

---

## Peer-to-Peer Systems: DHT and Swarming

10/5/2009

1

## Admin.

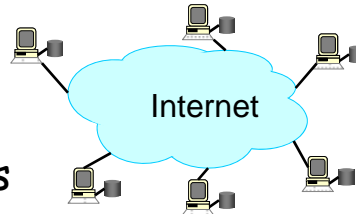
---

- Programming assignment 1
  - New due date: Oct. 12 11:59 pm EDT

2

## Recap: Objectives of P2P

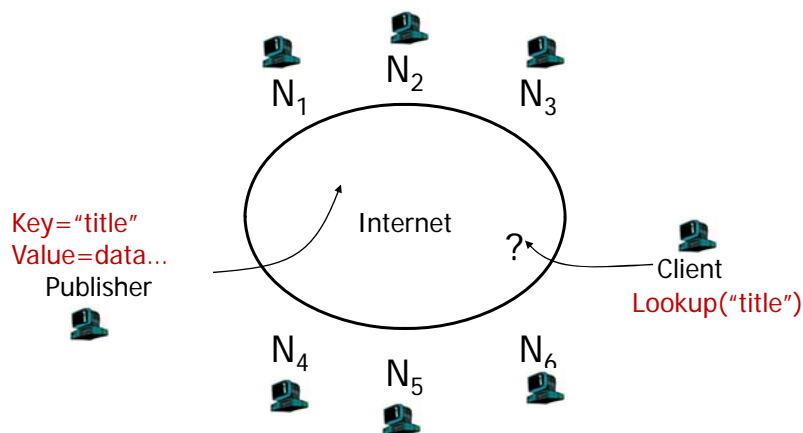
- Share the resources (storage *and* bandwidth) of individual clients to improve scalability/robustness



- Find clients with resources!

3

## Recap: The Lookup Problem



find where a particular document is stored

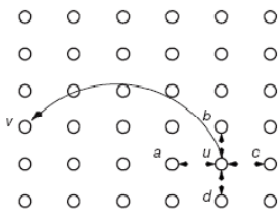
## Recap: The Lookup Problem

- Napster (central query server)
- Gnutella (decentralized, flooding)
- Freenet (search by routing)

5

## Recap: Kleinberg's Result on Distributed Search

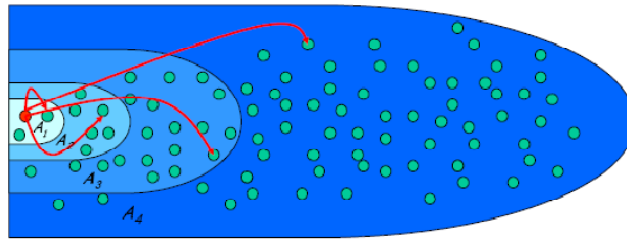
- Question: how many long distance links to maintain so that distributed (greedy) search is effective?
- Assume that the probability of a long link is some ( $\alpha$ ) inverse-power of the number of lattice steps
- Distributed algorithm exists only when probability is proportional to (lattice steps)<sup>-d</sup>, where  $d$  is the dimension of the space



6

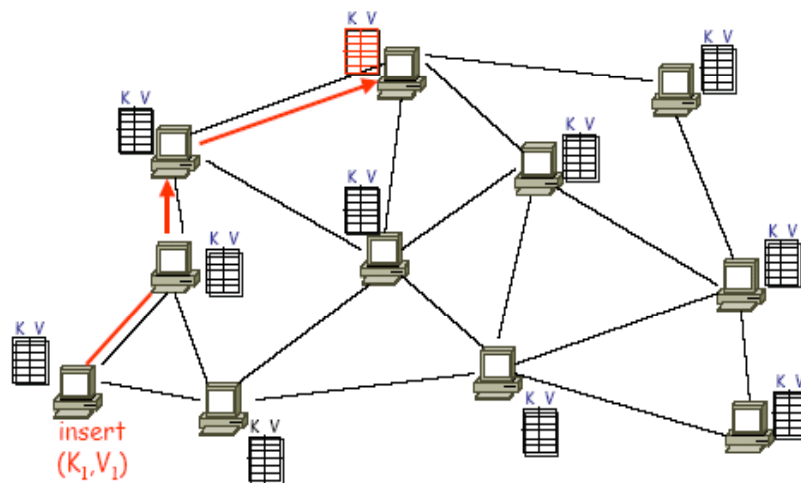
## Recap: Distributed Search

- In other words, a node connects to roughly the same number of nodes in each region  $A_1, A_2, A_4, A_8$ , where  $A_1$  is the set of nodes who are one lattice step away,  $A_2$  is those two steps away, ...



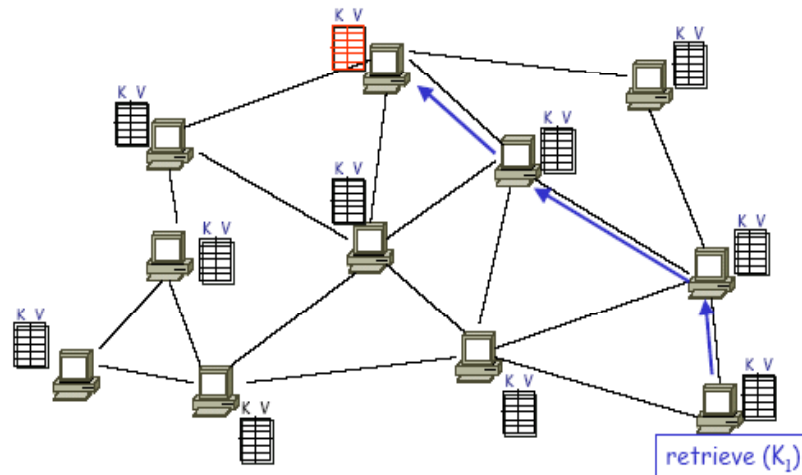
7

## DHT: API



8

## DHT: API



9

## Key Issues in Understanding a DHT Design

- How does it map the key to the internal representation (typically a metric space)?
- Which space is a node responsible?
- How are the nodes linked?

