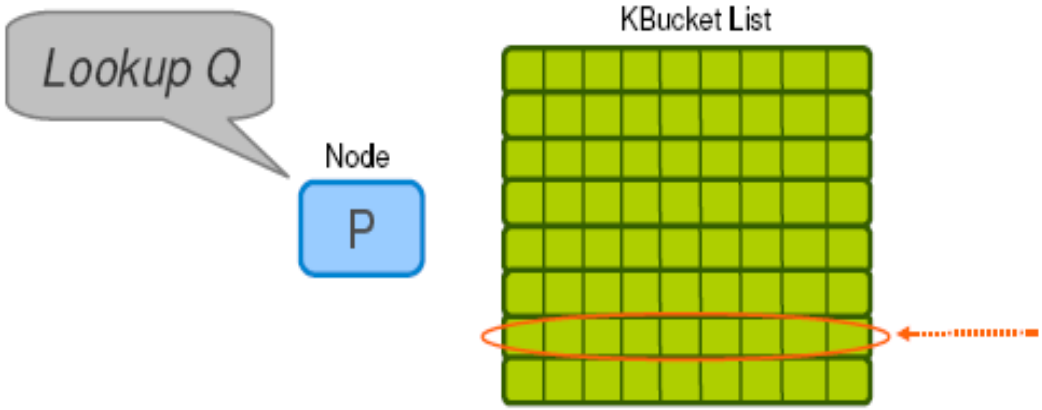
Nodi verde : nodi rappresentanti negli altri sottoalberi

Nodo Blu : restituisce alla sorgente un ulteriore nodo da contattare

KADEMLIA: IL ROUTING (α>1)



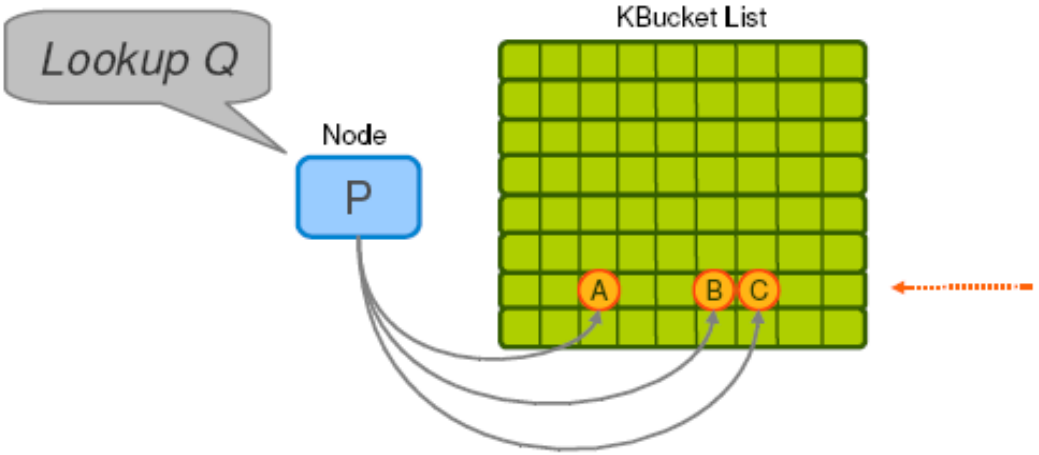
KADEMLIA: (α>1)



Memorizza i nodi più vicini a Q

- P deve ricercare la chiave K (può essere l'identificatore di un nodo o di un dato)
- P ricerca nella bucket-list i nodi che sono più vicini a Q
- Sono i nodi che presentano il prefisso più lungo a comune con Q
- Sono quelli contenuti nei sottoalberi più vicini

KADEMLIA: (α>1)

