Lezione n.2
RETI NON STRUTTURATE
GNUTELLA 0.4

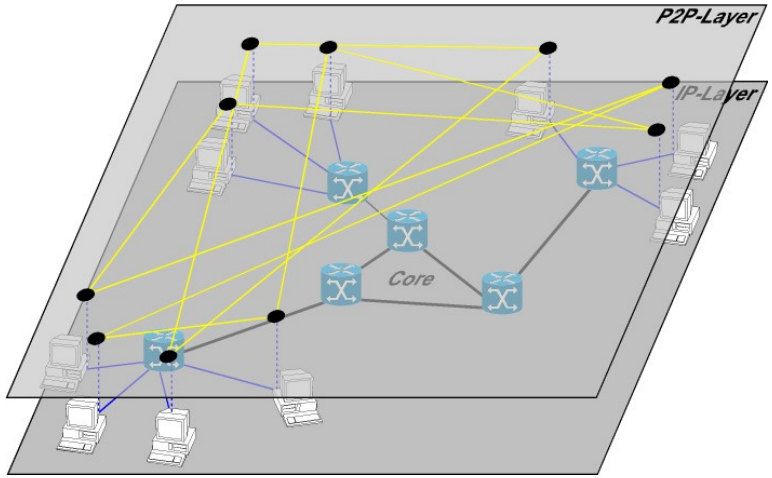
http://rfc-gnutella.sourceforge.net/developer/index.html

Laura Ricci

27-9-2013

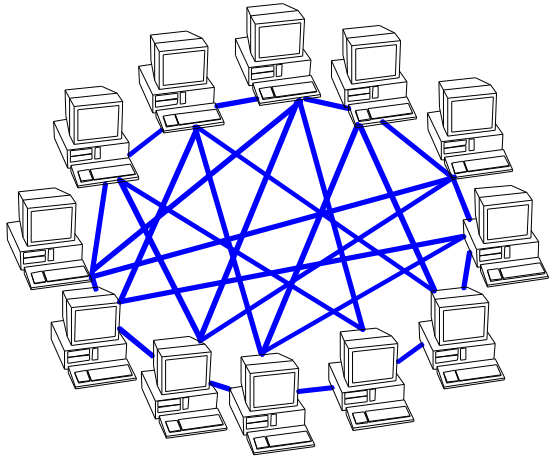
RIASSUNTO DELLA PRESENTAZIONE

1. Caratteristiche generali dei sistemi P2P di prima generazione
2. Reti P2P centralizzate
   - Caratteristiche Base
   - Protocollo
   - Dicussione
3. Reti Peer to Peer Pure, Overlay non strutturati
   - Caratteristiche Base
   - Protocollo
   - Discussione
4. Reti Peer to Peer Ibride, Overlay non Strutturati
   1. Caratteristiche Base
   2. Protocollo
   3. Discussione



OVERLAY NON STRUTTURATI

- Non esiste una entità centralizzata nella overlay network
- La rimozione di un peer non riduce le funzionalità del sistema
- Le connessioni tra i peer sono stabilite per:
  - propagare le queries e le risposte alle queries
  - acquisire conoscenza sullo stato della rete

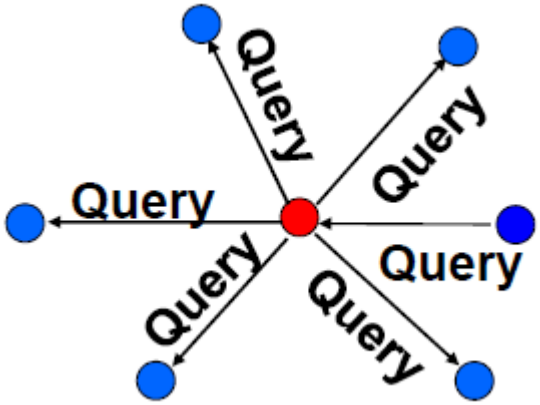


OVERLAY NON STRUTTURATI

- Reactive protocol
  - i peer non notificano i files che intendono condividere
    - non esiste un sistema di indicizzazione delle risorse
  - i cammini verso i peer che forniscono il file ricercato (content providers) vengono stabiliti dinamicamente (on demand)
- Ricerca: completamente decentralizzata, non esiste una entità centralizzata di coordinamento, necessari meccanismi:
  - decentralizzati per la diffusione delle query
  - per identificare univocamente i messaggi che transitano sulla rete ed evitare routing ciclici
- Invio delle risposte: backward routing.
  - la risposta positiva ad una query viene inoltrata a ritroso lungo lo stesso cammino seguito dalla query
  - richiede meccanismi per l'individuazione del backward path

Il contenuto viene trasferito direttamente dal peer che lo memorizza a quello che lo ha richiesto

MECCANISMI DI RICERCA: BREADTH FIRST

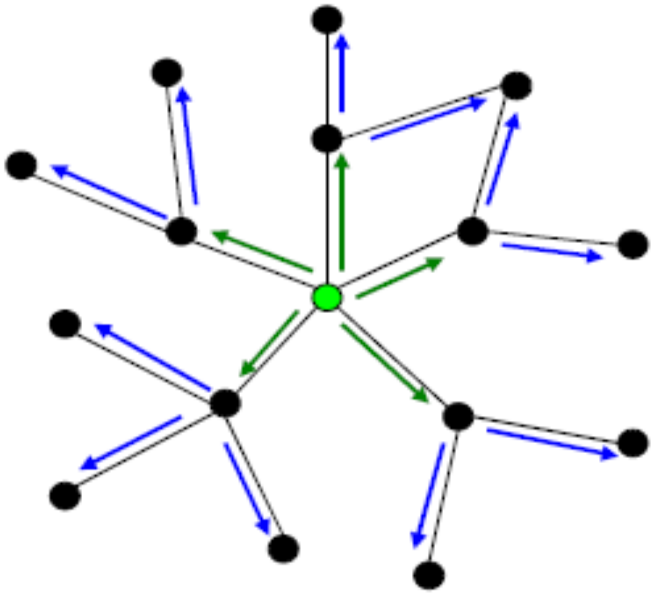


- la query viene diffusa a tutti i vicini, escluso quello da cui si è ricevuta
- meccanismo di routing: TTL Enhanced Flooding
- l'individuazione del documento non è garantita: possibilità di falsi negativi
- ridotta scalabilità.
- uso di GUID per evitare cicli nella ricerca
- adottato in Gnutella

MECCANISMI DI RICERCA: EXPANDING RING

- Meccanismo di base: sequenza di flood con TTL crescente
  - si inizia con un valore basso del TTL
  - se la ricerca non ha successo si incrementa successivamente il TTL

- Ancora bassa scalabilità
- Miglioramento in caso di distribuzione zipfiana degli oggetti (analisi in una lezione successiva)



MECCANISMI DI RICERCA: RANDOM WALK

- Inoltra la query ad un vicino scelto in modo casuale
  - riduzione del numero di messaggi
  - incremento della latenza
- Compromesso: Multiple Random Walk (k-Random Walk)
  - invia la query a k vicino scelti casualmente
  - incrementa numero messaggi
  - riduce la latenza
- Meccanismo di terminazione basato su TTL
- Analisi di algoritmi probabilistici

GNUTELLA 0.4

- Gnutella 0.4: una delle prime proposte di overlay P2P completamente decentralizzato

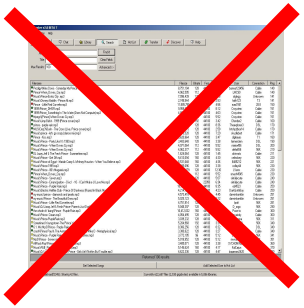
- Documentazione dettagliata del protocollo:
  http://rfc-gnutella.sourceforge.net/developer/index.html

- Un pò di storia:
  - sviluppato in non più di un mese da Frankel e Peppers, in Nullsoft, acquistata poi da AOL
  - Protocollo reso pubblico mediante reverse engineering
  - Gnutella = GNU (Not Unix, software libero) + Nutella (di cui gli autori andavano matti...)

- Le specifiche del protocollo sono molto ampie: diverse implementazioni disponibili. Le più recenti introducono diverse ottimizzazioni.

