Corso di Architetture Avanzate di Rete
Prof. Alfio Lombardo
A.A. 2005-2006

I sistemi peer-to-peer

Mirco Tribastone
mirco.tribastone@diit.unict.it

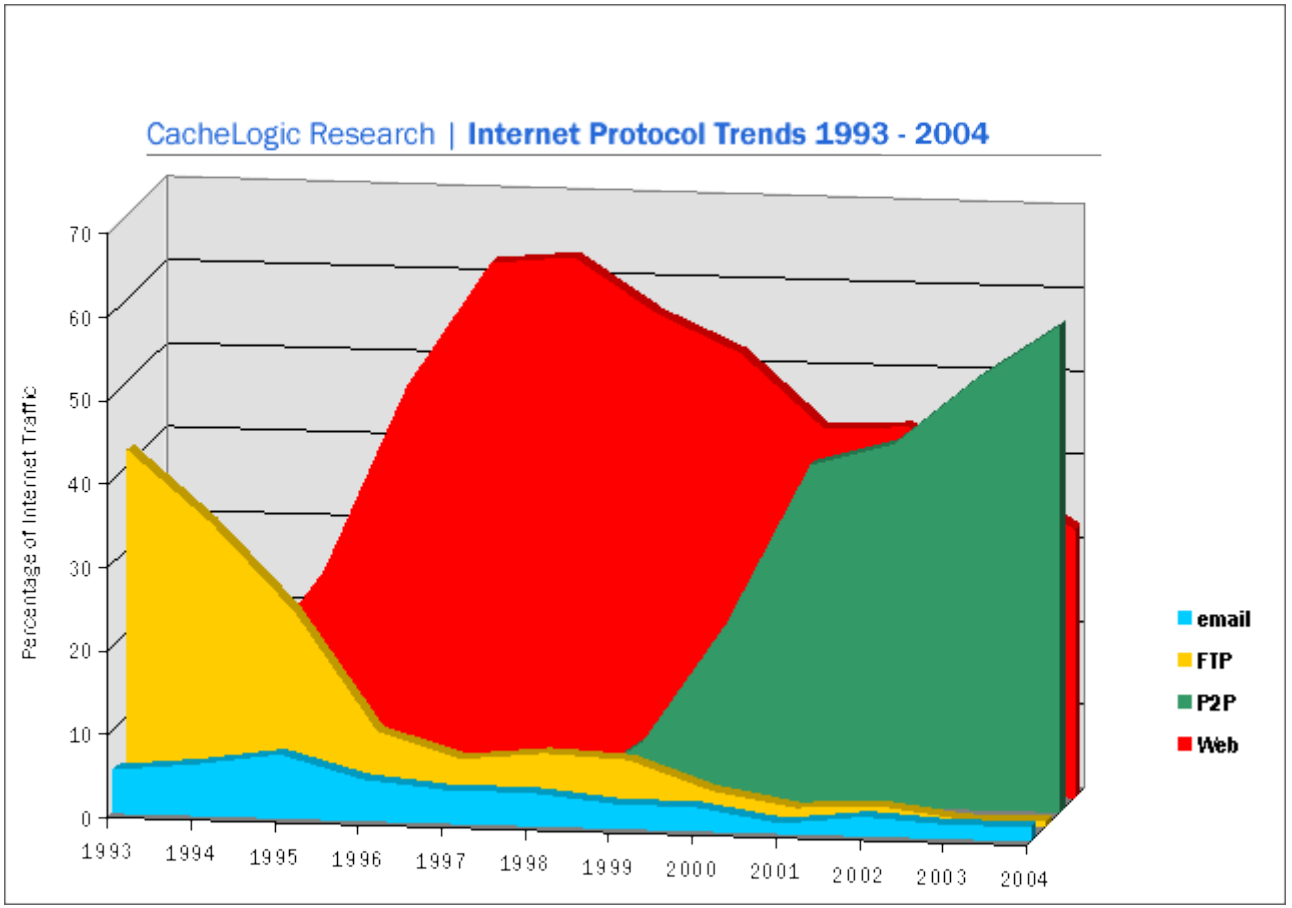
Outline

- Introduzione
- Modello client-server
- Modello peer-to-peer
- Proprietà dei sistemi peer-to-peer
- Tassonomia
  - Servizi
  - Architettura

Definizione di peer-to-peer

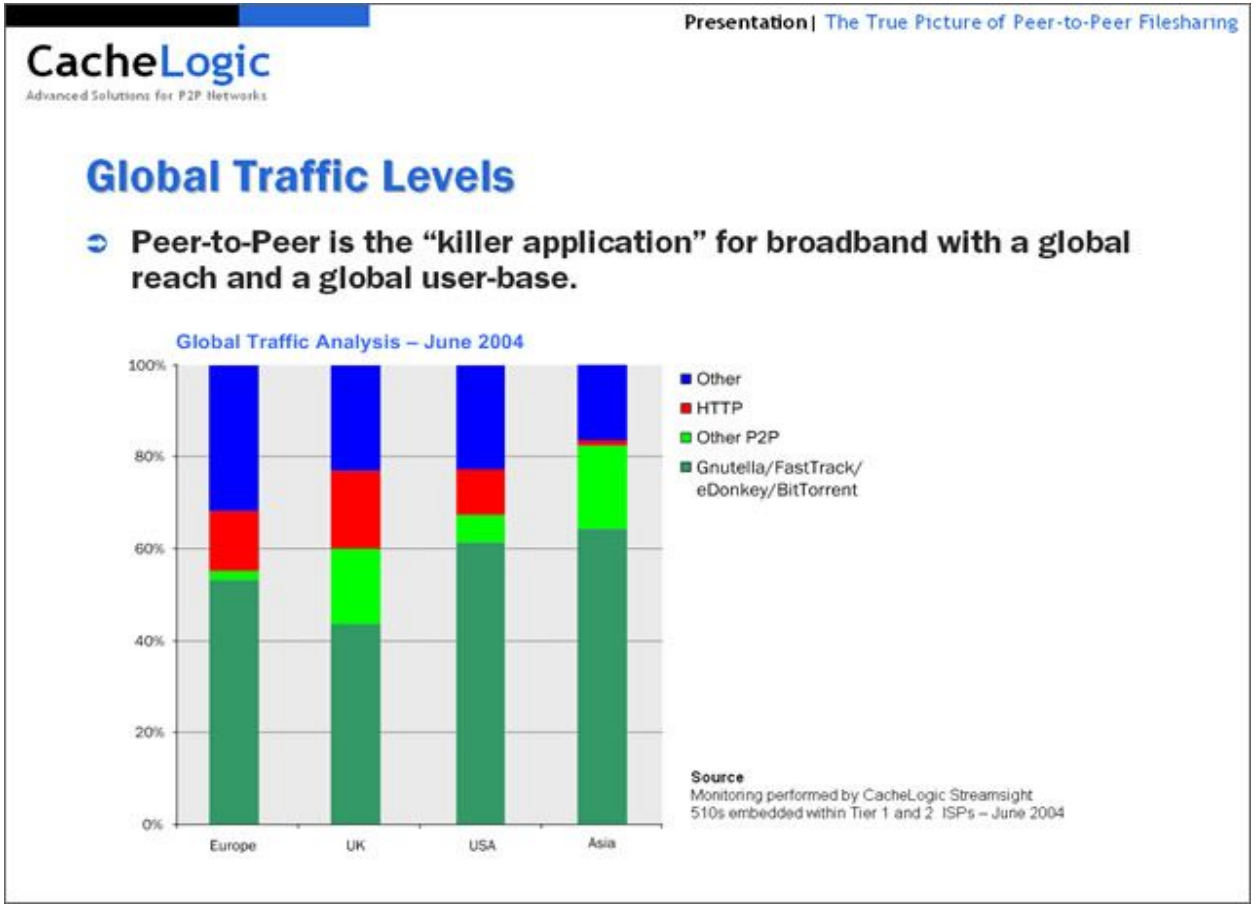
- Il modello **peer-to-peer** (P2P) è un paradigma di progettazione per le applicazioni **distribuite**
- In un sistema P2P le entità partecipanti **condividono** le proprie risorse per contribuire attivamente alla fornitura del **servizio**
- Il modello P2P si contrappone alla tradizionale architettura **client/server** (C/S)

Impatto del P2P su Internet



Fonte: cachelogic.com

Impatto del P2P su Internet

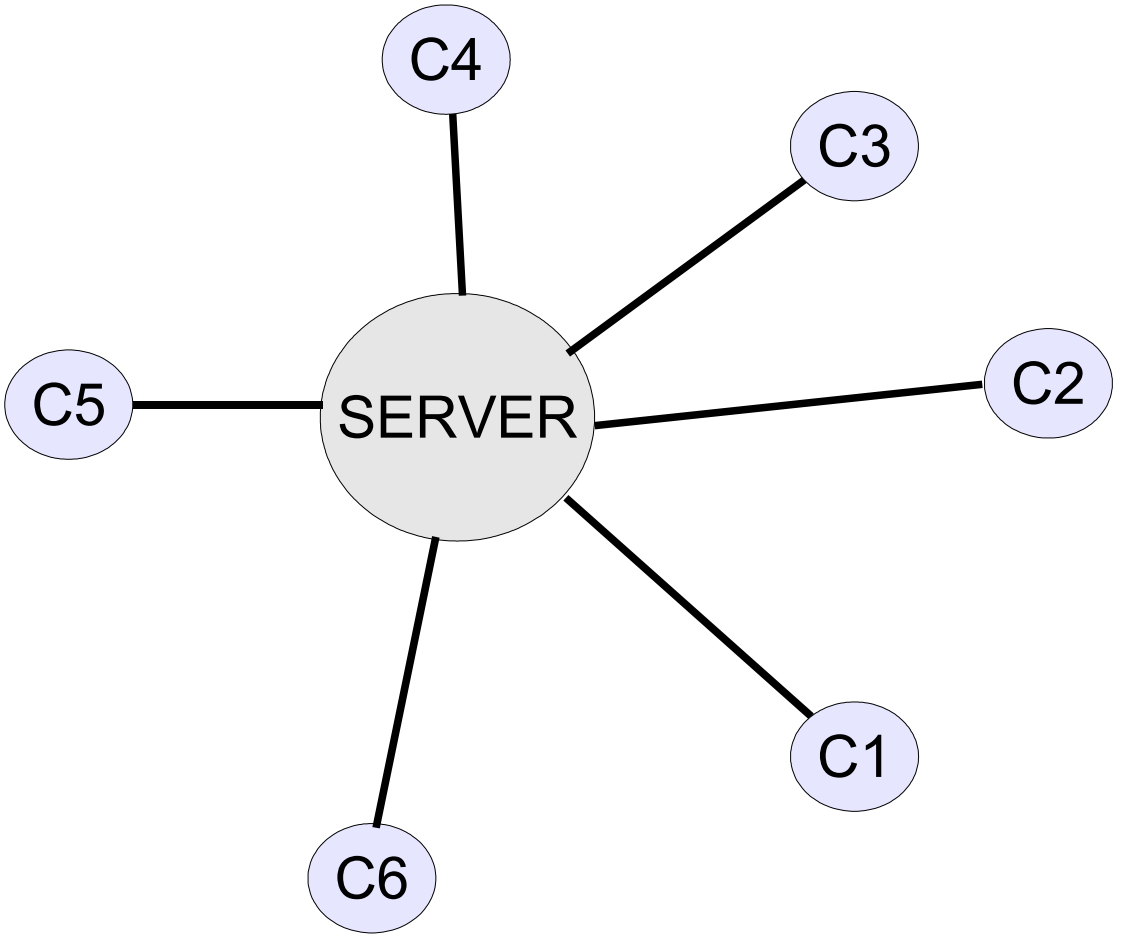


Fonte: cachelogic.com

Il modello client-server

- In un sistema C/S ogni nodo può comportarsi come client o come server
- Il server svolge un ruolo **passivo**
  - Resta in attesa di richieste di servizio da parte dei client
  - Serve ogni richiesta trasmettendo un messaggio di risposta
- Il client svolge un ruolo **attivo**
  - Manda una richiesta di servizio
  - Resta in attesa di un messaggio di risposta

Rappresentazione di un sistema C/S



L'accentramento delle risorse

- In un sistema C/S, le **risorse** (e quindi i **costi**) necessarie per fornire il servizio sono **concentrate** nei server
- Tipicamente il rapporto tra server e client è **uno-a-molti**: la responsabilità del servizio non è affidata omogeneamente a tutte le entità partecipanti

Scalabilità di un sistema C/S

- All'aumentare del numero di richieste, le prestazioni di un sistema C/S degradano
  - Per esempio, in un server FTP, al crescere degli utenti diminuisce la larghezza di banda disponibile per la singola connessione, e aumentano i tempi di attesa
- Le prestazioni del sistema dipendono dal server
  - I miglioramenti possono avvenire soltanto investendo risorse per l'aggiornamento della configurazione del server (e.g., aumento della capacità)

Disponibilità del servizio

- Il server è l'**unica entità** che può fornire il servizio
- In caso di **failure** del server (malfunzionamento hardware, di rete, etc.) il servizio non è più **disponibile**
- Per garantire un'adeguata **Qualità del Servizio** è necessario adottare opportuni accorgimenti (ridondanza nei sistemi di alimentazione, nello storage, etc.)