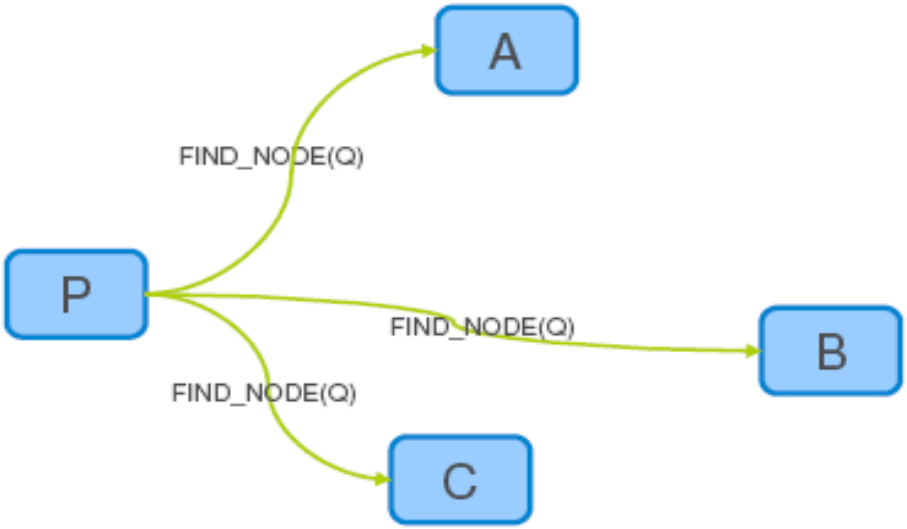


Memorizza i nodi

più vicini a Q

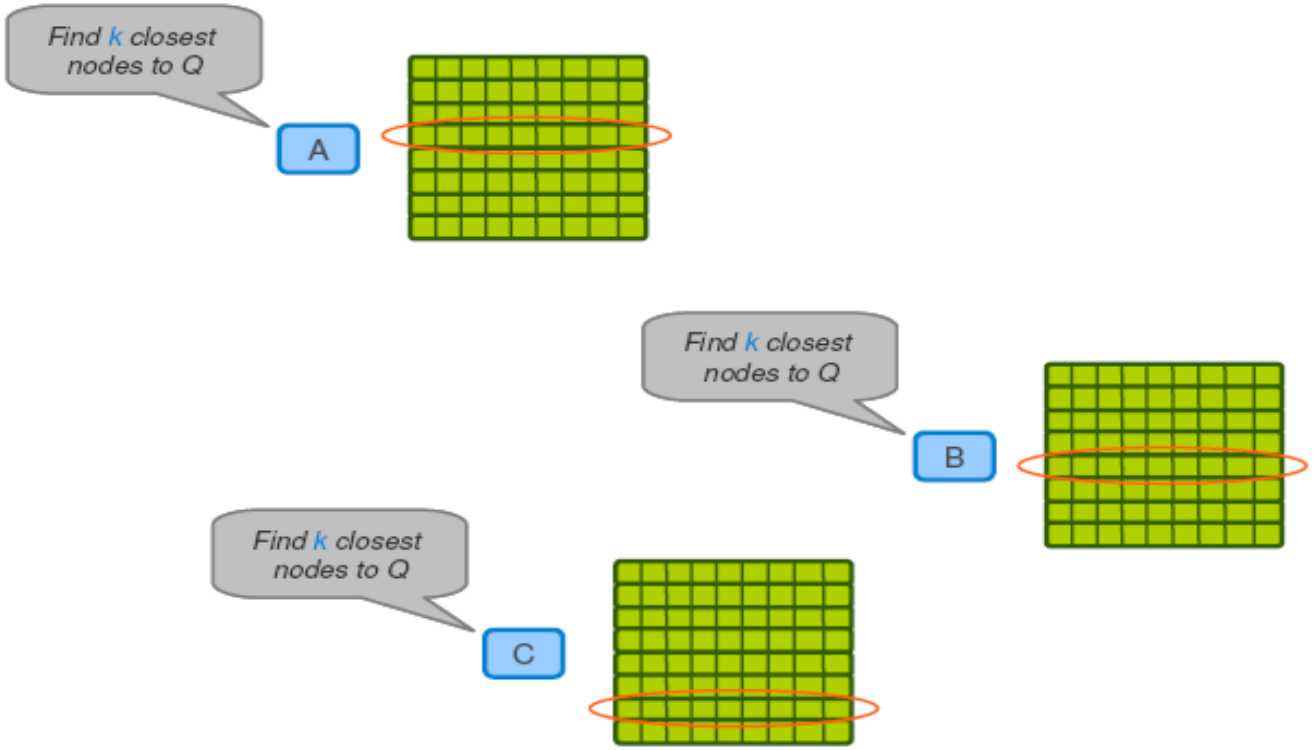
: P seleziona κ nodi dal bucket individuato

KADEMLIA: (α>1)



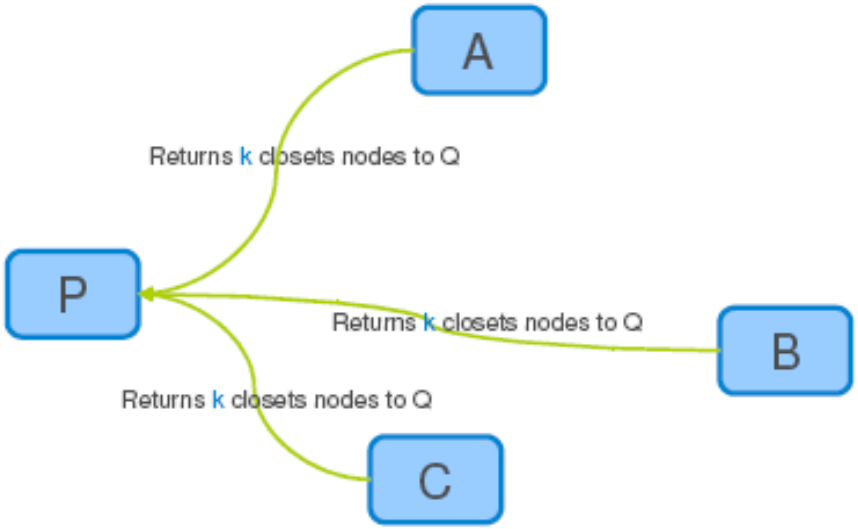
- P inoltra la query in parallelo a tutti i nodi individuati
- A differenza di Chord, la query può essere inviata in
- parallelo a più nodi

KADEMLIA: (α>1)



- Ogni nodo contattato individua a sua volta k nodi più vicini alla chiave
- Ogni nodo può sfruttare una diversa riga nella bucket list

KADEMLIA: (α>1)



