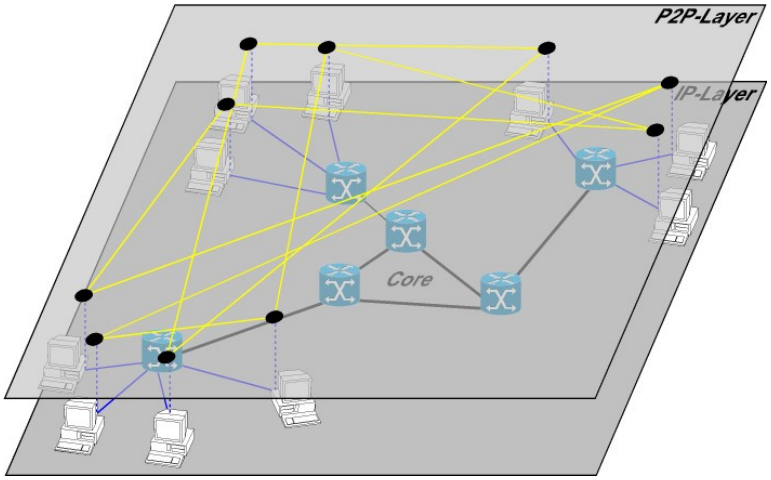
Lezione n.4
GNUTELLA 0.6, KaZAA, DISTRIBUTED HASH TABLES: Caratteristiche generali

Laura Ricci

Materiale didattico:
Peer-to-Peer Systems and Applications
Capitolo 5

RIASSUNTO DELLA PRESENTAZIONE

1. Caratteristiche Generali dei Sistemi P2P
2. Reti P2P Centralizzate
   1. Caratteristiche Base
   2. Protocolli
   3. Discussione
3. Reti P2P pure
   1. Caratteristiche Base
   2. Protocolli
   3. Discussione
4. Reti P2P Ibride
   1. Caratteristiche Base
   2. Protocolli
   3. Discussione

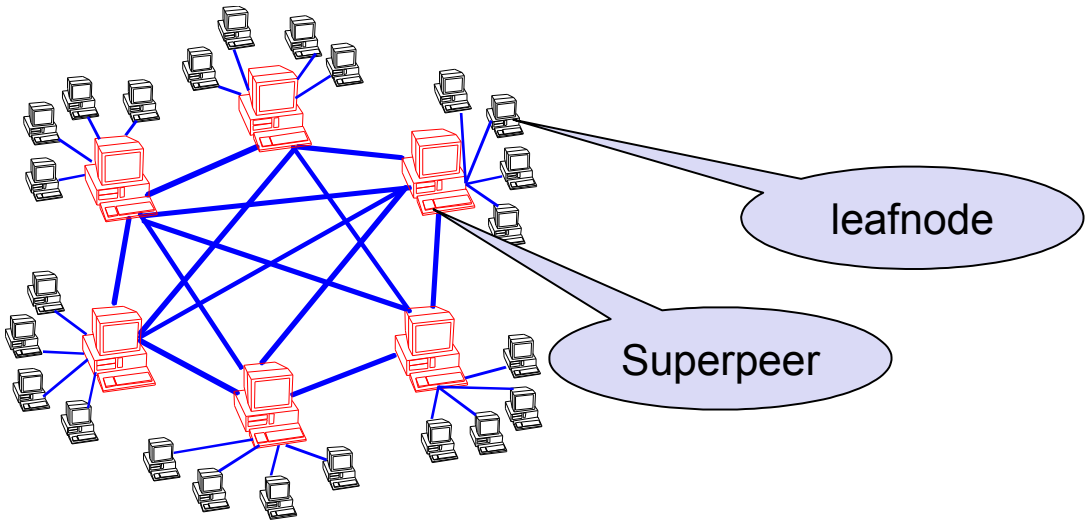


GNUTELLA 0.6: CARATTERISTICHE GENERALI

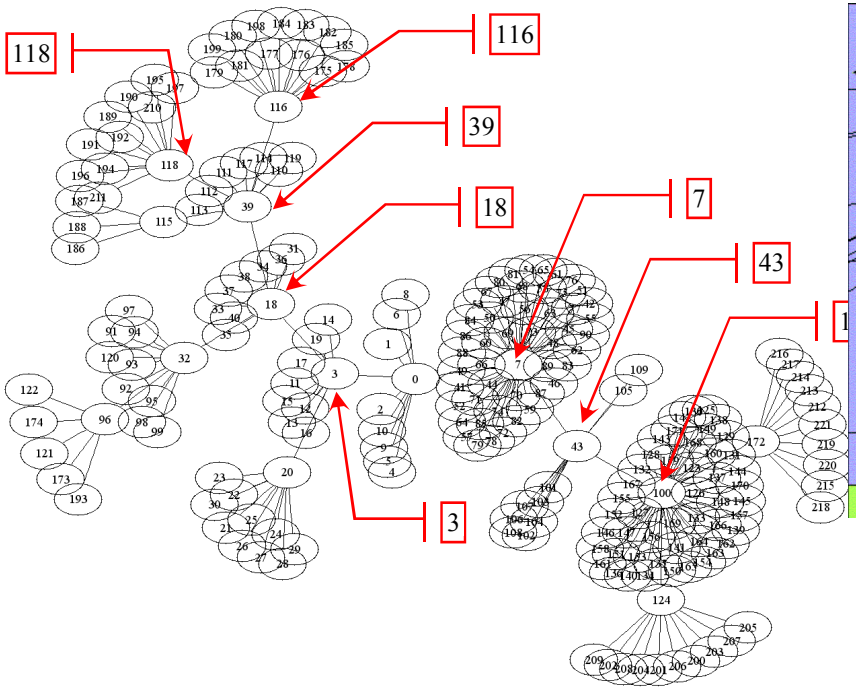
- Caratteristica principale, rispetto al P2P puro : definizione dinamica di un livello gerarchico nella rete
- Riduce il traffico relativo ai 'messaggi di controllo' senza ridurre l'affidabilità
- Superpeers (o UltraPeers): mantengono aperte molte connessioni con altri
- Leafnodes: mantengono aperte solo un numero limitato di connessioni.Tutte le connessioni sono verso SuperPeers (di solito uno)

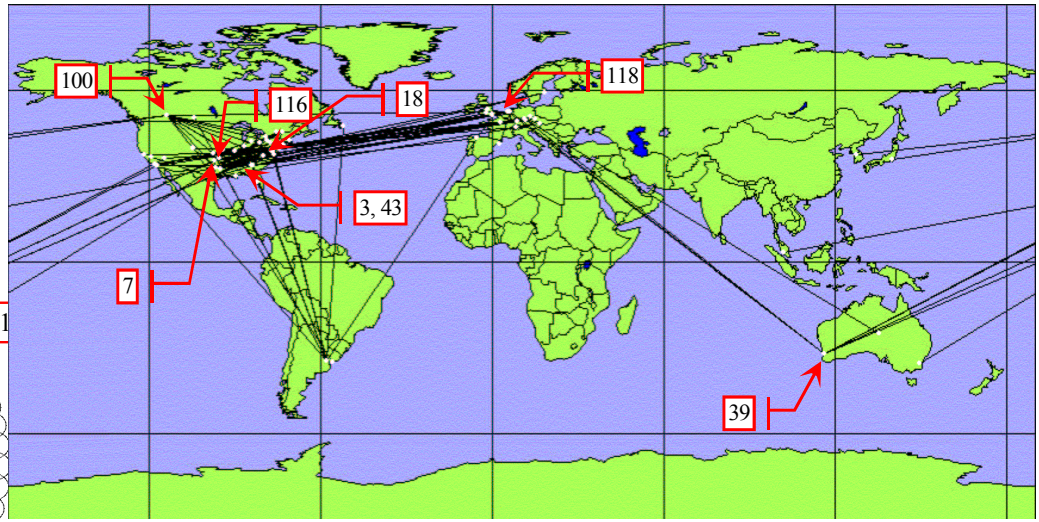
SuperPeer = Proxy verso la rete per i leafnodes gestiti

Hub based network



TOPOLOGIA DI UNA RETE P2P IBRIDA





Rete Gnutella (222 nodes).
Sulla destra è mostrata la collocazione fisica dei rilevata il giorno 01.08.2002

I numeri corrispondono ai numeri dei nodi mostrati nella rete astratta sulla sinistra

- La rete virtuale non corrisponde a quella fisica. Osservare il cammino dal nodo 118 al nodo 18.
- La struttura dei Superpeer (hub strucure) è chiaramente rilevabile dalla rete astratta

GNUTELLA 0.6

- Obiettivi:
  - Metodo decentralizzato di ricerca di files
  - Diminuzione dei messaggi scambiati nelle reti P2P pure
  - Affidabilità paragonabile a quella di Gnutella 0.4 (non esiste un unico punto di centralizzazione)

- Alla base di diverse altre implementazioni di sistemi P2P (Kazaa,....)

- Breve Storia:
  - Primavera 2001: sviluppata a partire da Gnutella 0.4

  - Da allora:
    - Disponibili diverse implemetazioni (Limewire, Bearshare,...)
    - Definite diverse ottimizzazioni (privacy, scalability, performance,...)