10

## Outline

### □ Admin. and review

#### ➤ *P2P*

##### ➤ *the lookup problem*

- Napster (central query server; distributed data server)
- Gnutella (decentralized, flooding)
- Freenet (search by routing)
- *Content Addressable Network*

11

## CAN

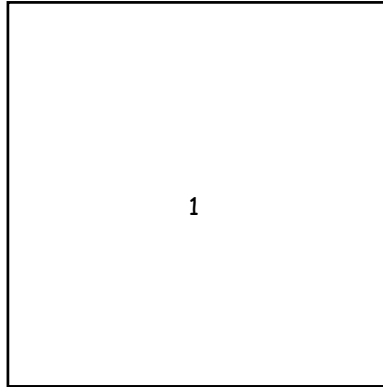
### □ Abstraction

- map a key to a "point" in the *multi-dimensional Cartesian space*
- a node "owns" a *zone* in the overall space
- route from one "point" to another

12

## CAN Example: Two Dimensional Space

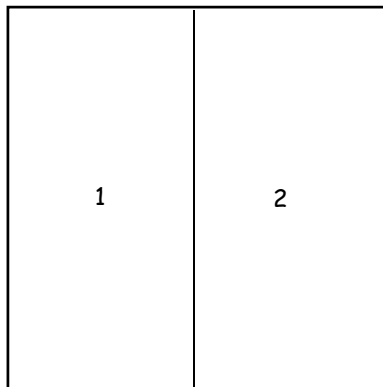
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



13

## CAN Example: Two Dimensional Space

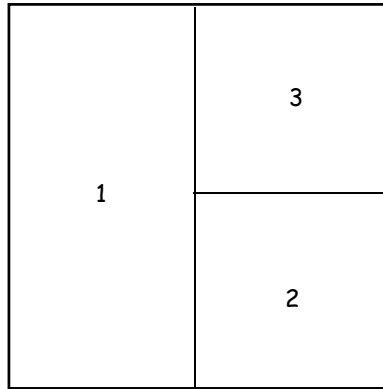
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



14

## CAN Example: Two Dimensional Space

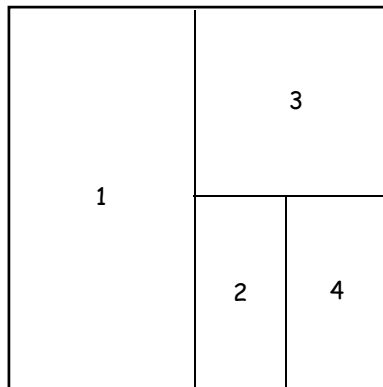
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



15

## CAN Example: Two Dimensional Space

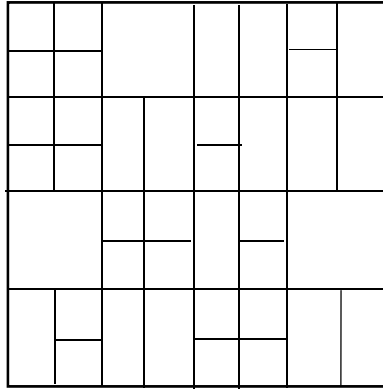
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



16

## CAN Example: Two Dimensional Space

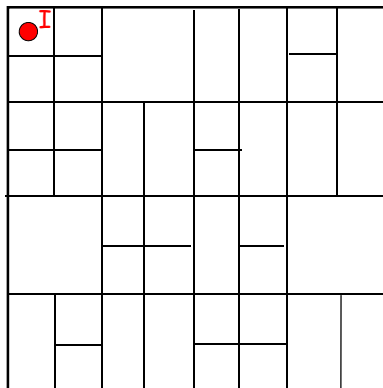
- Space divided among nodes
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1



17

## CAN Insert: Example (1)

node I::insert(K,V)



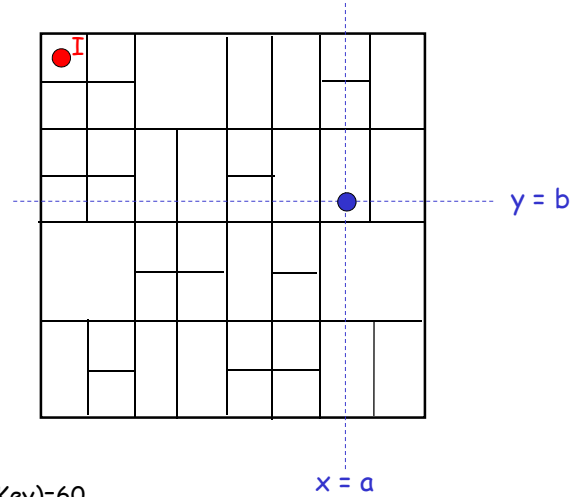
18

## CAN Insert: Example (2)

node I::insert(K,V)

(1)  $a = h_x(K)$

$b = h_y(K)$



Example: Key="Matrix3"  $h(\text{Key})=60$

19

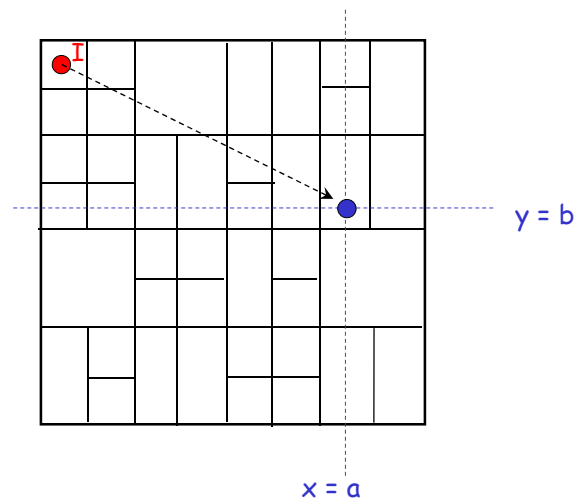
## CAN Insert: Example (3)

node I::insert(K,V)

