I sistemi peer to peer (P2P)
Una Introduzione

What Does Peer-to-Peer Mean?

- A generic name for systems in which **peers communicate directly and not through a server**
- Characteristics:
  - decentralized
  - self-organizing
  - distributed systems
  - all or most communication is symmetric
  - typically, large scale systems (up to millions)
  - Virtual organization

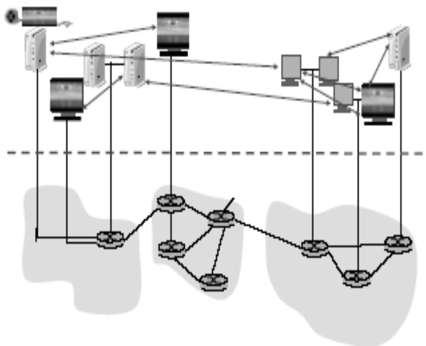
Typical Characteristics –details

- Large Scale: lots of nodes (up to millions)
- Dynamicity: frequent joins, leaves, failures
- Little or no infrastructure
  - No central server
- Symmetry: all nodes are "peers" – have same role
  - Not always true – "all nodes are equal, but some node are equal more"
- Communication possible between every pair of nodes (Internet)
  - Not always true – NAT, FW

P2P Applications

- File sharing (music, movies, …)
- Distributed computing
- VoIP - Skype
- Collaboration

P2P Networking



Focus on the apps

▷ 5

File Sharing Services

- **Publish** – insert a new file into the network
- **Lookup** – given a file name X, find the host that stores the file
- **Retrieval** – get a copy of the file
- **Join** – join the network
- **Leave** – leave the network

▷ 6

The main problem - Lookup

- Given a data item X, stored at some set of nodes, find it
- The main challenge
  - Do it in a **reliable, scalable and efficient** way
  - Despite the dynamicity and frequent changes

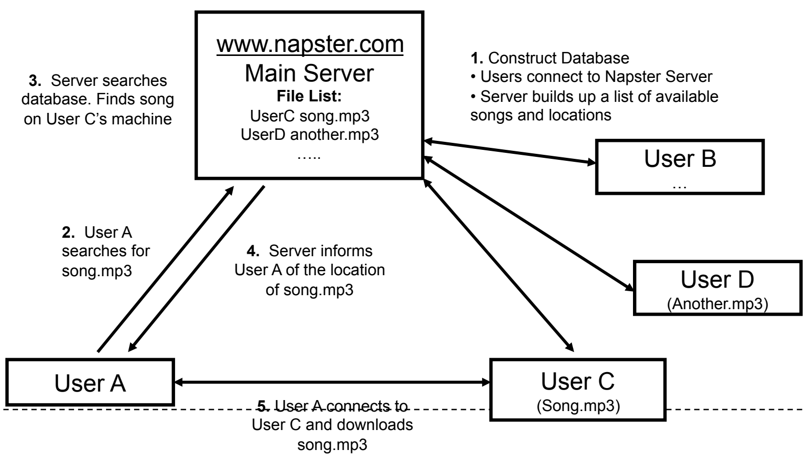
Take 1 – Napster (Jan '99)

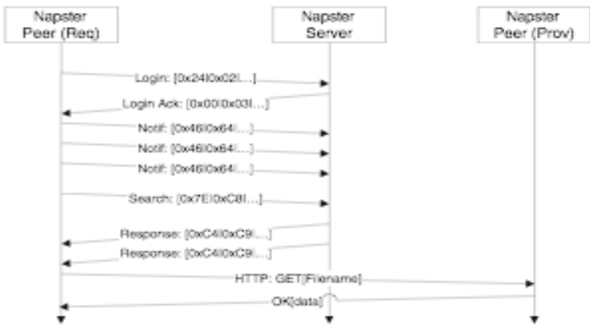
- Client – Server architecture (not P2P)
- **Publish** – send the key (file name) to the server
- **Lookup** – ask the server who has the requested file. The response contains the address of a node/nodes that hold the file
- **Retrieval** – get the file directly from the holder
- **Join** – send you file list to the server
- **Leave** – cancel your file list at the server

Take 1 – Napster (continued)