GNUTELLA 0.6: TIPI DI NODI

L'architettura prevede due tipi di nodi

- Superpeer (UltraPeer):
  - 10-100 connessioni con i nodi foglia, <10 connessioni con altri ultrapeer
  - Proxy per i rispettivi nodi foglia
  - deve possedere certe caratteristiche (es: connessione veloce ad Internet nonbloccate da firewall)
- Nodi Foglia
  - Non accettano connessioni Gnutella
  - Aprono solo alcune connessioni verso nodi SuperPeer (spesso solo una)
  - una foglia può chiedere dinamicamente di diventare SuperPeer
- L'overlay Gnutella 0.6 assume caratteristiche simili ad Internet: nodi caratterizzati da bassa banda sono connessi ai routers che trasmettono i dati su collegamenti a larga banda (backbones)

GNUTELLA 0.6: ELEZIONE DI SUPERPEERS

- SuperPeers: eliminano il carico della gestione del routing delle queries dai leaf nodes
- Routing delle queries: avviene solo tra SuperPeers
- Messaggi di ping/pong: scambiati solamente tra i Super Peers
- L'elezione dei SuperPeer avviene in modo completamente decentralizzato
- Elezione di SuperPeer: esempio di self organization
- Ogni host determina in modo autonomo, prima di connettersi alla rete Gnutella, se ha le caratteristiche necessarie per diventare un superpeer. In questo caso l'host si qualifica come SuperPeer Capable altrimenti come LeafNode
- Esempio: Limewire, richiesti 10kB/s per UpLoad, 20kB/s per DownLoad per poter essere eletto SuperPeer

SUPERPEERS: ELEZIONE DINAMICA

Criteri per la determinazione dei nodi SuperPeer Capable

- assenza di firewall. Di solito la verifica di questa proprietà viene approssimata verificando se il nodo ha ricevuto connessioni in ingresso
- banda di ingresso/uscita. Il calcolo della banda si approssima calcolando il throughput di upload/download
- quantità di RAM disponibile. Un SuperPeer deve memorizzare un insieme di routing tables, per indicizzare i files condivisi dai leaf nodes gestiti.
- Sistema Operativo. Alcuni sistemi operativi gestiscono un alto numero di sockets im modo più efficiente rispetto ad altri
- potenza di calcolo (ciclo di clock della CPU). Un SuperPeer deve essere in frado di gestire un alto numero di connessioni.
- Uptime tempo medio in cui un SuperPeer rimane attivo sulla rete Gnutella. Euristica: l'uptime futuro è proporzionale a quello passato

SUPERPEERS: APERTURA DI CONNESSIONI

- Foglia F → SuperPeer S
  - F diventa una foglia di S
  - F rifiuta connessioni di tipo 0.4
  - F invia ad S una QRP routing table
- Foglia F → Foglia G protetta da S: F diventa foglia di S
- Foglia F → Foglia G: Gnutella 0.4
- SuperPeer S1 → Superpeer S2
  - Se entrambi i superpeer gestiscono un numero sufficiente di foglie, entrambi rimangono superpeers e stabiliscono una connessione tra di loro
  - Altrimenti uno dei due può diventare un nodo foglia gestito dall'altro

CONNESSIONE FOGLIA-SUPERPEER

Nodo A

GNUTELLA CONNECT/0.6

User-Agent: LimeWire 1.9

X-Ultrapeer: False

X-Query-Routing: 0.1

X-My-Address: 10.254.0.16:6349

SuperPeer B

GNUTELLA/0.6 200 OK

User-Agent: LimeWire 1.9

X-Ultrapeer: True

X-Ultrapeer-Needed: false

X-Try-Ultrapeers: 23.35.1.146:6346,
X-Try: 24.37.144:6346,193.205.63.22:6346

X-Query-Routing: 0.1

X-My-Address: 10.254.0.16:6346

GNUTELLA/0.6 200 OK

GNUTELLA/0.6 200 OK

CONNESSIONE CON FOGLIA GESTITA DA SUPERPEER

Nodo A

GNUTELLA CONNECT/0.6

X-Ultrapeer: False

Nodo B

GNUTELLA/0.6 503 I am a shielded leaf node

X-Ultrapeer: False

X-Try-Ultrapeers: 18.2.3.14:6346,

18.1.17.2:6346

- Il nodo B rifiuta la connessione da A e '"ridirige" A verso il proprio ultrapeer S
- A non invia ok a B
- A tenta di stabilire una connessione con S e si comporta come nel caso visto nel lucido precedente

CONNESSIONE CON NODO GNUTELLA 0.4

- Se un nodo non è riuscito a trovare un SuperPeer, passa ad eseguire il vecchio protocollo Gnutella 0.4

Nodo A

GNUTELLA CONNECT/0.6

X-Ultrapeer: False

Nodo B

GNUTELLA/0.6 200

X-Ultrapeer: False

GNUTELLA/0.6 200 OK

APERTURA DELLE CONNESSIONI

- L'host SuperPeer Capable A si connette al SuperPeer B.

  Se B gestisce pochi nodi, può indicare ad A che non c'è necessità di diventare un SuperPeer. A diventa un LeafNode di B.

Nodo A	SuperPeer B
GNUTELLA CONNECT/0.6	
X-Ultrapeer: True	
	GNUTELLA/0.6 200 OK
	X-Ultrapeer: true
	X-UltraPeer-Needed:False
GNUTELLA/0.6 200 OK	
X-UltraPeer: False	

- Un leafNode F può richiedere successivamente (ma solo quando è rimasto collegato alla rete per un intervallo di tempo sufficientemente lungo) al proprio superpeer S se deve diventare un superPeer

APERTURA DELLE CONNESSIONI

- Un SuperPeer S può decidere di rifiutare la connessione richiesta da un leaf node

- In questo caso S fornisce, al momento dell'handshaking, indirizzi di altri SuperPeers, inviando connection pongs che sono stati ricevuti, in precedenza, dalla overlay network dei super-peers

  Esempio: X-try-SuperPeers:68.37.233.44:9376, ...

- Alcuni connections pongs vengono restituiti anche nel caso in cui la connessione venga accettata. In questo caso le informazioni ricevute possono essere utilizzate in seguito per stabilire connessioni con SuperPeers alternativi

QUERY ROUTING PROTOCOL

- I Leaf Nodes inviano ai SuperPeer informazioni sulle risorse condivise, in modo che il SuperPeer possa decidere verso quali Leaf Nodes inoltrare la query
- Il QRP protocol permette al nodo Leaf di inviare al SuperPeer una descrizione delle risorse gestite
- Le informazioni sono contenute in una QRP table
- Gnutella 0.6 utilizza un insieme di strategie per contenere la dimensione di questa tabella
  - Hashing
  - Compressione

GNUTELLA 0.6: QRP PROTOCOL

- Ogni risorsa condivisa è individuata da un insieme di parole chiave

- Bloom Filters: Ad ogni parola chiave viene applicata una funzione hash e viene inserito un "flag di presenza" nella corrispondente posizione della hash table

- Esempio:

  L'host A condivide il file individuato dalle parole chiave "Snow Hey Oh"

  h(Snow) = 2, h("Hey")=5, h("Oh")=7

  La routing table di A viene rappresentata mediante la bitmap 00100101....

  Ogni stringa viene rappresentata mediante un singolo bit in un vettore di booleani

- La hash table non memorizza le parole, solo un insieme di flags che indicano la presenza di alcune chiavi

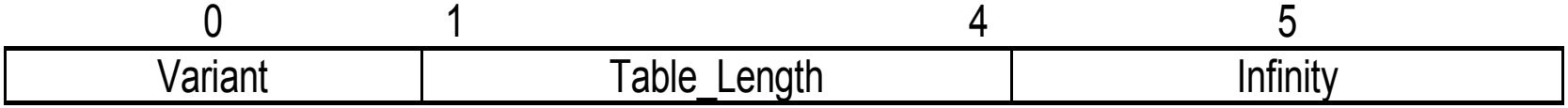
GNUTELLA 0.6: QRP PROTOCOL

- Il vettore booleano eventualmente compresso, viene suddiviso in blocchi e inviato al SuperPeer
- Il vettore viene memorizzato dal SuperPeer ed utilizzato come Query Routing Table per effettuare il routing delle queries
- Quando il SuperPeer riceve una query, applica la stessa funzione hash applicata dal leaf node ad ogni parola chiave della query
- Si controlla nella query routing table se la parola contenuta nella query corrisponde alla parola chiave che definisce la risorsa
- Diverse strategie di matching: es. tutte le parole contenute nella query devono corrispondere a parole chiave oppure solo alcune di esse,....

GNUTELLA 0.6: NOTIFICA DI FILES CONDIVISI

- Per implementare il QRP Protocol
- Aggiunto un nuovo tipo di di messaggio: ROUTE_TABLE_UPDATE, presente secondo due diverse varianti
  - ROUTE_TABLE_UPDATE (0x30), Reset variant (0x0): per eliminare una tabella di routing
  - ROUTE_TABLE_UPDATE (0x30), Patch variant(0x1): per inviare al SuperPeer una nuova tabella di routing

GNUTELLA 0.6: FORMATO DEI MESSAGGI



Reset Variant: utilizzata per eliminare la tabella di routing di un leaf node da un SuperPeer

Patch Variant: utilizzato per inviare una parte della tabella di routing di un Leaf Node

Compressor indica l'algoritmo utilizzato per la compressione

Seq_No, Seq_Size individuano il frammento di vettore inviato e la sua lunghezza

Data: Vettore di booleani

GNUTELLA 0.6: FORMATO DEI MESSAGGI

- Richiesta di files (Content Request)
  - QUERY (gli stessi di Gnutella 0.4)
  - QUERY_HIT (gli stessi di Gnutella 0.4)
- Keep alive:
  - PING (gli stessi di Gnutella 0.4)
  - PONG (gli stessi Gnutella 0.4)

GNUTELLA 0.6: ROUTING DELLE QUERIES

- Un LeafNode invia una richiesta al proprio Superpeer
- Il Superpeer controlla nella propria tabella di routing se il file richiesto è offerto da uno dei propri leafnodes.
- Il SuperPeer invia la query ad un leafnode L solo se L possiede una risorsa che soddisfa la query
- Il Superpeer inoltre invia la query ad altri superpeers (quelli a cui esso è connesso nella overlay network) per individuare altri hosts che condividono il file richiesto
- Ad ogni richiesta inoltrata tra i superpeer viene associato un opportuno valore di TTL, per limitare la diffusione della richiesta stessa sulla rete dei SupePeers.