Il modello peer-to-peer

- In un sistema peer-to-peer ogni entità (**peer**) partecipa alla fornitura del servizio
  - Per esempio, in un'applicazione di **file-sharing** ogni peer condivide una o più directory del proprio filesystem
- Un peer agisce **contemporaneamente** come client e come server (**servant**)
  - Come server, fornisce parte delle sue risorse
  - Come client, richiede le risorse degli altri peer

Decentramento delle risorse

- Il servizio è fornito in modo **distribuito** e **decentralizzato**
- La **capacità di servizio** del sistema è costituita dall'aggregazione delle risorse (cicli di CPU, storage, etc.) dei peer
- La qualità del servizio dipende dalle risorse che ogni peer **autonomamente** mette a disposizione

Localizzazione delle risorse

- Nelle applicazioni C/S la risorsa è localizzata facilmente perché è noto **a priori** l'identità del server
  - es. `http://www.unict.it`
- Nei sistemi P2P gli utenti accedono alle risorse in seguito ad una fase di **ricerca**
- I peer non utilizzano il DNS, e sono caratterizzati da connettività tipicamente **non permanente**

Overlay network

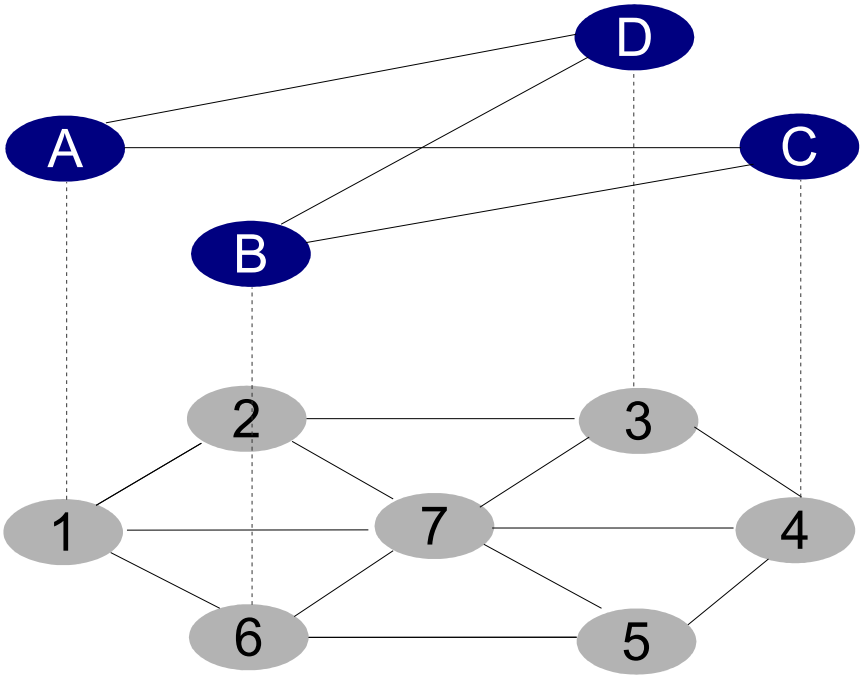
- E' necessario uno spazio di indirizzamento e un algoritmo di routing che siano **indipendenti** da IP
- I peer cooperano formando una **overlay network**

Overlay network

- Una **overlay network** (rete sovrapposta) è una rete **logica** di computer costruita su una rete fisica (**underlying network**)
- I nodi dell'overlay sono un sottoinsieme dei nodi della rete
- Le connessioni logiche si stabiliscono sfruttando le funzionalità della rete sottostante
  - In un'overlay su Internet, si stabiliscono connessioni TCP tra ogni coppia di nodi

Overlay network

- La connessione logica TCP (es. A-D) tra due nodi si risolve in un **path fisico** (presumibilmente) multi-hop (1-2-3)
- **Dominio dei nomi** indipendente dall'architettura di rete sottostante
- Possibilità di algoritmi di **routing** ad hoc
  - multicast



Overlay network

- In Internet, la maggior parte dei sistemi peer-to-peer sono costruiti su overlay network
  - I sistemi peer-to-peer sono anche chiamati **reti peer-to-peer**
- I messaggi di protocollo sono generalmente **incapsulati** in pacchetti TCP
  - Per esigenze di performance, alcuni protocolli si basano su pacchetti UDP

Proprietà delle reti P2P

- La decentralizzazione conferisce al modello P2P alcune importanti proprietà
  - Scalabilità
  - Condivisione/riduzione dei costi
  - Disponibilità del servizio
  - Autonomia
  - Anonimato/privacy

Scalabilità dei sistemi P2P

- Esistono sistemi P2P che supportano **milioni** di utenti
- Il carico viene **bilanciato** tra tutti i nodi dell'overlay
- La scalabilità è limitata dalle operazioni di **coordinamento** e **sincronizzazione** tra i peer

Condivisione dei costi

- I costi per la fornitura del servizio sono legati alle risorse che si mettono a disposizione:
  - Nel caso della pubblicazione di file-sharing, l'approccio C/S è quello di FTP: i costi legati allo storage sono totalmente a carico server
  - Approcci analoghi basati P2P sono le reti per il file-sharing (Napster, Gnutella, BitTorrent) in cui ogni peer condivide una **quota del proprio disco**

Riduzione dei costi

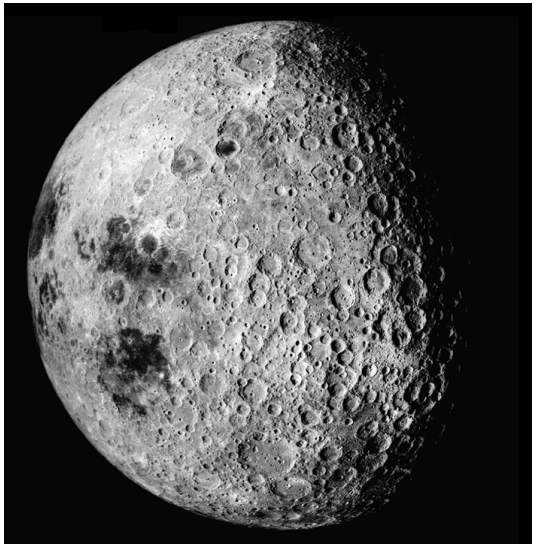
- L'approccio P2P può condurre eventualmente ad una riduzione dei costi
- Non c'è diretta proporzionalità tra capacità di servizio e costo
- Un sistema centralizzato che gestisce centinaia di TB di dati è molto più complesso (e costoso) che organizzare una rete P2P in cui ogni nodo condivide pochi GB!
  - I server devono essere sempre on-line (gruppi di continuità, alimentazione ridondante)
  - Devono gestire numerosi accessi (connessioni ad alta velocità, memoria adeguata)

Riduzione dei costi

- In alcuni casi, i costi si possono ridurre drasticamente, aggregando risorse **inutilizzate**
  - Cicli di CPU durante i momenti di inattivita' del PC (i.e., quando si attiva lo screen-saver)
  - Spazio su disco per utenti remoti quando quello effettivamente disponibile è molto elevato
- In questi casi, la spesa per la fornitura del servizio è praticamente nulla
  - La maggior parte dei PC domestici è effettivamente sovradimensionata rispetto al carico di lavoro svolto!

Riduzione dei costi

Commodore 64
1 Mhz – 64 KB RAM



La Luna

Disponibilità del servizio

- In un contesto C/S, se il server va giù il servizio non è più disponibile
- Nei sistemi P2P, se un peer si disconnette, il servizio **continua comunque** ad essere fornito dagli altri peer
- Meccanismi di **replicazione** per recuperare risorse disponibili presso il solo peer uscente

Autonomia

- Ogni peer decide **cosa** e **quanto** condividere
  - ✔ Pubblicazione di contenuti altrimenti sottoposti a **censura**
  - ✔ Scambio di materiale **protetto** da copyright
  - ✗ I contributi dei peer dovrebbero essere distribuiti **uniformemente**
  - ✗ Difficoltà nel controllo di materiale legato ad **attività illecite**

Anonimato

- Assicurare l'anonimato nei sistemi C/S è difficile perché il server **deve** essere individuabile (indirizzo IP)
- Nei sistemi P2P, si possono identificare diverse forme di anonimato:
  - Autore
  - Pubblicatore
  - Server
  - Lettore
  - Documento
  - Query

Tassonomia (servizio)

- Il **file-sharing** è la classe di sistemi P2P più utilizzata...
- ... ma il P2P **non è solo** file-sharing!

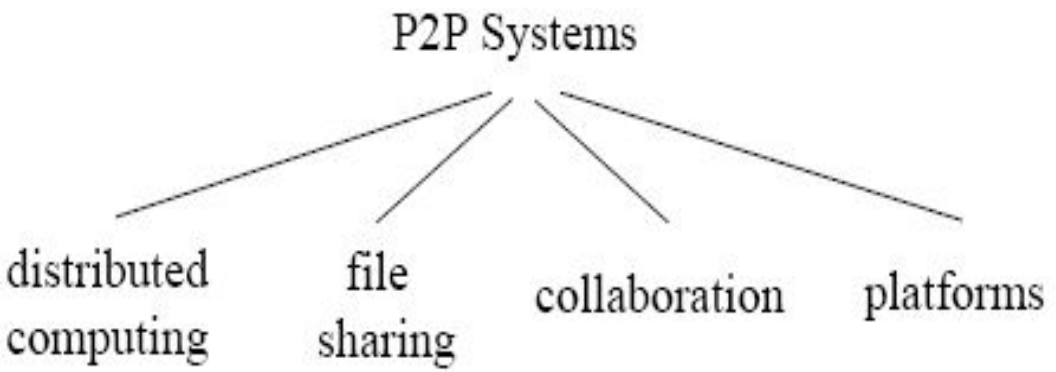
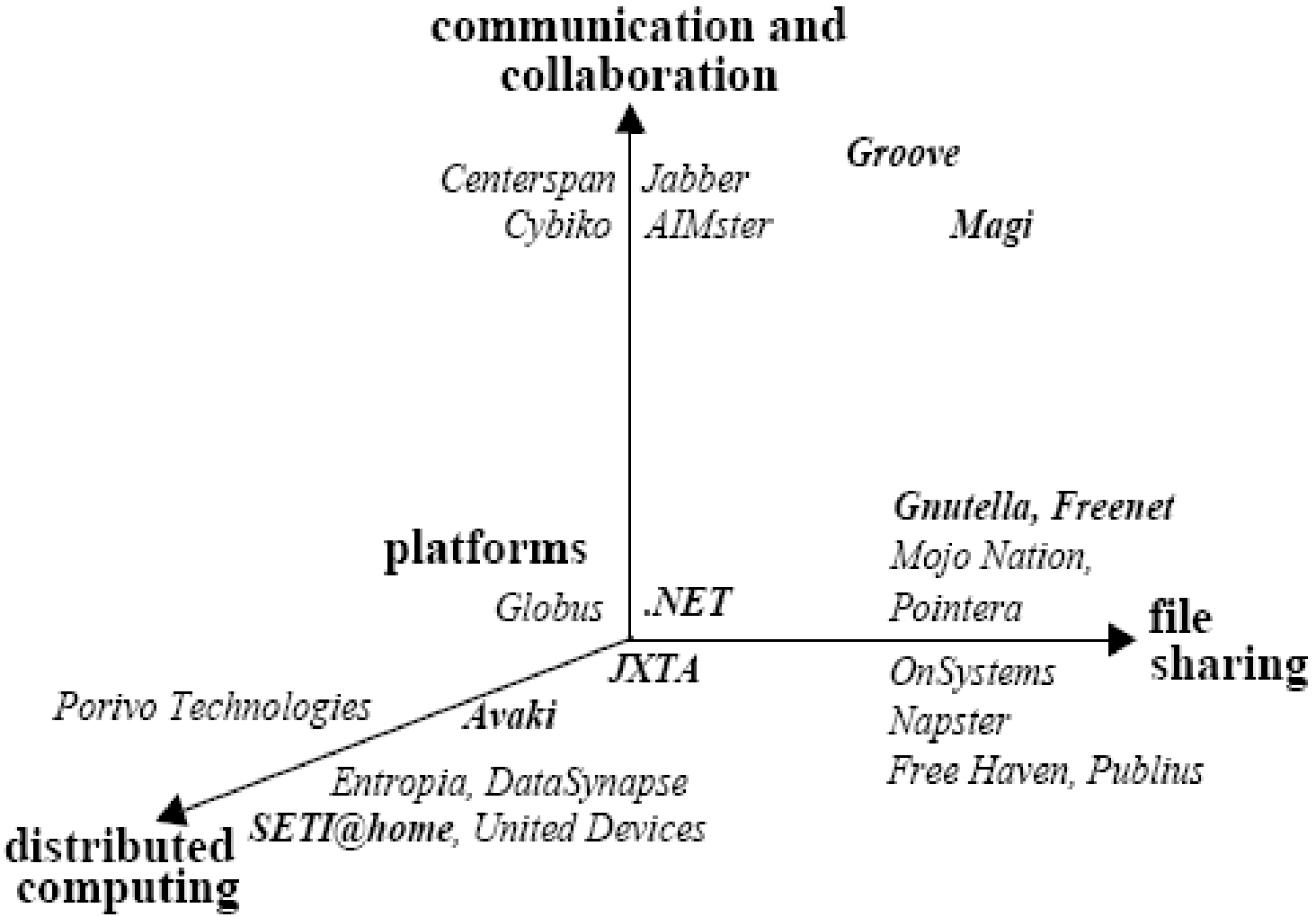


Figure 6: A Taxonomy of P2P Systems.

Alcuni esempi (2002)

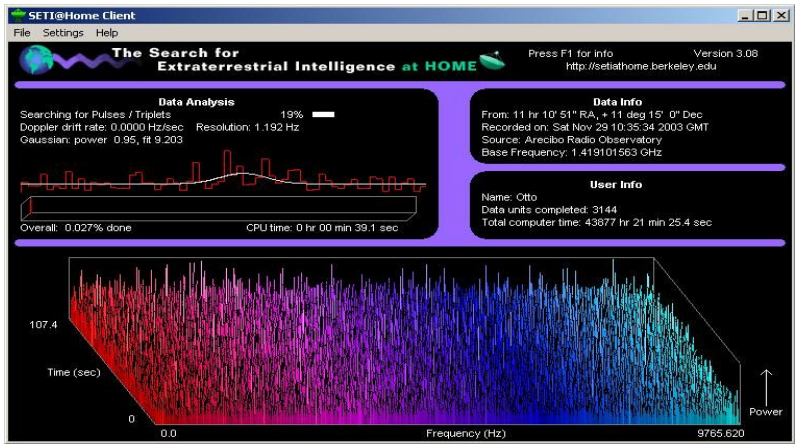


Distributed Computing

- La risorsa condivisa sono i **cicli di CPU** delle macchine in rete
- Un complesso problema di calcolo **intrinsicamente parallelo** è risolto distribuendololo tra i vari utenti
  - Ricerca di vita extra-terrestre
  - Mappa del genoma umano
  - Folding delle proteine

SETI@Home

- **S**earch for **E**xtra-**T**errestrial **I**ntelligence
- Progetto nato a Berkeley nel 1999
- Sfrutta i momenti di inattività di ogni PC collegato
  - Uno screensaver che raccoglie ed analizza blocchi di dati indipendenti (work-unit) di 350 KB provenienti dal radiotelescopio di Arecibo



SETI – Alcuni numeri

- Oltre **5 milioni** di partecipanti
- Già nel 2001, **10^21** operazioni in virgola mobile complessivamente eseguite
- **14 miliardi** di ore di CPU-time
- E' più veloce di qualunque supercomputer attualmente esistente, con un costo complessivo dell'**1%**
- http://setiathome.ssl.berkeley.edu/

Altri esempi...

- **Folding@home**
  - Promosso dall'Università di Standord (http://folding.stanford.edu/)
  - Simulazioni sul **"folding"** delle proteine per lo studio di malattie come la BSE o il morbo di Alzheimer
  - Più di **100.000 processori** coinvolti
  - La simulazione di **1 ns dura 1 giorno** di CPU-time!

Sistemi P2P per la comunicazione

- La risorsa condivisa è la **presenza umana** su Internet
- Esistono sistemi P2P per tutti i tipi di comunicazioni
  - Scrittura (chat)
  - Audio (telefonia su IP)
  - Video

Sistemi P2P per la comunicazione

- `talk` è stato il primo applicativo P2P di chat
  - È nato negli anni '80, disponibile in ambiente BSD 4.2
  - Entrambi gli interlocutori usavano lo **stesso software**
- L'evoluzione di questi sistemi ha portato a prodotti come
  - ICQ
  - MSN
  - Yahoo! Messenger
  - Skype (**VoIP**)

Video su P2P

- Si condividono contenuti audio/video
- PPLive è un'applicazione peer-to-peer per la **TV on-line**
  - È disponibile una lista di canali
  - Per ogni canale si forma un'overlay degli utenti interessati
  - Ogni peer riceve e replica lo stream ai nodi più vicini
- La **BBC** sta sperimentando un servizio P2P 'Interactive Media Player' per la condivisione dell'intero palinstesto televisivo

File-sharing

- E' l'applicazione P2P più conosciuta ed utilizzata
- Condivisione a livello di directory
  - L'utente specifica una directory del proprio filesystem in cui pubblica i propri file
- Alcuni esempi
  - Napster (1999)
  - Gnutella
  - Edonkey, FastTrack (KaZaA), BitTorrent, etc.