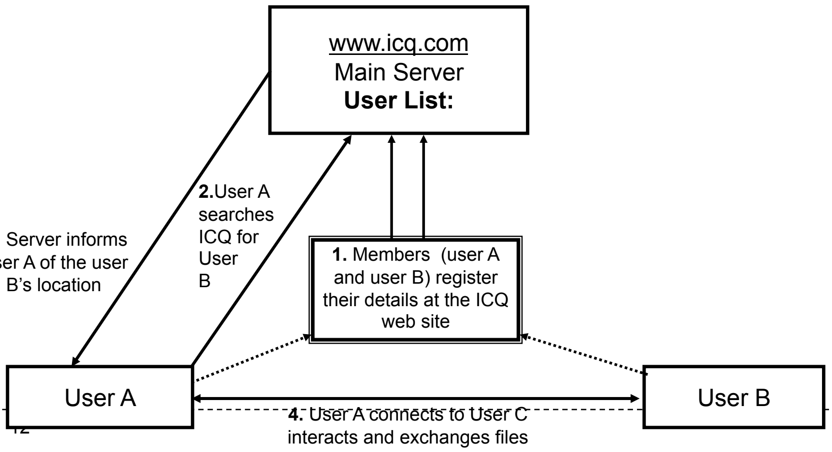
- Advantages
  - Low message overhead
  - Minimal overhead on the clients
  - 100% success rate (if the file is there, it will be found)
- Disadvantages
  - Single point of failure
  - Not scalable (server is too busy)

File Sharing with Napster





ICQ: Instant Messaging



Naspter

- centralized server:
  - single logical point of failure
  - potential for congestion
  - Napster "in control" (freedom is an illusion)
- no security:
  - passwords in plain text
  - no authentication
  - no anonymity

Lets distribute the server

- Every node is connected to every node
  - No scalable at all
- Every node is connected to a number of peers
  - Can communicate directly with immediate neighbors
  - Can communicate with other nodes through my direct neighbors
  - This is called **overlay**

Overlay Networks

- **Overlay is a virtual structure imposed over the physical network** (e.g., the Internet)
  - over the Internet, there is an (IP level) unicast channel between every pair of hosts
  - an overlay uses a fixed subset of these
  - nodes that have the capability to communicate directly with each other do not use it
- Allows scalable symmetric lookup algorithms

15

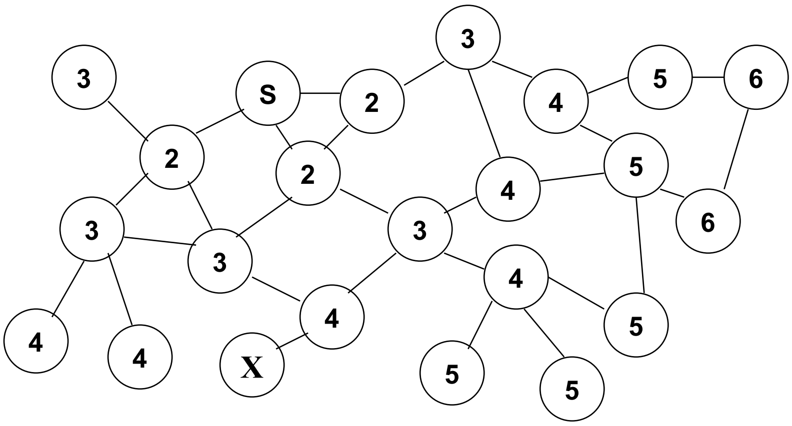
Take 2 - Gnutella (March '00)

- Build a decentralized unstructured overlay
  - each node has several neighbors
- **Publish** – store the file locally
- **Lookup** – check local database. If X is known return, if not, ask your neighbors. TTL limited.
- **Retrieval** – direct communication between 2 nodes
- **Join** – contact a list of nodes that are likely to be up, or collect such a list from a website.
  - Random, unstructured overlay
- What is the communication pattern?

flooding

16

Resolve Query by Flooding



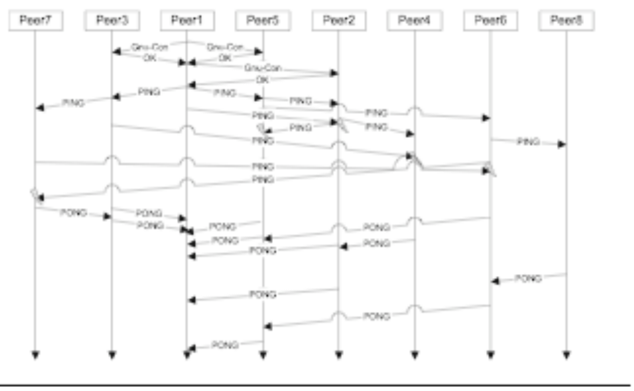
~~Time-To-Live (TTL)=5 would have been enough~~