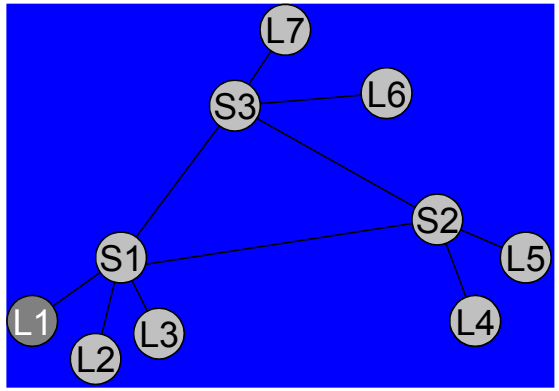
GNUTELLA 0.6: ROUTING DELLE QUERIES

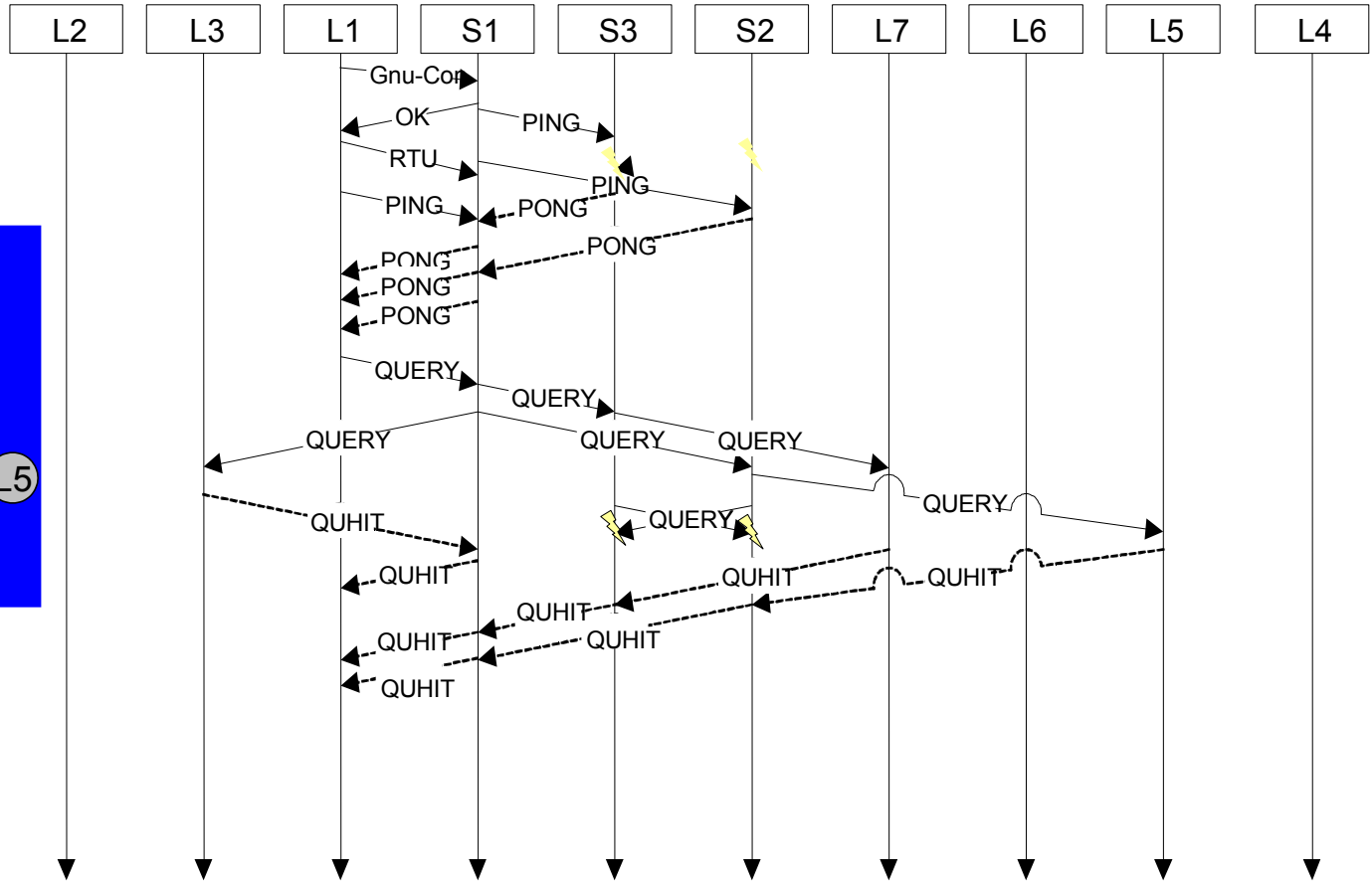
- Routing dei query hits:
  - Quando un LeafNode riceve una richiesta, controlla di possedere effettivamente il file richiesto
  - In caso positivo, il Leafnode invia un query hit al proprio SuperPeer
  - Il messaggio viene instradato all'indietro lungo lo stesso cammino che aveva percorso la query (backward routing)
- Scambio dei files:
  - Avviene direttamente tra i nodi interessati, tramite HTTP connessioni.

GNUTELLA 0.6: MESSAGGI DI KEEP-ALIVE

- I messaggi di ping pong vengono scambiati solamente tra i SuperPeers
- Un SuperPeer non propaga mai un messaggio di ping ai propri LeafNodes
- In questo modo ogni SuperPeer "mette al riparo" i propri LeafNodes dal traffico di keep-alive
- Un superPeer può ricevere un ping da un proprio LeafNode. In questo caso invia un pong con le informazioni ricevute da altri SuperPeers. Il LeafNode può utilizzare questa informazione nel caso in cui il proprio SuperPeer si disconnetta, oppure nel caso in cui voglia aprire connessioni con più SuperPeers.

GNUTELLA 0.6: MESSAGGI DI CONTROLLO





KAZAA: CARATTERISTICHE GENERALI

- Sistema P2P proprietario (2001), tutto il traffico è crittato (scarsa documentazione)
- Utilizza il protocollo FastTrack
- Tre milioni di utenti, si stima che il 76% del traffico P2P in USA nel 2003 sia dovuto a Kazaa/FastTrack e solo l'8% a Gnutella
- Molte similitudini con Gnutella 0.6
  - Definisce due tipi di nodi, Ordinary Nodes(ON), Super Nodes (SN (connessione TCP tra ON e SN)
  - ON comunica al proprio SN i metadati che descrivono le informazioni che condivide (nome file, dimensione file, content hash del file, decrizione del file mediante parole chiave fornite dall'utente)
  - Ogni Super Node tiene traccia solo delle informazioni relative ai propri ON

KAZAA: INGRESSO NELLA RETE

- Ogni nodo mantiene una cache di SN (SN cache list)
- Esistono default Server che possono essere interrogati se la cache è vuota
- SN Probing: All'inizio un ON prova ad inviare un pacchetto UDP a tutti gli SN nella sua cache list
- Al passo successivo, apre in parallelo più connessioni TCP con alcuni SN selezionati
- Alla fine sceglie un SN e si disconnette dagli altri
- Al momento della connessione ogni SN invia al proprio ON una lista contenente altri 200 SN. Queste informazioni sono utilizzate per popolare la SN cache list dell'ON
- Gli SN nella cache list di un ON O, sono quasi tutti vicini di O(RTT<50 ms).
- Analisi del protocollo basata su RRT e IP prefixes

KAZAA: UPLOAD DEI METADATI

- Ogni ON invia al proprio SN una lista dei file che intende condividere, che include, per ogni file: nome del file, dimensione del file, file descriptor (parole chiave introdotte dall'utente) e content hash (hash signature).
- File Descriptor organizzato in blocchi di metadati
- Il file descriptor è utilizzato per la risoluzione delle query

KAZAA: LOOK UP DEI DATI

- ON invia la query al proprio SN (Esempio: ricerca di qualsiasi file che contenga la keyworld "Sting"). La query contiene una lista di parole chiave.
- SN inoltra la query ad un piccolo insieme di SN (algoritmo non noto).
- Le risposte ricevute da SN contengono metadati che descrivono la risorsa individuata ed identificazione del peer che offre la risorsa
- Il download è diretto e utilizza il content hash per individuare il file (GET conthash)
- Per aumentare le prestazioni il download scarica il file a frammenti dall'insieme di peer che lo condividono
- Se il download fallisce si ricerca nuovamente il file utilizzando il content hash

KAZAA: CARATTERITICHE GENERALI

- Overlay altamente dinamico: La struttura della overlay network che collega i SuperNodes varia frequentemente (ogni 10 minuti i collegamenti sono modificati)

- Questo "rimescolamento" consente di esplorare nuovi segmenti di rete e di trovare nuove risposte alle queries