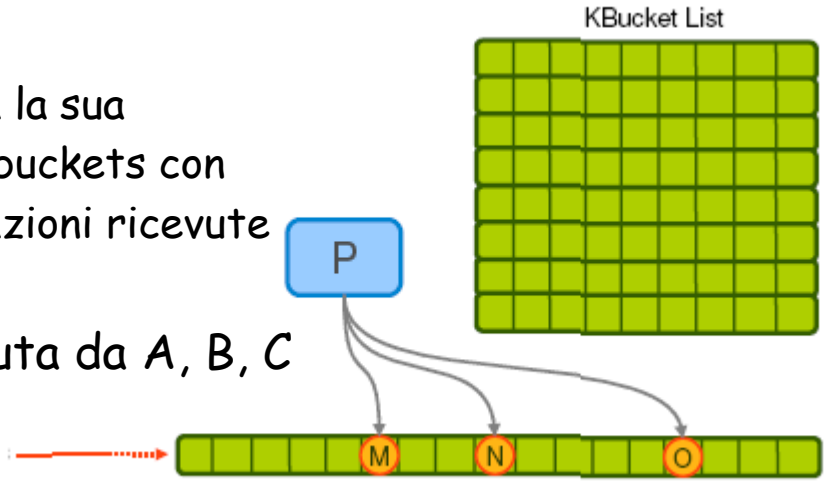
Routing Iterativo:

- ogni nodo restituisce i risultati individuati a P
- P continua il processo di routing con i risultati ricevuti

KADEMLIA: (α>1)

P aggiorna la sua
lista di k-buckets con
le informazioni ricevute

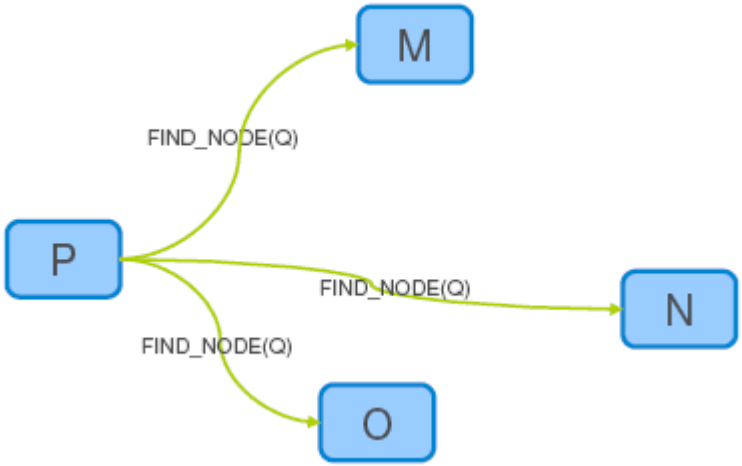
Informazione ricevuta da A, B, C



P seleziona nuovamente α nodi dalla

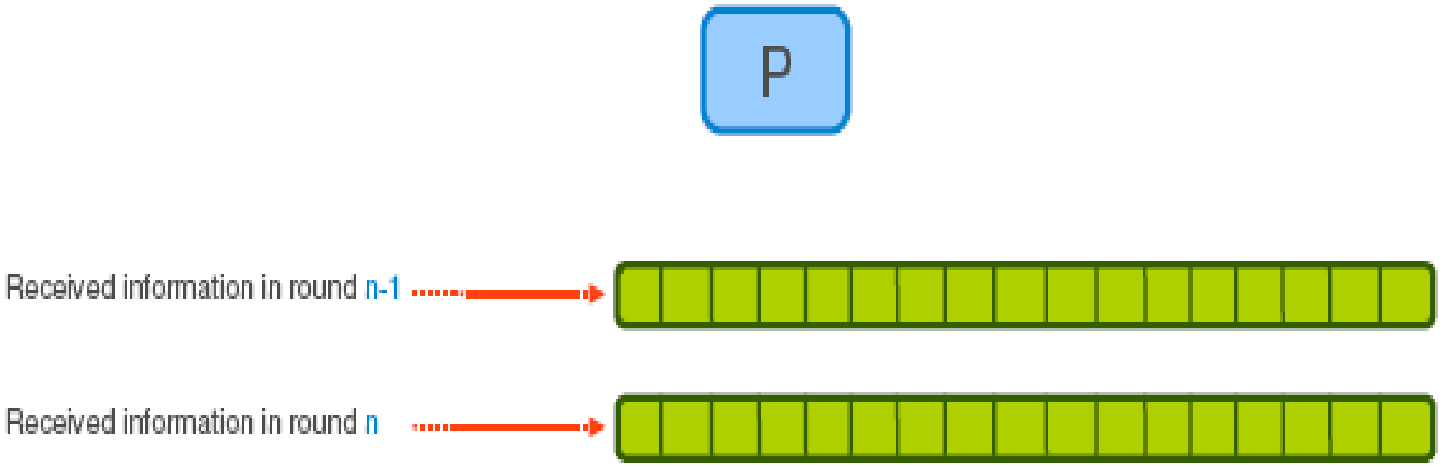
informazione ricevuta

KADEMLIA:(α>1)



- P inoltra la richiesta ai nuovi nodi individuati

KADEMLIA:(α>1)



Il procedimento termina quando le informazioni ricevute alla iterazione n sono le solite di quelle ricevute al passo n-1