La localizzazione delle risorse

Architetture per la localizzazione delle risorse

- Il routing a livello di overlay consente la localizzazione delle risorse distribuite nella rete P2P
- Esistono tre possibili architetture:
  - Modello **centralizzato**
  - Modello distribuito **non strutturato**
  - Modello distribuito **strutturato**

Modello centralizzato

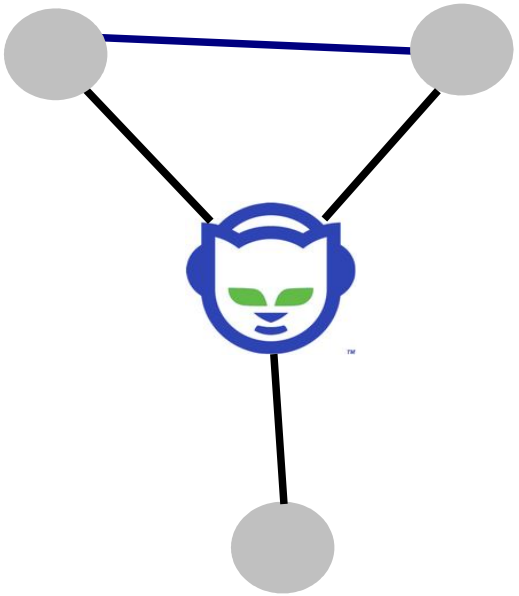
- E' il modello reso popolare da Napster
- Gli utenti del sistema connettono un **server centrale** in cui pubblicano **i nomi** delle risorse che condividono
- Le query sono trasmette al server, che risponde con **le identità** dei peer che soddisfano i criteri di ricerca

Modello centralizzato

- E' un'applicazione client-server oppure P2P?
  - La ricerca è basata sul modello C/S
  - Il trasferimento del file (**il Servizio**) è effettivamente P2P
  - Il server contiene solo **meta-informazioni**
    - (peer ip address, tcp port, resource name)

Il caso Napster

- Condivisione di brani musicali in formato MP3
- Attività dei peer
  - Registrazione/Autenticazione
  - Ingresso (e trasferimento degli indici al server)
  - Ricerca e download
  - Uscita (e cancellazione delle meta-info nei server)



Altri esempi

- I servizi di messaggistica istantanea
  - ICQ, MSN, Yahoo!Messenger, Google Talk, Skype, etc.
  - I server contengono le informazioni sui profili degli utenti
  - La comunicazione è diretta
- Altri sistemi per il file-sharing
  - AudioGalaxy

Caratteristiche del modello centralizzato

- Il sistema è **scalabile**?
  - È **più scalabile** rispetto ad un approccio client-server
  - Il server, invece di conservare il file completo, ne memorizza solo l'indice
    - E.g., il server localizza un file audio di **5 MB**, conservando al massimo **1 KB** di dati (titolo, tag, e indirizzo del peer), pari allo **0.02%**
  - L'architettura implementata in Naspter ha gestito un picco massimo di **6 milioni di utenti**!

Caratteristiche del modello centralizzato

- Il sistema è **anonimo**?
  - Il sistema **non è** anonimo
  - Gli utenti sono tipicamente sottoposti ad una fase di registrazione e autenticazione
  - Informazioni sulle query di ricerca sono memorizzabili nel server
  - Il publisher è direttamente rintracciabile in base al suo indirizzo IP

Caratteristiche del modello centralizzato

- Non è **fault-tolerant**
  - Se il server va giù, tutti i trasferimenti peer-to-peer non vengono influenzati
  - Non è più possibile l'accesso al sistema nè la ricerca di nuove risorse (solo il server risponde alle query)
- Il servizio è **deterministico**
  - Il server ha una **visibilità globale** del sistema: se la risorsa esiste, è sicuramente localizzabile!

Modello distribuito non strutturato

- Sono sistemi peer-to-peer **puri**
- L'accesso alla rete (**bootstrapping**) non avviene contattando un'entità gerarchicamente superiore
- Anche la **ricerca è distribuita**: ogni peer risponde solo dei file che condivide
- L'assenza del server implica la necessità di un'algoritmo di routing sull'overlay per i messaggi del protocollo
- **Non strutturato**: la formazione dell'overlay non è controllata

Il protocollo Gnutella

- E' stato il primo protocollo peer-to-peer puro per il file-sharing (2000)
- Circa 2 milioni di utenti
- Più di 30 implementazioni sviluppate
- Ogni peer gestisce 4 attività fondamentali
  - Accesso alla rete
  - Mantenimento della rete
  - Ricerca del file
  - Download del file

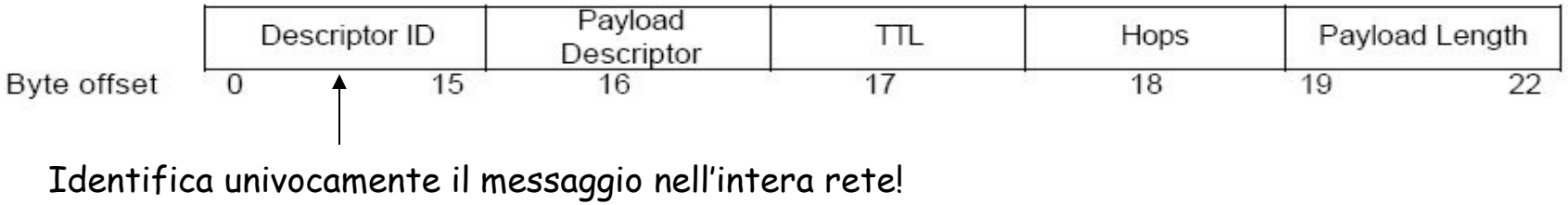
Ingresso nelle rete Gnutella

- Non esiste un server dedicato (come Napster)
- Il peer entrante deve conoscere l'identità di almeno un peer già connesso
  - Indirizzi conservati in cache da precedenti sessioni
  - Ricavati con meccanismi **out-of-band**: web, IRC, etc.
- L'**esplorazione del vicinato** avviene instradando messaggi sulle connessioni (TCP) di bootstrap effettuate

I messaggi di Gnutella

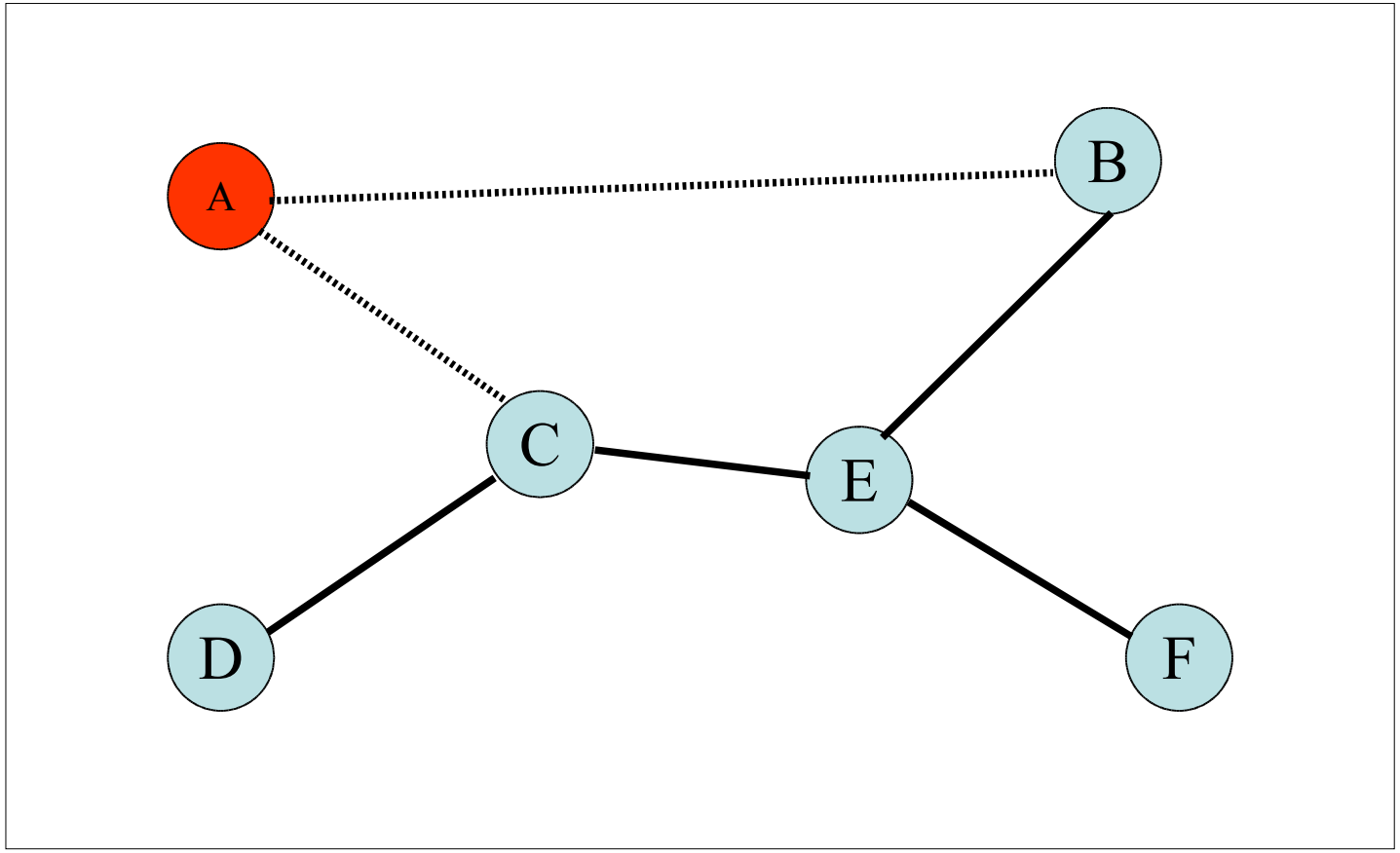
- I segenti messaggi sono incapsulati all'interno del pacchetto TCP della connessione P2P:
  - **PING**: request per il mantenimento della rete
  - **PONG**: reply di PONG
  - **QUERY**: request per la ricerca di una risorsa
  - **QUERY-HIT**: reply di QUERY
  - **PUSH**: trasferimento del file
- I messaggi di request sono trasmessi in **flooding**
- I messaggi sfruttano il **backward-learning**

Algoritmo di routing



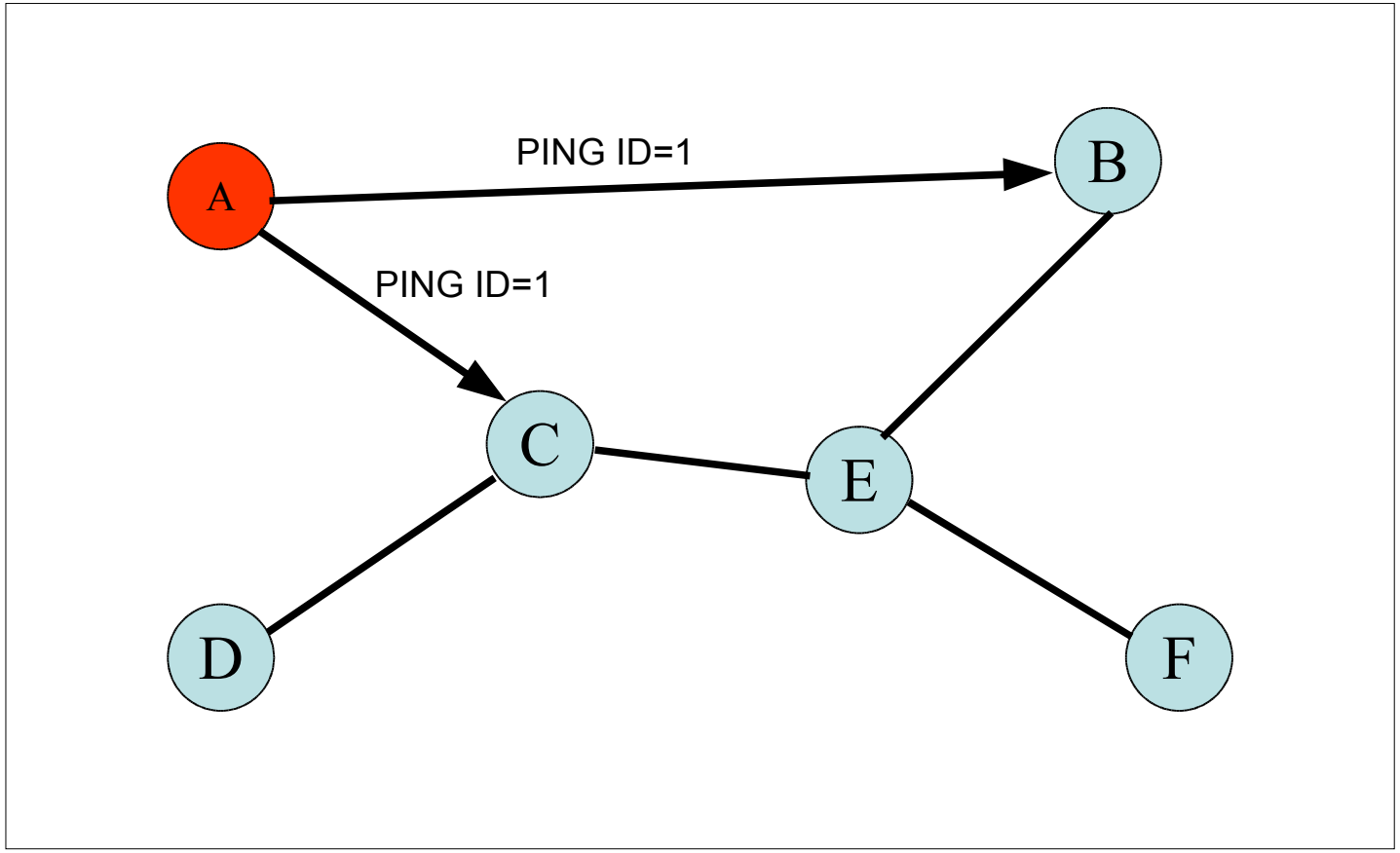
- La circolazione infinita di un pacchetto è prevenuta con i campi TTL e HOPS
  - Inizialmente, TTL è posto al massimo numero di hop (e.g., 10)
  - Quando TTL = 0 il pacchetto è scartato
- Ogni peer conserva gli ID dei messaggi che ha instradato!