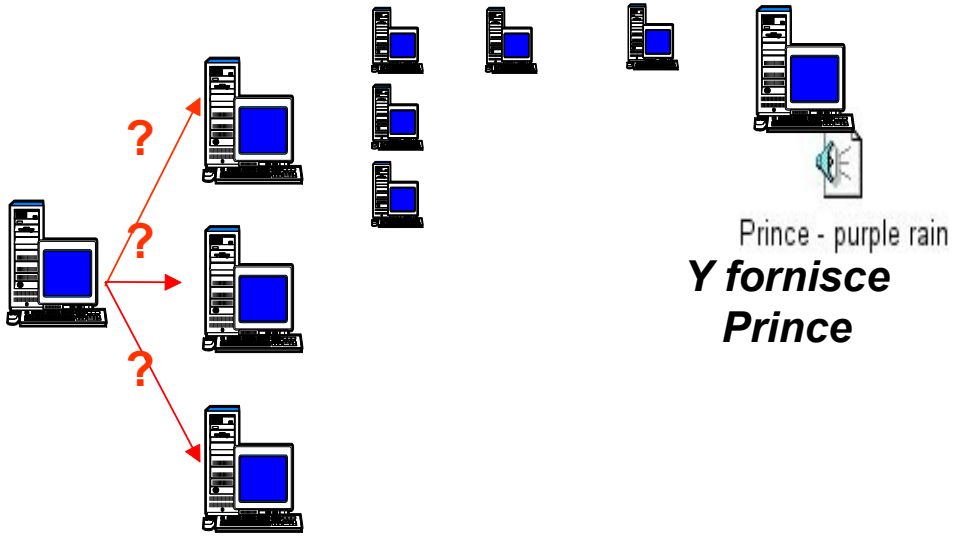
GNUTELLA 0.4

http://www.gnutelliums.com/



Non esiste un Database Centralizzato

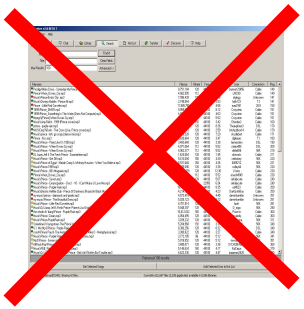
X ricerca Prince



Y fornisce Prince

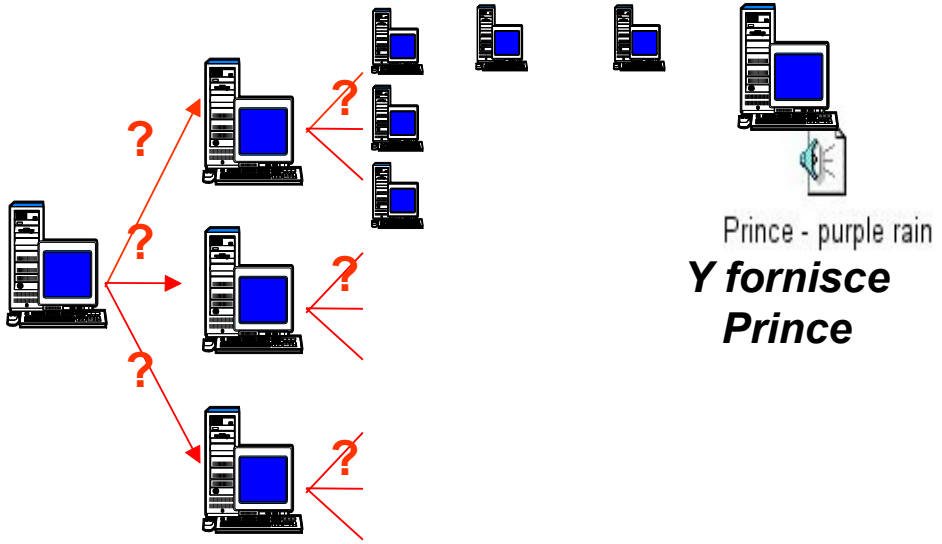
GNUTELLA 0.4

http://www.gnutelliums.com/



Non esiste un Database Centralizzato

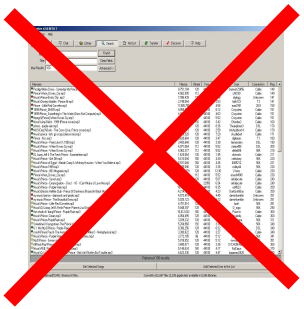
X ricerca Prince



Y fornisce Prince

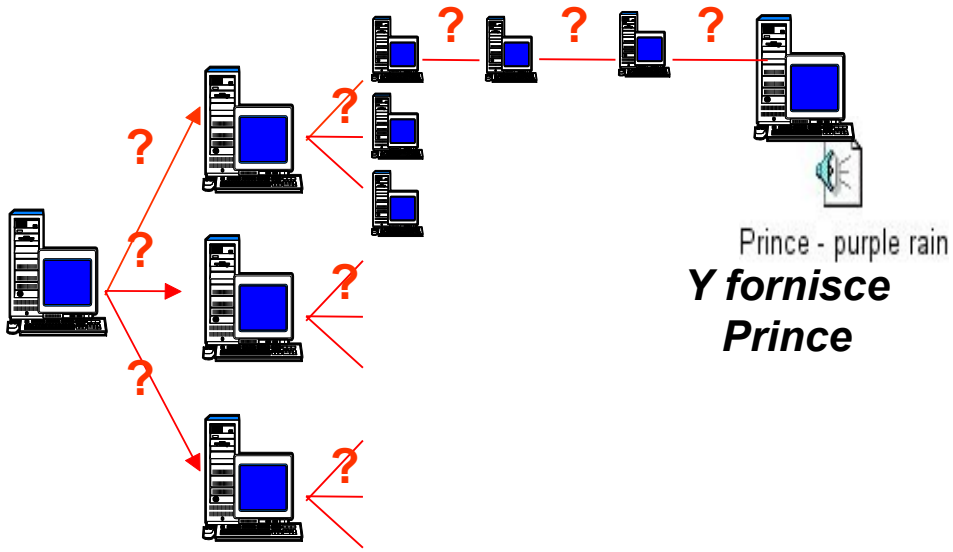
GNUTELLA 0.4

http://www.gnutelliums.com/



Non esiste un Database Centralizzato

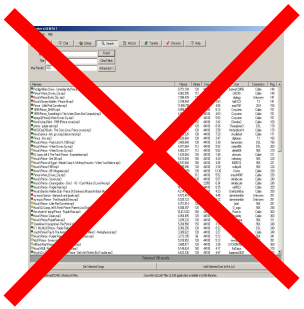
X ricerca Prince



Y fornisce Prince

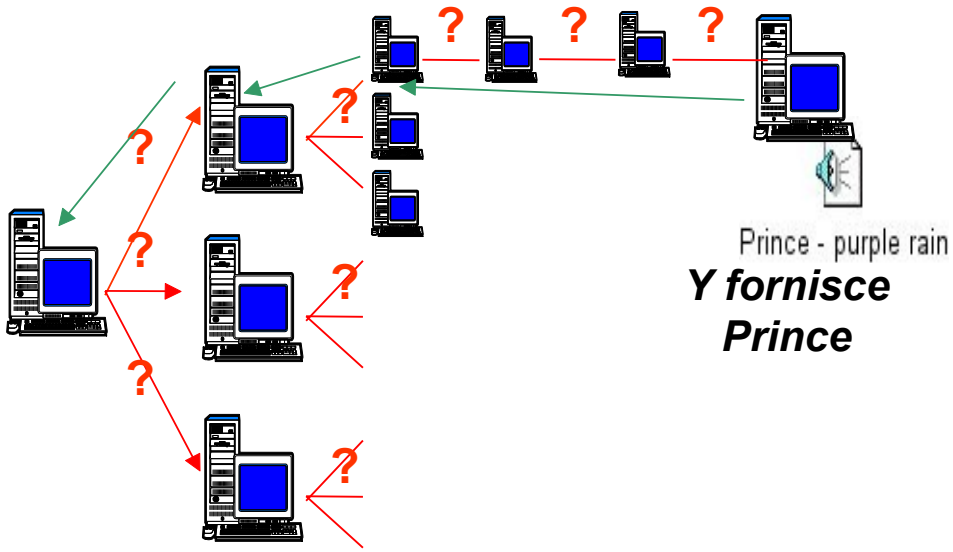
GNUTELLA 0.4

http://www.gnutelliums.com/



Non esiste un Database Centralizzato

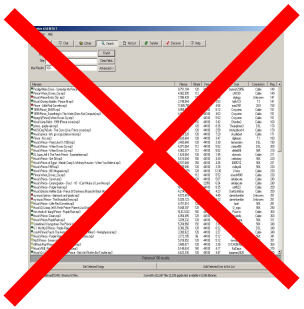
X ricerca Prince



Y fornisce Prince

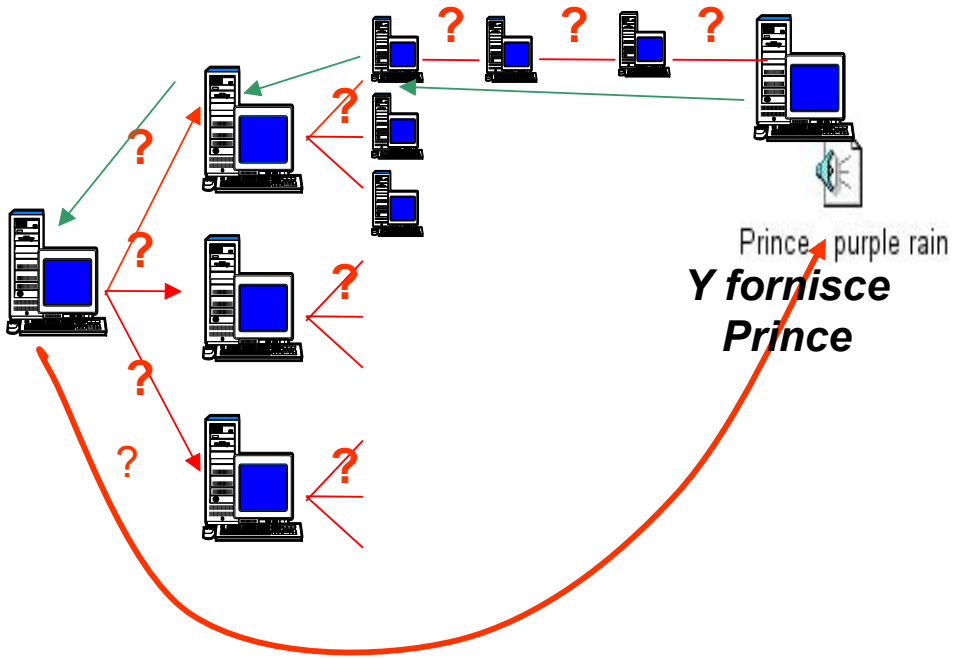
GNUTELLA 0.4

http://www.gnutelliums.com/



Non esiste un Database Centralizzato

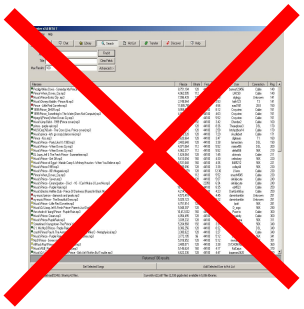
X ricerca Prince



Y fornisce Prince

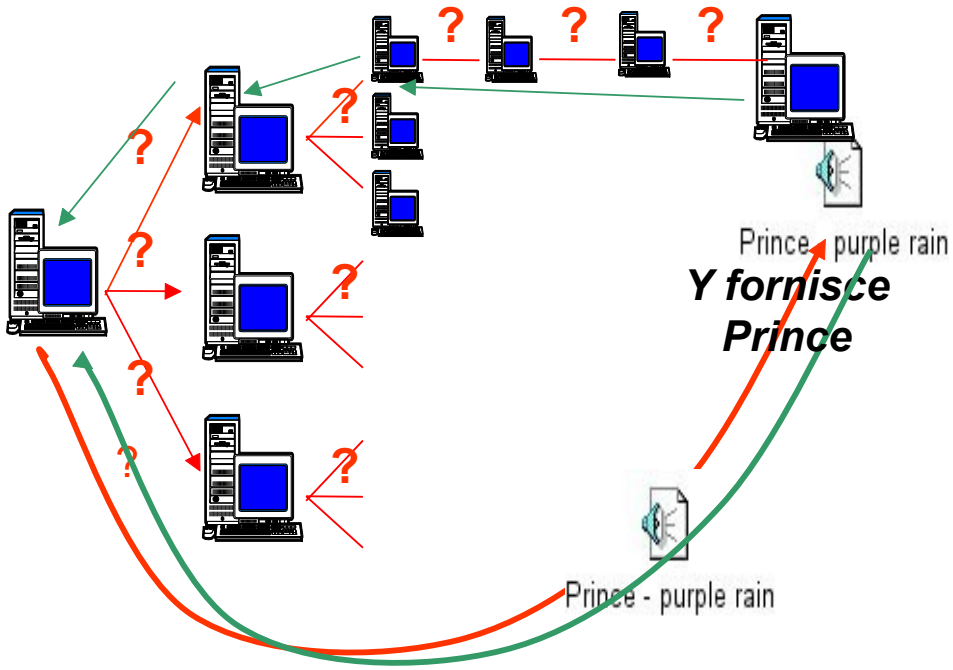
GNUTELLA 0.4

http://www.gnutelliums.com/



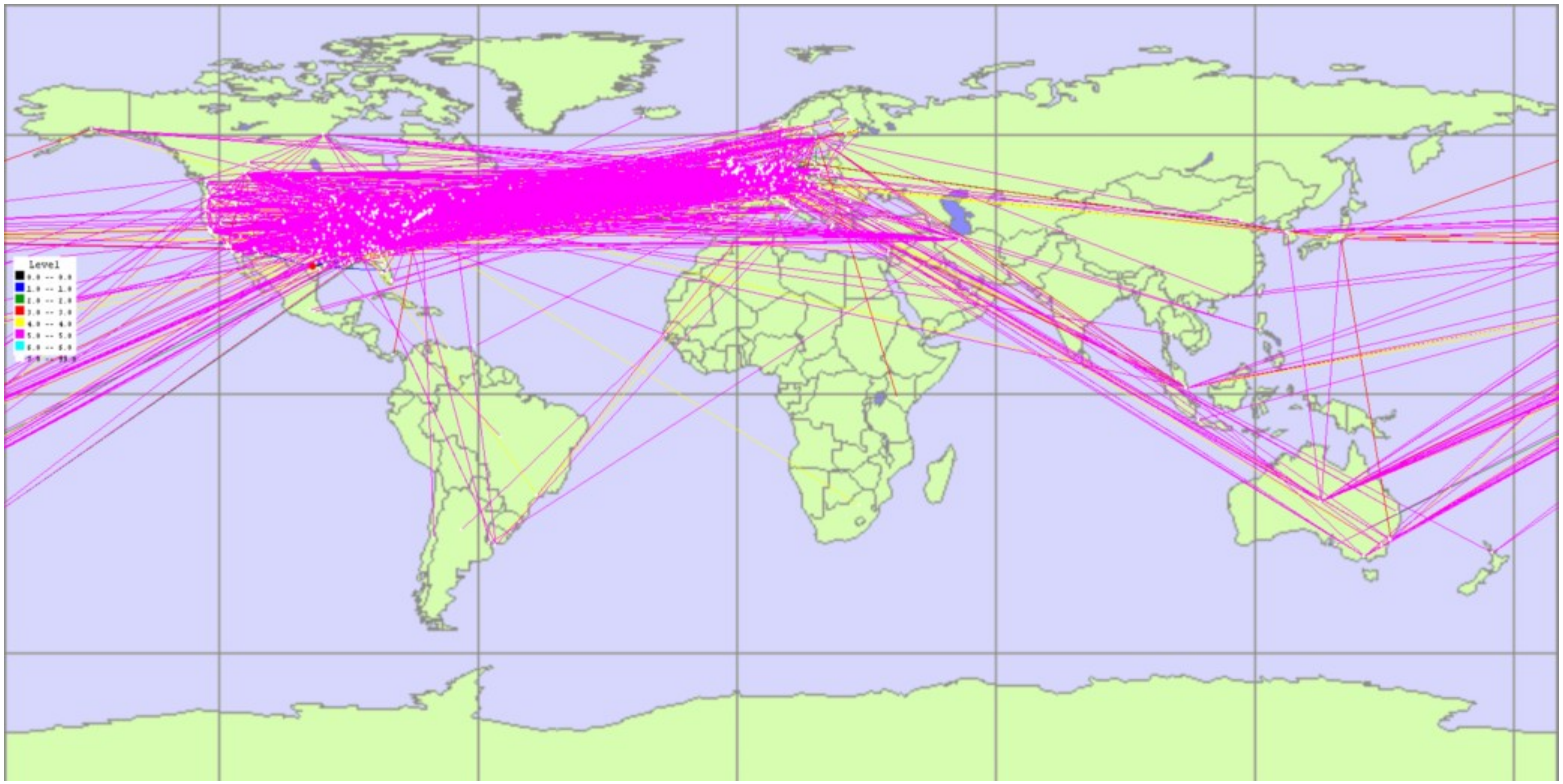
Non esiste un Database Centralizzato

X ricerca Prince



Y fornisce Prince

LA RETE GNUTELLA



Topologia della rete Gnutella rilevata
nell'agosto 2002

GNUTELLA: IL PROTOCOLLO IN BREVE

1) Bootstrap:

- Inserimento del peer nella rete mediante individuazione di peers attivi ed apertura di connessione essi.
- Meccanismi di bootstrap basati su
  - beacon (bootstrap) servers: memorizzano una lista di peers spesso attivi sulla rete
  - peer cache: memorizzano liste di hosts contattati in sessioni precendenti
- Evoluzione dei meccanismi di bootstrap in versioni recenti del servlent Gnutella

2) Esplorazione

- La rete viene esplorata mediante l'invio di messaggi di ping/pong (ping message, pong message). Questo consente di scoprire ulteriori peer sulla rete
- Eventuale creazione di connessioni con altri peer

GNUTELLA: IL PROTOCOLLO IN BREVE

3) Ricerca breadth first

- invio delle queries ai peer vicini (query message). I vicini a loro volta inoltrano la query mediante un meccanismo di enhanced flooding, limitato da TTL.