KADEMLIA: IL PROTOCOLLO

Funzioni definite dal protocollo Kademlia

- PING: utilizzato per verificare la presenza di un nodo
- FIND_NODE (ID): dato un identificatore logico ID appartenente allo spazio logico di Kademlia, restituisce k triple (Indirizzo IP, porta UDP, ID logico) corrispondenti ai k nodi più vicini al nodo
- FIND_VALUE (ID): si comporta come la FIND_NODE e restituisce il valore associato alla chiave
- STORE (KEY,VALUE): individua i k nodi più vicini alla chiave e memorizza su di essi la coppia chiave, valore

KADEMLIA: JOIN, LEAVE

- Per effettuare un join ad un'overlay Kademlia, un nodo n deve conoscere qualche nodo w appartenente all'overlay
- Procedura di join:
  - n inserisce w nel bucket relativo
  - effettua un look up di se stesso attraverso w
    - in questo modo trova alcuni nodi vicini
    - effettua un refresh sugli altri k buckets
  - L'uscita di un nodo non richiede operazioni aggiuntive
    - Se un nodo non risponde, esso viene eliminato dai k-bucket dei peer che gli inviano un ping

KADEMLIA: IL PROTOCOLLO

- Per assicurare la persistenza dei dati inseriti nell'overlay, ogni nodo ripubblica periodicamente i dati pubblicati in precedenza
- Il meccanismo di pubblicazione periodica dei dati è introdotto da Kademlia per
  - evitare la perdita di dati in conseguenza dell'uscita volontaria o dovuta al guasto di un nodo dall'overlay
  - considerare l'entrata di nuovi nodi nell'overlay Kademlia. Un nuovo nodo potrebbe avere un identificatore più vicino alla chiave
- Ogni coppia (chiave-valore) viene ripubblicata una volta per ogni ora
- Definite alcune ottimizzazione per diminuire il numero di messaggi scambiati
  - Se un nodo riceve una STORE può supponerre che la STORE sia stata inviata ad altri k-1 nodi vicini e non ripubblica la chiave nell'ora successiva

KADEMLIA: CARATTERISTICHE GENERALI

- La DHT è caratterizzata dai parametri
  - K = dimensione del bucket
  - α = numero di lookup mandati in parallelo

- I messaggi di lookup non vengono persi in seguito all'abbandono della rete da parte di un nodo

- E' possibile dimostrare che il routing di Kademlia è $O(\log(N))$ e che le tabelle di routing hanno dimensione $O(\log(N))$

CONCLUSIONI

- Confronto tra Chord e Kademlia:
  - In Chord la routing table è rigida e richiede costose operazioni nel caso di fallimento di un nodo ed in caso di uscita volontaria di un nodo
  - Kademlia offre una routing table flessibile:
    - Metrica simmetrica per la distanza
    - Esistono percorsi alternativi verso un nodo
    - Le ricerche sono parallelizzate
    - La manutenzione della routing table ha un costo inferiore

LA RETE KAD: CARATTERISTICHE GENERALI

- Overlay networks basate su Kademlia
  - Overnet, KAD: DHT implementate in Emule e basate su Kademlia
  - rete KAD supportata dal client eMule (a partire dalla versione 0.40)
  - recentemente anche BitTorrent implementa un protocollo basato su Kademlia
- Introdotta per limitare gli svantaggi della rete basata su servers
  - eliminare colli di bottiglia (30 servers gestiscono 90% della rete)
  - migliorare l'anonimato
- Caratteristiche generali del protocollo KAD
  - implementazione basata su Kademlia, ma contiene alcune modifiche rispetto alla proposta originaria
  - pubblicazione + ricerca delle fonti: basata su UDP (porta 4672)
  - download di file: basato su TCP (porta 4662)
  - controllo automatico dell'accessibilità delle porte (NAT +Firewalls)

LA RETE KAD: CARATTERISTICHE GENERALI

Rete completamente basata sull'utilizzo del protocollo UDP, motivazioni:

- Ogni peer contatta diversi altri peer, ed invia ad ogni peer una singola richiesta
- Alto livello di churn:
  - occorrerebbe aprire e chiudere diverse connessioni TCP
  - Una connessione TCP avrebbe breve durata

LA RETE KAD: CARATTERISTICHE GENERALE

- Spazio degli identificatori: 128 bits, metrica XOR per misurare la distanza tra gli identificatori
- Tabella di routing basata k-buckets, come in Kademlia
  - La struttura dell'albero binario corrispondente alla tabella di routing è modificata in modo da ottenere una struttura più regolare
- bootstrap basato sulla conoscenza di peer attivi sulla rete KAD. E' possibile
  - specificare l'indirizzo di un peer conosciuto
  - oppure usare una lista di client (nodes.dat) conosciuti memorizzata sul client nella cartella config di eMule
  - il file nodes.dat può essere scaricato da siti web conosciuti
- gestione di peers a monte di firewall o NAT
  - firewall check
  - buddy peers

PUBBLICAZIONE DI FONTI

- La pubblicazione di un intero file o di parti di esso sulla DHT provocherebbe un alto tasso di traffico sull'overlay
- Per questa ragione, quando un peer intende condividere un file F
  - memorizza F su un supporto locale
  - non utilizza la DHT per pubblicare F o parti di F
  - utilizza la DHT per memorizzare un insieme di riferimenti ad F
- La pubblicazione dei riferimenti ad F consente un rapido reperimento di F, senza l'introduzione di un insieme di servers centralizzati
- In questo modo ogni peer memorizza un indice per alcuni file condivisi in KAD
- Replicazione dei riferimenti: ogni riferimento R ad F viene memorizzato su un insieme di nodi della DHT. Questo insieme di nodi definiscono la tolerance zone di R