Take 2 – Gnutella (continued)

- Advantages
  - Fast lookup
  - Low join and leave overhead
  - Popular files are replicated many times, so lookup with small TLL will usually find the file
    - Can choose to retrieve from a number of sources
- Disadvantages
  - Not 100% success rate, since TTL is limited
  - Very high communication overhead
    - Limits scalability
    - But people do not care so much about wasting bandwidth
  - Uneven load distribution

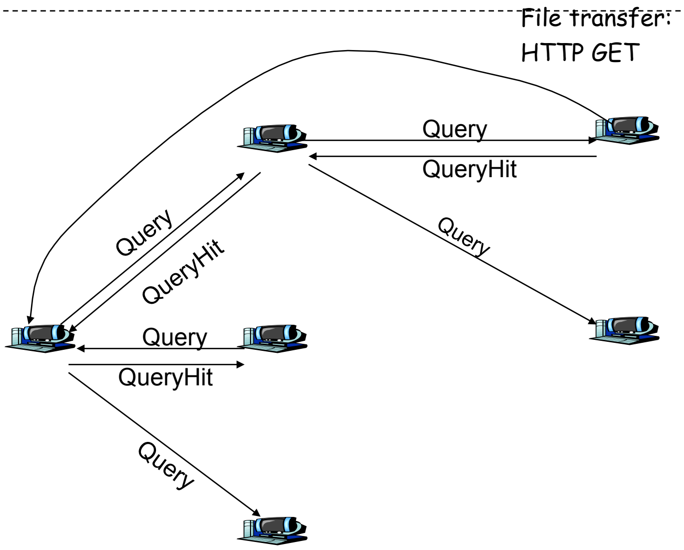
Gnutella

- Le query sono trasmesse sulle con. TCP
- Network exploration: PING-PONG
- Query
  - DescriptorID, PayloadID, TTL, Hops, Length, Paylod
- Query Forward: pari inoltrono i msg nella rete overlay
- QueryHit : risposta alla query lungo il cammino “inverso” della rete overlay
  - Hits, Port, IP, Speed, ResultSet, NodeId



Gnutella

❒



Gnutella: Peer joining

1. Joining peer X must find some other peer in Gnutella network: use list of candidate peers
2. X sequentially attempts to make TCP with peers on list until connection setup with Y
3. X sends Ping message to Y; Y forwards Ping message.
4. All peers receiving Ping message respond with Pong message
5. X receives many Pong messages. It can then setup additional TCP connections

Take 3 - FastTrack, KaZaA, eDonkey

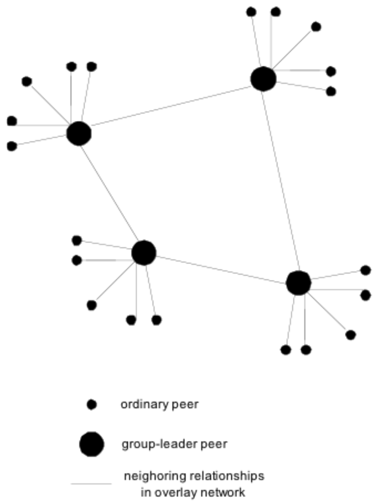
- Improve scalability by introducing a hierarchy
  - 2 tier system
    - super-peers: have more resources, more neighbors, know more keys
    - clients: regular/slow nodes
  - Client can decide if it is a super-peer when connecting
  - Super-peers accept connection from clients and establish connections to other super-peers
  - Search goes through super-peers

Take 3 - FastTrack, KaZaA, eDonkey (continued)

- Advantages
  - More stable than Gnutella. Higher success rate
  - More scalable
- Disadvantages
  - Not "pure" P2P
  - Still high communication overhead

KaZaA

- Strutturazione dei peer
  - Peer = group leader o e' associatto a un group leader.
  - Peeer -- Group leader TCP Con..
  - TCP cons tra coppie di group leader.
- Group leader: sono una sorta di directoly centralizzata per i peer associati al gruppo.