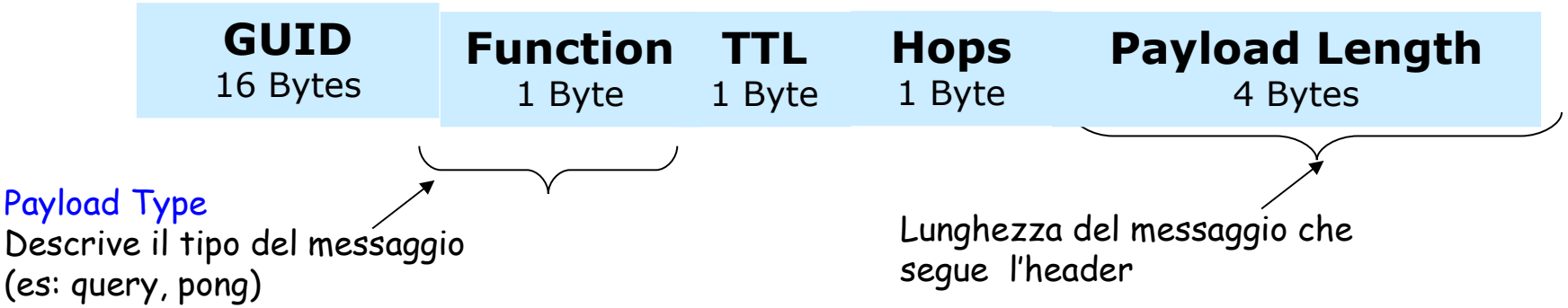
- ricezione di risposte dai vicini selezione della risposta "migliore" (query hit message)

4) Download

- connessione diretta al peer selezionato e download del file ricercato
- scambio dei dei dati mediante protocollo HTTP

GNUTELLA: IL FORMATO DEI MESSAGGI

HEADER DEL MESSAGGIO: 23Byte



- **GUID: Globally Unique IDentifier** stringa di 128 bits che identifica univocamente il messaggio
- **TTL (Time-To-Live):** quante volte il messaggio può ancora essere inoltrato dai servents prima di essere eliminato dalla rete
- **Hops:** quanti servent hanno già inoltrato il messaggio

$$TTL(0) = TTL(i) + Hops(i) \leq K$$

(K in genere = 7, moficabile da alcuni applicativi)

GNUTELLA: MESSAGE TYPES

PING (0x00)

Non contiene payload

PONG (0x01)

Port 2 Bytes	**IP Address** 4 Bytes	**Nb. of shared Files** 4 Bytes	**Nb. of Kbytes shared** 4 Bytes

- ogni messaggio di PONG corrisponde ad un PING
- GUID del messaggio di PONG = GUID corrispondente messaggio di PING
- indirizzo IP, port, identificano il peer che invia il PONG
- numero di files che il peer condivide + numero di kbytes condivisi

GNUTELLA: MESSAGE TYPES

QUERY (Function:0x80)

Minimum Speed 2 Bytes	**Search Criteria** n Bytes

- **`Minimum Speed:`**

Nelle prime versioni: banda minima (kb/sec) richiesta per i servent che offrono informazioni condivise.

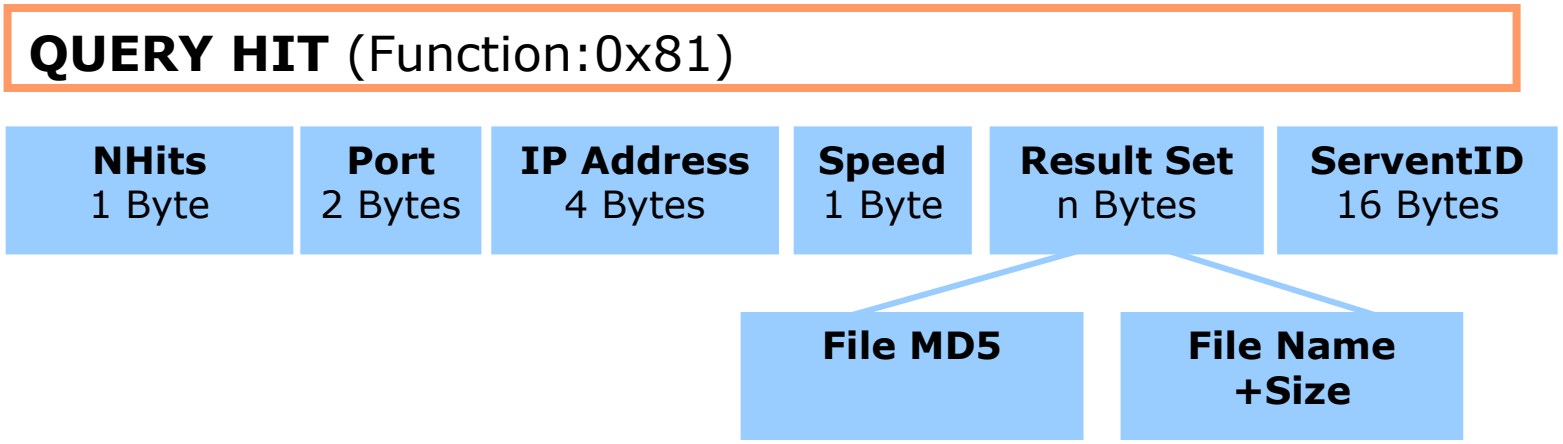
Nelle versioni più recenti sostituito da un insieme di flags, tra cui

- **`Firewalled Indicator:`** indica che l'host che ha inviato la query non può accettare connessioni perché si trova a **monte di un firewall/NAT**. In questo modo un peer che può soddisfare la query evita di inviare un query hit se si trova a sua volta a monte di un firewall.
- altri flag di supporto ad estensioni al protocollo base

- **`Search Criteria:`**

Contiene una stringa costituita mediante la concatenazione di un insieme di keyworld, separate da spazi bianchi.

GNUTELLA: MESSAGE TYPES

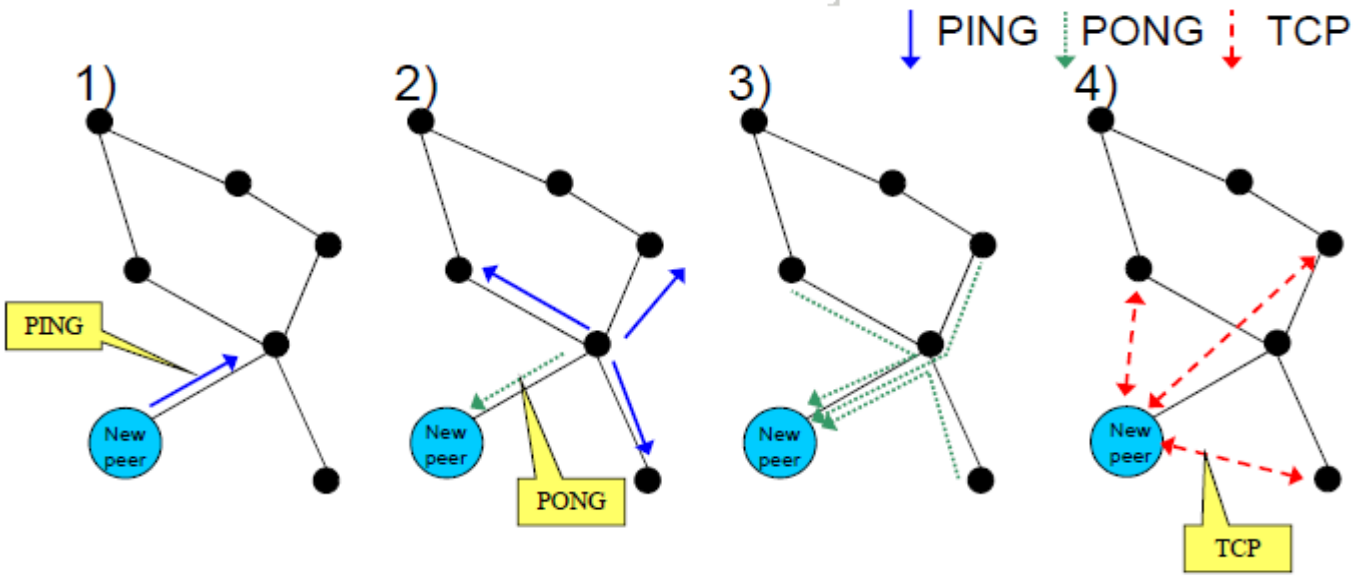


- **nHits** = Numero di elementi nel result set
- **Port + IP Address** = individuano l'host che ha generato l'hit
- **Speed** = Banda (kb/sec) dell'host che ha generato l' hit
- **Result Set** = Contiene nhits elementi e descrive i files che hanno generato l'hit
- **ServentID** = Identifica univocamente il nodo che ha inviato il query hit ed è utilizzata per il routing dei messaggi di PUSH

GNUTELLA: IL PROTOCOLLO IN BREVE

- Il servent Gnutella individua uno o più hosts nella rete (tecniche di bootstrap analizzate in dettaglio in seguito)
- Conosce indirizzo IP e numero di porta di ogni hosts individuati
- Stabilisce una connessione TCP con ciascuno di questi hosts
- Per ogni connessione TCP stabilita
  - Invio di una stringa `GNUTELLA CONNECT/<protocol version> \n\n`
  - Ricezione della risposta `GNUTELLA OK \n\n`
- Il nodo inizia ad esplorare la rete inviando messaggi di ping ai vicini

GNUTELLA: IL PROTOCOLLO IN BREVE



- il nuovo peer si connette ad (almeno) un host conosciuto della rete
- invia tramite esso un messaggio di ping sulla rete
- riceve un certo numero di pong tramite le connessioni già esistenti
- sceglie un sottoinsieme S di pong
- stabilisce connessioni TCP con i nodi in S

GNUTELLA: IL PROTOCOLLO IN BREVE

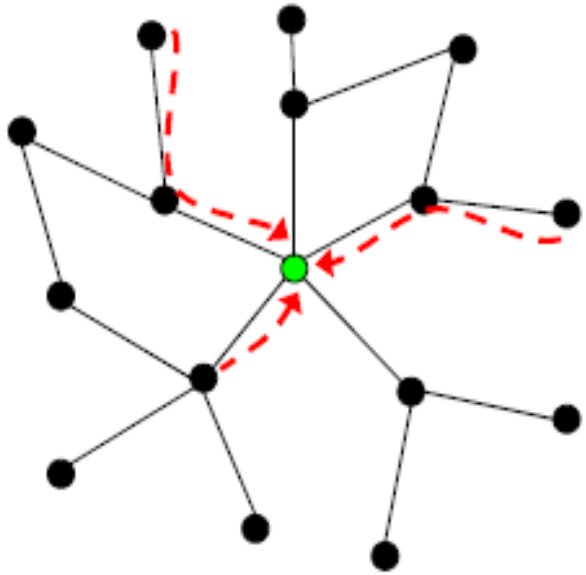
- TTL Enhanced Flooding: quando un peer riceve un ping message oppure un query message
  - incrementa il campo hops, diminuisce TTL, il messaggio viene scartato se
    - TTL=0
    - se il peer ha già inoltrato lo stesso messaggio (utilizzo di identificatori unici di messaggio).
  - altrimenti il messaggio viene spedito a tutti i vicini, escluso il peer da cui si è ricevuto il messaggio (o ad un sottoinsieme di k vicini)

- Valori tipici: TTL=7, k=4

- Quando un peer riceve un pong oppure un query hit
  - Il messaggio viene inoltrato al peer che aveva spedito il corrispondente messaggio di ping o il corrispondente messaggio di query
  - il peer memorizza le relazioni tra messaggi ricevuti e messaggi inoltrati

GNUTELLA: IL PROTOCOLLO IN BREVE

- Quando un nodo riceve una query controlla se possiede contenuti che possano soddisfare la query
- In questo caso invia un messaggio di QueryHit
  - il Query Hit contiene
    - indirizzo IP e porta del mittente
    - informazioni sui files che soddisfano il criterio di ricerca
  - Backward routing: la risposta percorre a ritroso il percorso della query



GNUTELLA: IL PROTOCOLLO IN BREVE

- Perché i messaggi di query hit e di pong vengono inviati mediante backward routing, invece che mediante una connessione diretta con l'host H che li ha originati?
  - H dovrebbe accettare una gran quantità di connessioni
  - queste connessioni sarebbero utilizzata solo per inviare un numero limitato di dati
  - le connessioni verrebbero aperte ed immediatamente disattivate, dopo la ricezione di un pong o di un query hit
  - Il backward routing consente di implementare politiche di caching (pong caching).

GNUTELLA: IL PROTOCOLLO IN BREVE

- Quando un servent A riceve un messaggio di query hit B, A può decidere di scaricare il file richiesto da B . In questo caso apre una connessione diretta con B.

- Il download
  - non avviene tramite la overlay network,
  - è implementato mediante protocollo HTTP, su una connessione diretta tra servents

- Connection Reversal per peer nattati: se B si trova a monte di un firewall, può non essere in grado di accettare connessioni
  - B avverte A di trovarsi a monte di un firewall (esiste un flag nel messaggio di query hit)
  - Al momento della ricezione del query hit, A invia indietro a B un messaggio (messaggio di push)
  - Quando B riceve il push, apre una connessione verso A, quindi invia il file.