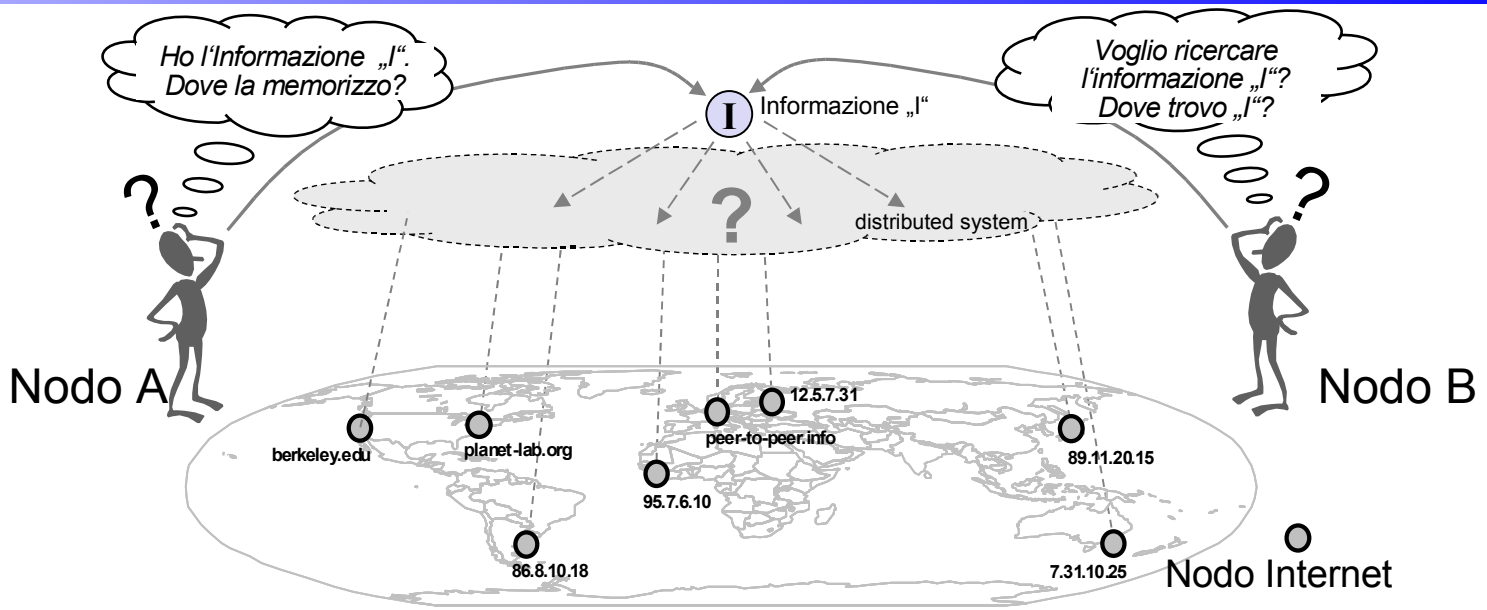
- Ogni ON si disconnette frequentemente dal proprio SN e si connette ad altri SN. Ogni query viene inviate al nuovo SN per la ricerca di nuovi dati

- Un ON rinvia i propri metadati al nuovo SN ed il vecchio SN cancella i dati relativi all'ON disconnesso

DISTRIBUTED HASH TABLES:INTRODUZIONE

1. Distributed Hash Tables (DHT): Introduzione
   1. Motivazioni
   2. Caratteristiche
   3. Confronti
2. DHT: Aspetti Fondamentali
   1. Gestione distribuita dei dati
   2. Indirizzamento nelle Distributed Hash Tables
   3. Routing
   4. Memorizzazione dei dati
3. DHT: I meccanismi
   1. Inserzione di nuovi nodi
   2. Fallimento/Uscita volontaria di nodi dal sistema
4. DHT: Le interfacce
5. Conclusioni

SISTEMI P2P: LOOK UP



- A vuole memorizzare una informazione I all'interno del sistema distribuito, B vuole reperire I senza conoscere a priori l'effettiva locazione di I
- Come organizzare il sistema distribuito? In particolar modo, quali sono i meccanismi utilizzati per decidere dove memorizzare l'informazione e come reperirla?
- Qualsiasi soluzione deve tenere particolarmente in considerazione
  - Scalabilità del Sistema. Occorre controllare l'overhead di comunicazione e la memoria utilizzata da ogni nodo, in funzione del numero dei nodi del sistema
  - Robustezza ed adattabilità in caso di faults e frequenti cambiamenti

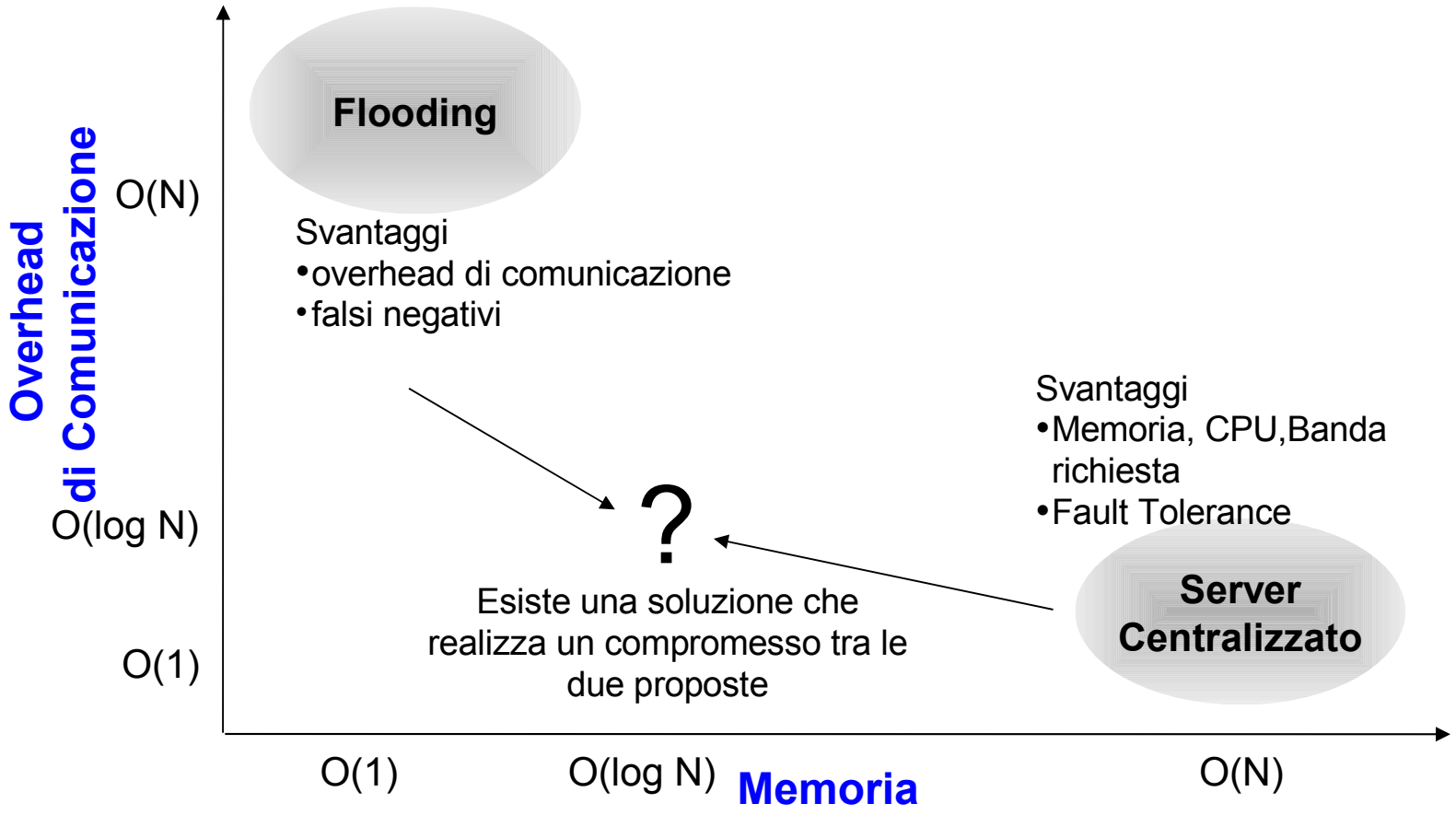
P2P:ANALISI e CONFRONTI

- Approccio Centralizzato
  - *Ricerca:* O(1) – "basta chiedere al server"
  - Quantità di Memoria Richiesta sul Server: O(N) (N = numero di informazioni disponibili nel sistema)
  - Banda richiesta (connessione server/rete): O(N)
  - Possibilità di sottomettere al sistema queries complesse