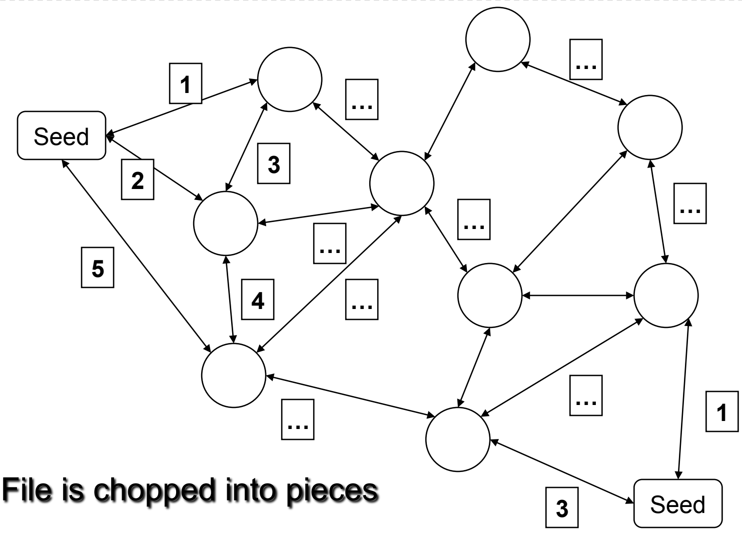
KaZaA: Querying

- Each file has a hash and a descriptor
- Client sends keyword query to its group leader
- Group leader responds with matches:
  - For each match: metadata, hash, IP address
- If group leader forwards query to other group leaders, they respond with matches
- Client then selects files for downloading
  - HTTP requests using hash as identifier sent to peers holding desired file

Bit Torrent

- New approach
  - Content distribution
- Main Goal:
  - Replicate file to large number of clients
- Each file is brocken into chuncks (torrent file details metadata)
  - Size chuncks
  - Tracker (server which keeps track of the current acrive clients)

BitTorrent



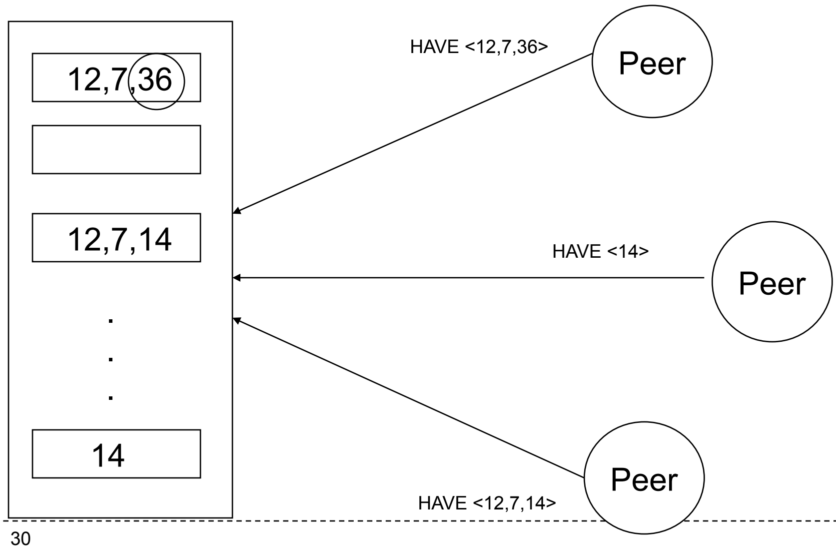
❖ File is chopped into pieces

How BitTorrent works

- Downloaders exchange blocks with each other
- Tracker keeps track of connected peers
- Salient features
  - Which block to download first?
    - Locally rarest block
  - Which peers should I upload blocks to?
    - Tit-for-tat: peers which give best download rates