GNUTELLA: TIPI DI CONNESSIONE

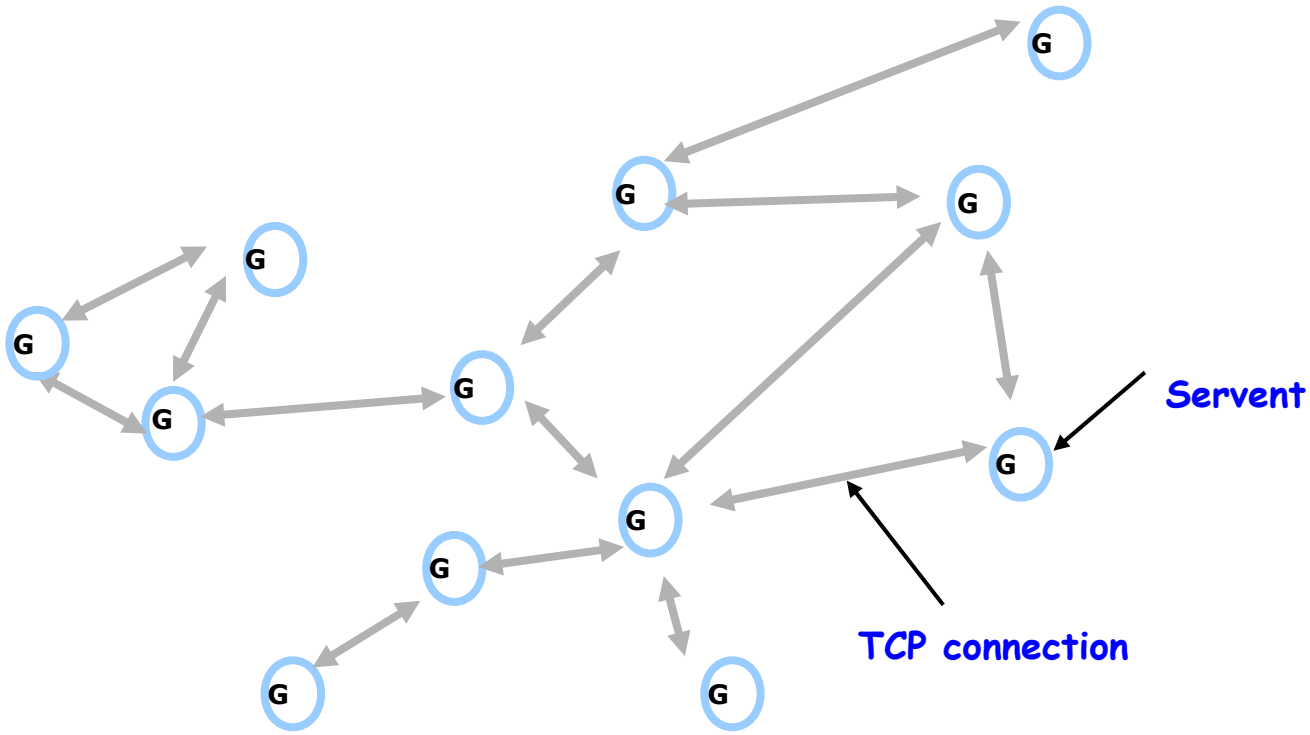
Signaling Connections

- Definiscono la overlay network
- Utilizzate per
  - la ricerca di informazione condivisa
  - invio di messaggi di keep-alive
- stabili, vengono modificate solo quando cambiano i vicini sulla overlay network
- Basate su TCP

Content Transfer Connections

- Utilizzate per il trasferimento dei dati
- temporanee
- Basate su HTTP

GNUTELLA: L'OVERLAY NETWORK



- I peer sono connessi mediante una rete logica, l'overlay network (rete di copertura)

- Archi= connessioni TCP tra i peers, corrispondono a collegamenti astratti a cui possono corrispondere diversi collegamenti fisici
Es: un arco tra un peer situato negli Stati Uniti ed uno in Polonia

ANALISI DETTAGLIATA DEL PROTOCOLLO

- In ogni sistema completamente decentralizzato è necessario risolvere il problema dell'individuazione di un punto iniziale di connessione:
  dove e come trovare un servent Gnutella a cui connettersi?

- Il meccanismo di bootstrap consente di stabilire la prima connessione ad un host della overlay network

- Un semplice meccanismo:
  - invio di messaggi di ping ad un insieme di hosts scelti in modo completamente casuale sulla rete, senza accertarsi preventivamente se fanno parte della rete Gnutella
  - meccanismo inutilizzabile in quanto
    - produce una grossa mole di traffico inutile
    - il tempo di individuazione di un servent è in generale inaccettabile

GNUTELLA:MECCANISMI DI BOOTSTRAP

Meccanismi implementati nelle prime versioni del protocollo:

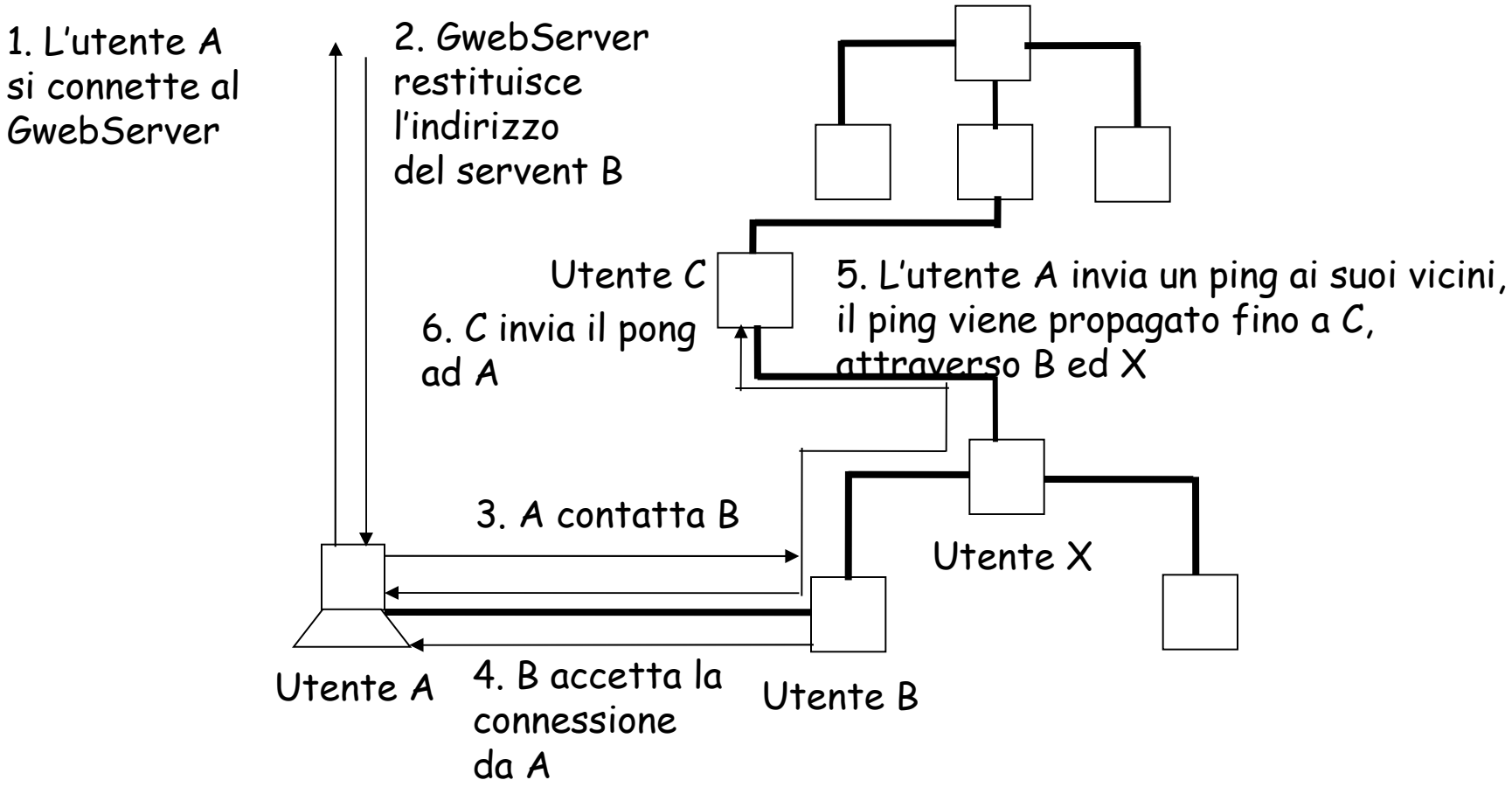
- Out of band methods: gli indirizzi di alcuni servent Gnutella sono pubblicati su alcune pagine web (o si possono ottenere mediante chat). L'utente passa uno di questi indirizzi al servent Gnutella, al momento della attivazione.

I meccanismi definiti nelle versioni successive sono basati su strategie di caching:

- GWebCache: web servers su cui è in esecuzione uno script che consente di interagire con servent Gnutella. Memorizzano gli IP di alcuni servent ed i riferimenti ad altri GWebCaches. E' aggiornato dinamicamente dai servents, in modo automatico.

- Cache interna al servent: memorizza gli indirizzi IP degli hosts con cui il servent è venuto a contatto durante la sessione attuale e le sessioni precedenti(ricavati da messaggi di pong, queryhit ed X-try)

GNUTELLA: MECCANISMI DI BOOTSTRAP



GNUTELLA: GWEBCACHES

- Una ricerca di GWebCaches effettuata con Google restituisce migliaia di risultati (tuttavia molti sono relativi a GWebCaches non più attivi….).
- Definiscono un'ulteriore overlay network, definita "sopra" la rete Gnutella
- Ogni servent
  - può richiedere al GWebServer l'indirizzo IP di un host oppure un riferimento ad altri GWebServers
  - memorizza le informazioni ricevute dal GWebServer in una sua cache interna
  - informa periodicamente il GWebServer della sua presenza nella rete
    - gli aggiornamenti vengono inviati solo dopo che il servent è rimasto attivo sulla rete per un certo intervallo di tempo, uptime (esempio: un'ora). Dopo questo intervallo di tempo gli aggiornamenti vengono inviati periodicamente
    - un servent non invia aggiornamenti al GWebServer se si trova a monte di un firewall

GNUTELLA: LOCAL HOST CACHES

- Ogni servent Gnutella implementa un Local Cache. Il Local Cache memorizza in modo permanente i riferimenti ad un centinaio di servent.
- La lista di servent viene caricata in memoria quando il servent viene lanciato e viene salvata al momento della sua disconnessione dalla rete.
- Una lista iniziale di hosts e GwebServers viene distribuita con il servent.
- La probabilità che uno degli hosts nella local cache sia attivo è alta, anche se il local host non è stato aggiornato da alcune settimane.
- Il local cache viene aggiornato dinamicamente mediante i meccanismi di
  - GWebCache
  - Pong Reply
  - Query Hit
  - Xtry, Xtry UltraPeer

GNUTELLA: LOCAL HOST CACHES

- Alcune implementazioni di Gnutella mantengono nella RAM una lista molto grande di hosts contattati, ma al momento della disconnessione, salvano solo una parte della lista. Gli hosts sono scelti in base a
  - Uptime
  - Numero e dimensione dei files condivisi
  - Ultima volta in cui il servent remoto è stato contattato
  - ……
- Il local cache deve essere utilizzato per evitare di contattare più volte durante la stessa sessione il GWebServer
- Il local cache viene utilizzato anche per evitare di ricontattare più volte lo stesso host

GNUTELLA: UN ALGORITMO DI BOOTSTRAP

- Lanciare il servent e caricare i dati dal Local Cache
- Provare a connettersi alla rete Gnutella, utilizzando i dati ricavati dal Local Cache (evitare di aumentare il carico sul GWebServer)
- Se dopo X secondi non si sono stabilite connessioni, inviare una query al GWebServer
- Se dopo Y secondi non si è avuta risposta dal GWebServer, provare a contattare un altro GWebServer e così via, fino a che non si riesce a stabilire un contatto
- Non ricontattare il medesimo GWebServer durante la solita sessione
- Valori tipici: X= 5 secondi, Y= 10 secondi

GNUTELLA HANDSHAKING

Dopo che il servent A ha scelto un servent B a cui connettersi

- A richiede una connessione TCP con B
- A invia un messaggio a B costituito da una serie di righe. La struttura del messaggio è simile a quella di un messaggio HTTP
  - ogni linea è costituita da una sequenza di caratteri ASCII.
  - le linee sono separate dai caratteri <cr> <lf>
  - la prima linea è quella di richiesta/concessione connessione
  - le linee successive sono righe di intestazione (headers)

- B può decidere di accettare la connessione o di rifiutarla (in base ad una politica di gestione delle connessioni)

GNUTELLA HANDSHAKING

SERVLENT A

GNUTELLA CONNECT/0.6<cr><lf>
User Agent: Bearshare/1.0<cr><lf>
Ultra-Peer: False<cr><lf>
Pong-caching: 0.1<cr><lf><cr><lf>

GNUTELLA/0.6 200 OK<cr><lf>
<cr><lf>

SEVLENT B

GNUTELLA/0.6 200 OK <cr><lf>
User Agent:Limewire/1.0<cr><lf>
Pong Caching: 0.1 <cr><lf>
X-try:141.2.89.3:6436, 212.43.98.71:6436<cr><lf>

- **`X-try headers`** = can be seen as connection pongs
- Contengono una lista di coppie (indirizzo IP, porta) che individuano una lista di servent Gnutella noti al servlent a cui ci si tenta di connettere. Vengono inviati anche se l'host decide di non accettare la connessione
- Esempio:
  X-try: 1.2.3.4: 1234, 3.4.5.6:3456