2-LEVEL PUBLISHING MODEL

- Ad ogni file ed ad ogni peer vengono assegnati identificatori logici calcolati nello spazio degli identificatori della DHT.
- Per ogni file vengono pubblicati due diversi tipi di riferimenti

1) Location information(Fonti) (SourceID, Location)

- SourceID: identificatore logico del file calcolato mediante la funzione hash nello spazio di 128 bits della DHT
- Location: lista di identificatori logici dei peer (nello spazio di identificatori della DHT) che hanno pubblicato il file e che quindi memorizzano il file

2) Metadati (Keyword, SourceID)

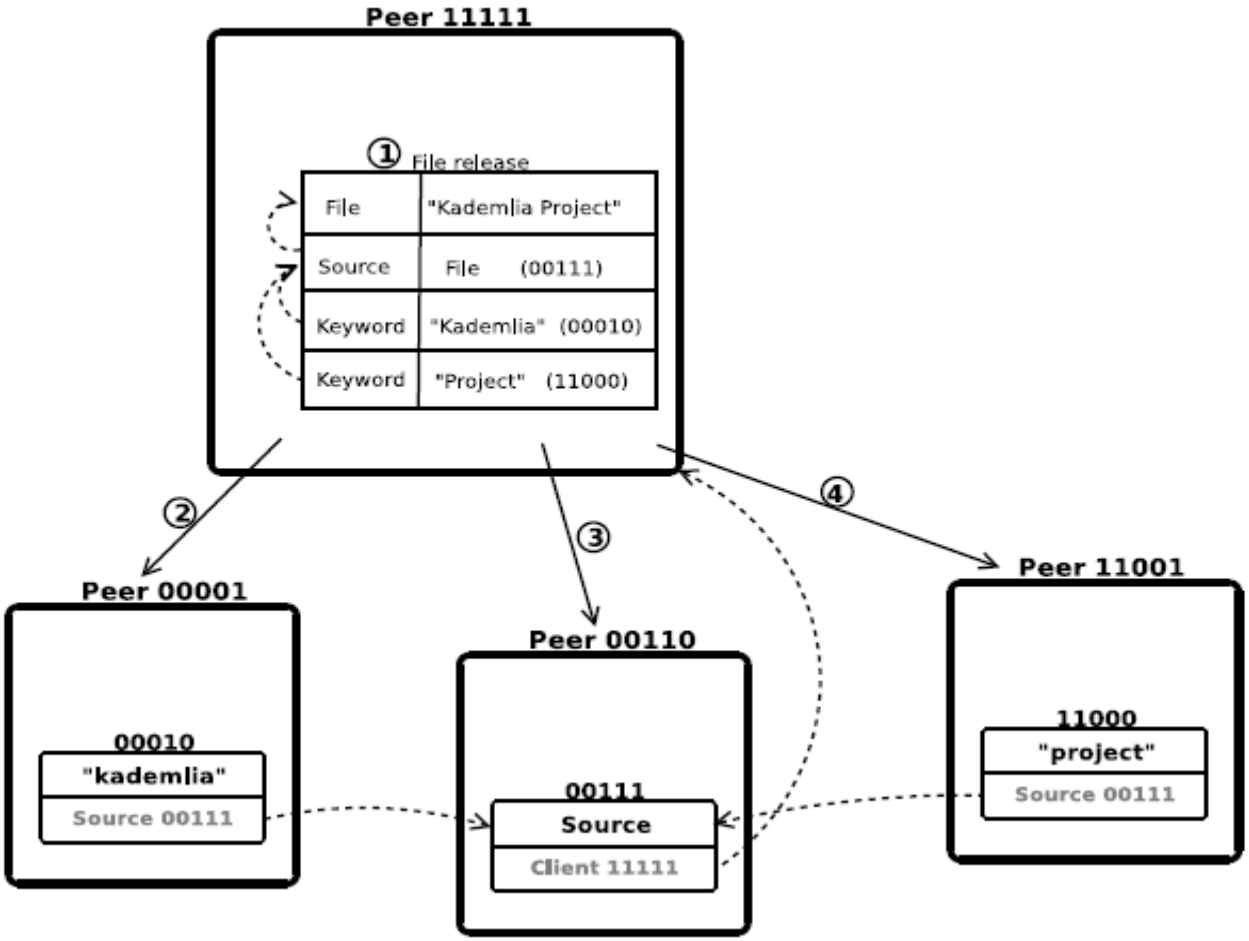
- Keyword: identificatore logico calcolato a partire da una delle parole chiavi che identificano il file (Es: Ligabue, Buonanotte, Italia,.....)
- SourceID: identificatore logico del file

Le informazioni 1) sono dette di primo livello, le 2) di secondo livello

2-LEVEL PUBLISHING MODEL

- Un riferimento di primo livello (Location Information) 'punta' direttamente al peer che ha pubblicato e memorizza il file
- Un riferimento di secondo livello (Metadati) punta ad un riferimento di primo livello
- I puntatore sono logici, cioè sono identificatori logici all'interno della DHT