(1)  $a = h_x(K)$

$b = h_y(K)$

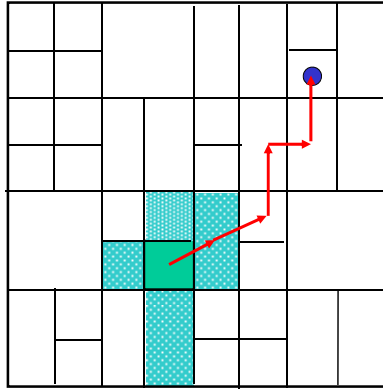
(2)  $\text{route}(K,V) \rightarrow (a,b)$



20

## CAN Insert: Routing

- A node maintains state only for its immediate neighboring nodes
- Forward to neighbor which is closest to the target point
  - a type of greedy, local routing scheme



21

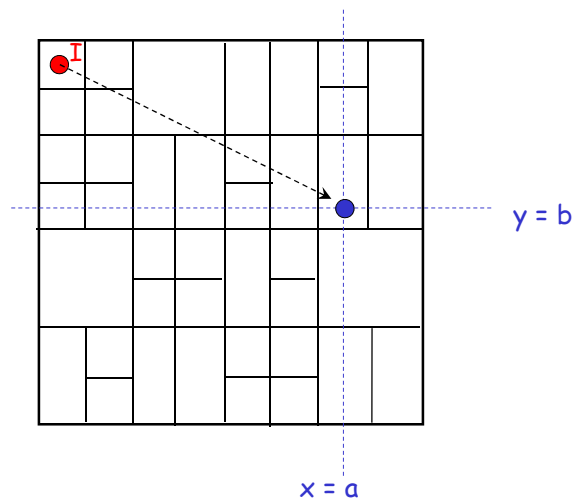
## CAN Insert: Example (4)

node I::insert(K,V)

(1)  $a = h_x(K)$   
 $b = h_y(K)$

(2)  $\text{route}(K,V) \rightarrow (a,b)$

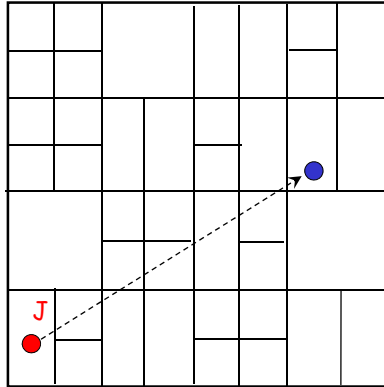
(3)  $(K,V)$  is stored at  $(a,b)$



22

## CAN Retrieve: Example

node J::retrieve(K)



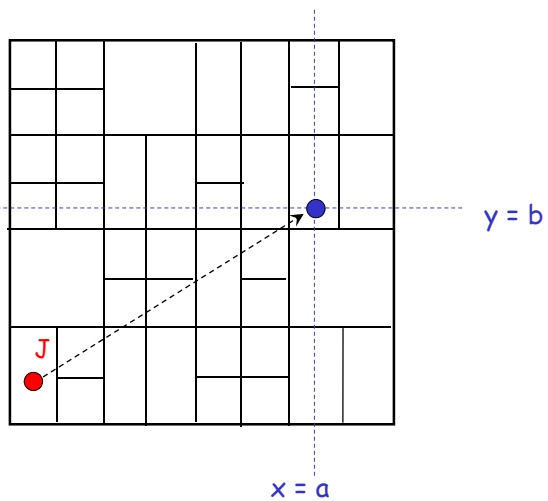
23

## CAN Retrieve: Example

node J::retrieve(K)

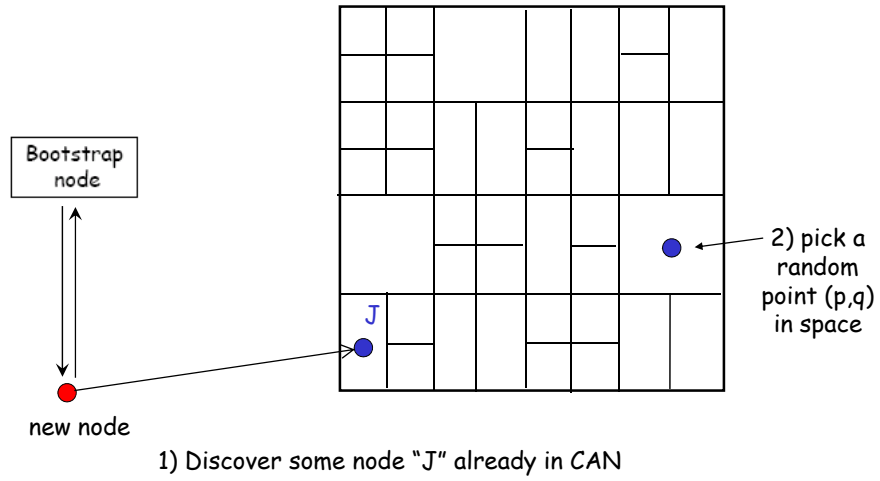
(1)  $a = h_x(K)$   
 $b = h_y(K)$

(2) route "retrieve(K)" to  
(a,b)