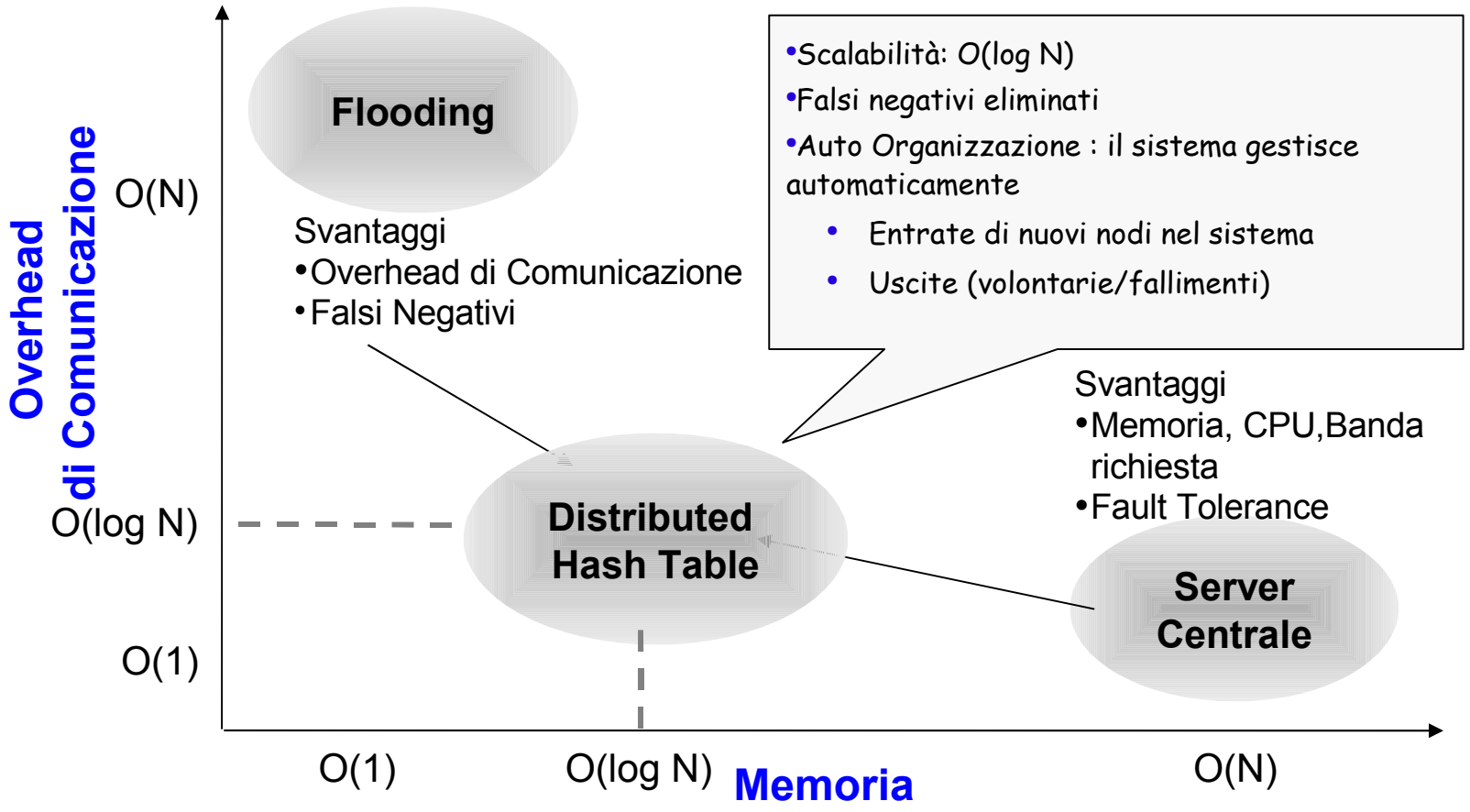
- Approccio Completamente Distribuito
  - Ricerca: caso pessimo $O(N^2)$ - "ogni nodo chiede a tutti i vicini ". Possibili ottimizzazioni (TTL, identificatori per evitare cammini ciclici)
  - Quantità di memoria richiesta : O(1)
    - Informazione condivisa: non dipende dal numero di nodi del sistema
    - Non si utilizzano strutture dati per ottimizzare il routing della query (flooding)

DISTRIBUTED HASH TABLES:MOTIVAZIONI

Analisi dei sistemi Esistenti

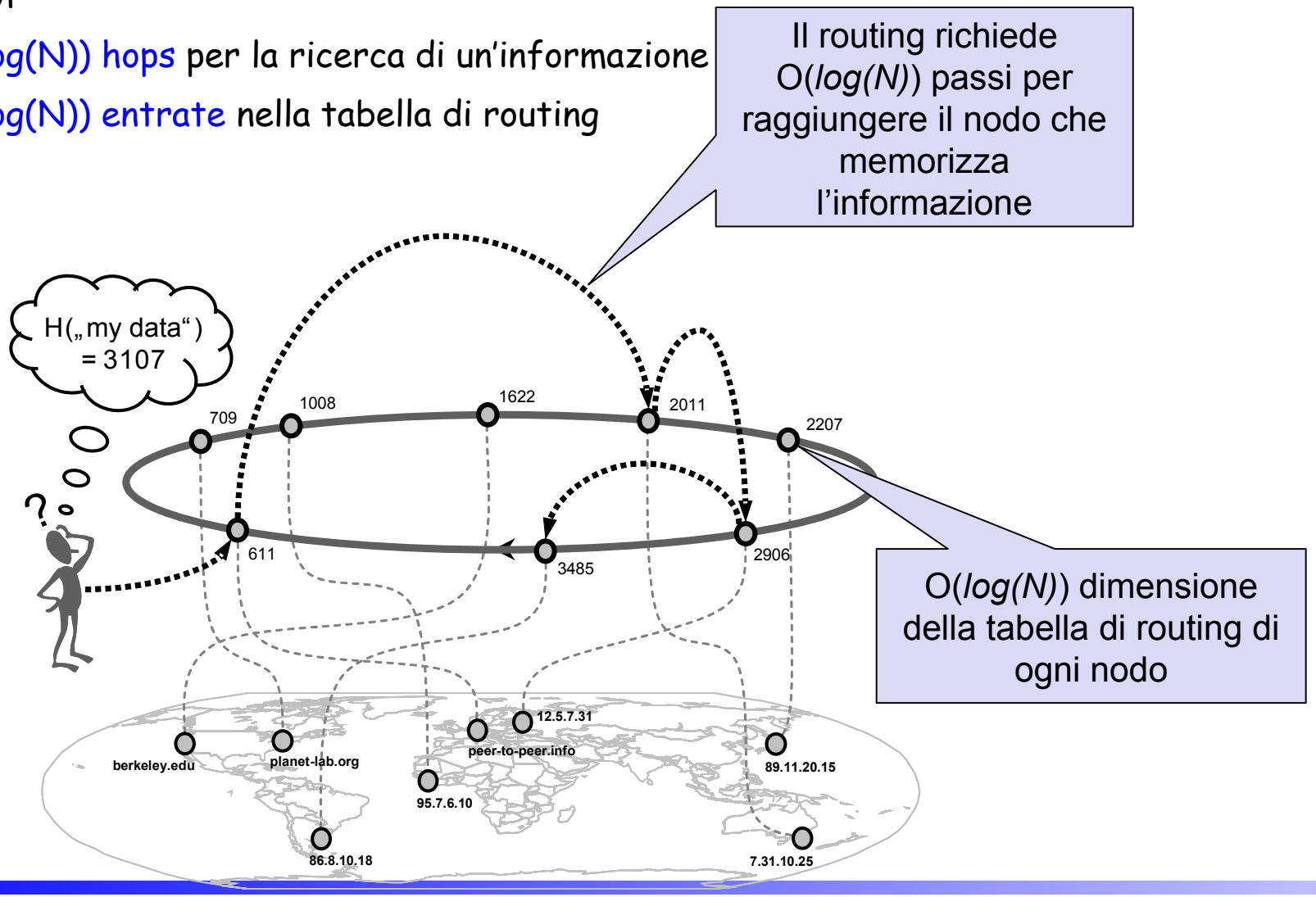


DISTRIBUTED HASH TABLES: MOTIVAZIONI



DISTRIBUTED HASH TABLES: CONCETTI GENERALI

- Obiettivi
  - O(log(N)) hops per la ricerca di un'informazione
  - O(log(N)) entrate nella tabella di routing



DISTRIBUTED HASH TABLES: OBIETTIVI

- DHT: Obiettivi
  - Scalabilità
  - Flessibilità
  - Affidabilità

- Adattabilità a fallimenti, inserimento ed eliminazione di nodi
  - Assegnamento di informazioni ai nuovi nodi
  - Re-assegnamento e re-distribuzione delle informazioni in caso di fallimento o disconnessione volontaria dei nodi dalla rete

- Bilanciamento delle informazioni tra i nodi
  - Fondamentale per l'efficienza della ricerca

CONFRONTI TRA I DIVERSI APPROCCI

Approccio	Memoria per Nodo	Overhead di Comunicazione	Queries Complesse	Falsi Negativi	Robustezzza
Server Centrale	O(N)	O(1)	✓	✓	✗
P2P puro (flooding)	O(1)	$O(N^2)$	✓	✗	✓
DHT	O(log N)	O(log N)	✗	✓	✓