29

Locally rarest block



30

Structured Lookup Overlays

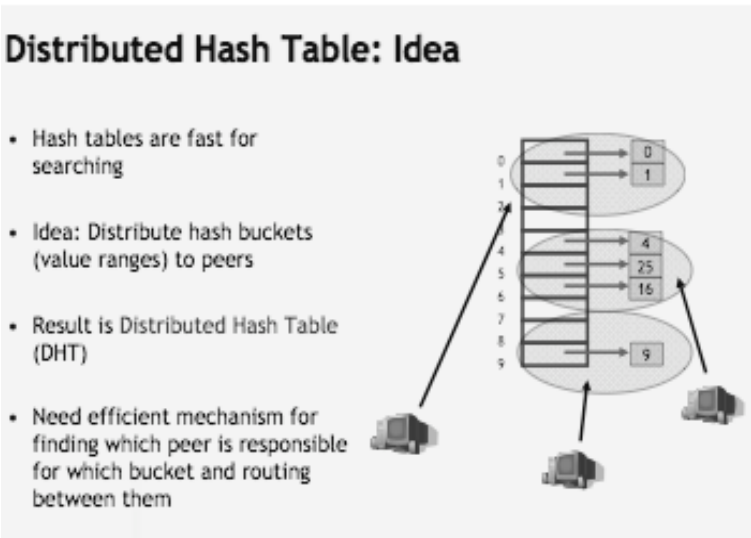
- Structured overlay – data stored in a defined place, search goes on a defined path
- Implement Distributed Hash Table (DHT) abstraction
- Symmetric, no hierarchy
- Decentralized self management
- Many recent academic systems –
  - CAN, Chord , D2B, Kademlia, Koorde, Pastry, Tapestry, Viceroy, OverNet (based on the Kademlia algorithm)

Reminder: Hashing

- Data structure supporting the operations:
  - void **insert**(key, item)
  - item **search**(key)
- Implementation uses *hash function* for mapping keys to array cells
- Expected search time O(1)
  - provided that there are few collisions

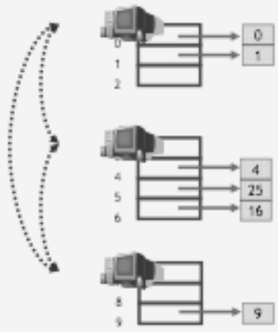
Distributed Hash Tables (DHTs)

- Nodes store table entries
  - Key ->IP of the node currently responsible for this key
- **lookup(key)**
  - returns the IP of the node responsible for the key
  - **key** usually numeric (in some range)
- Requirements for an application being able to use DHTs:
  - data identified with unique keys
  - nodes can (agree to) store keys for each other
    - location of object or actual object



DHT: Principle

- In a DHT, each node is responsible for one or more hash buckets
  - As nodes join and leave, the responsibilities change
- Nodes communicate among themselves to find the responsible node
  - Scalable communications make DHTs efficient
- DHTs support all the normal hash table operations



35

Summary of DHT Principles

- Hash buckets distributed over nodes
- Nodes form an overlay network
  - Route messages in overlay to find responsible node
- Routing scheme in the overlay network is the difference between different DHTs
- DHT behavior and usage:
  - Node knows “object” name and wants to find it
    - Unique and known object names assumed
  - Node routes a message in overlay to the responsible node
  - Responsible node replies with “object”
    - Semantics of “object” are application defined

36

Using the DHT Interface

- How do you publish a file?
- How do you find a file?

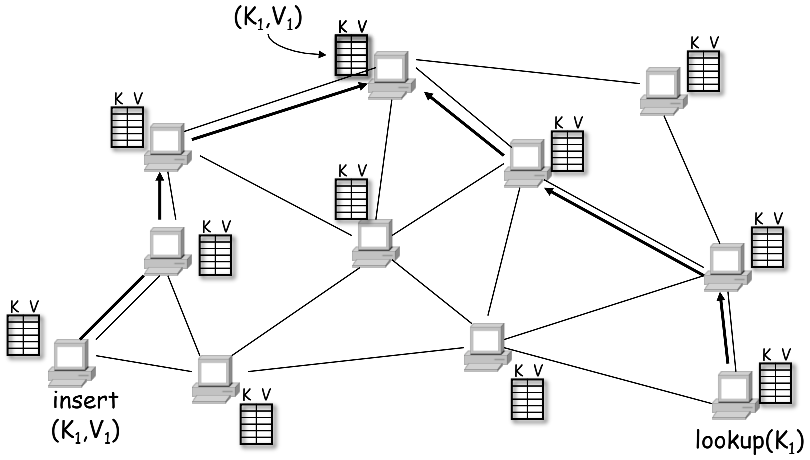
37

What Does a DHT Implementation Need to Do?