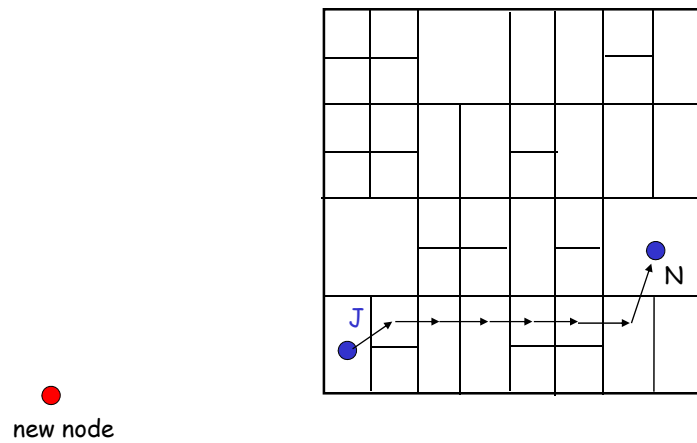
24

## CAN Insert: Join (1)



25

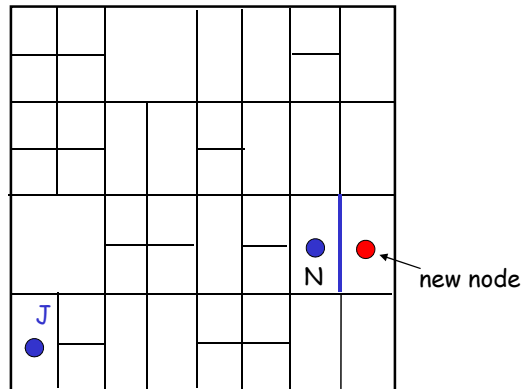
## CAN Insert: Join (2)



26

## CAN Insert: Join (3)

Inserting a new node affects only a single other node and its immediate neighbors



4) split N's zone in half... new node owns one half

27

## CAN Evaluations

- Guarantee to find an item if in the network
- Load balancing
  - hashing achieves some load balancing
  - overloaded node replicates popular entries at neighbors
- Scalability
  - for a uniform (regularly) partitioned space with  $n$  nodes and  $d$  dimensions
  - storage:
    - per node, number of neighbors is  $2d$
  - routing
    - average routing path is  $(dn^{1/d})/3$  hops (due to Manhattan distance routing, expected hops in each dimension is dimension length \*  $1/3$ )
    - a fixed  $d$  can scale the network without increasing per-node state

28

## Outline

- Admin. and review

- *P2P*

- *the lookup problem*

- Napster (central query server; distributed data server)
- Gnutella (decentralized, flooding)
- Freenet (search by routing)
- Content addressable networks
  - *Chord (search by routing/consistent hashing)*

29

## Chord

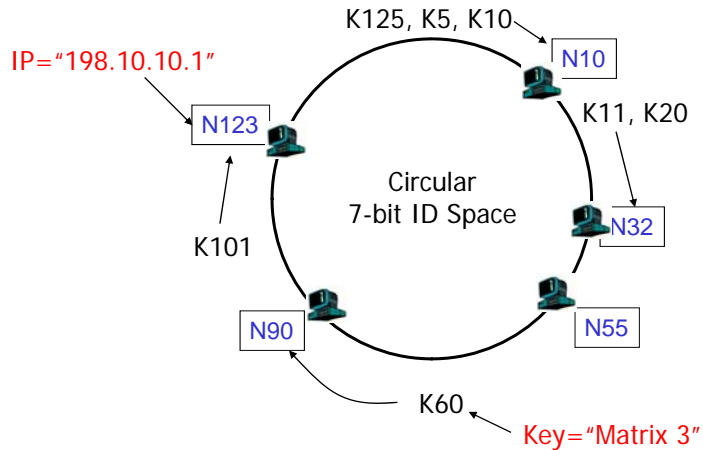
- Space is a ring

- Consistent hashing:  $m$  bit identifier space for both keys and nodes

- key identifier =  $\text{SHA-1}(\text{key})$ , where  $\text{SHA-1}()$  is a popular hash function,  
Key="Matrix3" → ID=60
- node identifier =  $\text{SHA-1}(\text{IP address})$ 
  - IP="198.10.10.1" → ID=123

30

## Chord: Storage using a Ring

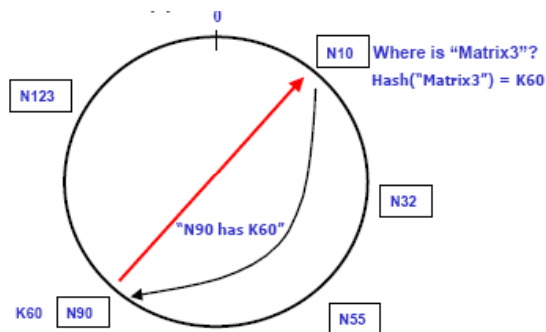


- A key is stored at its successor: node with next higher or equal ID

31

## How to Search: One Extreme

- Every node knows of every other node
- Routing tables are large  $O(N)$
- Lookups are fast  $O(1)$

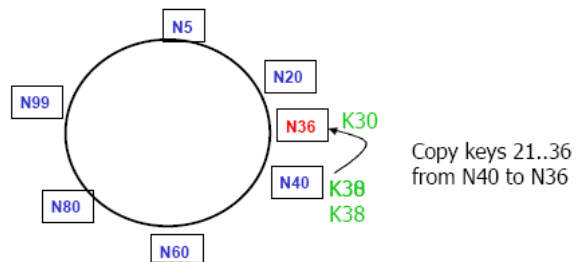


32



## Joining the Ring

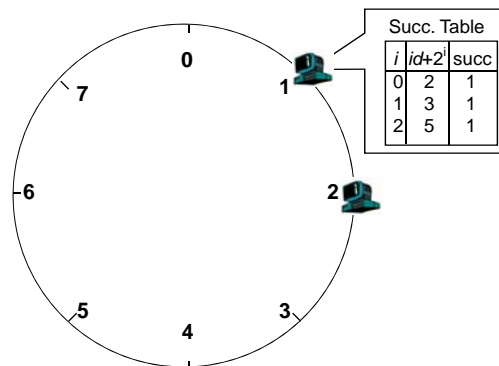
- use a contact node to obtain info
- transfer keys from successor node to new node
- updating fingers of existing nodes