GNUTELLA: GESTIONE DELLE CONNESSIONI

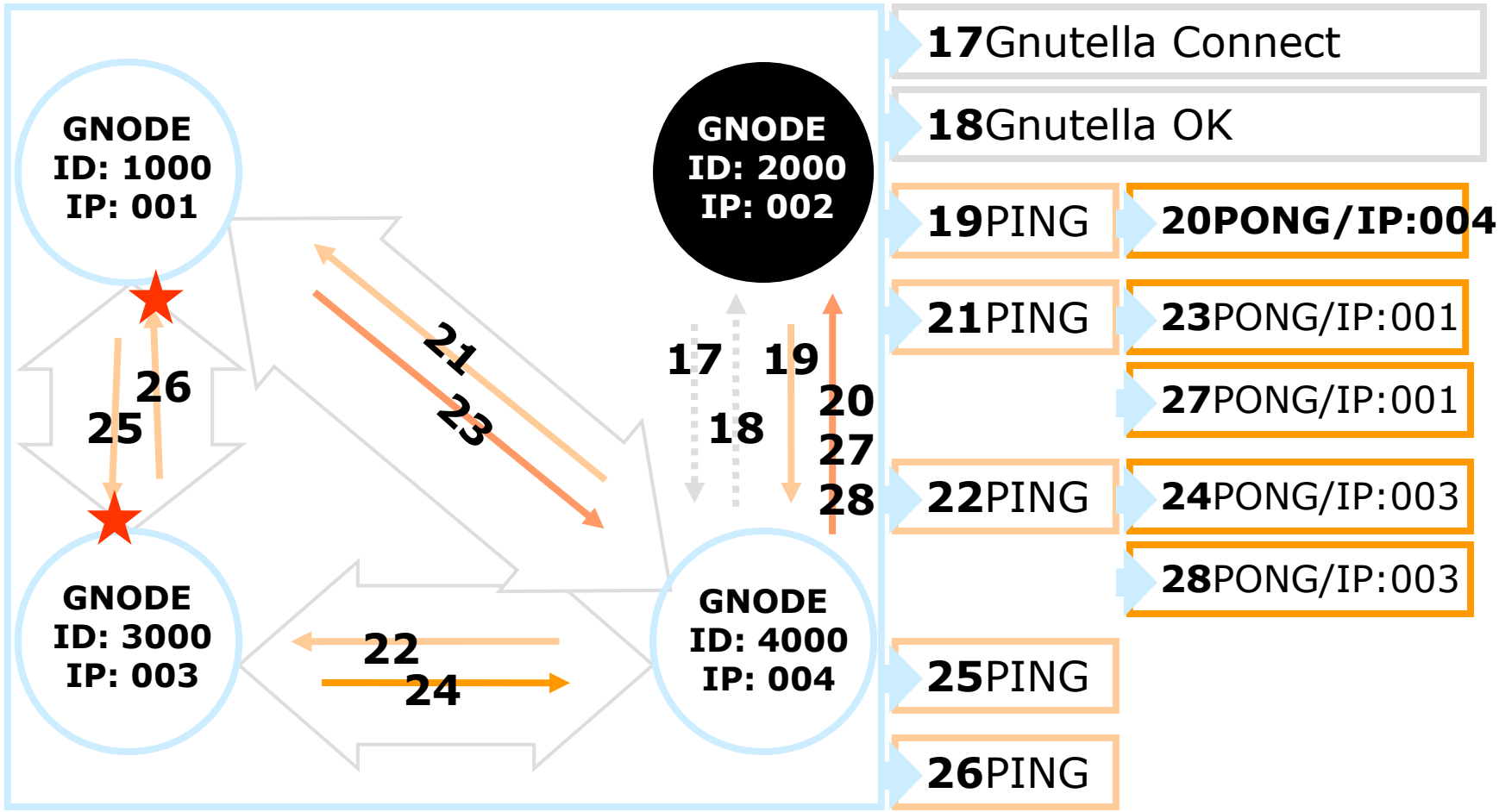
Gestione delle Connessioni

- Il numero di connessioni aperte dipende dalla banda di entrata/uscita del servent

- Una semplice politicha per gestire le connessioni:
  - stabilire un valore k che indica il massimo numero di connessioni che il servent può accettare.
  - durante la fase di bootstrap aprire n (n<k) connessioni verso altri servent.
  - i rimanenti n-k slots sono utilizzati per accettare connessioni in ingresso, durante la permanenza dell'host all'inteno della rete Gnutella.

GNUTELLA: ESPLORAZIONE DELL'OVERLAY

Il nodo 2000 stabilisce una connessione con il 4000

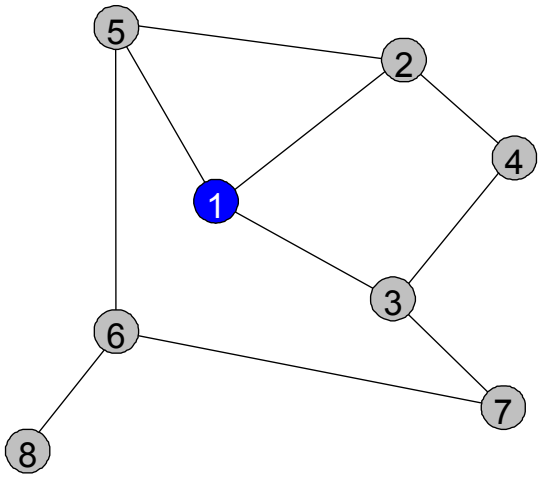


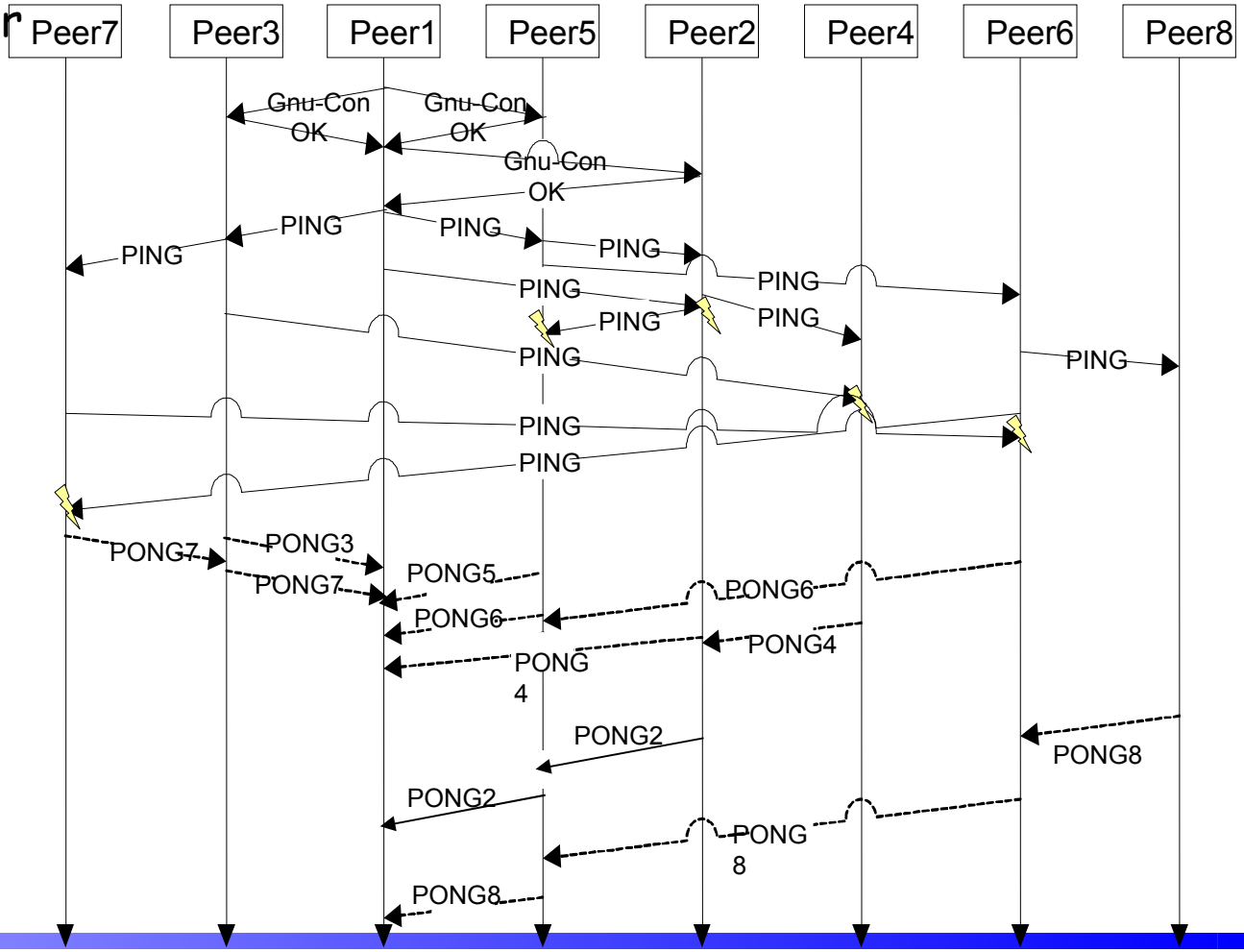
GNUTELLA: ESPLORAZIONE DELL'OVERLAY

Sequenza di messaggi scambiati nella fase di connessione ed esplorazione della rete

numero di messsaggi per esplorare la rete:

- 6 msg di connessione
- 12 PING
- 12 PONG





GNUTELLA: SCALABILITA'

Il principale problema di gnutella 0.4 è la sua bassa scalabilità

- Il numero di ping e di query cresce in modo esponenziale ad ogni hop
- Esempio: un nodo invia un messaggio di query con TTL=7 (valore di default per la maggior parte delle implementazioni), ogni nodo lo propaga a x vicini (esempio x=4, TTL=7)
  - il numero totale di queries propagate sulla rete è calcolabile come una serie geometrica

    $\Sigma_{i=1,7} x^i = (4^{i+1} - 1)/(4-1) = (4^{7+1} - 1)/(4-1) = 21845$
  - il numero di risposte dipende dalla popolarità del file richiesto
- Soluzione
  - Introduzione di appocci ibridi basati su superpeers
  - Definizione di meccanismi di caching

GNUTELLA 0.4: SCALABILITA'

- Frequenza dei diversi tipi di messaggi nella rete Gnutella

QUERY	54.8%
PONG	26.9%
PING	14.8%
QUERY HIT	2.8%
PUSH	0.7%

- 41.7% dei messaggi per l'esplorazione della rete
- per diminuire il numero dei messaggi: pong caching
  - quando un host ricelve un ping da un altro host, può:
    - inviare il ping originale a tutti i suoi vicini, e restituire al mittente del ping i pong ottenuti come risposta
    - utilizzare una cache dei pong, in modo da restituire i pong dei suoi vicini (memorizzati in precedenza) senza doverli interrogare con ulteriori ping.

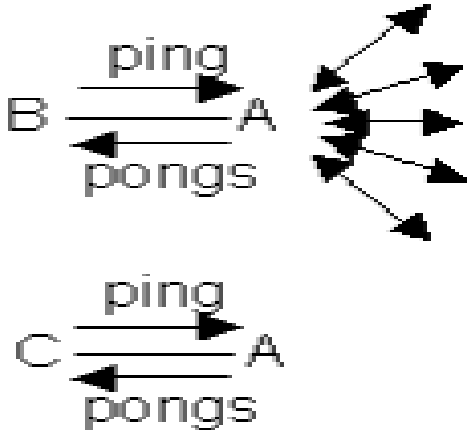
PROTOCOLLO PING/PONG: PROPRIETA' GENERALI

Ogni implementazione del protocollo di PING/PONG deve garantire le seguenti proprietà

- alla ricezione di un PING da una connessione
  - se è passato almeno un certo intervallo di tempo dal momento in cui si è ricevuto un altro PING dalla stessa connessione, il servent deve inviare un certo numero di messaggi di PONG. Strategie:
    - propagazione messaggio di PING
    - PONG caching
- ad ogni PONG viene associato lo stesso GUID del PING corrispondente
- TTL(PONG) ≥ HOP (PING) (non ovvio con pong caching)

PONG CACHING: IDEA BASE

- introdotto per diminuire il numero di messaggi sulla overlay network
- osservazione base: le informazioni contenute in un PONG sono statiche: solo quantità di dati condivisi varia, ma non rapidamente.
- I PONG ricevuti possono essere memorizzato in una cache



A propaga il PING ricevuto da B e memorizza nella cache i PONG ricevuti in risposta, mentre non propaga il PING di C, sfrutta invece la cache

- necessità di rinfrescare la cache periodicamente a causa della dinamicità della rete

GNUTELLA PONG CACHING

- Obiettivi di una buona strategia di pong caching:
  - messaggi di PONG devono riferire, con alta probabilità, host on-line
  - dovrebbe essere garantita una scelta uniforme tra gli host della rete
  - ottimizzazione della banda per messaggi di PING/PONG
- Diverse implementazioni del pong caching a seconda del client utilizzato