- Map keys to nodes
  - needs to be dynamic as nodes join and leave
- Route a request to the appropriate node
  - routing on the overlay

38

Lookup Example



Mapping Keys to Nodes

- Keys and nodes are mapped to **the same identifier space**
  - **NodeID=hash(node's IP address)**
  - **KeyID=hash(key).** Key is a file name
  - *m*-bit identifier
  - Cryptographic hash function (e.g., SHA-1)
- Goal: load balancing, achieved by hashing
- Typical DHT implementation:
  - **map key to node whose id is "close" to the key** (need distance function).

Routing Issues

- Each node must be able to forward each lookup query to a node closer to the destination
- Maintain routing tables adaptively
  - each node knows some other nodes
  - must adapt to changes (joins, leaves, failures)
  - Goals
    - Efficient - use as few nodes as possible on the routing path
    - …

41

Handling Join/Leave

- When a node joins it needs to assume responsibility for some keys
  - In order for future lookups to succeed
  - ask the application to move these keys to it
  - how many keys will need to be moved?
- When a nodes fails or leaves, its keys have to be moved to others
  - what else is needed in order to implement this?

42

P2P System Interface

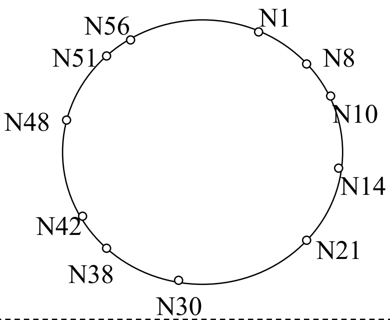
- Lookup
- Join
- Move keys

Chord

Stoica, Morris, Karger, Kaashoek, and Balakrishnan
2001

Chord Logical Structure

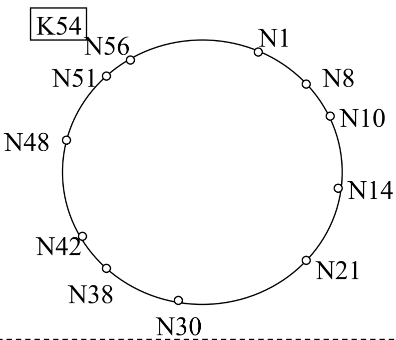
- *m*-bit ID space (2^m IDs), usually *m*=160.
- Think of nodes as organized in a logical ring according to their IDs.



45

Assigning Keys to Nodes

- KeyID *k* is assigned to first node whose NodeID >= k (clockwise from k)
  - Denoted: *successor(k)*

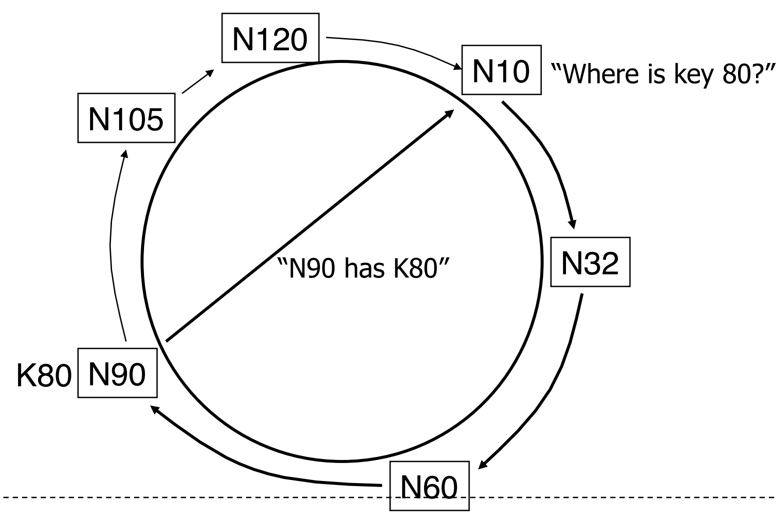


46

Simple Routing Solutions

- Each node knows only its successor
  - routing around the circle
  - O(N) routing
  - O(1) memory
- Each node knows all other nodes
  - O(1) routing
  - O(N) memory

Routing around the circle



Routing around the circle - code

Lookup(my-id, key-id)

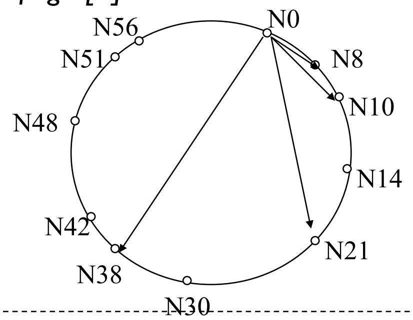
```
q = my_successor
if my-id < key-id < q
  return my_successor          // done
else
  call Lookup(id) on q     // next hop
```