35

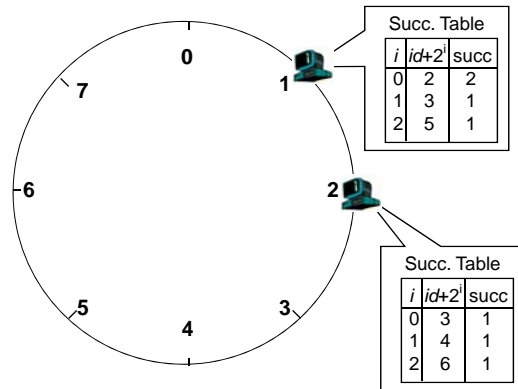
## DHT: Chord Node Join

- Assume an identifier space [0..8]
- Node n1 joins



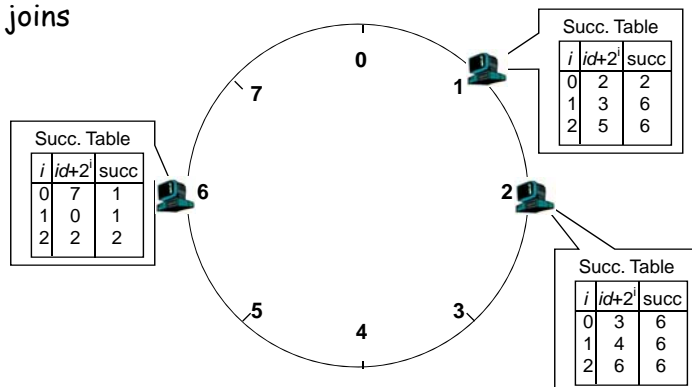
## DHT: Chord Node Join

□ Node n2 joins



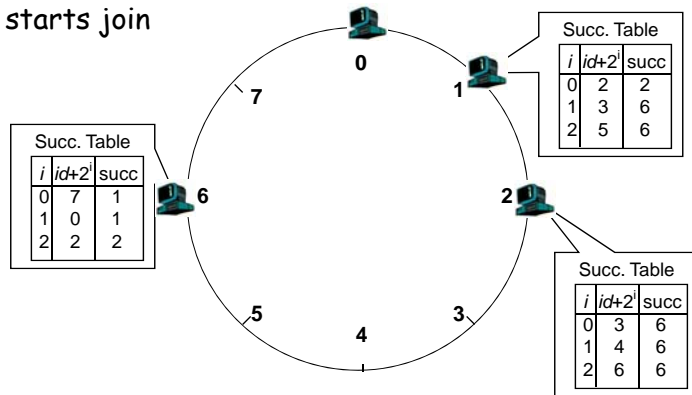
## DHT: Chord Node Join

□ Node n6 joins



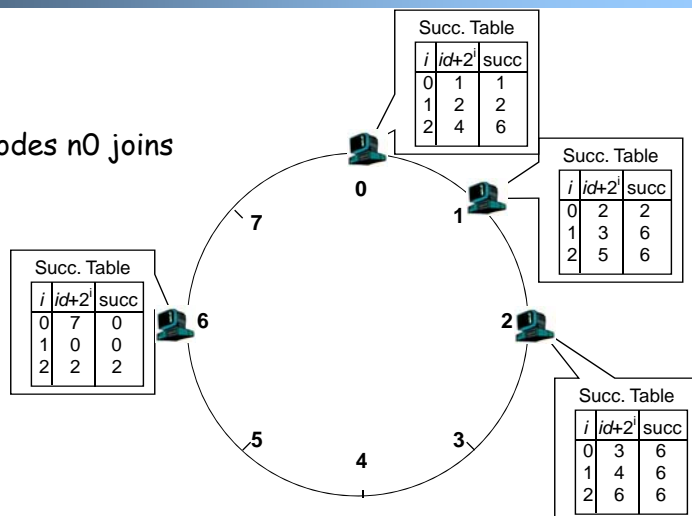
## DHT: Chord Node Join

- Node n0 starts join



## DHT: Chord Node Join

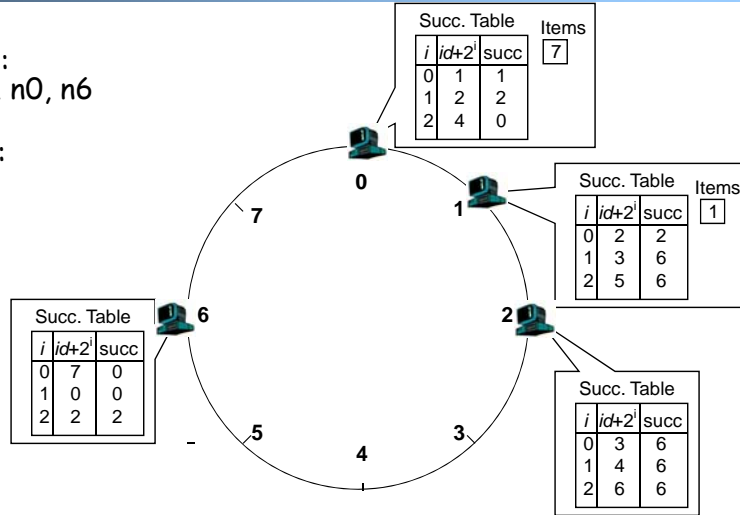
- After Nodes n0 joins



## DHT: Chord Insert Items

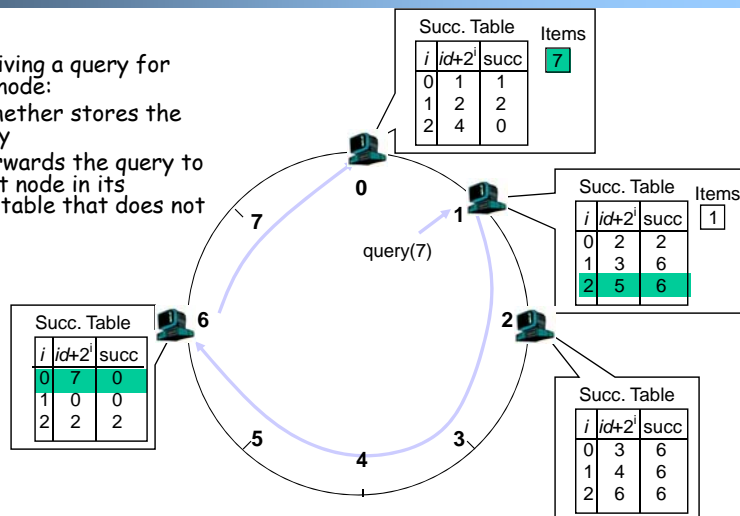
- Nodes:  $n_1, n_2, n_0, n_6$

- Items:  $f_7, f_1$



## DHT: Chord Routing

- Upon receiving a query for item  $id$ , a node:
  - checks whether stores the item locally
  - if not, forwards the query to the largest node in its successor table that does not exceed  $id$