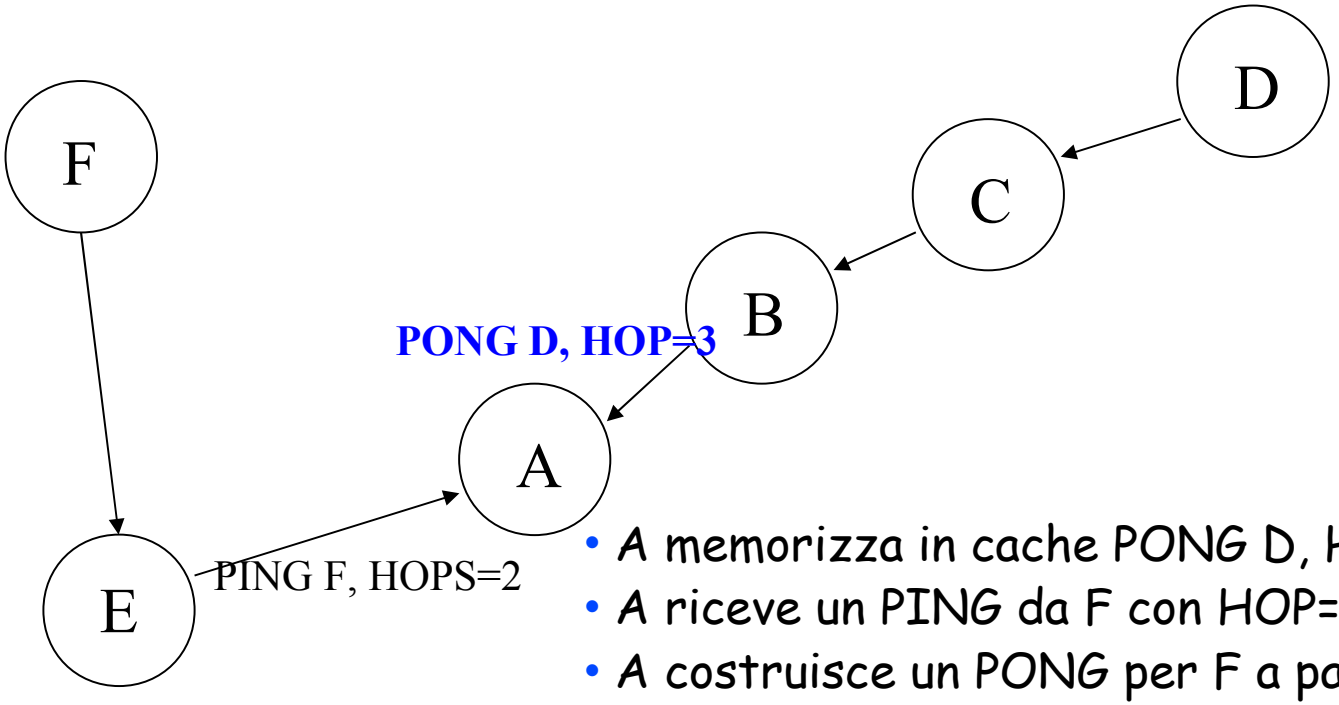
PONG CACHING: UNA PRIMA SOLUZIONE

- Se possibile, rispondere ad un PING con PONG prelevati dalla cache
  - scelta di PONG provenienti da connessioni diverse e con valori di hops diversi
- "rinfrescare" periodicamente la cache
  - inviare un PING (TTL=7, HOPS=0), su tutte le connessioni ogni X secondi: trade-off intervallo di refresh/banda impiegata
  - X varia a seconda che il vicino implementi o meno pong caching (handshaking header)
    - vicino supporta pong caching, diminuire X (3sec.) (il PING non viene inviato in broadcast)
    - vicino non supporta pong caching, aumentare X (1 minuto) (ogni PING ricevuto, inviato a tutti)

COSTRUZIONE PONG MESSAGES

- Come ricavare i messaggi di PONG dalle informazioni contenute nella cache
- Al momento della ricezione di un ping Pi, il servent invia un insieme di messaggi di pong, costruiti a partire dai messaggi presenti nella pong cache
- Per ogni messaggio di pong Po di risposta al messaggio di ping Pi costruito a partire da un messaggio di pong Pc reperito nella cache:
  - `GUID(Po) = GUID(Pi)`
  - `HOPS(Po) = HOPS (Pc) +1`
  - `TTL(Po) = 7 - HOPS(Pc)`
  - `Se TTL(Po) < HOPS(Pi): Po non viene spedito`

PONG CACHING: IMPLEMENTAZIONE

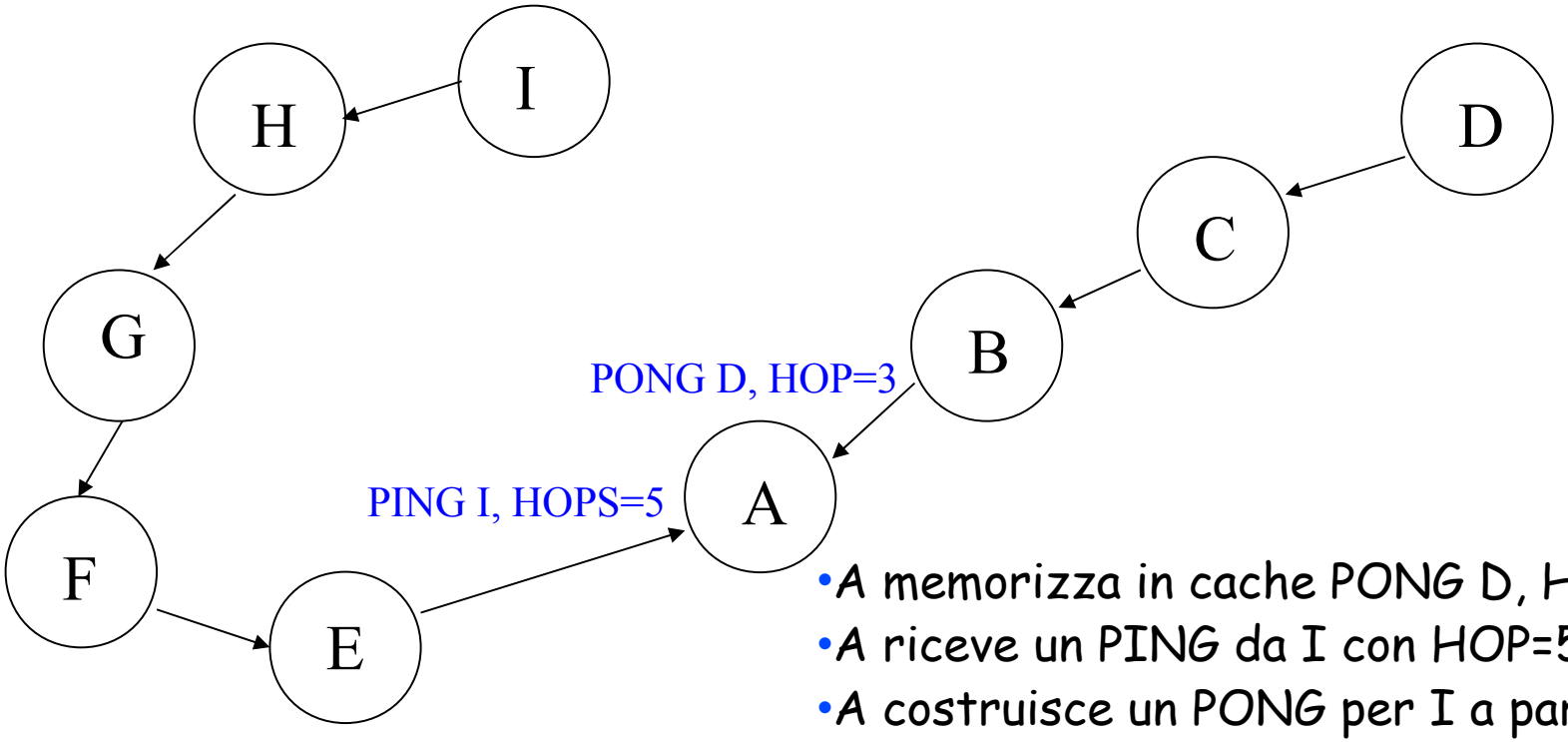


- A memorizza in cache PONG D, HOP=3
- A riceve un PING da F con HOP=2
- A costruisce un PONG per F a partire dal PONG D reperito nella propria cache
- TTL(PONG destinato ad F) = 7-3= 4
- poichè

$$4 \geq 2 \text{ (HOP(PING di F))}$$

il PONG viene inviato ad F

PONG CACHING: IMPLEMENTAZIONE



- A memorizza in cache PONG D, HOP=3
- A riceve un PING da I con HOP=5
- A costruisce un PONG per I a partire dal PONG D reperito nella propria cache
  TTL(PONG destinato ad I) = 7-3= 4
- poichè

  4 < 5 (HOP(PING di F))

  il PONG non viene inoltrato

PONG CACHING: UNA SOLUZIONE PIU' AVANZATA

- Mantenere nella cache le seguenti informazioni:
  - IP Address, Porta dell'host che ha inviato il PONG
  - Numero Files condivisi
  - Dimensione totale dei files condivisi,
  - Unique Connection Identifier: Handle del Socket da cui è stato ricevuto il PONG) e porta,
  - Timestamp
- Timestamp e UCI non sono contenuti nel messaggio di PONG, ma impostati dall'host che lo riceve

PONG CACHING: UNA SOLUZIONE PIU' AVANZATA

- all'inizio la cache è vuota
- stabilito numero minimo k di PONG da spedire per ogni PING
- se la cache non contiene almeno k PONG, PING broadcast, altrimenti PING drop e utilizzo i PONG nella cache
- i PONG corrispondenti ai PING vengono sempre memorizzati nella cache, quindi spediti al mittente: ogni PONG memorizzato con time stamp
- periodicamente, filtrare le entrate della cache più vecchie in base ad una soglia di permanenza in cache

PONG CACHING: POLITICA DI FILTERING

Diverse opzioni possibili:

- alla ricezione di un PING filtro le entrate della cache, poi decido PING broadcast o PING drop, in base al numero di entrate rimaste nella cache
- definisco un thread eseguito in background: filtering della cache ad intervalli regolar. Adottata dalla maggioranza dei clients
- Alla ricezione di un nuovo PONG effettuo il filtering della cache

PONG CACHING: POLITICA DI FILTERING

Compito per casa (non obbligatorio): scrivere un programma JAVA che implementi il protocollo di PING/PONG utilizzando le diverse opzioni presentate nei lucidi precedenti. Valutare le diverse strategie implementate.

PONG CACHING

- Perchè un messaggio Gnutella include sia il campo HOP che il TTL?
- Motivazioni
  - Il servent può decrementare il valore del TTL (alcuni servent utilizzano un TTL =5): in questo non è sempre possibile risalire dal valore del TTL contenuto nel messaggio al numero di HOPS.
  - Utilizzo di PING con funzioni particolari
  - PING con TTL=1 ed HOPS=0: remote host probing
  - PING con TTL=2 ed HOPS=0: crawler PING utilizzato per ottenere informazioni sull'host vicino e sui suoi vicini

GNUTELLA: GLI IDENTIFICATORI

- Gli identificatori sono utilizzati per il routing dei messaggi
  - GUID: identifica univocamente un messaggio, contenuto in ogni messaggio spedito sulla rete Gnutella
  - Server Identifier: contenuto solamente nei messaggi di QueryHit e di Push

- Gli identificatore vengono utilizzati
  - per supportare il backward routing (vedremo in seguito l'uso diverso dei due tipi di identificatori)
  - per per evitare la duplicazione dei messaggi sulla rete

GNUTELLA: IL ROUTING

- i servent svolgono funzioni di routing, per l'instradamento delle risposte, i messaggi da gestire sono
  - QueryHit
  - Pong
  - Push

- routing dei messaggi di QueryHit e di Pong:
  - memorizzare il GUID dei messaggi di richiesta (esempio: Query/Ping)
  - aspettare la risposta (esempio: QueryHit/Pong)
  - instradare la risposta sulla connessione da cui era arrivata la richiesta

- per quanto tempo mantenere in memoria i GUID?
  - un servent non può mantenere la storia di tutti i messaggi ricevuti in una sessione (fino ad un milione di messaggi al giorno)
    - mantere per un intervallo di tempo il GUID, dopo di cui si scarta
    - intervalli diversi per tipi diversi di messaggi (vedere in seguito)

QUERY HIT/PONG ROUTING: IMPLEMENTAZIONE

- utilizza una tabella associativa, con chiave il GUID e con valore la connessione
  - per ogni query ricevuta si inserisce nella tabella l'associazione
    GUID ⇒ connessione da cui la query è stata ricevuto
  - se la Query è stata generata da S imposta a null il riferimento alla connessione.

- Associa ad ogni Query Hit lo stesso GUID della corrispondente Query. Al momento della ricezione di un Query Hit
  - se esiste una connessione associata al GUID nella tabella
    - il messaggio viene instradato su quella connessione
  - se c'è il valore NULL associato al GUID nella tabella
    - il messaggio è una risposta alla query del servent
  - non esiste il GUID nella tabella
    - il GUID è scaduto ed è stato rimosso dalla tabella prima di ricevere il messaggio

PUSH: CONNECTION REVERSING

- Supponiamo che B sia nattato e che non possa accettare connessioni dall'esterno

- Punto fondamentale: B apre comunque un insieme di connessioni verso l'esterno per inserirsi nell'overlay

- B quindi può ricevere la Query mediante le connessioni dell'overlay network che esso stesso ha aperto verso altri peer. Può inviare i messaggi di risposta a query (query hit) mediante le queste connessioni

- A invia una Query e B ha il contenuto che la soddisfa. B invia il query hit mediante le connessioni dell'overlay ed inserisce nel query hit
  - il proprio ServentID
  - setta il firewalled flag

PUSH: CONNECTION REVERSING

- A riceve il QueryHit e ed invia un messaggio di Push a B.
- Il messaggio di PUSH contiene
  - il ServentId di B,
  - un riferimento al file che A vuole scaricare,
  - indirizzo IP e porta di A
- Il messaggio di Push segue a ritroso lo stesso cammino percorso dal messaggio di QueryHit.
- I ServentId viene utilizzato per implementare il backward routing del messaggio di Push
- Quando B riceve il messaggio di Push, apre una connessione verso A ed invia il file richiesto

GNUTELLA: IL FORMATO DEI MESSAGGI

PUSH : (Function 0x40)

Servent Identifier 16 Bytes	**File Index** 4 bytes	**IP Address** 4 Bytes	**Port** 4 Bytes

- Servent Identifier: identifica l'host che si trova a monte del firewall (il servent che fornisce il file)
- File Index identificatore del file che si intende scaricare
- IP Address IP address dell'host a cui il file deve essere inviato
- Port porta dell' l'host a cui il file deve essere inviato

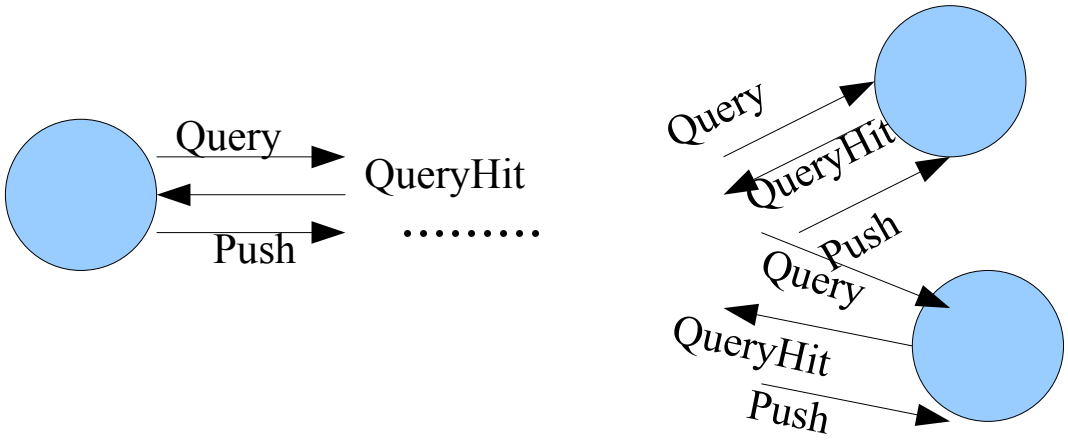
PUSH ROUTING: IMPLEMENTAZIONE

- ogni volta che un peer riceve una QueryHit inserisce nella tabella la associazione ServerIdentifier ⇒ connessione
  - Non si utilizza il GUID del messaggio!