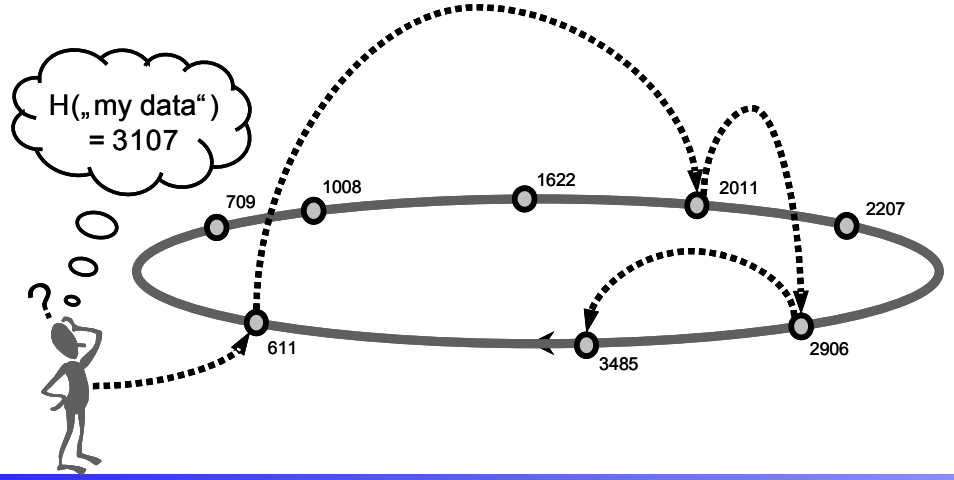
DHT: GESTIONE DISTRIBUITA DEI DATI

Operazioni fondamentali

- Mapping dei nodi e dei dati nello stesso spazio di indirizzamento
  - Ai peers ed ai dati sono associati degli identificatori unici (ID), che li individuano
  - univocamante all'interno del sistema
  - Esiste uno spazio comune degli indirizzi per i dati e per i peer.
  - I nodi sono responsabili della gestione di una porzione dello spazio degli indirizzi
  - corrispondente ad un sottoinsieme dei dati
  - La corrispondenza tra i dati ed i nodi può variare per l'inserimento/cancellazione
  - di nodi
- Memorizzazione/ Ricerca dei dati
  - Ricerca di un dato = routing verso il nodo responsabile
  - Ogni nodo mantiene una tabella di routing, che fornisce al nodo una visibilità
  - parziale del sistema
  - Il routing è guidato dalla conoscenza dell'ID del dato ricercato
  - Falsi negativi eliminati

DHT: INDIRIZZAMENTO

- Tecniche di indicizzazione distribuita
  - Spazio degli indirizzi in cui vengono mappati sia i dati che i nodi
  - I nodi intermedi mantengono delle tabelle di routing
    - Instradamento efficiente verso il nodo "destinazione"
    - Content routing (guidato dagli ID dei dati),
- Problemi
  - Gestione dinamica delle tabelle di Routing (inserzioni, eliminazioni)
  - Queries complesse non supportate (e.g, ricerche contenenti wildcard)

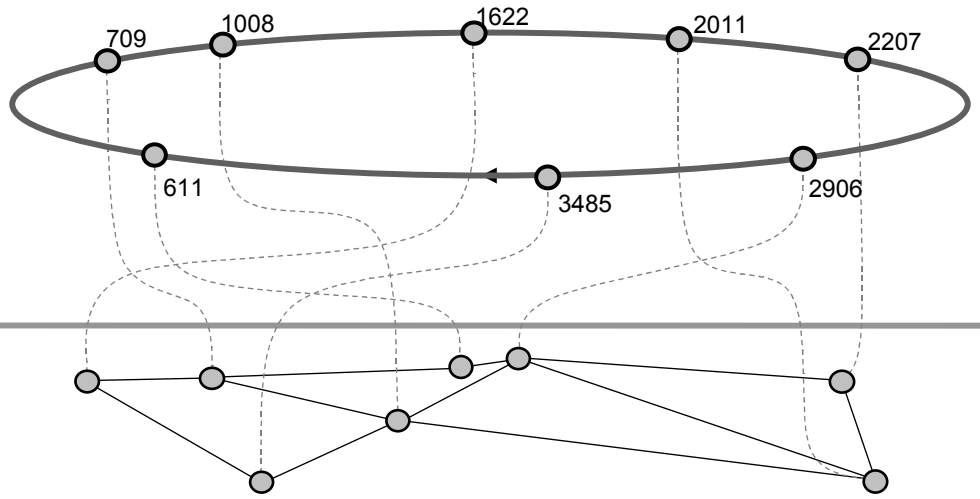


DHT: INDIRIZZAMENTO

Passo 1:

- Definizione dello spazio degli indirizzi. Esempio: Spazio degli indirizzi strutturato secondo un anello logico. S
  - spazio lineare degli indirizzi 0, …, 2^m-1 >> del numero di oggetti da memorizzare (es m=160), su cui è definito un ordinamento totale (operazioni in modulo)
- Associazione nodi-indirizzi, mediante una funzione hash
- La topologia reale e logica (overlay network) non sono in genere correlate

Distributed Hash Table: Visione logica



Mapping sulla topologia reale

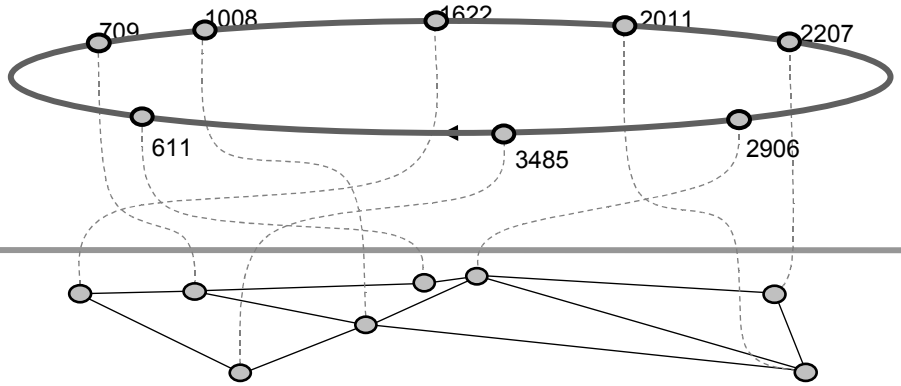
DHT: INDIRIZZAMENTO

Passo 2:

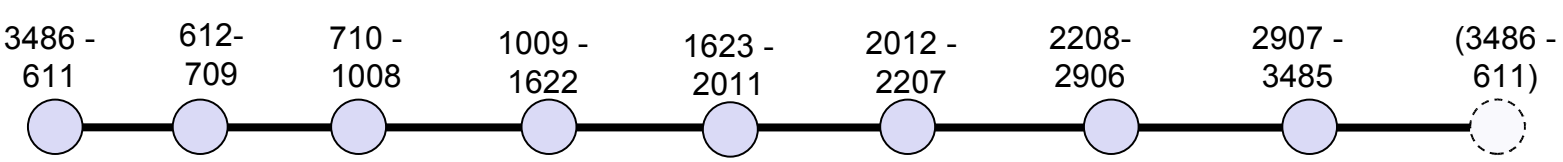
- Ogni nodo è responsabile di una parte dei dati memorizzati nella DHT
- I dati vengono mappati nello stesso spazio degli indirizzi dei nodi, mediante la funzione hash
  - E.g., Hash(*String*): H(„ *LucidiLezione12-03-06* “) → 2313
  - Esempi: hashing del nome del file o del suo intero contenuto
- Ad ogni nodo viene assegnata una porzione contigua dello spazio degli indirizzi.
- Ogni nodo memorizza informazioni relative ai dati mappati sulla propria porzione di indirizzi
- Spesso si introduce una certa ridondanza (overlapping)

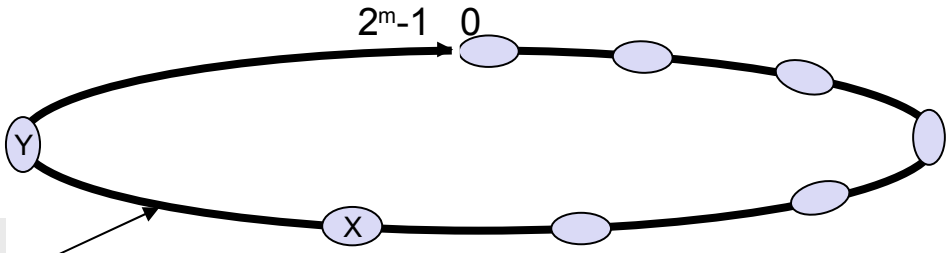
DHT: INDIRIZZAMENTO

Distributed Hash Table: Visione logica



Mapping sulla topologia reale





H(Node Y)=3485

Dato "D": H("D")=3107

H(Node X)=2906

DHT: BILANCIAMENTO DEL CARICO

- Problema: distribuzione uniforme degli indirizzi tra i peers che definiscono la DHT