## Chord/CAN Summary

- Each node "owns" some portion of the key-space
  - in CAN, it is a multi-dimensional "zone"
  - in Chord, it is the key-id-space between two nodes in 1-D ring
- Files and nodes are assigned random locations in key-space
  - provides some load balancing
    - probabilistically equal division of keys to nodes
- Routing/search is local (distributed) and greedy
  - node X does not know of a path to a key Z
  - but if it appears that node Y is the closest to Z among all of the nodes known to X
  - so route to Y

43

## There are other DHT algorithms

- Kadmelia
- Tapestry (Zhao et al)
  - Keys interpreted as a sequence of digits
  - Incremental suffix routing
    - » Source to target route is accomplished by correcting one digit at a time
    - » For instance: (to route from 0312 → 1643)
    - » 0312 → 2173 → 3243 → 2643 → 1643
  - Each node has a routing table
- Skip Graphs (Aspnes and Shah)

44

## Summary: DHT

- Underlying metric space.
- Nodes embedded in metric space
- Location determined by key
- Hashing to balance load
- Greedy routing
- Typically
  - $O(\log n)$  space at each node
  - $O(\log n)$  routing time

45

## Summary: the Lookup Problem

- Unstructured P2P design
  - Napster (central query server)
  - Gnutella (decentralized, flooding)
  - Freenet (search by routing)
- Structured P2P design (search by routing)
  - Chord (a ring)
  - CAN (zones)

46

## Outline

### □ Recap

#### ➤ *P2P*

#### □ *the lookup problem*

- Napster (central query server; distributed data server)
- Gnutella (decentralized, flooding)
- Freenet (search by routing)
- Content Addressable Network (virtual zones)
- Chord (search by routing on a virtual ring)

#### ➤ *the scalability problem*

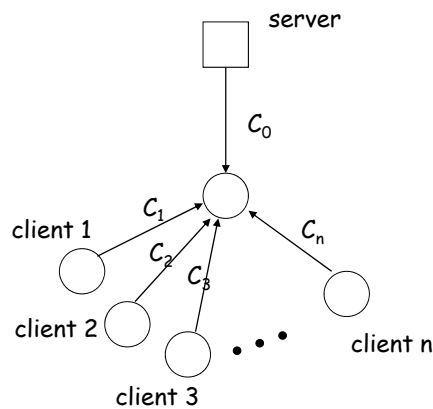
47

## An Upper Bound on Scalability

### □ Assume

- need to achieve same rate to all clients
- only uplinks can be bottlenecks

### □ What is an upper bound on scalability?



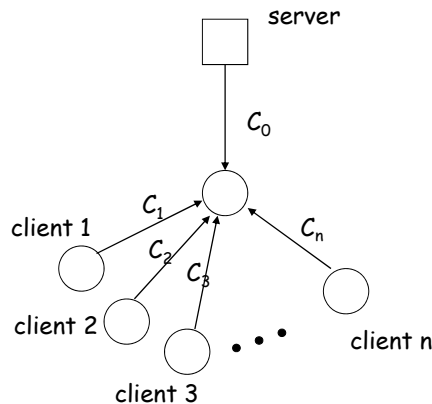
48

## The Scalability Problem

- Maximum throughput

$$R = \min\{C_0, (C_0 + \sum C_i)/n\}$$

- The bound is theoretically approachable

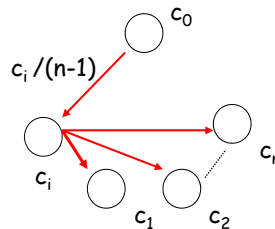


49

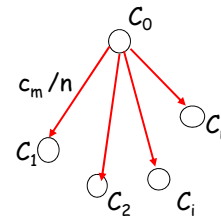
## Theoretical Capacity: upload is bottleneck

$$R = \min\{C_0, (C_0 + \sum C_i)/n\}$$

- Assume  $C_0 > (C_0 + \sum C_i)/n$
- Tree i:  
server  $\rightarrow$  client i:  $c_i/(n-1)$   
client i  $\rightarrow$  other n-1 clients

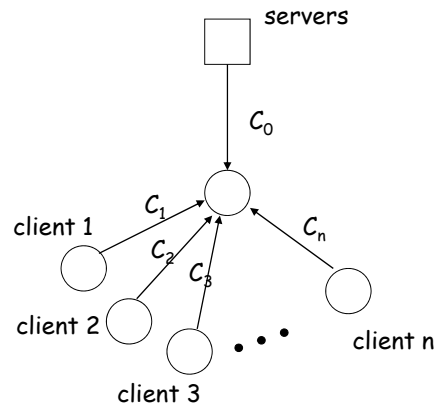
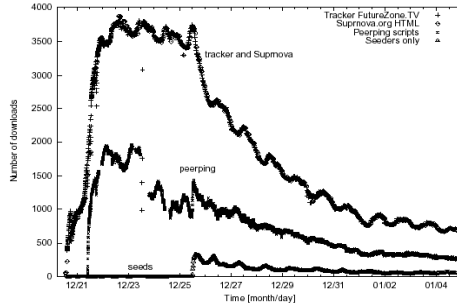