- i messaggi di Push
  - ad essi viene associato il server identifier del corrispondente query hit
  - seguono a ritroso il percorso effettuato dal corrispondente messaggio di QueryHit
  - Per il backward routing dei messaggi di Push: ricerca in tabella con chiave il nome del Servent presente nel messaggio di Push
  - sono in numero inferiore rispetto ai QueryHit: non a tutti i query hit corrisponde un messaggio di Push

- memorizzati in una hashtable separata, per un intervallo di tempo maggiore

PERCHE' DUE MECCANISMI DIVERSI?



- Per ogni query/ping possono essere generati più query hit/pong
- Ad ogni query hit/ping viene associato lo stesso identificatore della query/ping acorrispondente
- Più query hit con lo stesso identificatore provenienti da servlet diversi
- Il messaggio di push deve essere inoltrato al server che ha generato il query hit
- Non posso usare il GUID del query hit per identificare il servlet che l'ha generato, utilizzo il Server Identifier

CONTROLLARE LA DUPLICATIZIONE DEI MESSAGGI

- è necessario evitare di inviare più volte lo stesso messaggio sulla rete
- strategie diverse a seconda del tipo di messaggio
- PING duplicati
  - molto rari, soprattutto se è implementato il PONG caching
  - si memorizza il GUID del PING in una PING duplicates table
  - quando si riceve un ping si controlla la tabella e se il GUID del messaggio è già presente, il messaggio viene scartato
- PONG duplicati
  - a differenza dei PING il controllo del GUID non è sufficiente
  - il GUID di tutti i messaggi di PONG in risposta allo stesso PING è il medesimo
  - si memorizza GUID+IP+porta, controllo simile a PING

CONTROLLARE LA DUPLICAZIONE DEI MESSAGGI

- Query duplicate:
  - molto frequenti, causate dalla presenza di cammini ciclici sull'overlay
  - come per i PING si può memorizzare il GUID ed applicare lo stesso procedimento di controllo
  - GUID già memorizzato per implementare il QueryHit

- QueryHit duplicati
  - un messaggio di QueryHit può contenere riferimenti a diversi file
  - un 'grosso' QueryHit può essere ''fragmentato'' in una sequenza di QueryHit più piccoli,
  - quindi potremmo avere
    - più query hit con lo stesso GUID della query che le ha originate
      - GUID del messaggio + Servent Identifier(SUID)
    - più query hit con lo stesso GUID della query che le ha originate e lo stesso servent identifier
      - GUID del messaggio + SUID + l'hash(payload)

GNUTELLA: IL CONTROLLO DEL FLUSSO

- Controllo del flusso:
  - regolare la quantità di dati che passano attraverso una connessione dell'overlay
  - obiettivo: diminuire la congestione e quindi il ritardo di trasmissione di un messaggio su quella connessione

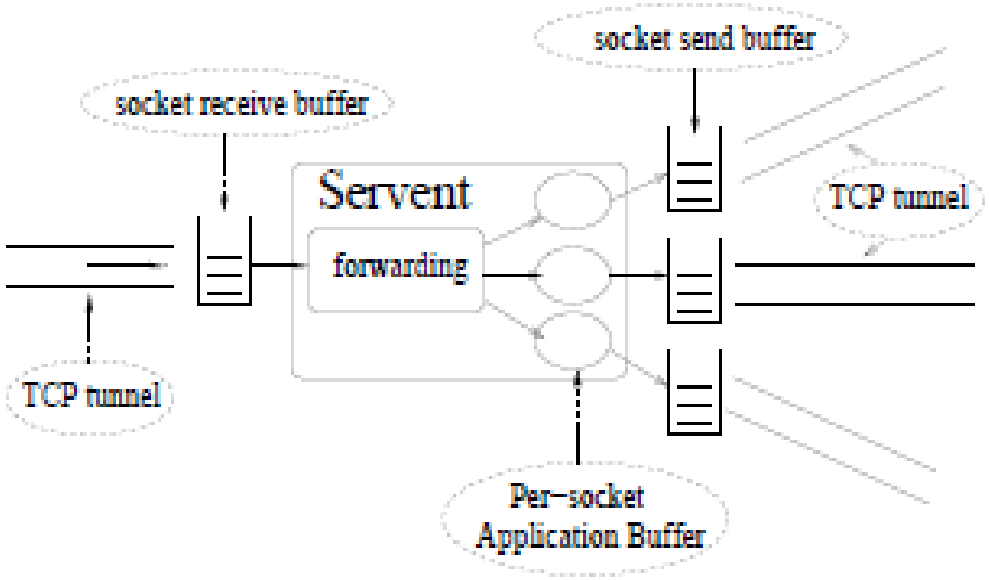
- Il servent si rende conto che vi sono problemi di latenza su una connessione. Soluzioni semplici:
  - chiudere la connessione
    - provoca una continua modifica dell'overlay, messaggi persi, sceta di nuovi vicini
  - eliminare in modo casuale alcuni messaggi diretti verso quella connessione
    - spreco di banda
  - utilizzare un buffer per i messaggi
    - i buffer possono saturarsi provocando il blocco totale del servent
    - la banda totale della rete non è sufficiente per supportare il traffico
    - la rete Gnutella entra in uno stato di meltdown

MECCANISMI DI CONTROLLO DEL FLUSSO

Politica più sofisticata:

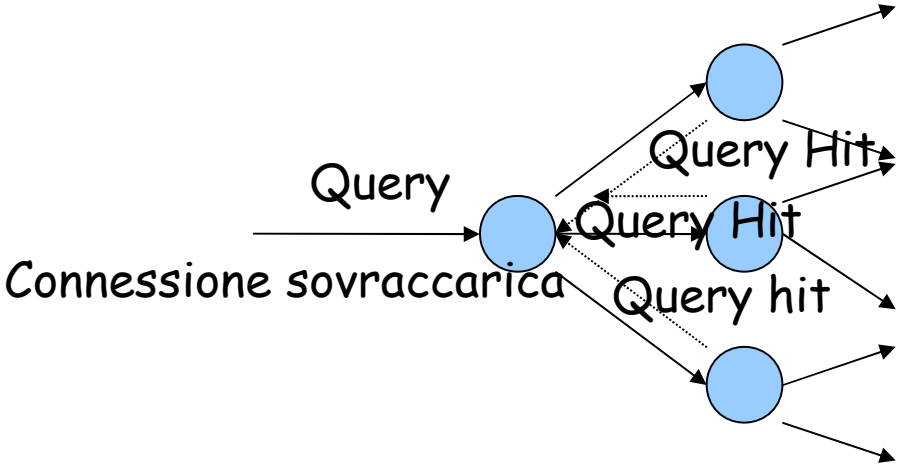
- associare coda di uscita ad ogni connessione
- dimensione coda 150% dimensione del massimo messaggio
- stabilire un valore soglia (es: 25%) rispetto alla sua capacità massima
- se la dimensione della coda supera la soglia,
  - si innesca la modalità di controllo del flusso
  - in modalità di controllo del flusso, si scartano alcuni messaggi
    - obiettivo: cercare di diminuire la 'pressione' su code congestionate
  - si rimane in modalità controllo del flusso finchè la dimensione della coda non scende al di sotto della soglia

CONTROLLO DEL FLUSSO: ARCHITETTURA



PRIORITA' DEI MESSAGGI

- Quali messaggi scartare in modalità di controllo del flusso?



- se si accetta una query da una connessione sovraccarica e si propaga quella query in broadcast, in seguito sarà necessario inviare un insieme R di risultati (query hit) per quella query sulla stessa connessione
- la cardinalità di R può essere potenzialmente elevata
- Si scarta quella query e le successive provenienti dalla connessione congestionata. La ricerca dei risultati per quelle query prosegue comunque lungo altri cammini

PRIORITA' DEI MESSAGGI

Definire una priorità per i messaggi:

- messaggi inviati in flooding (Query e Ping): più alto è il numero di hop, più bassa la priorità
  - le query inviate dal nodo stesso hanno la priorità massima

- messaggi singoli (Query-Hit e Pong): la priorità è direttamente proporzionale al numero di hop percorsi dal messaggi
  - massimizza l'utilizzazione complessiva della banda.
  - quando il messaggio ha attraversato diversi link se viene scartato, la banda utilizzata per la sua trasmissione è completamente sprecata

Priorità per tipo di messaggio, definita in ordine decrescente come segue:

- Push, Query Hit, Pong, Query, Ping

Quando un messaggio riempie la coda oltre il 100%, si eliminano dalla coda i messaggi di priorità inferirore

L'ALGORITMO SACHRIFC

Algoritmo di controllo di flusso utilizzato da diversi client Gnutella (LimeWire, BearShare) e sviluppato con i seguenti obiettivi:

- cercare di eliminare il minor numero possibile di messaggi a meno che la banda non sia eccessivamente limitata.
- minimizzare i ritardi per i messaggi: cercare di evitare di bufferizzare troppo a lungo un messaggio
- Preferire i messaggi nel seguente ordine:
  - Push, QueryHit, Query, Pong, Ping
- evitare che un certo tipo di messaggio possa dominare il traffico.
  - per esempio, un flusso di Push non dovrebbe impedire a tutte le Query Hit di essere instradate.
- favorire Query Hit meno popolari rispetto a quelli più popolari
  - esempio: 10 repliche per la query Q1, 1000 repliche per la query Q2. E' meglio inoltrare un query hit per Q1

L'ALGORITMO SACHRIFC

utilizza

- un buffer per ogni connessione

- ciascun buffer è diviso a sua volta in 5 strutture separate, una per ogni tipo di messaggio
  - i QueryHit sono ordinate in modo crescente secondo il volume GUID (numero di risposte generate per quel GUID)
  - gli altri tipi di messaggio sono ordinati in modo decrescente rispetto al timestamp
    - per inserire un messaggio M di tipo T in un buffer B

```
if B[T] isFull()
    B[T].removeTail();
B[T].insertHead(M)
```

    - strategia LIFO

L'ALGORITMO SACHRIFC

- invia i messaggi in coda attraverso un strategia di tipo Round Robin.