Routing su Gnutella



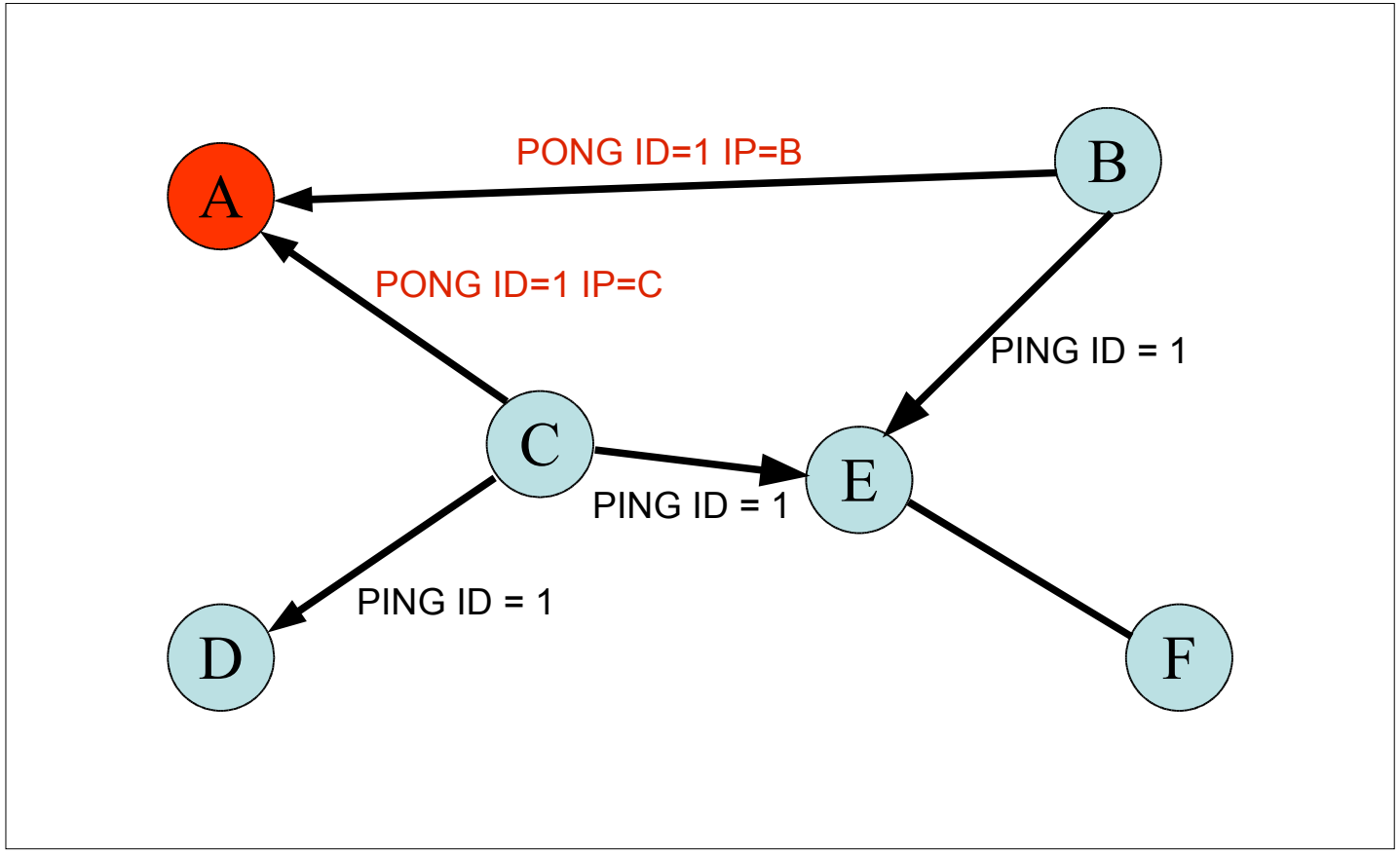
Time 0

Routing su Gnutella



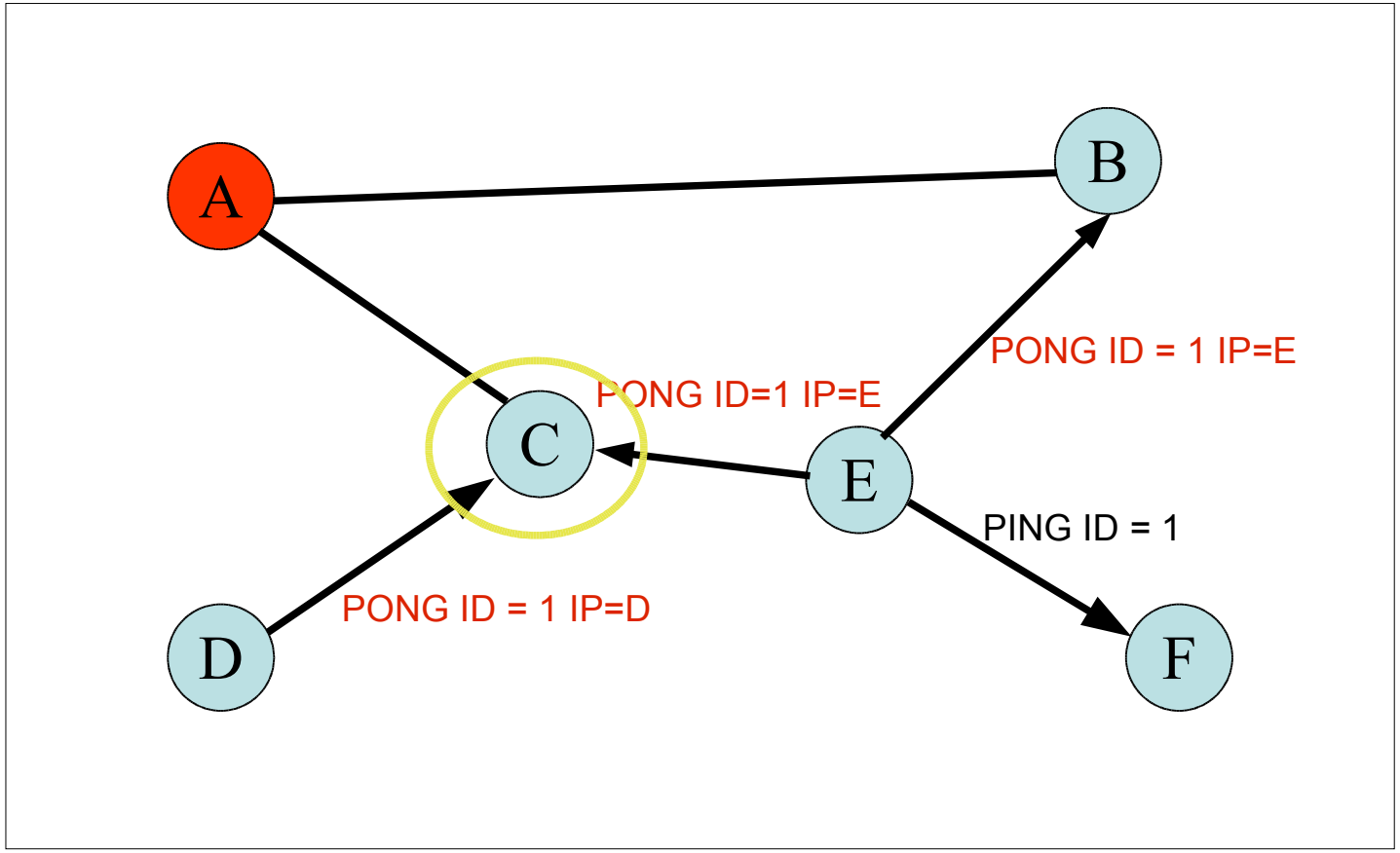
Hop 1

Routing su Gnutella



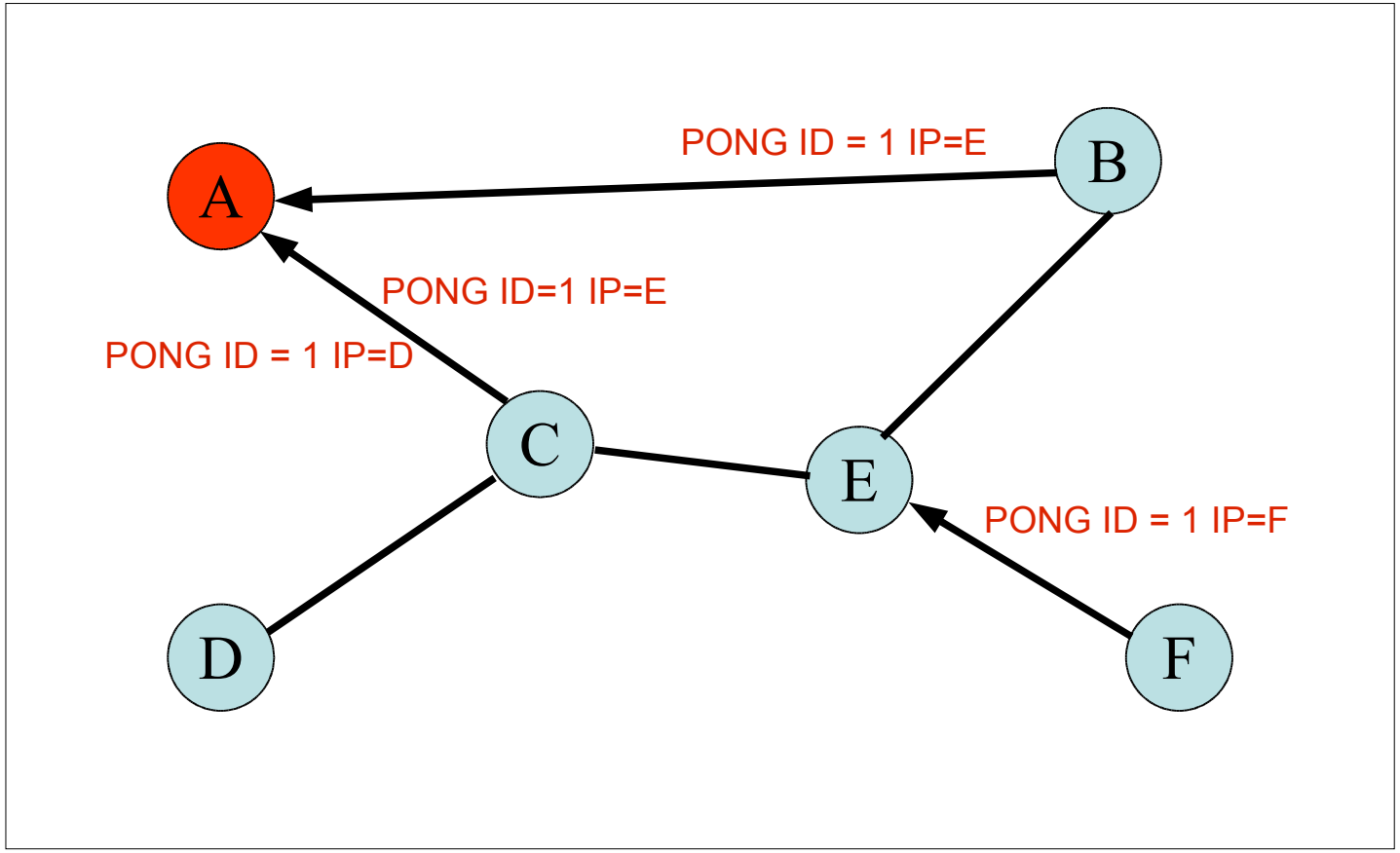
Hop 2

Routing su Gnutella



Hop 3

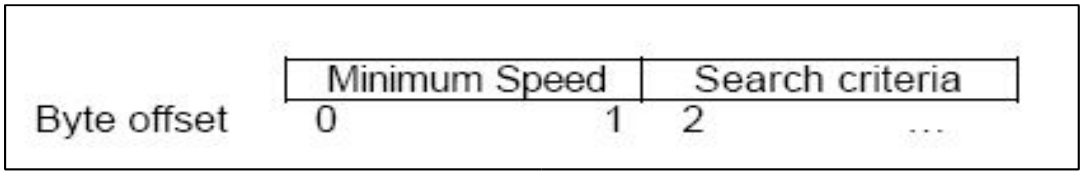
Routing su Gnutella



Hop 4

Messaggio di Query

- Implementa il servizio di ricerca all'interno della rete

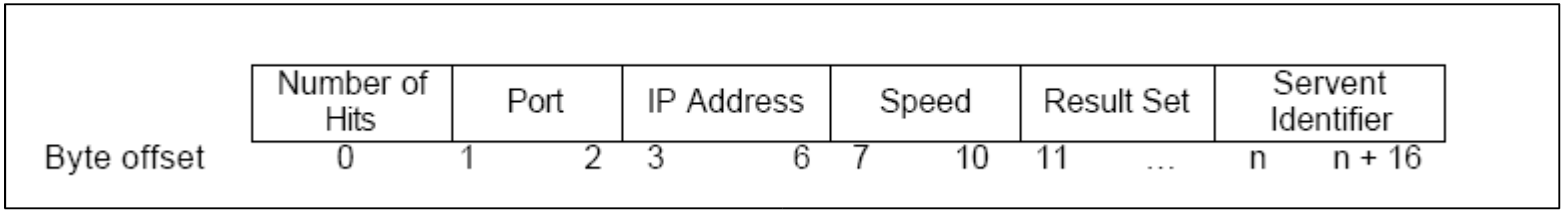


Minimum Speed = velocità minima richiesta al server che risponde

Search Criteria = stringa ASCII

- E' instradato con lo stesso meccanismo del messaggio PING

Query-Hit



Number of Hits: numero di elementi di Result Set

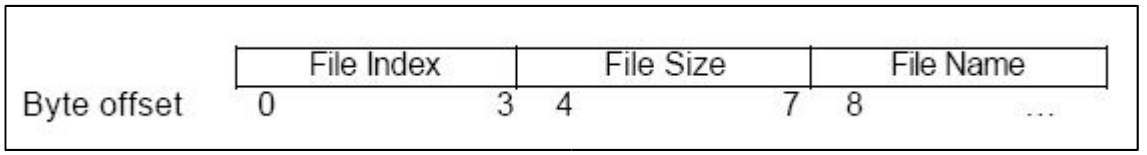
Port: porta per le connessioni in ingresso

IP Address: indirizzo IP di chi risponde

Speed: la velocità (Kb/s) della connessione del peer

Servent Identifier: identifica univocamente il peer nella rete (è necessario per l'operazione di PUSH)

Result Set: contiene i risultati della ricerca, formattati come segue



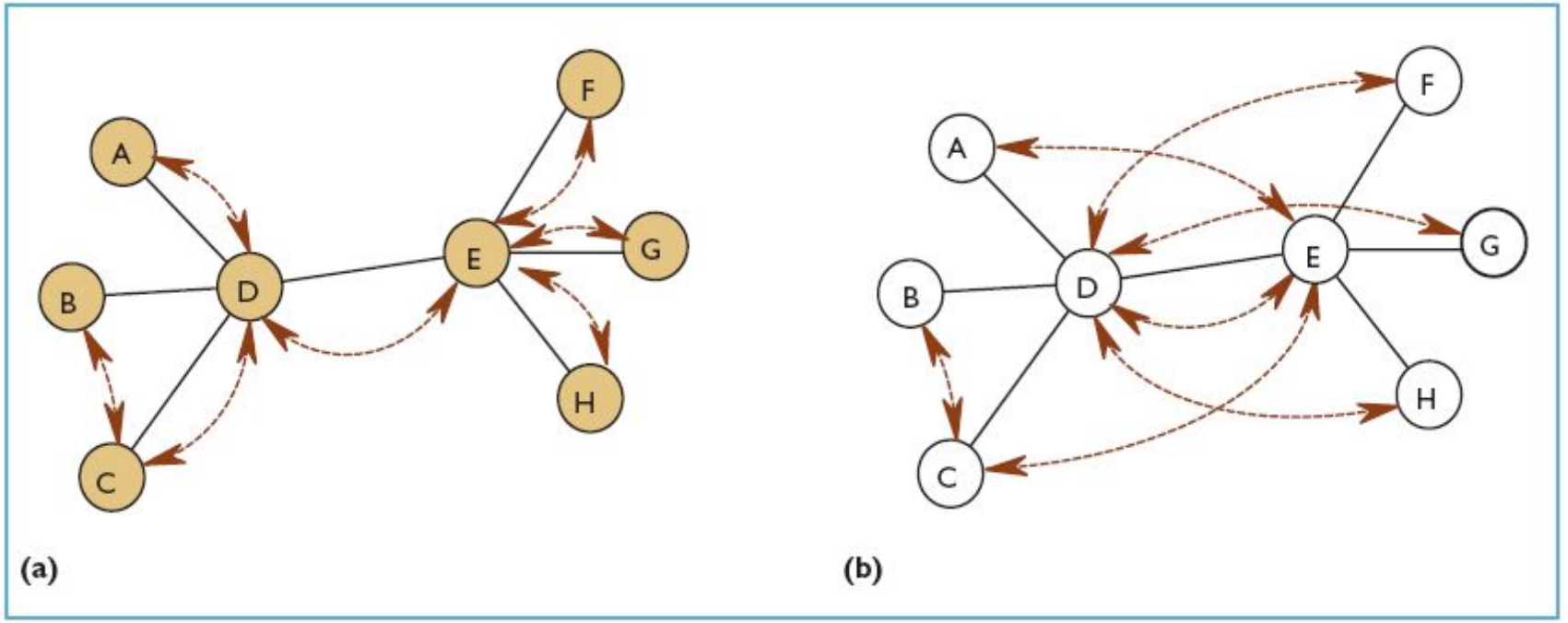
File Index: identificatore locale del file