- Tree 0:  
server has remaining  $c_m = c_0 - (c_1 + c_2 + \dots + c_n)/(n-1)$   
send to client i:  $c_m/n$



50

## Why not Building the Trees?

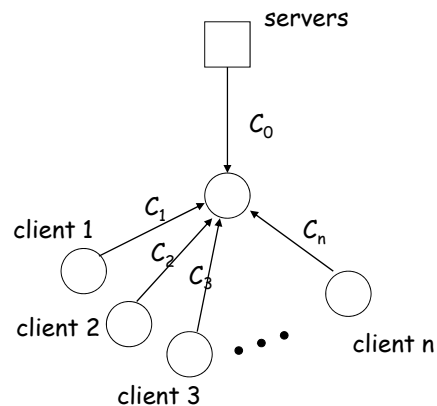


- ❑ Clients come and go (churn): maintaining the trees is too expensive

51

## Key Design Issues

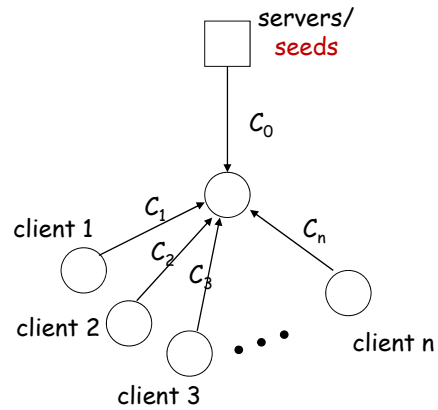
- ❑ Robustness
  - Resistant to churn and failures
- ❑ Efficiency
  - A client has content that others need; otherwise, its upload capacity may not be utilized
- ❑ Incentive: clients are willing to upload
  - 70% of Gnutella users share no files,
  - nearly 50% of all responses are returned by the top 1% of sharing hosts
- ❑ Lookup problem



52

## Discussion: How to handle the issues?

- Robustness
- Efficiency
- Incentive



53

## Outline

- Recap
  - *P2P*
    - *the lookup problem*
    - *the scalability problem*
    - *BitTorrent*

54

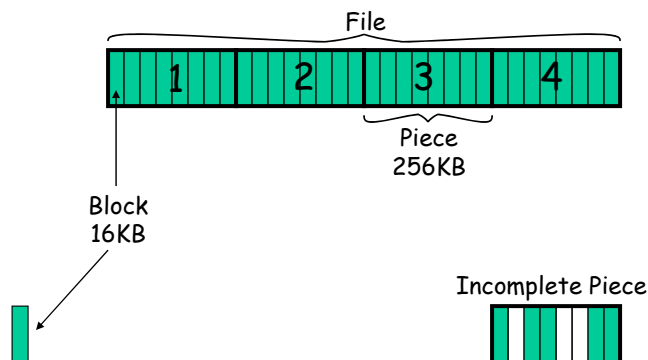
## BitTorrent

- ❑ A P2P file sharing protocol
- ❑ Created by Bram Cohen in 2004
- ❑ A peer can download pieces concurrently from multiple locations

55

## File Organization

Piece: unit of information exchange among peers  
Block: unit of download



56

## Outline

---

- Recap
  - *P2P*
    - *the lookup problem*
    - *the scalability problem*
    - *BitTorrent*
      - *Lookup*

57

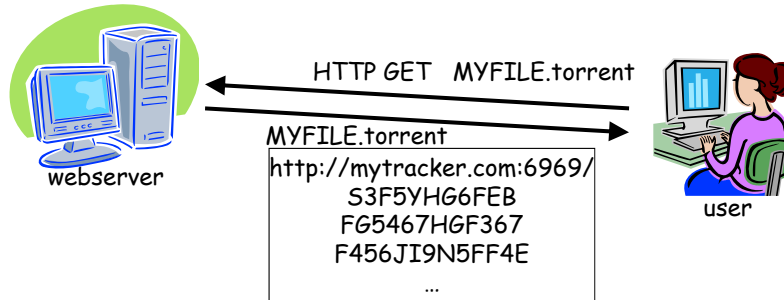
## BitTorrent

---

- Mostly tracker based
  
- Tracker-less mode; based on the Kademlia DHT

58

## BitTorrent: Initialization

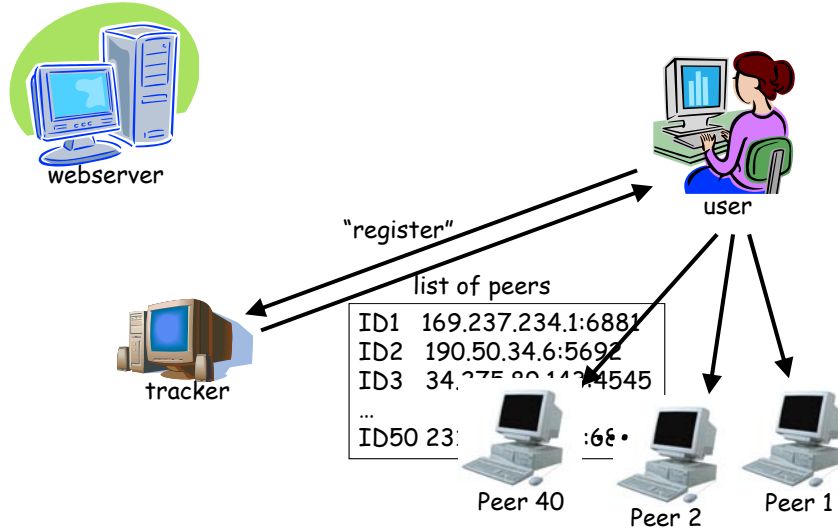


59

## Metadata File Structure

- Meta info contains information necessary to contact the tracker and describes the files in the torrent
  - announce URL of tracker
  - file name
  - file length
  - piece length (typically 256KB)
  - SHA-1 hashes of pieces for verification
  - also creation date, comment, creator, ...