- Correctness depends only on successors

49

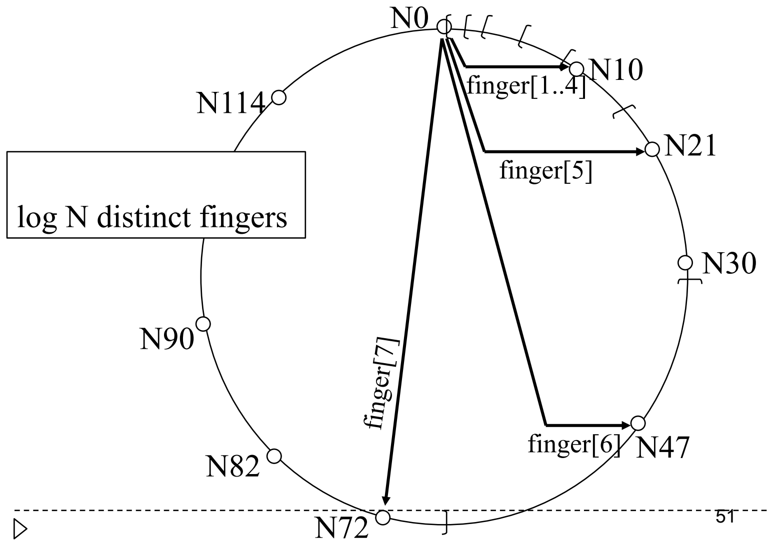
Chord Fingers

- Each node has "fingers" to nodes ½ way around the ID space from it, ¼ the way…
- *finger[i]* at *p* contains *successor*($p+2^{i-1}$)
- *successor* is *finger[1]*



50

Example: Chord Fingers

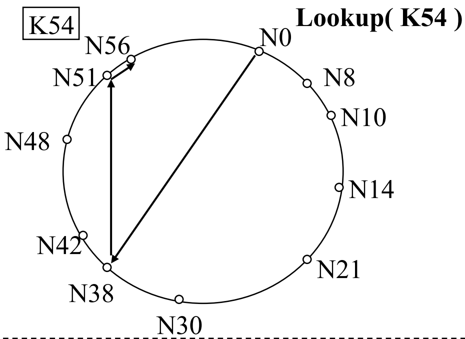


Chord Data Structures

- At Each Node
  - Finger table
  - First finger is successor
  - Predecessor

Forwarding Queries

- Query for key *k* is forwarded to finger with highest ID not exceeding *k*



53

Chord Routing – the code

Lookup(my-id, key-id)

```
look in local finger table for
  highest node q s.t. my-id < q < key-id
if q exists
  call Lookup(id) on node q          // next hop
else
  return my_successor          // done
```

54

Routing Time - O(*log(N)*) steps

- maximum *m* steps
- Assuming uniform node distribution around the circle, the number of nodes in the search space is halved at each step:
  - expected number of steps: **log N**

55

Joining Chord

- Goals?
- Steps:
  - Find your successor
  - Initialize finger table and predecessor
  - Notify other nodes that need to change their finger table and predecessor pointer
    - $O(\log^2 n)$
  - Learn the keys that you are responsible for; notify others that you assume control over them

56

Join Algorithm: Take II

- Observation: for correctness, successors suffice
  - fingers only needed for performance
- Upon join, update successor only
- Periodically,
  - check that successors and predecessors are consistent
  - fix fingers

Failure Handling

- Periodically fixing fingers
- List of *r* successors instead of one successor
- Periodically probing predecessors

Moving Keys upon Join/Leave

- Left up to the application
- When a node joins, it becomes responsible for some keys previously assigned to its successor
  - local change
  - how many keys should move, on average?
- And what happens when a node leaves?
  - List of *r* successors instead of one successor
  - Replicate keys:
    - Store every key in r successors, instead of only one
- Or do key maintenance periodically

Summary: DHT Advantages

- Peer-to-peer: no centralized control or infrastructure
- Scalability: O(log N) routing, routing tables, join time
- Load-balancing
- Overlay robustness

DHT Disadvantages

- No control where data is stored
- Complex queries are not possible

61

Resources

- http://en.wikipedia.org/wiki/Peer-to-peer

62