15-440 Distributed Systems

CDN & Peer-to-Peer

Server Selection

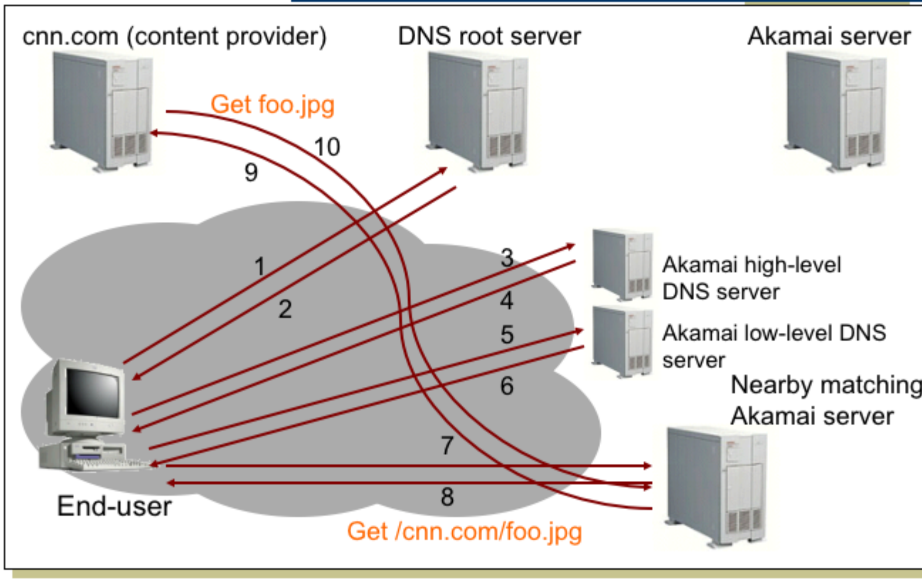
- Which server?
  - Lowest load → to balance load on servers
  - Best performance → to improve client performance
    - Based on Geography? RTT? Throughput? Load?
  - Any alive node → to provide fault tolerance
- How to direct clients to a particular server?
  - As part of routing → anycast, cluster load balancing
    - Not covered ☹
  - As part of application → HTTP redirect
  - **As part of naming → DNS**

2

How Akamai Works

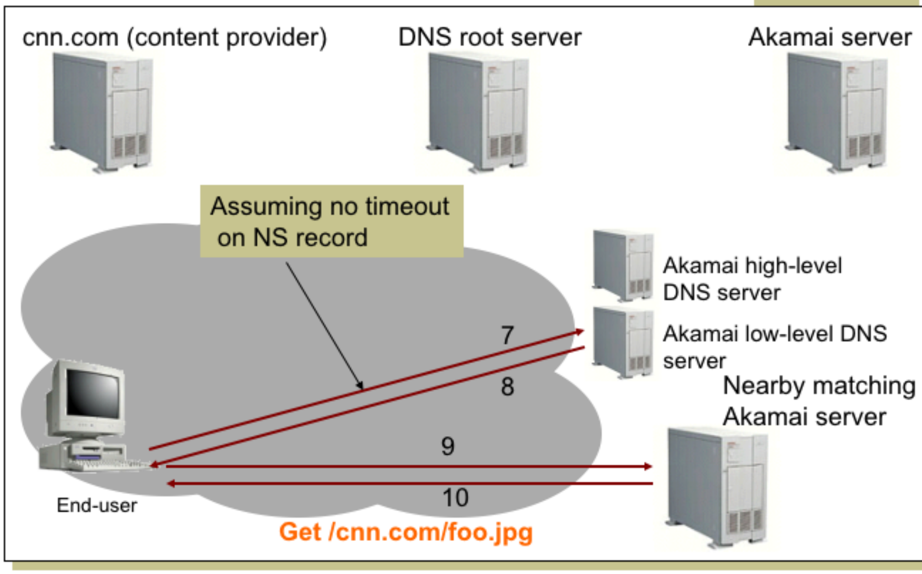
- Clients delegate domain to akamai
  - ibm.com. 172800 IN   NS   usw2.akam.net.
- CNAME records eventually lead to
  - Something like e2874.x.akamaiedge.net For IBM
  - Or a1686.q.akamai.net for IKEA….
- Client is forced to resolve eXYZ.x.akamaiedge.net hostname

How Akamai Works



4

Akamai – Subsequent Requests



5

How Akamai Works

- How is content replicated?
- Akamai only replicates static content (*)
- Modified name contains original file name
- Akamai server is asked for content
  - First checks local cache
  - If not in cache, requests file from primary server and caches file

* (At least, the version we're talking about today. Akamai actually lets sites write code that can run on Akamai's servers, but that's a pretty different beast)

How Akamai Works

- Root server gives NS record for akamai.net
- Akamai.net name server returns NS record for x.akamaiedge.net
  - Name server chosen to be in region of client's name server
  - TTL is large
- x.akamaiedge.net nameserver chooses server in region
  - Should try to chose server that has file in cache - How to choose?
  - Uses eXYZ name and consistent hashing
  - TTL is small → why?