Download del file

- Il download del file avviene in modalità **out-of-network**
- Si stabilisce una connessione diretta tra i due peer (è iniziata da chi riceve il messaggio Query-Hit)
- Il protocollo utilizzato è **HTTP**
- Supponiamo che il peer che possiede il file non accetti connessioni in ingresso (perché protetto da firewall)
- Il peer che tenta il download allora può mandare un messaggio di PUSH, invitando il peer remoto ad iniziare al posto suo la connessione TCP

Limitazioni delle reti non strutturate

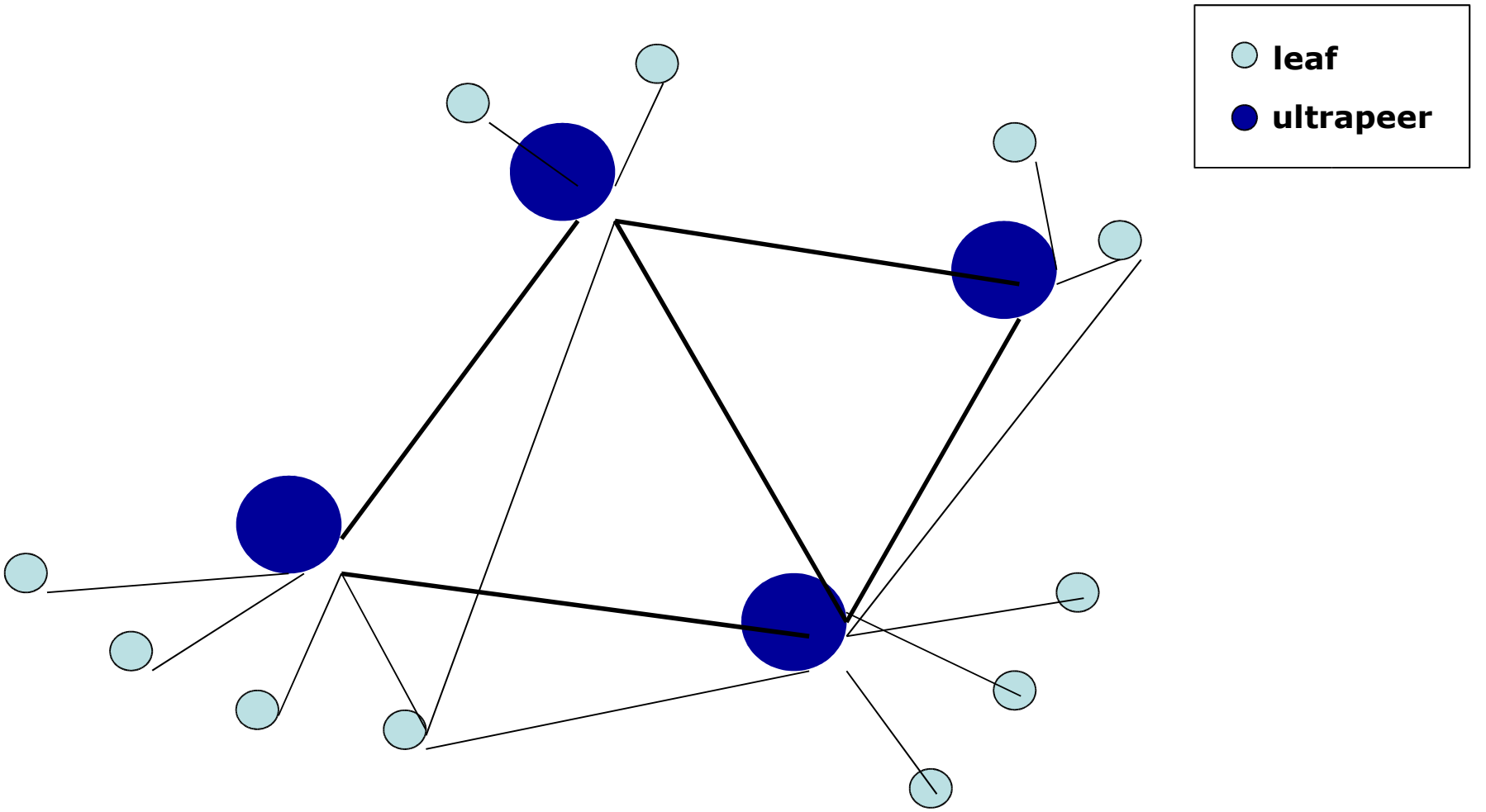
- Non è garantito il determinismo della ricerca
  - Non si ha una visione globale del sistema
  - Hop count limitato
- La fase di mantenimento e di ricerca del file consumano molta larghezza di banda



I sistemi ibridi

- Compromesso tra il determinismo del modello centralizzato e la scalabilità del sistema puro
- L'overlay è l'interconnessione di **cluster**
- Si cerca di limitare la ricerca all'interno di un cluster; eventualmente è diffusa all'esterno
- Alcuni sistemi ibridi: FastTrack, E-Donkey, DirectConnect, Gnutella2, etc...

Gnutella 2



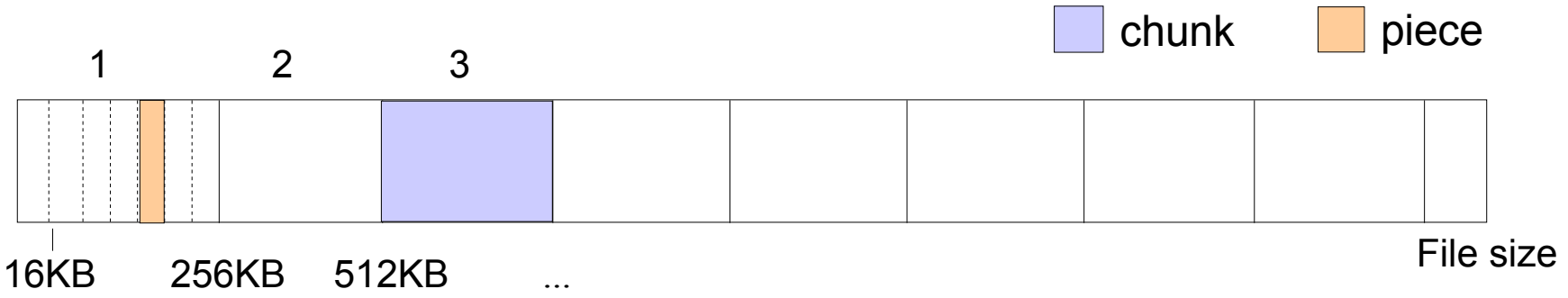
1) Non sono possibili connessioni leaf-leaf
2) Un leaf può connettersi anche a più ultrapeer

Gnutella 2

- Elezione **autonoma** dello stato del peer
  - Caratteristiche hardware
  - Configurazione della rete
  - Profilo dell'utente (tempo di vita nella rete)
- **Query Routing Protocol**
  - Si occupa dell'instradamento dei messaggi
  - Ogni leaf manda le query all'ultrapeer
  - L'ultrapeer inoltra le query solo ai leef connessi che si presume posseggano le risorse

Sistemi P2P per la distribuzione di un file: BitTorrent

- La ricerca avviene in modo **out-of-band**:
  - Web, mail, messaggistica istantanea, etc.
- Si costruisce un'overlay network **per ogni file** condiviso:
  - Il file è suddiviso logicamente in piccole unità scaricabili **parallelamente**
  - I peer condividono i chunk che già posseggono



Architettura di BitTorrent (BT)

- **Il file torrent**
  - E' generato da chi pubblica la risorsa
  - Contiene meta-informazioni per la localizzazione e l'accesso nell'overlay del file desiderato
- **Il tracker**
  - Servizio centralizzato dell'overlay per il bootstrapping del nodo entrante
  - Assegna una lista di peer presenti nella rete
- **Il client BitTorrent**
  - Implementa il protocollo BitTorrent

Terminologia BT

- **Swarm** (o torrent): insieme dei peer presenti nella rete
- **Seed**: peer che posseggono già la copia completa del file
- **Leecher**: peer che devono ancora completare il download
- **Share ratio**: rapporto tra volumi di traffico in upload e in download

Il file torrent

- Si ottiene con attività out-of-band
- Lo strumento più utilizzato è il Web
- Es. linuxtracker.org

Browse Torrents

Monday December 05, 2005.

Categories

Distro	Name	Added	Type	Rating	Size	Files	Complete	Seeders	Leechers
	VectorLinux 5.1 SOHO beta 2	2005-12-05 06:43:35	LOCAL	---	697.59 MB	2	0	1	3
	MEPISLite-3.3.2-1.rc1.iso	2005-12-05 04:53:15	LOCAL	---	626.36 MB	1	0	1	2
	dyne:bolic 2.0 beta 5	2005-12-05 03:48:57	LOCAL	---	425.49 MB	2	4	3	1
	Zenwalk Linux 2.0.1	2005-12-04 23:33:42	LOCAL	---	477.13 MB	1	0	0	0
	Linux From Scratch LiveCD 6.1.1-1 (x86)	2005-12-04 21:32:13	EXT	---	390.05 MB	1	6	6	1
	Kurumin 5.1 beta1	2005-12-04 20:03:48	LOCAL	---	464.62 MB	1	0	0	0
	Kurumin 5.0	2005-12-04 20:02:45	LOCAL	---	420.32 MB	1	0	0	0
	Scientific Linux 4.2 for x86_64 (4 CDs)	2005-12-04 10:03:25	LOCAL	---	2.48 GB	5	0	0	0
	Scientific Linux 4.2 for i386 (4 CDs)	2005-12-04 10:02:04	LOCAL	---	2.24 GB	5	0	1	2
Linux Live CDs	limp ver 1.2	2005-12-03 23:17:22	LOCAL	---	49.54 MB	1	117	6	1

Il file torrent

- E' il file d'ingresso di un client BitTorrent
- Il peer conosce le seguenti informazioni:

```
file: Gentoo-RR4-2.65.1.iso.torrent

info hash: 3f1e120b3ef2f29b354bb4d1608dbcac43708841

file name.....: Gentoo-RR4-2.65.1.iso

file size.....: 2568243200 (1224 * 2097152 + 1329152)

announce url..: http://linuxtracker.org/announce.php
```

Il tracker

- E' un server che risponde a messaggi HTTP
- Consente il bootstrap di un nuovo nodo
- Può gestire contemporaneamente diversi torrent
- Opzionalmente supporta il servizio di pagine Web che contengono informazioni statistiche

Il tracker

BitTorrent download info

- **tracker version:** 4.0.0
- **server time:** 2005-12-05 12:43 UTC

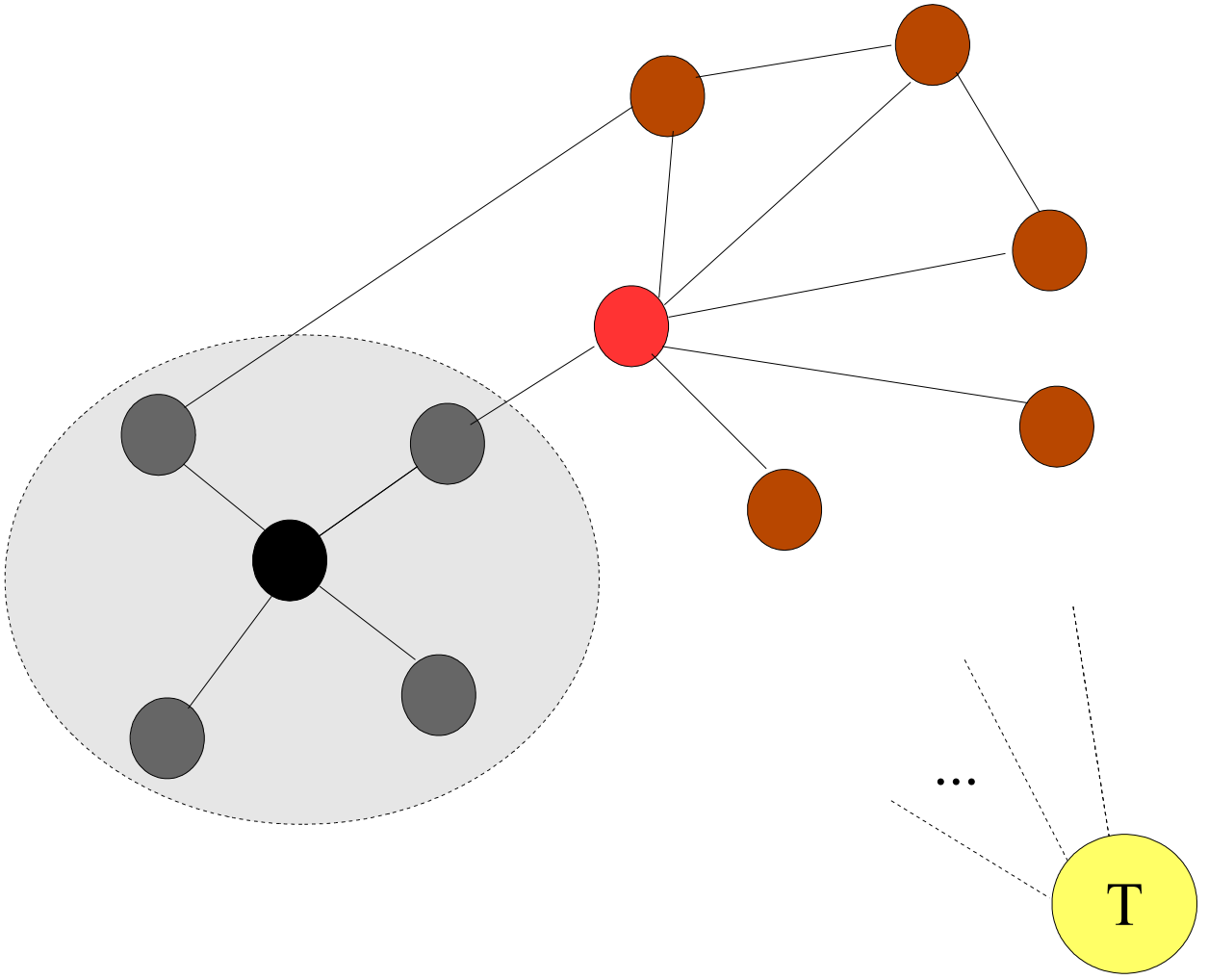
info hash	torrent name	size	complete	downloading	downloaded	transferred
5f5964c1727e071fcada25893a7aff6104137599	lunar-1.4.0.iso.bz2	113MiB	25	48	29	3.22GiB
0bd7218a85f8ac4e971cc432c7c8818e50a29c08	lunar-1.5.0-i686.iso.bz2	241MiB	248	469	337	79.44GiB
a547f6eb423e583454dc1b9f05890e93ac76c88f	lunar-1.5.1-i386.iso.bz2	204MiB	64	25	15	2.99GiB
d9fd544b446736913e78b6614e06c5844c841ce7	lunar-1.5.1-i686.iso.bz2	234MiB	100	108	41	9.4GiB
7f8862bbefe045ada89e6b256d324af3aaa0f0f6	xfce-fosdem2005-ofourdan.avi	374MiB	5	2	5	1.82GiB
5bf88cb08bcb34b7a7bb7f7b2025bdbc2a13f105	xfld-0_2-de.iso	641MiB	46	243	14	8.76GiB
4de523967f8900b2f0497c31bca8b47647256a1b	xfld-0_2-en.iso	641MiB	472	706	329	206.0GiB
	7 files	2.39GiB	960	1601	770/770	311.66GiB