2-LEVEL PUBLISHING MODEL

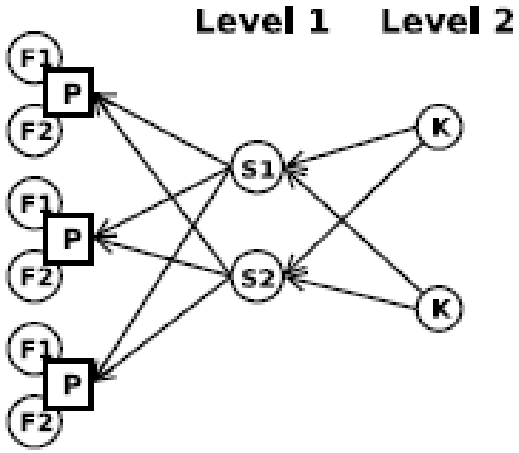


2-LEVEL PUBLISHING MODEL

Nello schema presentato nel lucido precedente occorre considerare che

- ad una stessa parola chiave possono corrispondere più files
  - in generale è quindi necessario associare una lista di identificatori logici di files all'hash che identifica la parola chiave
- lo stesso file può essere messo in condivisione da più utenti
  - in generale è quindi necessario associare una lista di locazioni logiche ad ogni file. La lista contiene gli identificatori logici di tutti i peer che hanno messo in condivisione il file

2-LEVEL PUBLISHING: VANTAGGI



Consideriamo 2 files F1, F2

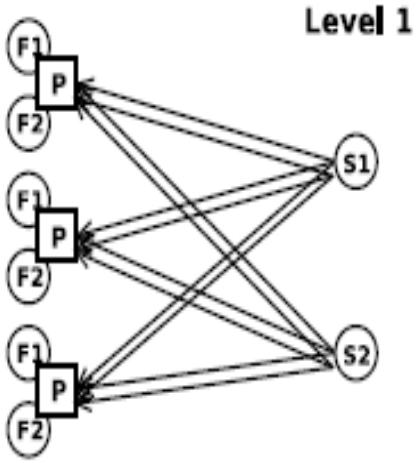
- entrambi i files sono stati pubblicati da 3 diversi peer
- ognuno dei due files contiene due parole chiave

Esempio:

- esistono due file diversi, F1,F2.F1 contiene la prima lezione di P2P, F2 contiene la seconda lezione di P2P
- ogni file è individuato dalle parole chiave dispensa, P2P
- ognuno dei 3 peer pubblica i due files, identificati dalle stesse parole chiave

- 2 Level publishing:
  richiede 10 riferimenti

2-LEVEL PUBLISHING: VANTAGGI



1 Level publishing

- Vengono accorpati il livello delle parole chiave con il livello degli identificatori dei file
- In un unico livello corrispondenza (parola chiave, identificatore logico del file)
- Es: S1 contiene l'identificatore logico del file F1, uno dei 2 file identificati dalle parole chiave Dispensa, P2P
- Per ogni peer S1 mantiene due riferimenti, uno per ogni parola chiave che identifica il file F1
- richiede 12 riferimenti

Se il tasso di replicazione dei files è alto il primo schema può ridurre notevolmente il numero dei riferimenti nella rete

PUBBLICAZIONE DI METADATI E FONTI

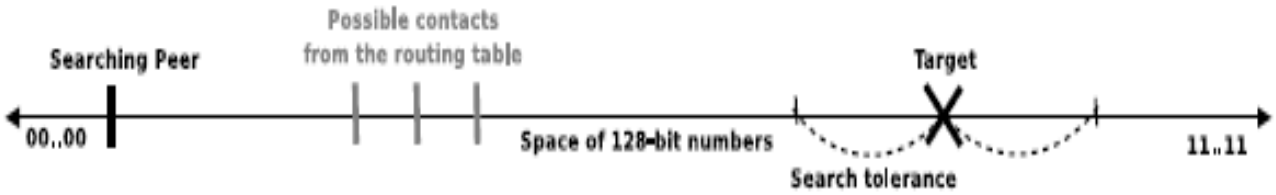
Pubblicazione di fonti:

- il peer pubblica i riferimenti di primo e di secondo livello utilizzando come chiave, rispettivamente
  - l'identificatore del file
  - l'identificatore della chiave

- content replication:
  - La replicazione è utilizzata per far fronte all'alto tasso di churn (alla lettera agitazione) della rete
  - la memorizzazione di un riferimento su un unico peer non è indicata in un ambiente con un alto livello di churn, cioè altamente dinamico, in cui i peer possono entrare ed uscire dalla rete molto frequentemente
  - ogni riferimento viene replicato su un insieme di peers appartenenti alla tolerance zone del riferimento