7

Summary

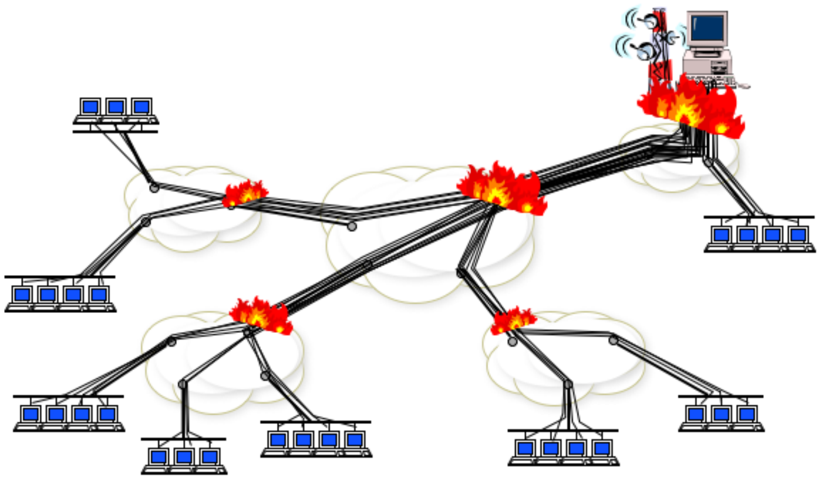
- DNS
- Content Delivery Networks move data closer to user, maintain consistency, balance load
  - Consistent hashing maps keys AND buckets into the same space

Outline

- Content Distribution Networks
- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord

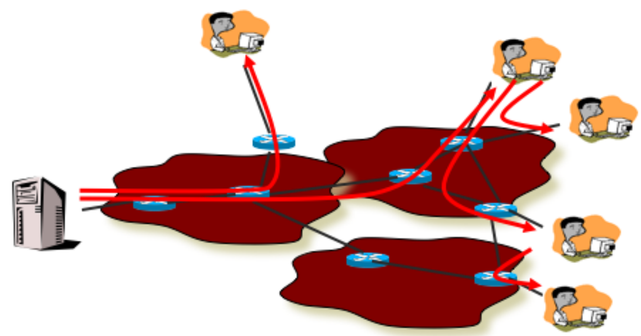
Scaling Problem

- Millions of clients ⇒ server and network meltdown



10

P2P System



- Leverage the resources of client machines (peers)
  - Computation, storage, bandwidth

11

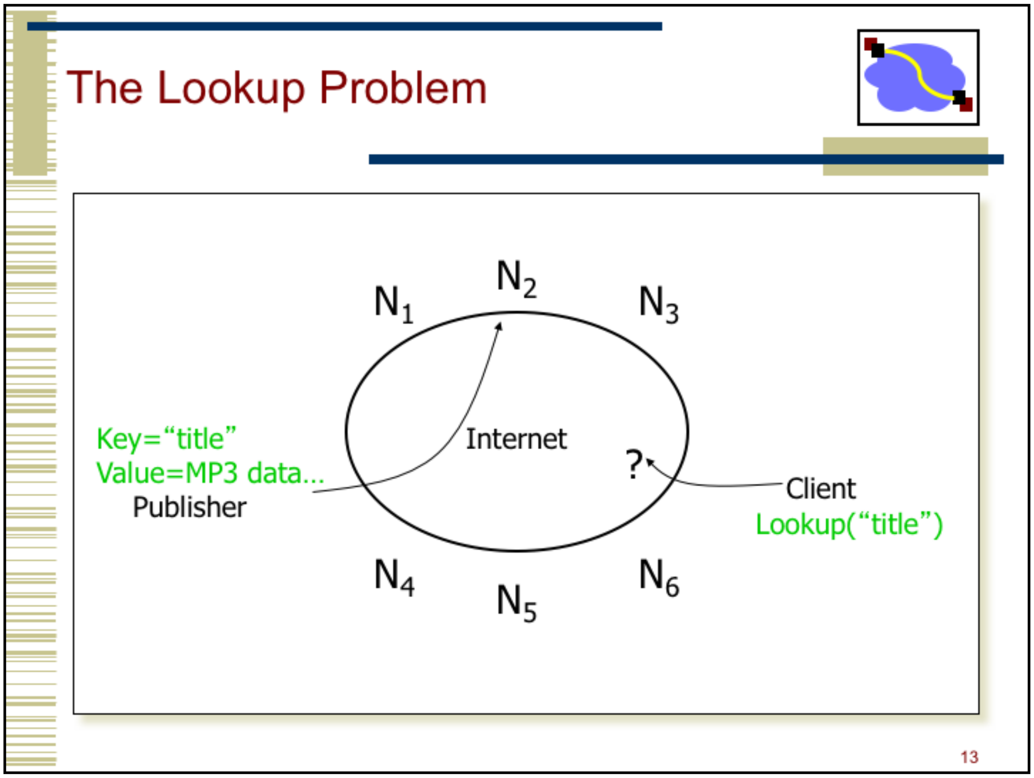
App end-point vs. Infrastructure vs. waypoints

Transition: infrastructure -> remove them

Peer-to-Peer Networks

- Typically each member stores/provides access to content
- Basically a replication system for files
  - Always a tradeoff between possible location of files and searching difficulty
  - Peer-to-peer allow files to be anywhere → searching is the challenge
  - Dynamic member list makes it more difficult

12



1000s of nodes.