- *info hash:* SHA1 hash of the "info" section of the metainfo (*.torrent)
- *complete:* number of connected clients with the complete file
- *downloading:* number of connected clients still downloading
- *downloaded:* reported complete downloads (total: current/all)
- *transferred:* torrent size * total downloaded (does not include partial transfers)

Il protocollo BT

- Ogni nodo riceve una lista di peer (ip, port) già presenti nell'overlay e stabilisce le relative connessioni P2P (su TCP)
  - L'insieme delle connessioni è il **peer-set**
  - La dimensione del peer-set è un parametro **specificato dall'utente**
  - Il peer-set può essere un sottoinsieme del torrent!
- Meccanismo di **hand-shaking**
  - Scambio del messaggio **bitfield** per la conoscenza dei chunk del peer remoto

Topologia di una rete BT

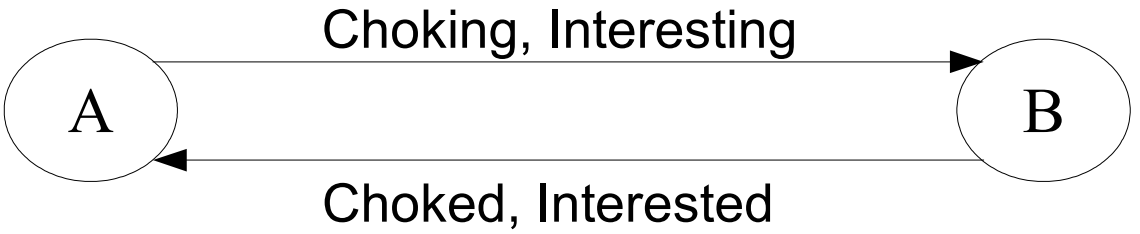


Lo stato logico della connessione BT

- Ogni peer associa **quattro** variabili di stato booleane ad ogni connessione:
  - **Choking**: il peer blocca l'upload verso il peer remoto
  - **Choked**: il peer non può scaricare dal peer remoto
  - **Interesting**: il peer possiede chunk che il peer remoto non ha
  - **Interested**: il peer remoto possiede chunk che il peer non ha

Lo stato della connessione

- Inizialmente, la connessione si assume bloccata in entrambi i sensi:
  - **Choked = Choking = True**
  - **Interested = Interesting = False**



I messaggi del protocollo

- **Have**
  - annuncia al peer-set la disponibilità di un chunk
- **Choke**
  - Blocca l'upload
- **Unchoke**
  - Sblocca l'upload
- **Interested**
  - Segnala l'interesse verso qualche chunk del peer remoto
- **Not Interested**
  - Comunica la perdita di interesse verso il peer remoto

I messaggi del protocollo

- **Request**
  - Richiede un segmento (16KB) di chunk (256KB) al peer remoto
- **Piece**
  - Trasmette la section indicata
- **Cancel**
  - Annulla la richiesta di un chunk

Esempio

Esempio

A si connette a B e manda il bitfield

Esempio

B risponde

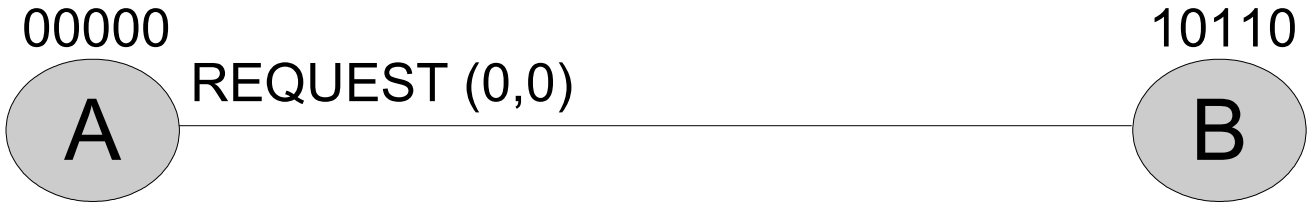
Esempio

A è interessato ai chunk di B

Esempio

B sblocca la connessione

Esempio



A richiede il segmento 0 del chunk 0

Esempio

B lo trasmette

Esempio

... A ottiene TUTTI i segmenti...

Esempio

A completa il chunk

Strategie per l'upload

- La suddivisione in chunk del file consente il **trasferimento parallelo**
  - Il download può avvenire da un **numero illimitato** di peer
  - L'upload è consentito verso un **numero prefissato** di peer N (N = 4 di default)
- Algoritmo di **unchoking**
  - Implementa una strategia **tit-for-tat** basata sul rate di upload nella connessione
  - Ogni 10 secondi si scelgono gli N-1 peer che stanno fornendo dati più velocemente

Strategie per l'upload

- **Optimistic unchoking**
  - Ogni tre turni (30 secondi) un peer rimanente è sbloccato **indipendentemente** dalla sua velocità di upload
  - Il peer è scelto in modo **casuale**
  - Consente un'esplorazione del vicinato per la ricerca di peer più veloci
  - Consente il download ai nuovi peer entrati

Strategie per la scelta dei chunk

- **Local Rarest First Policy**
  - Ogni peer cerca di scaricare il chunk più raro del suo peer set
  - Aumenta la probabilità che ogni peer sia sempre interessante
  - I primi chunk sono scaricati senza rispettare questa regola (per ridurre i tempi di inizio del download)
- **Strict Policy**
  - Il chunk deve essere scaricato completamente prima di cominciarne un'altro

Una variante per il seed

- Quando un peer termina il download, contatta il tracker
- Gli viene assegnata una lista di soli leecher
- Il seed non può applicare una strategia tit-for-tat
- Mantiene l'optimistic unchoking
- Ogni 10 secondi seleziona i **più veloci downloader**

Conclusioni su BitTorrent

- La distribuzione del file avviene molto efficacemente
  - Elevata utilizzazione della capacità di upload di ogni peer
- La ricerca della risorsa è fortemente centralizzata
  - La maggior parte dei servizi Web dedicati sono stati chiusi perché illegali

Conclusioni su BitTorrent

- Il tracker è l'elemento centrale dell'overlay
  - La sua disponibilità influenza solo il boot-strapping del nodo
  - I peer già entrati nella rete possono continuare il download
- Due alternative per migliorare le prestazioni
  - Messaggi tracker-peer su UDP
  - **Distributed Hash Table**

I sistemi P2P strutturati

Efficienza degli algoritmi

- Misure naturali di bontà
  - Spazio utilizzato
  - Tempo di esecuzione richiesto
- Approccio empirico (a posteriori)
  - Esecuzione dell'algoritmo per la valutazione di
    - Tempo medio
    - Tempo migliore
    - Tempo peggiore

Efficienza degli algoritmi

ordinamento di un vettore

| 1 | 20 | 5 | 7 | 2 | 15 | 3 | 16 |

- Il tempo di esecuzione dipende da diversi fattori:
  - Linguaggio di programmazione
  - Compilatore
  - Architettura del calcolatore
  - Distribuzione degli ingressi
  - **Dimensione dell'ingresso**

Efficienza dell'algoritmo

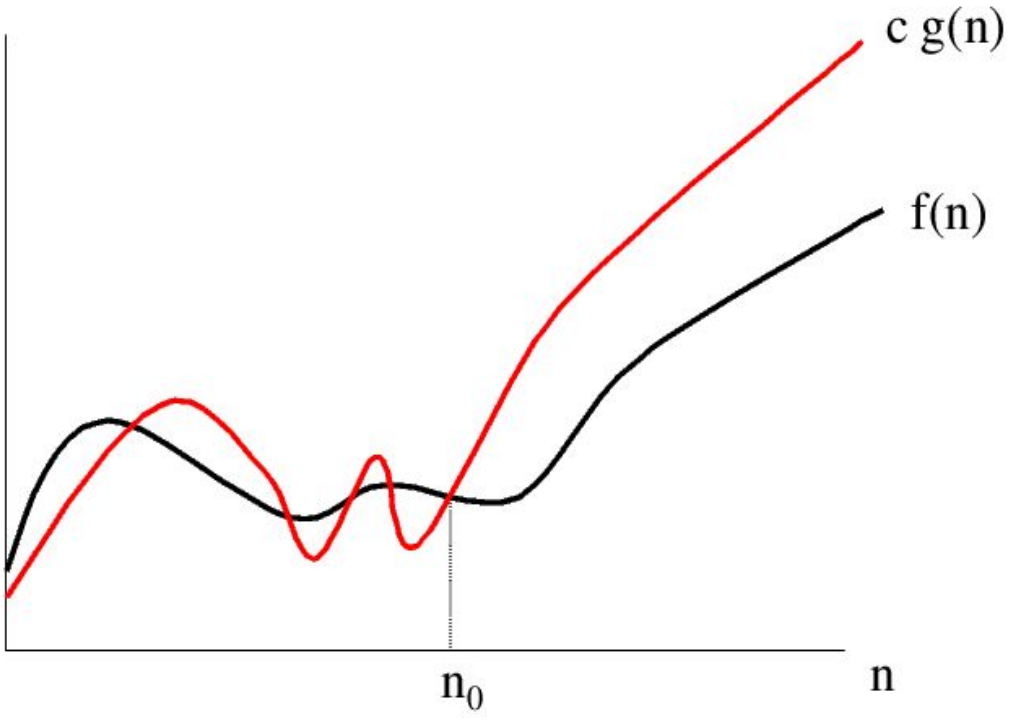
- Siamo interessati agli aspetti che **dipendono dal particolare algoritmo scelto**
- Valutare la funzione **T(n)** che fornisce la complessità computazionale dell'algoritmo quando la dimensione dell'input è **n**
- Approccio teorico (a priori)
  - Macchina di riferimento
  - Ogni istruzione semplice ha **costo unitario**

Notazione asintotica

- Non ci riferiamo alla valutazione delle prestazioni per una dimensione prefissata degli ingressi
- Analisi al limite:
  - Comportamento dell'algoritmo per un **n** sufficientemente grande
- Tre notazioni
  - O, Ω, Θ

La notazione O-grande

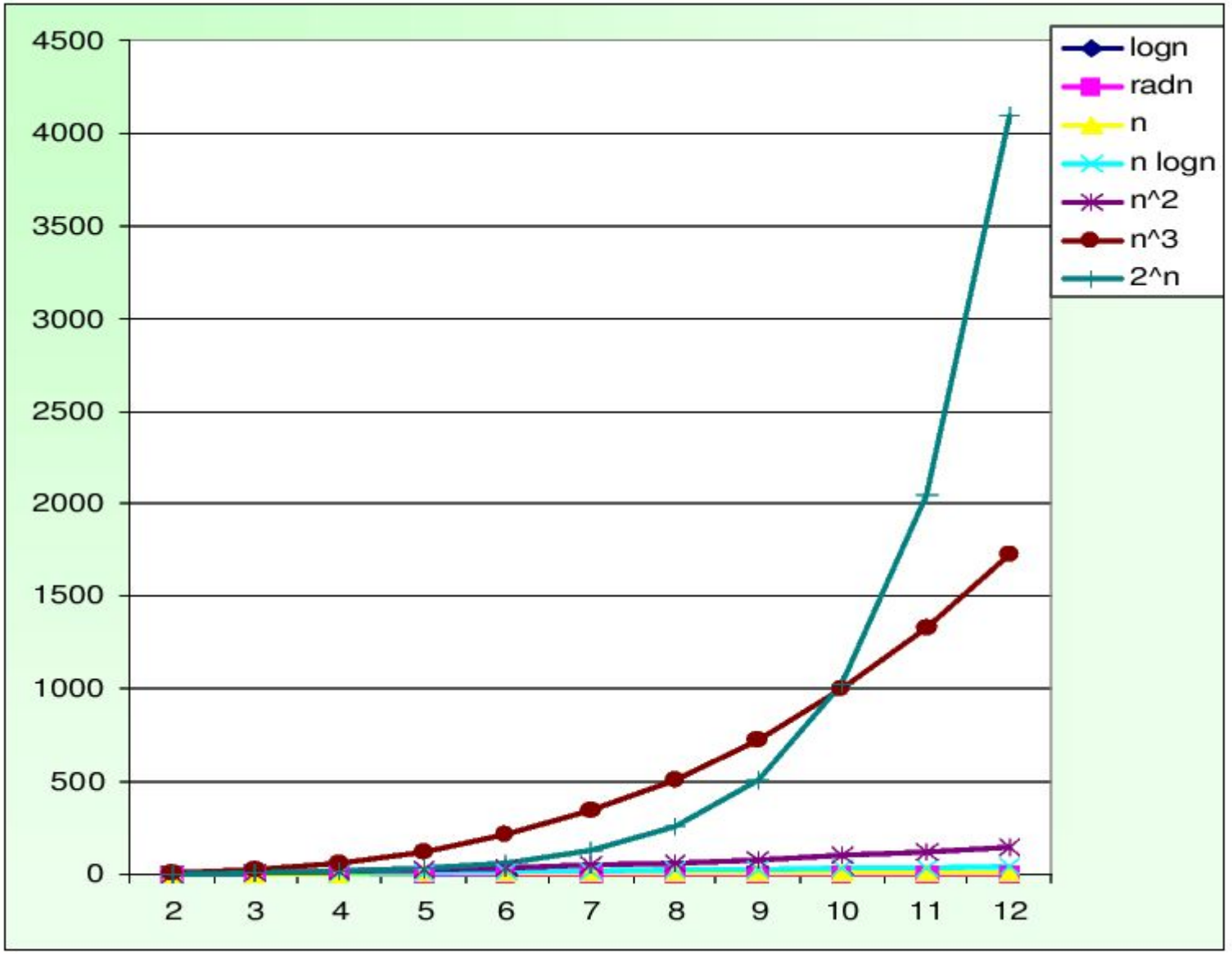
- Fornisce un **limite superiore** alla complessità computazionale di un algoritmo
- Una funzione f(n) appartiene alla classe O(g(n)) se esistono c, n_0: f(n) < c·g(n) per ogni $n > n_0$