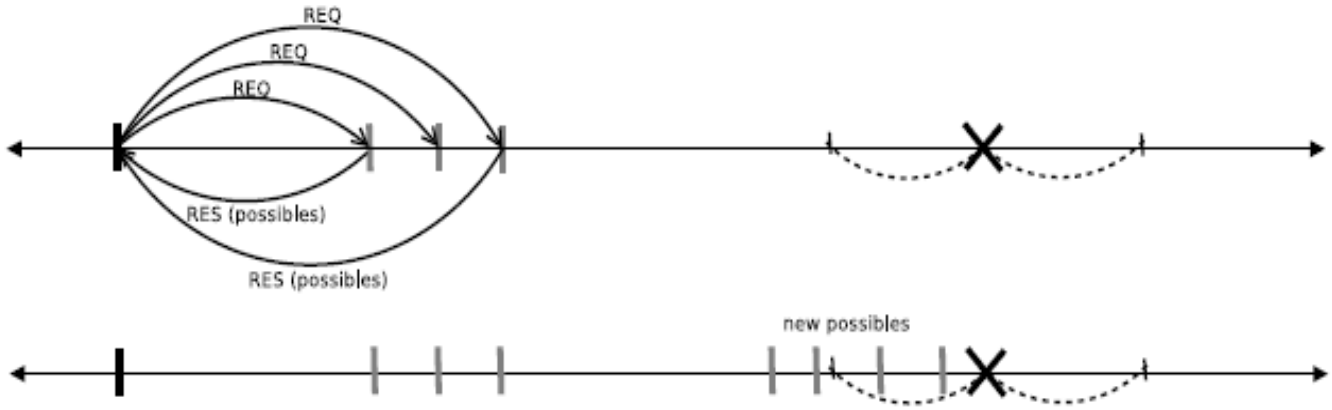
- Routing iterativo e parallelo per la ricerca dei peer appartenenti alla tolerance zone del riferimento

KAD: RICERCA ITERATIVA CON TOLERANCE ZONE



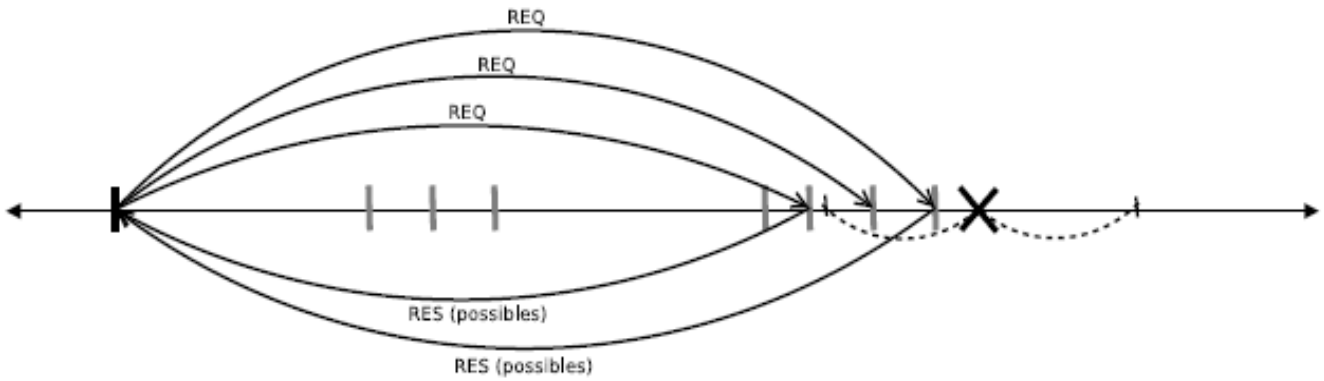
- Il peer che intende ricercare una chiave o memorizzare una nuova chiave ricerca nella propria routing table gli identificatori dei peer più vicini alla chiave
- Tutti i contatti in possesso del peer non sono ancora 'abbastanza vicini' alla chiave, cioè non appartengono alla zona di tolleranza
- La tolerance zone è centrata sul peer target della ricerca

KAD:RICERCA ITERATIVA CON TOLERANCE ZONE



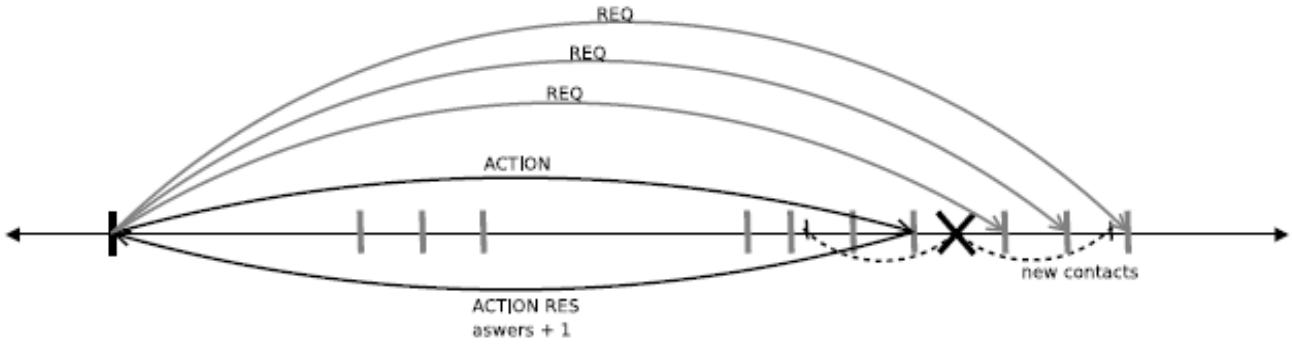
- Il peer invia la chiave ai contatti identificati mediante la sua routing table
- i contatti che sono ancora presenti sulla rete, inviano al richiedente una lista di contatti più vicini alla chiave
- 4 nuovi contatti vengono comunicati al peer che effettua la ricerca durante la prima iterazione. Due di questi si trovano nella tolerance zone della chiave