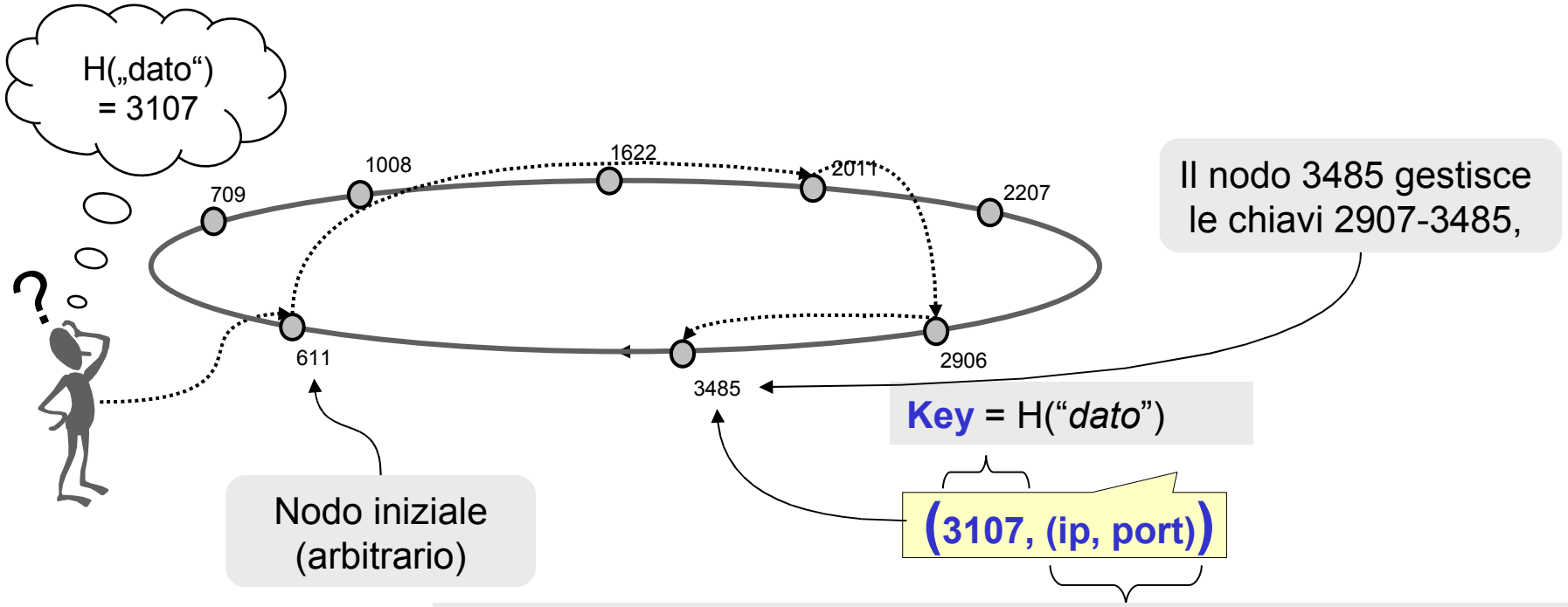
- Cause di possibili sbilanciamenti del carico:
  - Un nodo deve gestire una grossa porzione dello spazio degli indirizzi
  - Gli spazi degli indirizzi sono distribuiti in modo uniforme tra i nodi, ma gli indirizzi gestiti da un nodo corrispondono a molti dati
  - Un nodo deve gestire diverse queries, perché i dati corrispondenti agli indirizzi gestiti sono molto richiesti

- Sbilanciamento del carico comporta minor robustezza del sistema, minor scalabilità. Complessità $O(\log N)$ non garantita

- Definiti algoritmi di bilanciamento del carico

DHT: ROUTING

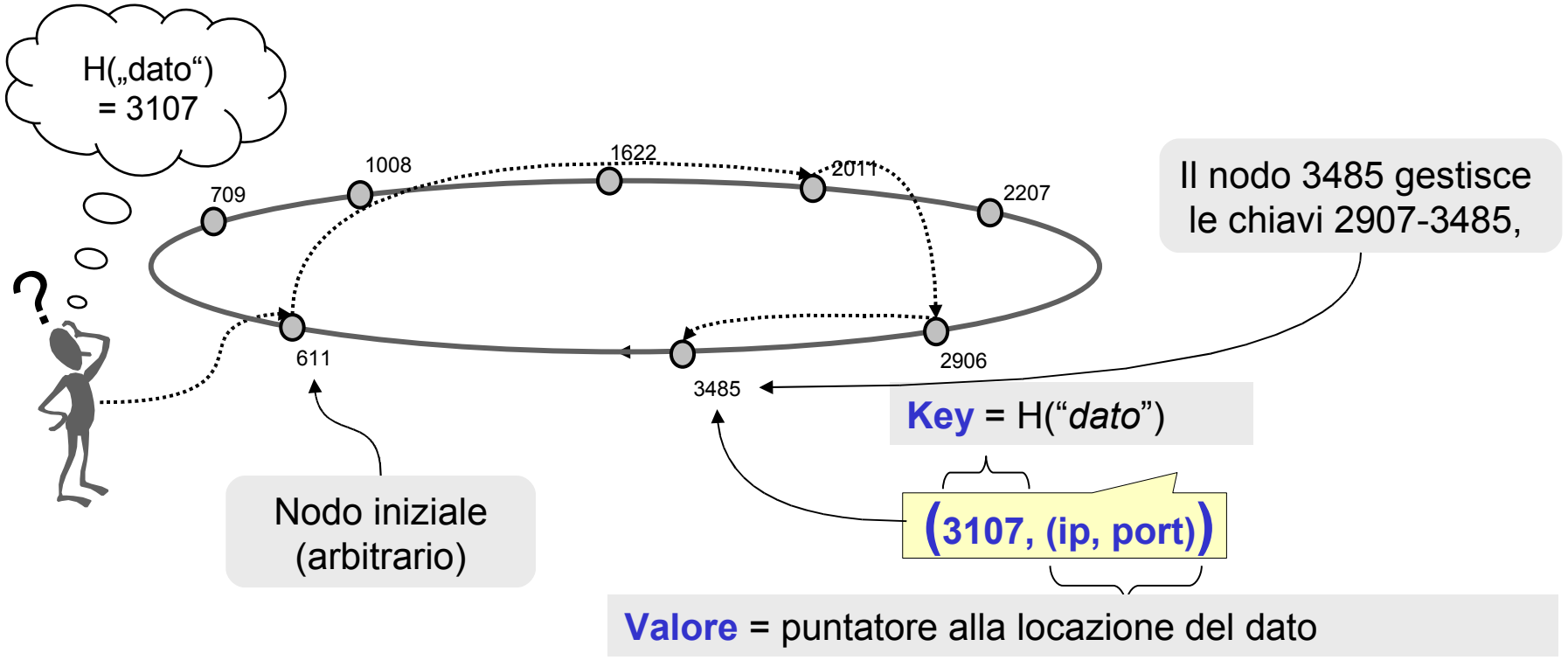
- Problema: dato D, cercare il nodo che gestisce key=H(D)
- La ricerca inizia in un nodo arbitrario della DHT
- La ricerca è guidata da H(D)



Valore = può essere un puntatore alla locazione effettiva del dato

DHT: ROUTING

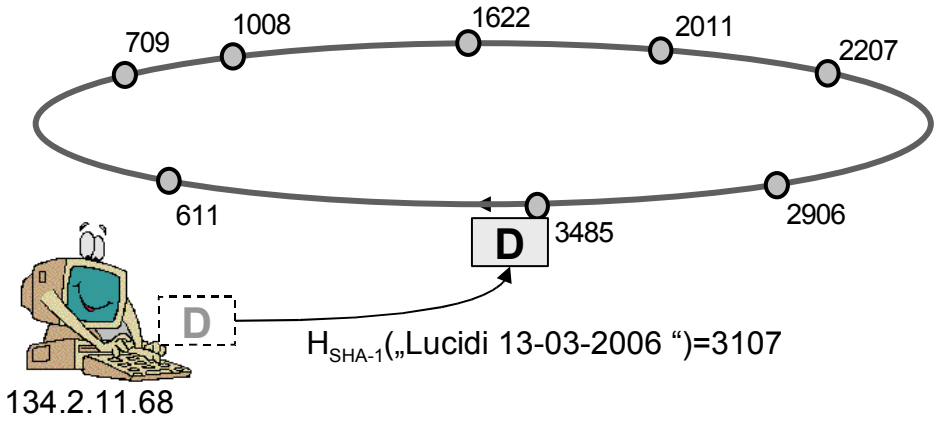
- Ogni nodo ha in genere una visione limitata della DHT
- Next hop: dipende dall'algoritmo di routing.
  - Esempio: basato sulla "vicinanza" tra l'ID del dato e l'ID del nodo (routing content based). Sistemi non strutturati: routing basato solo su connessioni tra nodi vicini



DHT: MEMORIZZAZIONE DIRETTA

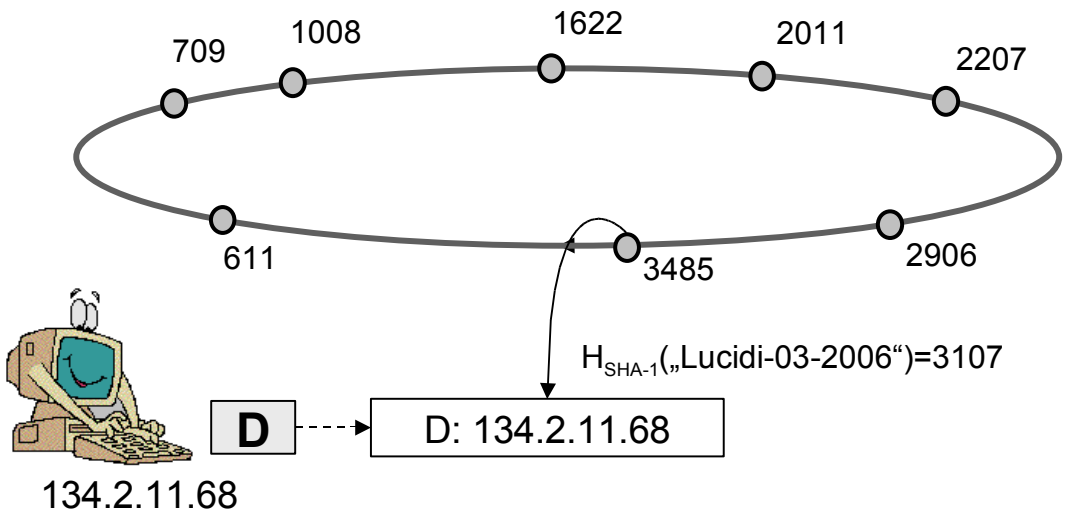
- La DHT memorizza coppie del tipo (key,valore)
- Valore = valore del dato ricercato
- Il dato viene copiato, al momento del suo insermento nella DHT, nel nodo che ne è responsabile (secondo il mapping dati-nodi). NOTA BENE: tale nodo non è in generale il nodo che ha inserito il dato nella DHT.