Velocità di crescita

n	log n	$\sqrt{n}$	n	n log n	n^2	n^3	2^n
2	1	1,41	2	2	4	8	4
4	2	2	4	8	16	64	16
8	3	2,83	8	24	64	512	256
16	4	4	16	64	256	4.096	65.536
32	5	5,66	32	160	1.024	32.768	4.294.967.296
64	6	8	64	384	4.096	262.144	$1,84 \times 10^{19}$
128	7	11,31	128	896	16.384	2.097.152	$3,40 \times 10^{38}$
256	8	16	256	2.048	65.536	16.777.216	$1,15 \times 10^{77}$
512	9	22,63	512	4.608	262.144	134.217.728	$1,34 \times 10^{154}$
1.024	10	32	1.024	10.240	1.048.576	1.073.741.824	$1,79 \times 10^{308}$

Scalabilità di un algoritmo



Le funzioni hash

- Una funzione hash **f()** è una funzione molti-a-uno che mappa i valori in ingresso in valori appartenenti ad un insieme finito
- Il codominio di una funzione hash è un sottoinsieme dei numeri naturali
- Esempio
  - f(x) = 0
  - f(x) = x mod 5, f(x) in (0,1,2,3,4)

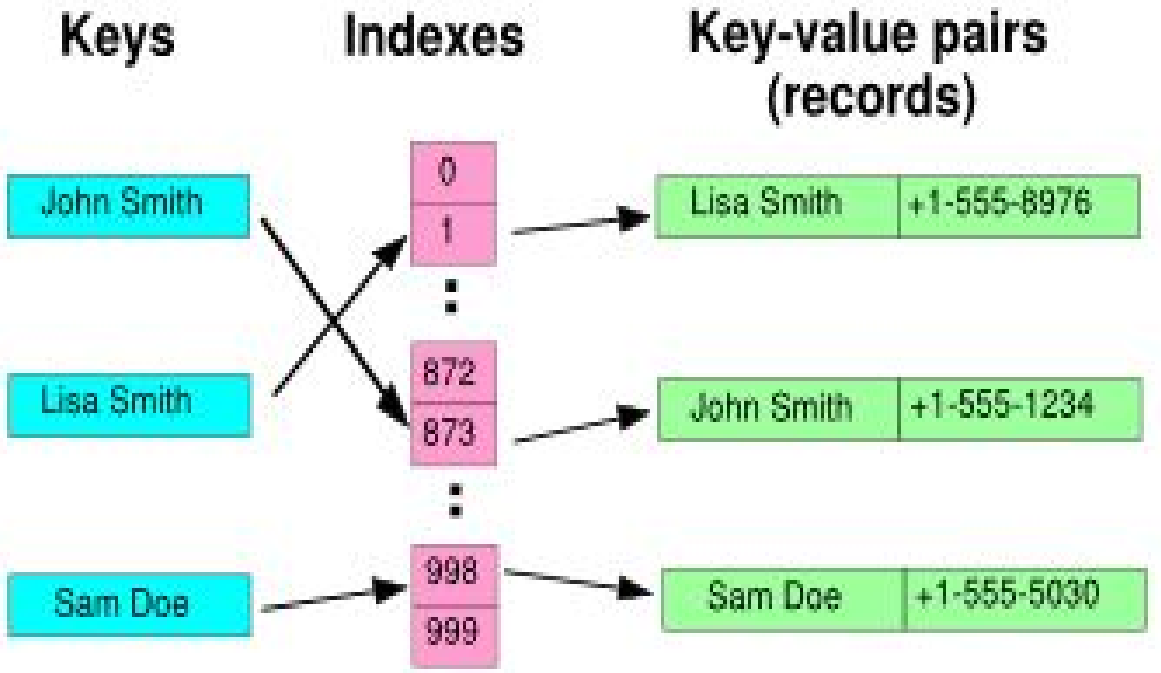
Le tabelle hash

- Sono strutture dati che supporta un meccanismo di ricerca efficiente (O(1)) per coppie **(chiave,valore)**
- Possono implementare memorie cache, array associativi, insiemi, etc...
- Supportano due funzioni:
  - insert(key, value)
  - lookup(key)

Le tabelle hash

- Usano un array capace di memorizzare m **record** (chiave,valore)
- Implementazione di **lookup(k)**
  - Si accede agli elementi di un array **A** con un indice i in [0, m-1]
  - Data la chiave **k**, la funzione hash **h()** calcola **i** = h(k)
  - Il record richiesto è **A[i]**
- Implementazione di **insert(k,v)**
  - **A[h(k)] = v**

Esempio

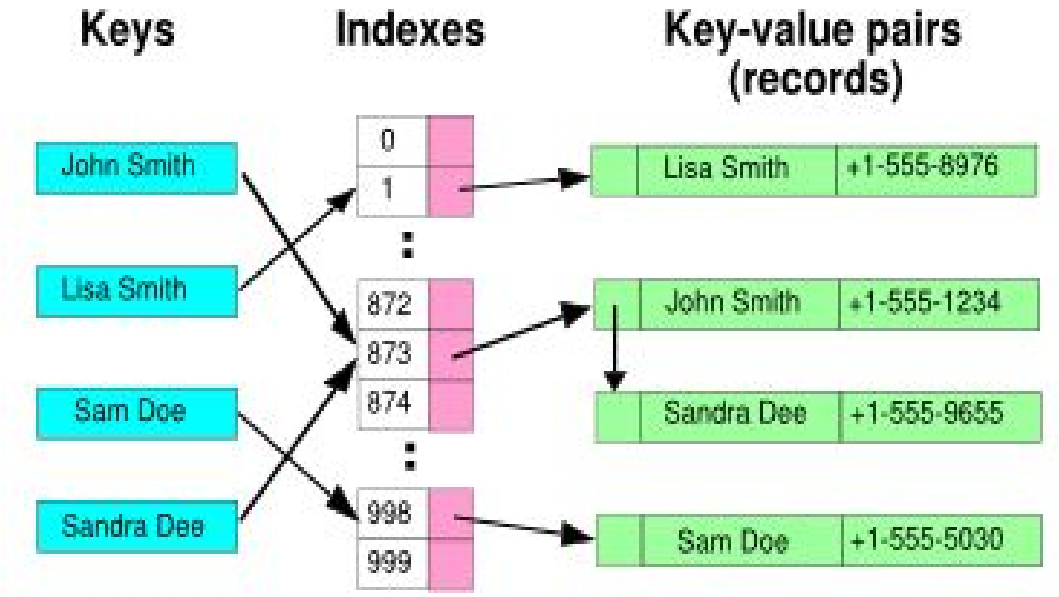


- **Elenco telefonico**
- Array di 1000 elementi
- Funzione hash:
  (somma dei valori ASCII dei caratteri) % 1000

Gestione delle collisioni

- Due chiavi **diverse** con lo stesso **hash**
  - Es: h(x) = x % 5 -> 10,15,20 sono chiavi che collidono
- Gestione della collisione:
  - Ogni indice (bucket) dell'array contiene una **lista concatenata di record**
  - Scansione lineare della lista

Gestione delle collisioni



Liste concatenate

- Prevenzione delle collisioni:
  - Utilizzare funzioni hash “buone”
  - Allargare l'array quando necessario

Message Digest

- **MD** è una funzione hash utilizzata per l'autenticazione dei messaggi
- Gode di quattro proprietà
  - Dato il messaggio P, è facile calcolare MD(P)
  - Dato MD(P) è impossibile ottenere P
  - Dato P, nessuno è in grado di trovare P' tale che MD(P) = MD(P')
  - Se l'input cambia di 1 solo bit, l'output è completamente diverso

SHA

- **S**ecure **H**ash **A**lgorithm mappa gli ingressi (di qualunque dimensione) in valori di uscita a 160 bit
- L'input è partizionato in blocchi di 512 bit ciascuno
  - Zero padding per ingressi non multipli di 512

```
$ echo sha | sha1sum

4a0c3544feaab0c75ba8623f66f9dbe91cb443e7
```

Le Hash Table Distribuite (DHT)

- Si forma un'overlay network in cui ogni nodo è responsabile della memorizzazione di un sottoinsieme di coppie <chiave, valore>
- La rete peer-to-peer è **strutturata**
  - La topologia è controllata
  - I messaggi del protocollo sono instradati a nodi sempre più **vicini** al nodo che conserva il record
- Sistemi P2P di **seconda generazione**

I servizi delle DHT

- I servizi fondamentali offerti dalle DHT sono due:
  - Lookup(chiave)
  - Insert(chiave, valore)
- Le applicazioni possono essere costruite usando le DHT come **middleware**
  - File-sharing (chiave = titolo, valore = indirizzo del peer)

Proprietà delle DHT

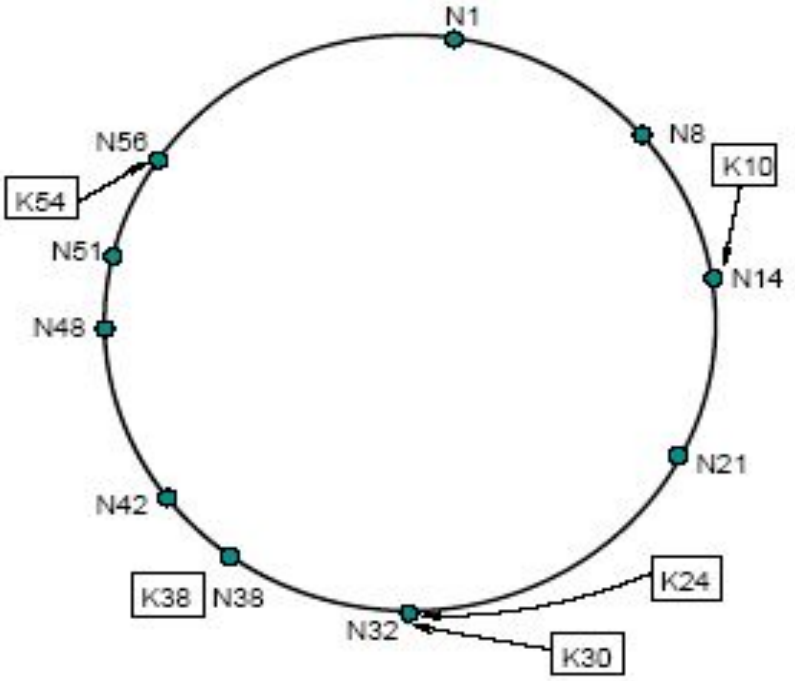
- **Load balance**: si utilizzano funzioni hash che distribuiscono uniformemente i record presso i nodi (bucket) della rete
- **Decentralization**: tutte le operazioni sono completamente distribuite; non c'è nessun nodo gerarchicamente superiore agli altri
- **Scalability**: il numero di nodi attraversati dai messaggi è O(logN)
- **Availability**: gestione automatica delle tabelle di routing dopo ingressi/uscite dei nodi

Chord

- DHT sviluppata nel 2001 in MIT e Berkeley
- I nodi e le chiavi condividono lo stesso **spazio degli identificatori**: consistent hashing **SHA-1** (160 bit)
  - L'id di un nodo è ottenuto dall'hash dell'indirizzo IP
  - La chiave di un record è ottenuta dall'hash della chiave originale

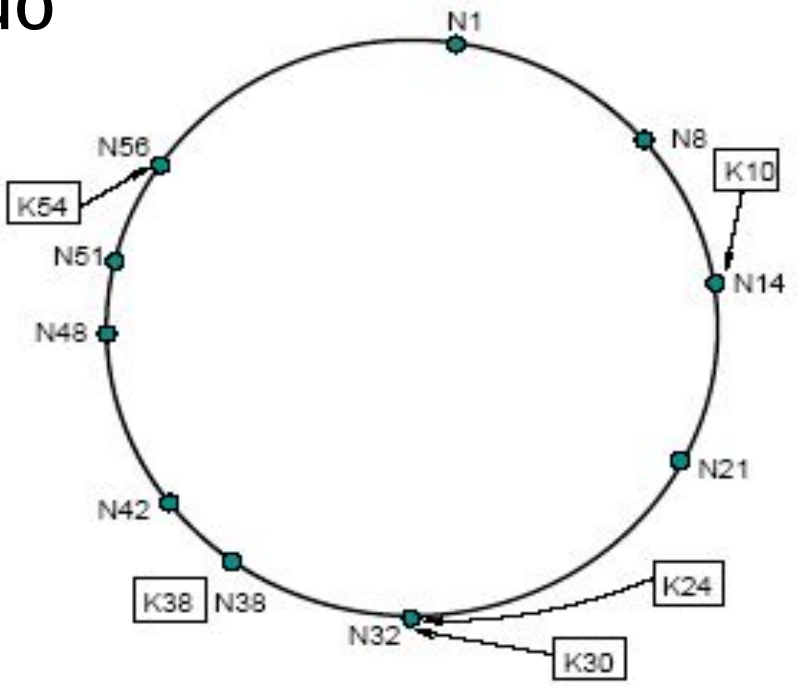
Topologia della rete

- I 2^{160} possibili identificatori sono organizzati in ordine crescente in un **cerchio** (modulo 2^{160})
- Una chiave **k** è assegnata al primo nodo il cui identificatore è uguale o segue **k**
  - **successor(k)**
- L'hashing consistente assegna le chiavi in modo uniforme ai nodi esistenti



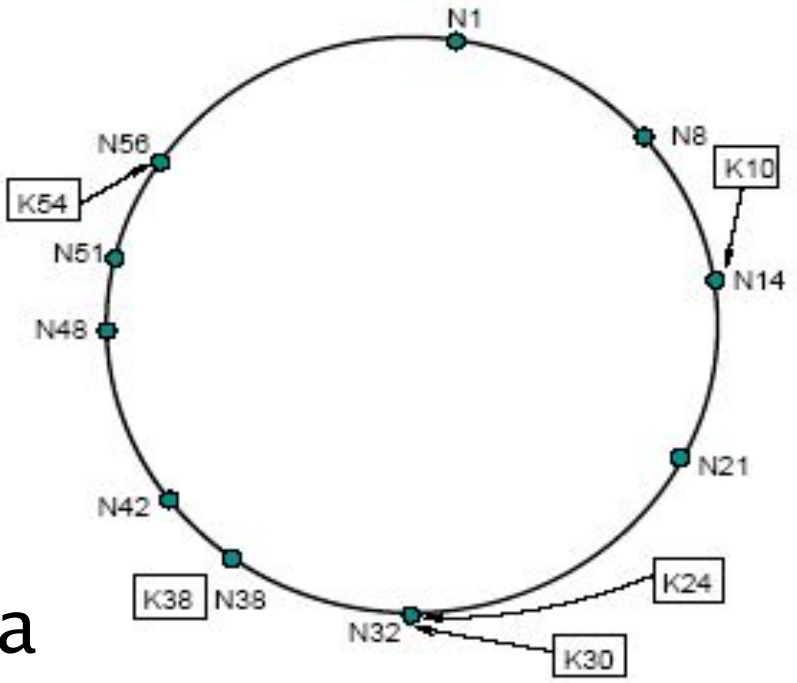
Routing in Chord

- Il nodo N1 richiede il lookup(K30)
- Il routing è possibile se ogni nodo Nx conosce l'indirizzo del suo successore **successor(Nx)**
- Dopo 4 hop il messaggio raggiunge
  N32 = successor(K30)
- Il pacchetto di lookup contiene l'indirizzo IP di N1
- Il record <chiave, valore> è trasferito direttamente a N1