## Tracker Protocol



61

## Tracker Protocol

- Communicates with clients via HTTP/HTTPS
- Client GET request
  - info\_hash: uniquely identifies the file
  - peer\_id: chosen by and uniquely identifies the client
  - client IP and port
  - numwant: how many peers to return (defaults to 50)
  - stats: e.g., bytes uploaded, downloaded
- Tracker GET response
  - interval: how often to contact the tracker
  - list of peers, containing peer id, IP and port
  - stats

## Outline

---

- Recap
  - *P2P*
    - *the lookup problem*
    - *the scalability problem*
    - *BitTorrent*
      - *Lookup*
      - *Robustness*

63

## Robustness

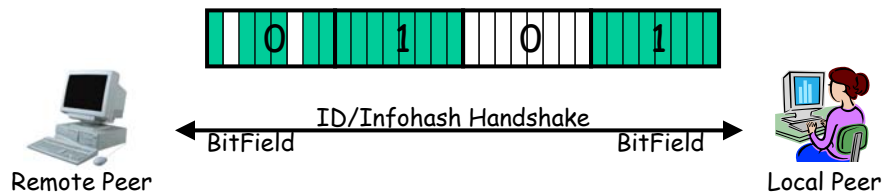
---

- A swarming protocol
  - Peers exchange info about other peers in the system
  - Peers exchange piece availability and request blocks from peers

64

## Peer Protocol

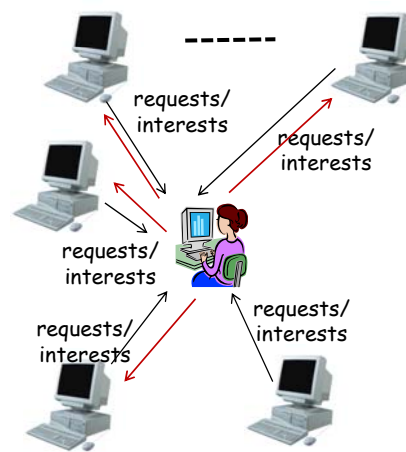
(Over TCP)



65

## Peer Protocol

- Unchoke: indicate if a allows b to download
- Interest/request: indicate which block to send from b to a



66

## Outline

### □ Recap

#### ➤ P2P

- *the lookup problem*
- *the scalability problem*
- *BitTorrent*
  - *Lookup*
  - *Robustness*
  - *Incentive*

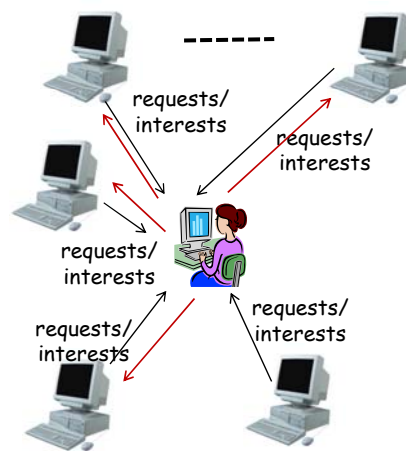
67

## Peer Selection - Response/Unchoking

□ Periodically (typically every 10 seconds) calculate data-receiving rates from all peers

□ Upload to (*unchoke*) the fastest

- constant number (4) of unchoking slots
- partition upload bw equally among unchoked



commonly referred to as "tit-for-tat" strategy

## Optimistic Unchoking

- Periodically select a peer at random and upload to it
  - typically every 3 unchoking rounds (30 seconds)
  
- Multi-purpose mechanism
  - allow bootstrapping of new clients
  - continuously look for the fastest peers (exploitation vs exploration)

## Outline

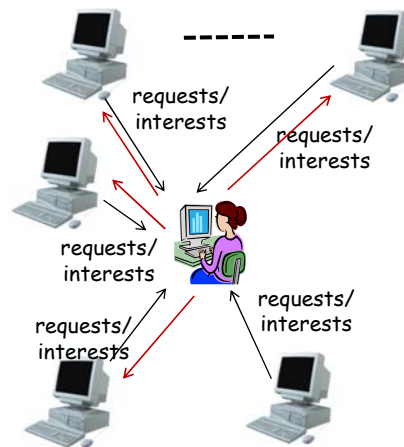
- Recap
  - *P2P*
    - *the lookup problem*
    - *the scalability problem*
    - *BitTorrent*
      - *Lookup*
      - *Robustness*
      - *Incentive*
      - *Efficiency*

## Piece Selection: Requests/Interests

- When downloading starts: choose at random and request them from the peers
  - get pieces as quickly as possible
  - obtain something to offer to others
- After 4 pieces: request (local) **rarest first**
  - achieves the fastest replication of rare pieces
  - obtain something of value
- Endgame mode
  - defense against the "last-block problem": cannot finish because missing a few last pieces
  - send requests for missing sub-pieces to all peers in our peer list
  - send cancel messages upon receipt of a sub-piece

## BitTorrent: Summary

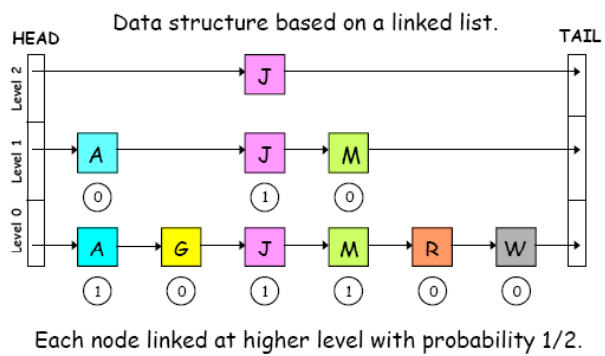
- Very widely used
  - mainline: written in Python
  - Azureus: the most popular, written in Java
  - Other popular clients: ABC, BitComet, BitLord, BitTornado, µTorrent, Opera browser
- Many explorations, e.g.,
  - BitThief
  - BitTyrant
- Better understanding is needed



## Optional Slides

73

## Skip List



74