  Esempio: key = H("*Lucidi 13-03-2006* ") = 3107 . Il dato viene memorizzato sul nodo responsabile dell'indirizzo 3107.



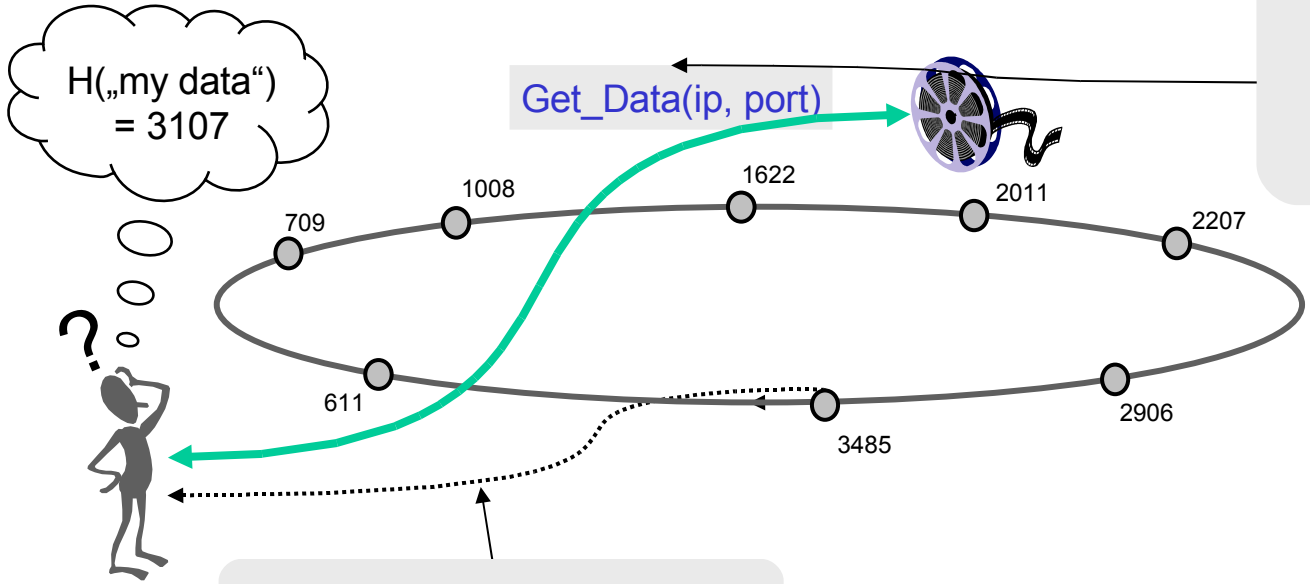
DHT:MEMORIZZAZIONE INDIRETTA

- Valore = riferimento al dato ricercato (es: indirizzo fisico del nodo che memorizza il contenuto)
- Il nodo che memorizza il dato può essere quello che lo ha inserito nel sistema
- Più flessibile, richiede un passo in più per l'accesso al dato



DHT: MEMORIZZAZIONE INDIRETTA

- Trasferimento dei dati
  - Si spediscono indirizzo IP e porta al richiedente
  - Il richidente effettua il download dei dati

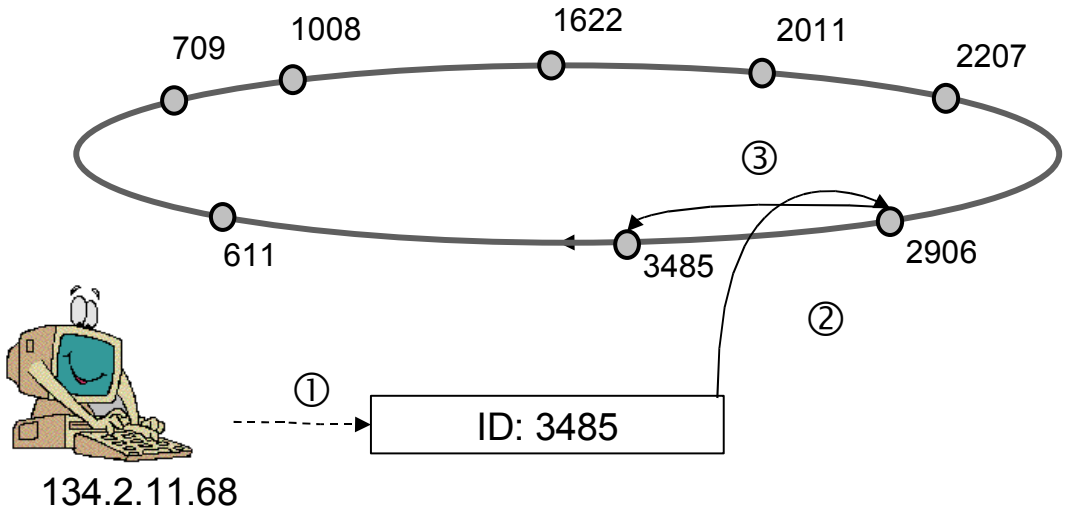


Memorizzazione Indiretta:
dopo aver ricevuto l'esatta
locazione del dato,
Il dato viene scaricato

Il nodo 3485 invia
(3107, (ip/port)) al richiedente

DHT:INSERZIONE DI NUOVI NODI

- Calcolo dell' identificatore del nodo, ID
- Il nuovo nodo contatta un nodo arbitrario della DHT (bootstrap)
- Assegnamento di una porzione dello spazio degli indirizzi ad ID
- Copia delle coppie K/V assegnate (in genere si utilizza ridondanza)
- Inserzione nella DHT (collegamento con nodi vicini)



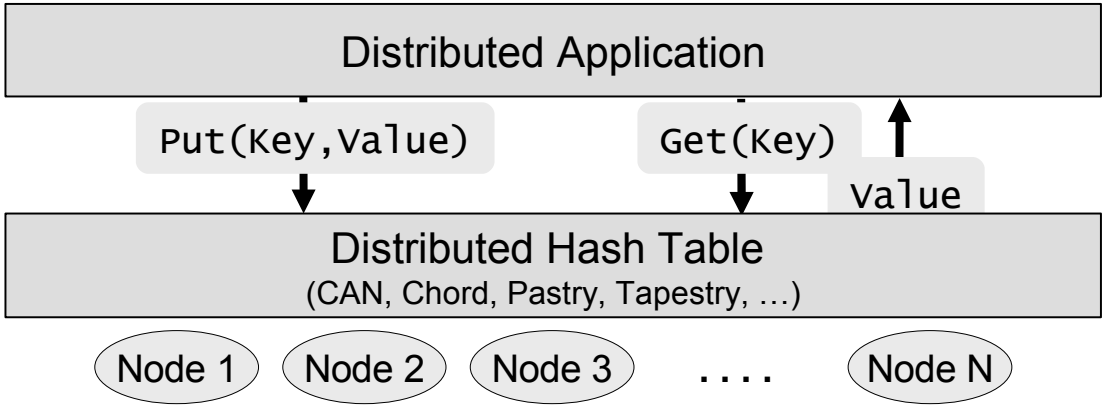
DHT:RITIRO/FALLIMENTO DI NODI

- Ritiro Volontario di un nodo
  - Partizionamento della propria porzione degli indirizzi sui nodi vicini
  - Copia delle coppi chiave/valore sui nodi corrispondenti
  - Eliminazione del nodo dalle tabelle di routing

- Fallimento di un Nodo
  - Se un nodo si disconnette in modo inatteso, tutti i dati memorizzati vengono persi a meno che non siano memorizzati su altri nodi
    - Memorizzazione di informazioni ridondanti (replicazione)
    - Perdita delle informazioni; refreshing periodico delle informazioni
  - Utilizzo di percorsi di routing alternativi/ridondanti
    - Probing periodico dei nodi vicini per verificarne la operatività. In caso di fault, aggiornamento delle routing tables

DHT: APPLICAZIONI

- Interfaccia (API) per l'accesso alla DHT
  - Inserimento di Informazione Condivisa
    - PUT(key,value)
  - Richiesta di Informazione (content serach)
    - GET(key)
  - Risposte
    - Value
- L'interfaccia è comune a molti sistemi basati su DHT



DHT: APPLICAZIONI

- Le DHT offrono un servizio generico distribuito per la memorizzazione e l'indicizzazione di informazioni

- Il valore memorizzato in corrispondenza di una chiave può essere
  - Un file
  - Un indirizzo IP
  - O qualsiasi altro dato……

- Esempi di applicazioni che possono utilizzare le DHT
  - Realizzazione di DNS
    - Chiave: hostname, valore: lista di indirizzi IP corrispondenti
  - P2P storage systems: es. Freenet

  ……

CONCLUSIONI

Proprietà delle DHT

- Il routing è basato sul contenuto della query
- Le chiavi sono equamente distribuite tra i nodi della DHT
  - Si evitano i colli di bottiglia
  - Supportano l'inserzione incrementale di chiavi nel sistema
  - Tolleranti ai guasti
- Sistemi auto-organizzanti
- Realizzazione semplice ed efficiente
- Supportano un ampio spettro di applicazioni
  - I valori associati alle chiavi dipendono dalla applicazione

DHT: SISTEMI ESISTENTI

- Chord

  UC Berkeley, MIT
- Pastry

  Microsoft Research, Rice University
- Tapestry

  UC Berkeley
- CAN

  UC Berkeley, ICSI
- P-Grid

  EPFL Lausanne

- … ed altri: Kademlia, Symphony, Viceroy, …