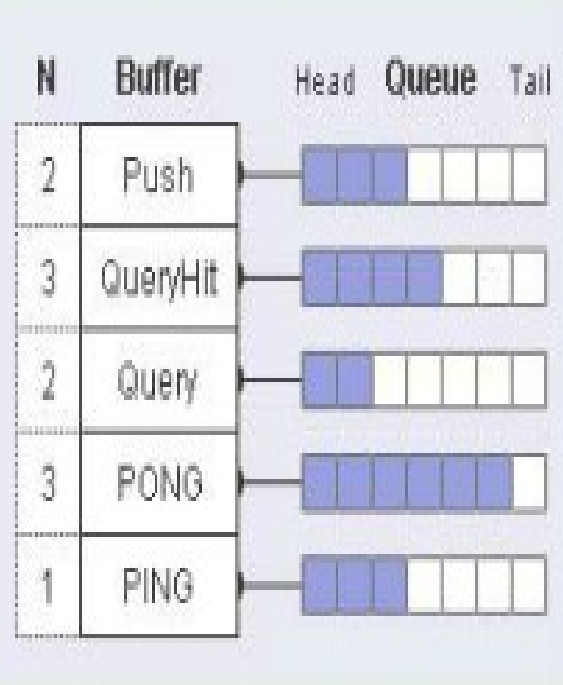
```
forEach (Q_i in B){
        for (j=0; j<min (|Q_i|, N_i); j++){
            M:= Q_i.head();
            if (M.isOld()) M.drop()
            else M.Send();}}
```

B buffer dei messaggi, Q_i coda$\in$ B

- N_i è un valore che indicare la quantità di messaggi da inviare ad ogni ciclo, per ogni tipo di messaggio (cioè per ogni tipo di coda)
  - modificando il rapporto tra i diversi N_i si può dare diversa priorità ai diversi tipi di messaggi
- `IsOld( )` valuta se quel messaggio è obsoleto

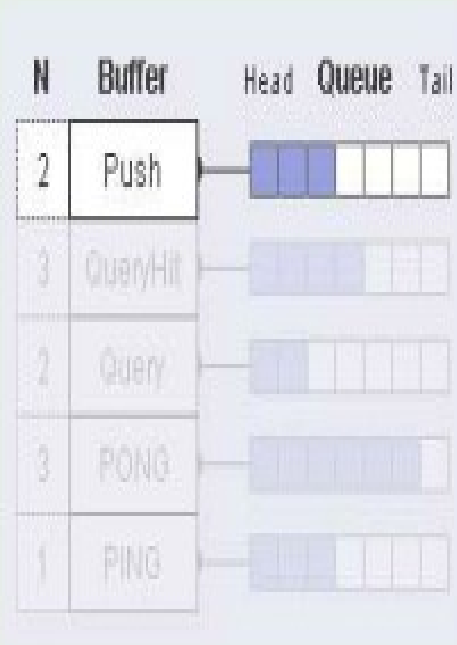
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



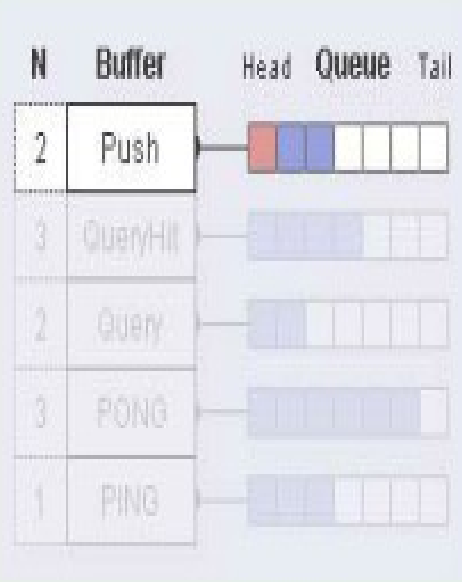
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



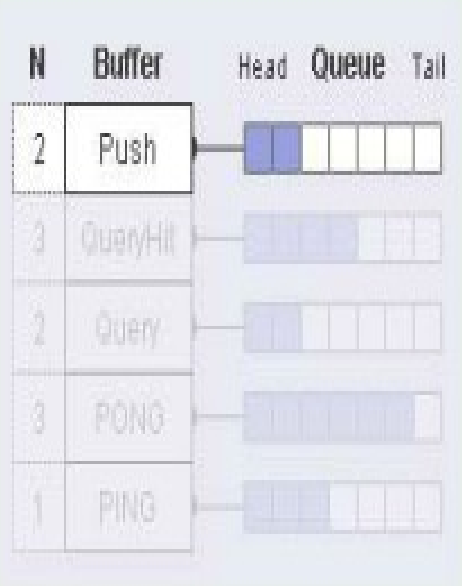
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



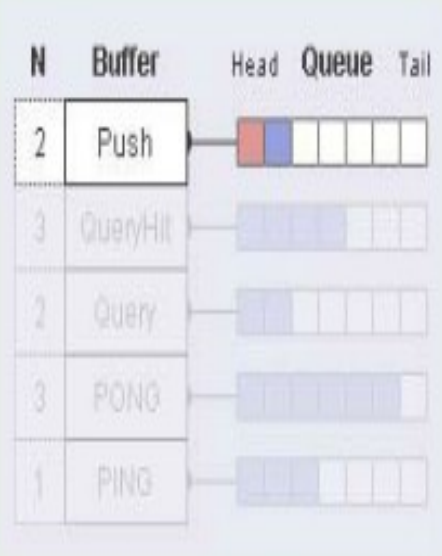
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



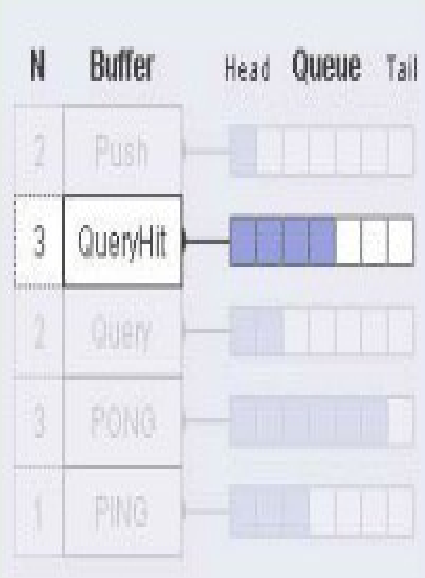
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



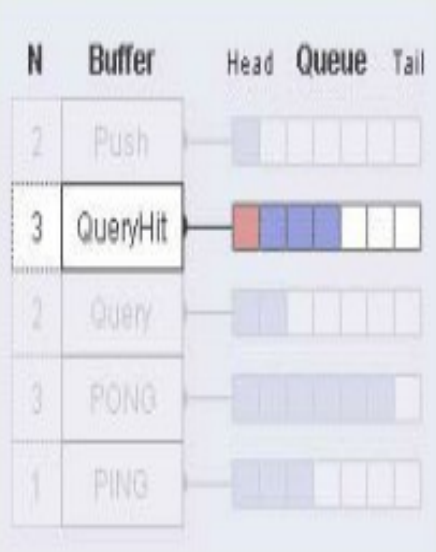
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



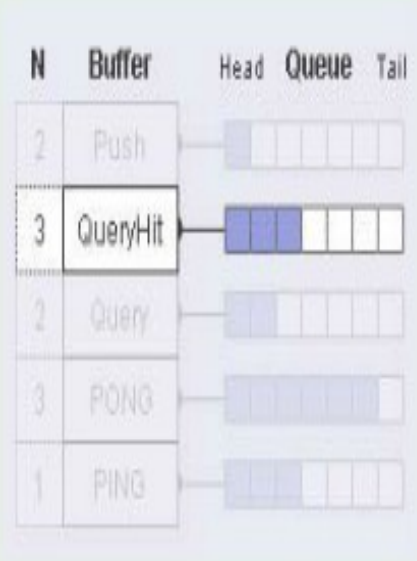
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



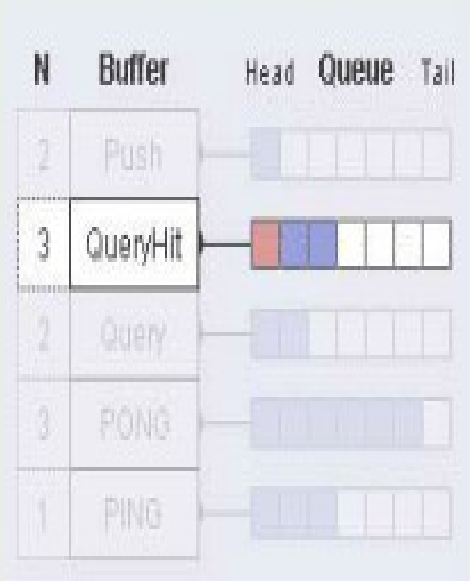
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



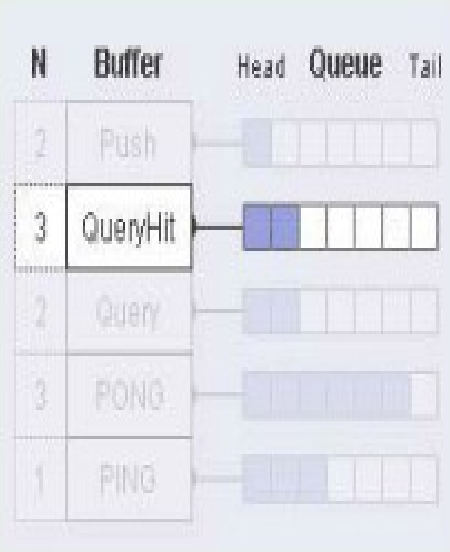
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



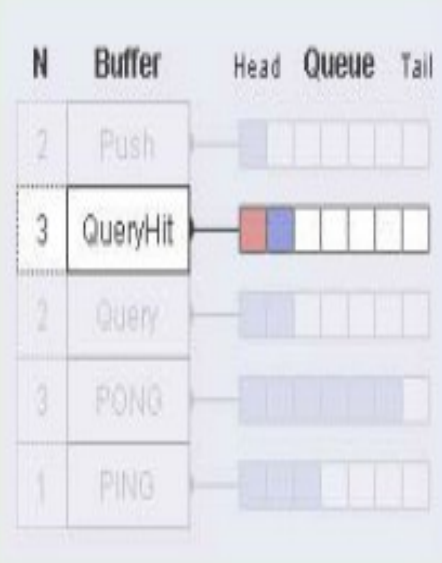
L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```



L'ALGORITMO SACHRIFC

```
foreach(Qi in B) {
    for (j = 0; j < min(|Qi|, Ni); j++) {
        M = Qi.head();
        if (M.isOld()) M.drop();
        else M.send();
    }
}
```

L'ALGORITMO SACHRIFC

- Il fattore N serve per dare diverse priorità diverse ai diversi tipi di messaggio
  - modificando il rapporto tra i diversi Ni si possono modificare le priorità
    - Push a priorità più alta, ma più rari
    - Ping/pong più comuni, ma a priorità più bassa
  - stabilire i fattori Ni in modo da evitare l'overflow di un buffer
- Perchè LIFO?
  - diminuire la permanenza media dei messaggi in coda
- QueryHit ordinate in base a popolarità della query

LIFO: SHORTEST MESSAGES FIRST

- messaggi ricevuti da una connessione C1 vengono inviati a C2

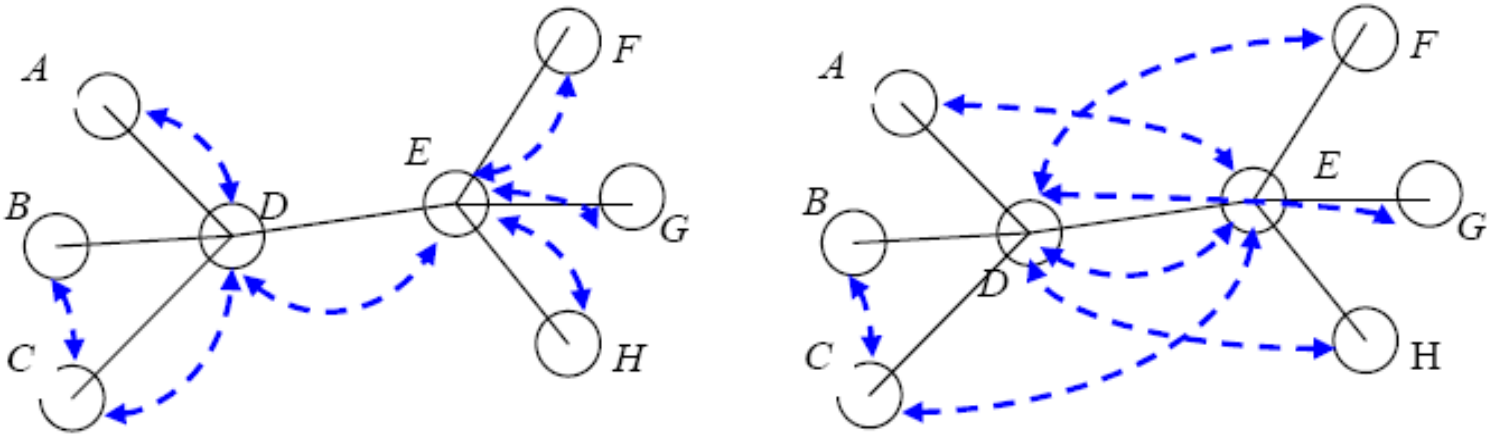
  scenario: C1 (infinita) -> Router -> C2 (1 KB/s)

- inizialmente nessun messaggio viene scartato
- successivamente, si riceve da C1 un messaggio di 10 k, l'host legge il messaggio a 1kB/s e quando il messaggio è completo lo invia su C2
  - Per inviarlo a C2 sono necessari 10 secondi
  - con FIFO
    - si continuano a leggere messaggi da C1 e si mettono in coda
    - la latenza di tutti i messaggi successivi è di 10 secondi
  - con LIFO
    - se arriva un nuovo messaggio di 1k da C1: viene inserito in testa ed inviato immediatamente su C2
    - Il blocco di 10 k viene inviato successivamente, se non obsoleto

IL PROBLEMA DELLO ZIG ZAG ROUTING

- L'overlay può differire notevolmente dalla struttura della rete fisica
- Zig Zag Routing: servent collocati in Europa ed negli States possono rimbalzare il messaggio più volte attraverso l'Atlantico
- Maggiore è 'il mismatch' tra l'overlay e la rete fisica sottostante, maggiore è l'utilizzo della struttura sottostante
- Utilizzo di una gran quantità di banda tra Europo e States
- Ritardi di comunicazione
- Proposta: adattare la topologia dell'overlay network a quella della rete fisica

IL PROBLEMA DELLO ZIGZAG ROUTING



- i collegamenti
  - tratteggiati in blu definiscono l'overlay network
  - continui in nero defiscono la struttura della rete fisica

- supponiamo che A invii un messaggio a tutti gli altri peer

  overlay di sinistra: il messaggio raggiunge tutti gli altri nodi dell'overlay attraversando il link D-E una sola volta

  overlay di destra, il messaggio deve attraversare il link fisico D-E più volte per raggiungere tutti gli altri nodi della rete