KAD: RICERCA ITERATIVA CON TOLERANCE ZONE



- tra i nuovi contatti di cui è venuto a conoscenza, il peer sceglie quelli più vicini al target e li contatta
- solo quelli presenti sulla rete rispondono comunicando nuovi contatti

KAD:RICERCA ITERATIVA CON TOLERANCE ZONE



- quando il peer individua un contatto attivo ed appartenete alla tolerance zone, invia al contatto il messaggio di action, che può corrispondere ad una ricerca oppure alla ad una richiesta di pubblicazione
- il contatto che ha eseguito l'azione invia un messaggio di ack (ACTION RES) il peer incrementa un contatore e se il contatore ha raggiunto il massimo interrompe la ricerca

PUBBLICAZIONE DI METADATI:OVERLOAD PROTECTION

- Oveload Protection per la pubblicazione delle parole chiavi Le parole chiave più popolari (ad esempio mp3) possono provocare hot spots nella rete
- Un peer che riceve una parola chiave molto popolare riceverà molti riferiemnti a files che contengono quella parola chiave
- Ogni peer KAD può memorizzare al massimo 60000 metadati
- Se un peer diventa un hot spot per una parola chiave molto popolare, si rischia di utilizzare tutto lo spazio di memorizzazione a disposizione del peer(i 60000 riferimenti) per quell'unica parola chiave
- In questo caso il peer è costretto a rifiutare la memorizzazione di riferimenti a parole chiave rare o poco conosciute
- Overload Protection: un peer P
  - non accetta ulteriori fonti se ne memorizza già 60000
  - non accetta più di 50000 fonti per ogni parola chiave
  - se una parola chiave ha già 45000 fonti, accetta una nuova fonte solo se completamente sconosciuta

PUBBLICAZIONE DI FONTI: OVERLOAD PROTECTION

Overload protection per la pubblicazione delle fonti

- Deve garantire di fornire più fonti possibili per un file, dato che le fonti disponibili rappresentano il vero bottleneck di un sistema di file sharing
- Massimo 300 fonti per ogni file
- Quando si reggiunge la soglia delle 300 fonti, si rimpiazza il riferimento inserito meno di recente

OVERLOAD PROTECTION

- Ogni riferimento pubblicato sulla rete KAD viene ripubblicato periodicamente
- Supponiamo che il peer P pubblichi un metadato D su un insieme di peer appartenenti alla tolerance zone
- Ogni peer a cui viene inviata la richiesta di pubblicazione risponde con un valore load proporzionale al numero di riferimenti associati al metadato
- P calcola la media M di tutti i valori di load restituiti dai peer che hanno pubblicato il metadato
- Il numero di valori ricevuti dipende dal grado di replicazione utilizzato
- P calcola l'intervallo che deve trascorrere prima di una ripubblicazione della stessa chiave in funzione del valore di M
  - se M <20 l'intervallo è di 24 ore
  - altrimenti intervallo = 7 giorni * (average load)/ 100
  - ad esempio: load medio = 30 (numero medio di riferimenti=15000), intervallo= 50 ore

RICERCA DI FONTI

Ricerca delle fonti: l'utente specifica almeno una parola chiave K

Primo passo: ricerca dei metadati

- si calcola l'hash H di K
- la chiave K viene ricercata nella tolerance zone di H, cioè nei peer con identificatore vicino ad H
- il client che ha iniziato la ricerca riceve un elenco di riferimenti a files che contengono quella parola chiave
- i file vengono aggiunti ad una lista di download
- strategia di routing basato su Kademlia
  - iterativo
  - parallelo

Secondo passo: ricerca delle fonti

- analoga alla ricerca dei metadati (routing iterativo + parallelo con zona di tolleranza)

DOWNLAOD DI FILES

Analogo a Emule e2k

- Basato sulla definizione di code di attesa nei clients
- Gestione dei crediti
- Protocollo TCP

LA RETE KAD: FIREWALL CHECK

- Nella rete ed2K, il client Nattato o a monte di un firewall viene riconosciuto dal server che gli assegna un identificatore basso
- Occorre definire un meccanismo analogo nella rete senza server

Se un peer P intende connettersi alla rete KAD

- individua un punto di contatto C sulla rete KAD
- viene effettuato automaticamente il firewall check
  - P invia a C un messaggio che include la porta TCP su cui P attende le connessioni
  - C individua l'indirizzo IP contenuto nel pacchetto ricevuto da P e lo rispedisce a P
  - P verifica se il suo indirizzo IP è uguale a quello ricevuto da C.
  - se i due indirizzi sono diversi P si trova a monte di un NAT e lo stato del peer viene impostato a firewalled
  - stato firewalled equivale al low ID della rete basate su servers

LA RETE KAD: BUDDY PEER

- Ogni peer P con stato firewalled deve ricercare un peer buddy (Buddy = compagno, amico caro)
- Il buddy riceve richieste per il P firewalled e gliele reinoltra
- Buddy peers = offrono supporto a peer in stato firewalled
- Se lo stato del peer risulta firewalled, KAD ricerca automaticamente un altro client con KAD non firewalled da utilizzare come compagno (buddy) per la ricerca delle fonti
- Ogni client con KAD non firewalled può gestire al massimo un peer con stato firewalled