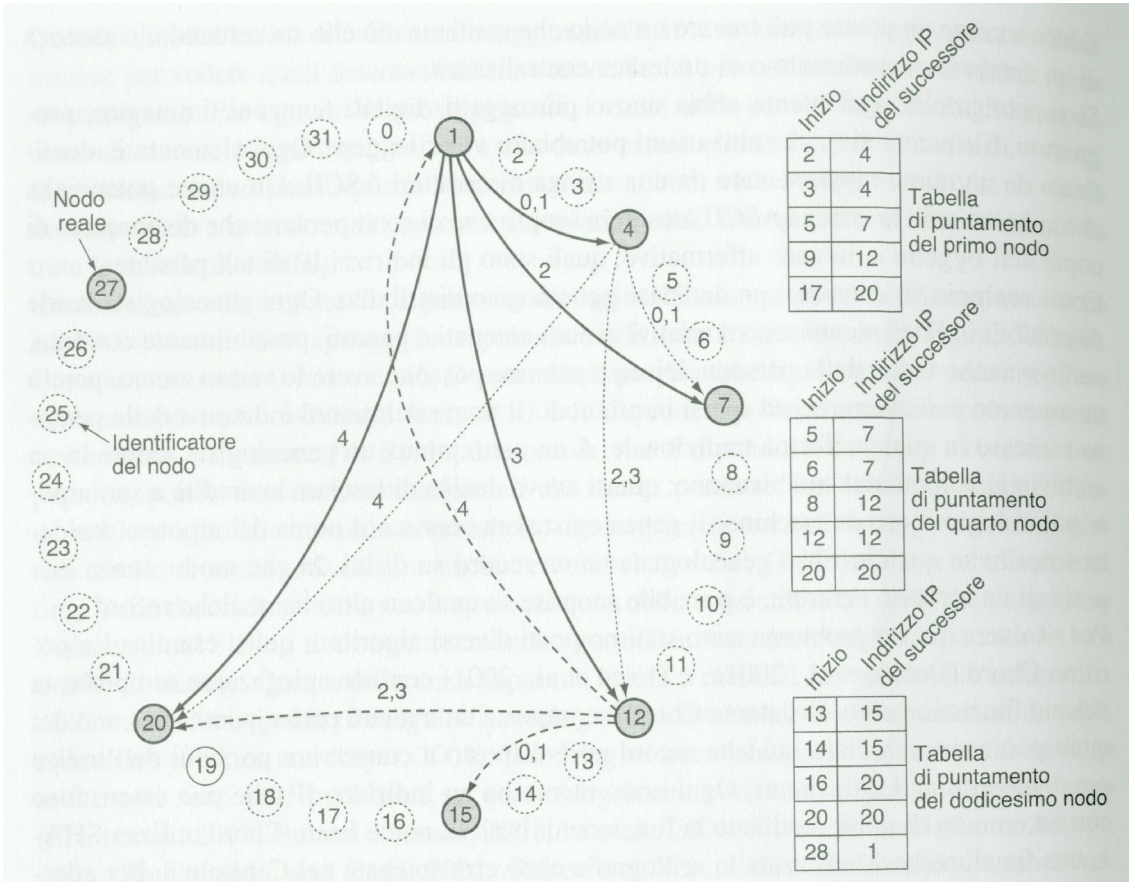
Miglioramento delle prestazioni

- Il routing ha complessità **O(N)**
  - La scansione dei nodi è lineare
  - E.g. N8 richiede il lookup(K1)
- La dimensione della tabella di routing è **O(1)**
  - È necessario conservare l'indirizzo del proprio successor
- E' possibile migliorare l'efficienza del numero di messaggi di routing
  - Aumentando la dimensione delle tabelle



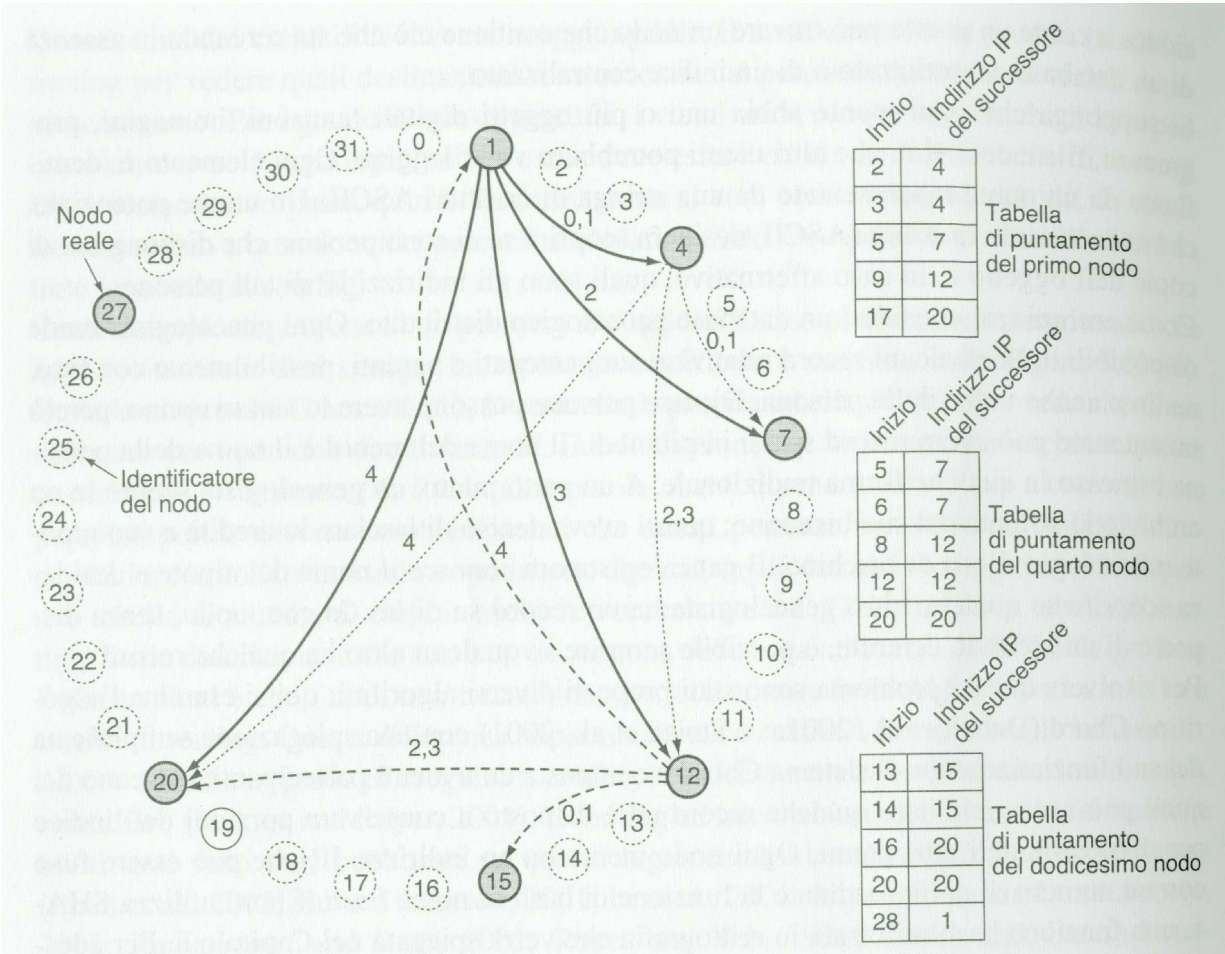
Finger Table

- Ogni nodo **k** conserva una **tabella di puntamento** di 160 voci (da 0 a 159)
- L'entry di posizione **i** voce ha due campi
  - Start = $k + 2^i$ (modulo 2^{160})
  - Indirizzo IP di successor(start)



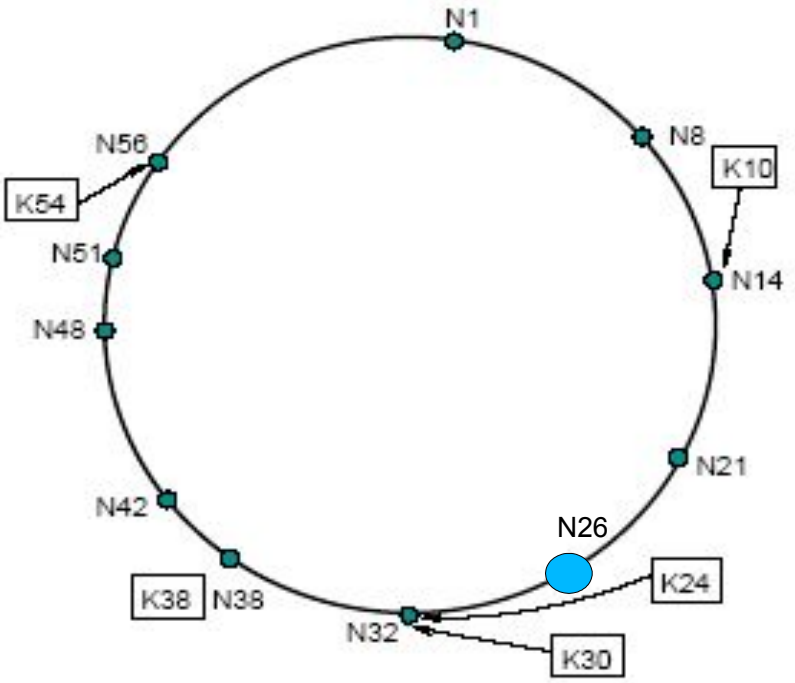
Routing Ottimizzato

- Un nodo **N** ricerca la chiave **k**.
  - Se k è compresa tra N e successor(N), successor(N) è responsabile per k
  - Altrimenti, la richiesta è instradata all'indirizzo IP corrispondente al più grande valore di **start** inferiore a **k**
- Ricerca effettuata in O(log N) passi



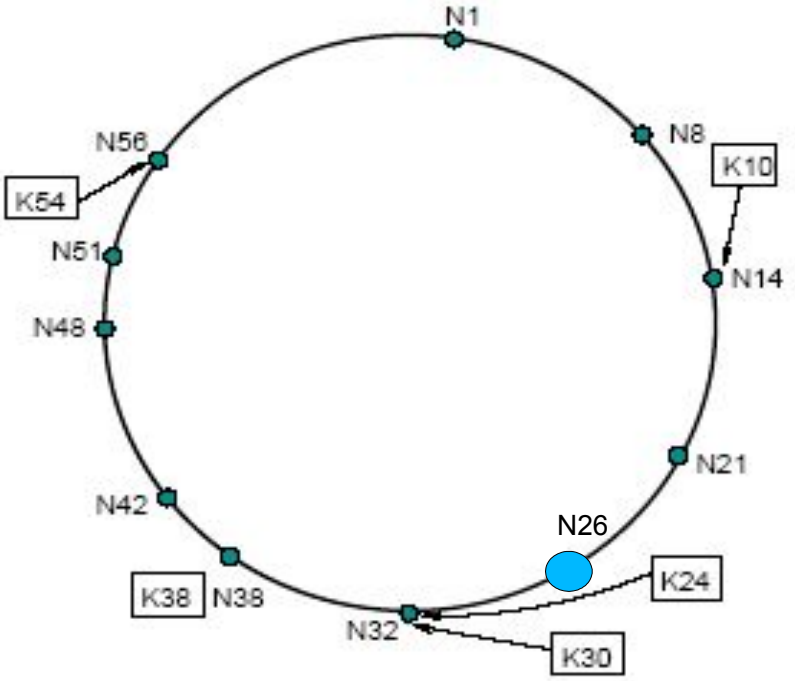
Ingresso di un nodo

- N26 vuole accedere alla rete
- Per il **boot-strapping** è necessario l'indirizzo di un qualunque nodo (e.g. N8)
  - A quel nodo sarà instradato il lookup(N26), che fornirà l'indirizzo IP del successore (N32)
  - N26 chiederà a N32 l'indirizzo del suo attuale **predecessore** (N21)
  - N21 userà N26 come successore
  - N26 userà N32 come successore
  - N26 **sarà responsabile per K24**



Uscita di un nodo

- N26 vuole uscire dalla rete
- Se l'uscita è regolare
  - N26 consegna i suoi record a N32
  - Informa dell'uscita N21
  - N21 userà N32 come successore
- Altrimenti,
  - N21 non avrà un successore valido!
  - Ogni nodo conserva **s** successori per ripristinare il cerchio



Altre DHT...

- **Kadmelia**
  - Usa lo stesso spazio degli identificatori di Chord
  - La nozione di distanza è basata sulla metrica dello **XOR**
  - Un'implementazione di DHT Kademlia-like è presente nella **versione trackerless** del mainline BitTorrent (www.bittorrent.com)

Altre DHT...

- **Tapestry**
  - Lo spazio degli identificatori a **m** bit (m=160) viene considerato in base 2^b (b = 4)
  - L'identificatore è una sequenza di 40 cifre esadecimali
  - L'algoritmo di routing è di tipo **longest prefix matching**: la query di lookup è instrada verso nodi che condividono un numero di cifre sempre maggiore dell'identificatore richiesto

4* -> 4A* -> 4AE* -> 4AEF

Conclusioni

- **Proximity Routing**: due nodi **vicini** geograficamente (nell'underlying network) possono essere mappati con identificati **lontani** (secondo la metrica dell'overlay)
  - Due nodi di una stessa sottorete possono avere differenze di pochi bit negli indirizzi IP
  - Il consistent hashing produce output completamente diversi
  - I nodi possono comunicare attraversando un elevato numero di nodi fisici
  - Aumentano le latenze dell'applicazione

Conclusioni

- **Exact Matching**: sotto ipotesi di stabilità della rete, le DHT garantiscono il lookup di un record in modo deterministico
  - L'identificatore deve essere esattamente specificato!
  - Alcuni sistemi P2P (e.g., file-sharing) supportano una ricerca basata sulle **parole chiave** che consente di ottenere dati anche se la risorsa richiesta non è specificata esattamente

Keyword searching con le DHT

- La stringa di ricerca viene suddivisa in **token**
  - Es: la ricerca di “Pink Floyd” produce i token <pink> e <floyd>
- Ad ogni token è applicata la funzione hash
  - **pink**
    `8c39bc628eaa9e6efe4481ebaf96914dccaba037`
  - **floyd**
    `f5d969487c86a48844f39eb315c42f28638ec528`
- Le chiavi vengono usate per memorizzare il **valore** (titolo, indirizzo ip) nella DHT

Esempio: BitTorrent “decentralizzato”

- La versione originale di BitTorrent prevede il tracker
- Nel Maggio 2005 è stata annunciata la versione trackerless
  - Una DHT basata su **Kademlia**
    - La chiave è l'hash SHA1 di informazioni contenute nel file torrent
    - Il valore è la lista dei peer che partecipano al torrent

Esempio: BitTorrent “decentralizzato”

- Un peer **P1** accede alla DHT usando un altro peer già connesso per il boot-strapping
- Ottiene (o genera) il file torrent:
  - key = SHA1(torrent_file.infohash)
  - Peer List = DHT.lookup(key)
  - DHT.insert(key, P1)
- Se è il seed iniziale la Peer List è inizialmente vuota!