Set of nodes may change…

Searching

- Needles vs. Haystacks
  - Searching for top 40, or an obscure punk track from 1981 that nobody's heard of?
- Search expressiveness
  - Whole word? Regular expressions? File names? Attributes? Whole-text search?
    - (e.g., p2p gnutella or p2p google?)

Framework

- Common Primitives:
  - **Join**: how to I begin participating?
  - **Publish**: how do I advertise my file?
  - **Search**: how to I find a file?
  - **Fetch**: how to I retrieve a file?

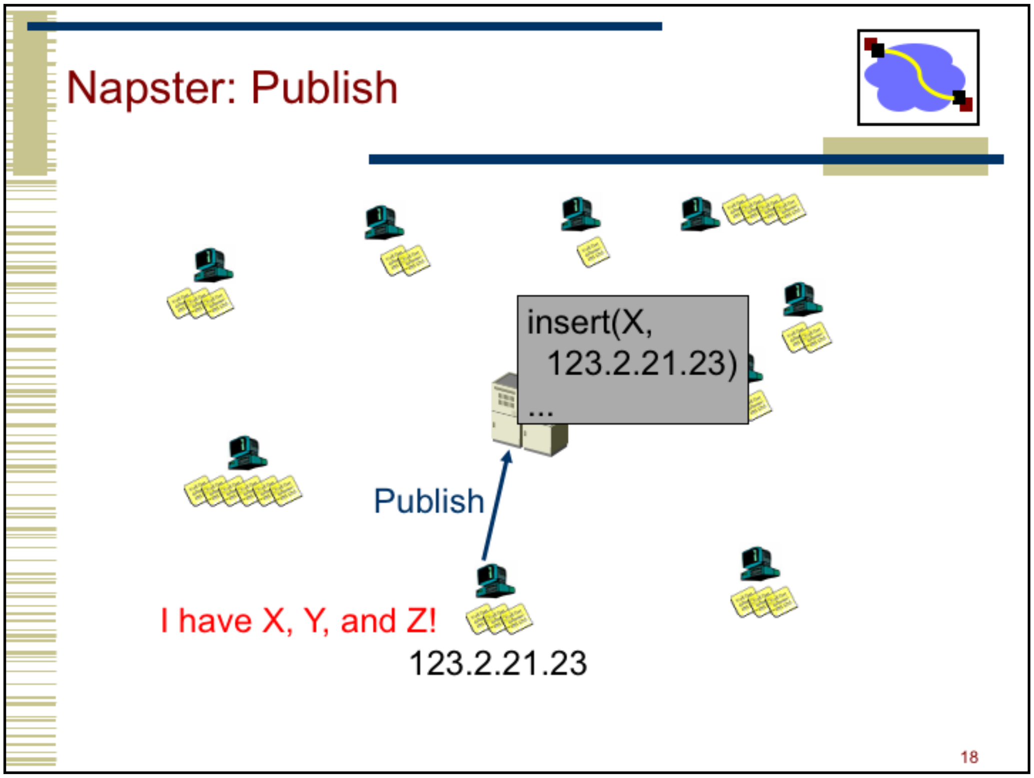
Outline

- Content Distribution Networks
- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord

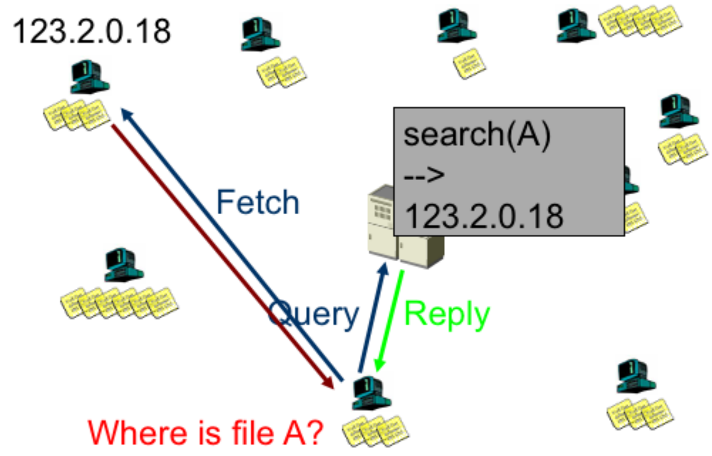
16

Napster: Overiew

- Centralized Database:
  - **Join:** on startup, client contacts central server
  - **Publish:** reports list of files to central server
  - **Search:** query the server => return someone that stores the requested file
  - **Fetch:** get the file directly from peer


Napster: Publish
insert(X,
123.2.21.23)
...
Publish
I have X, Y, and Z!
123.2.21.23
18

Napster: Search



19

Napster: Discussion

- Pros:
  - Simple
  - Search scope is O(1)
  - Controllable (pro or con?)
- Cons:
  - Server maintains O(N) State
  - Server does all processing
  - Single point of failure

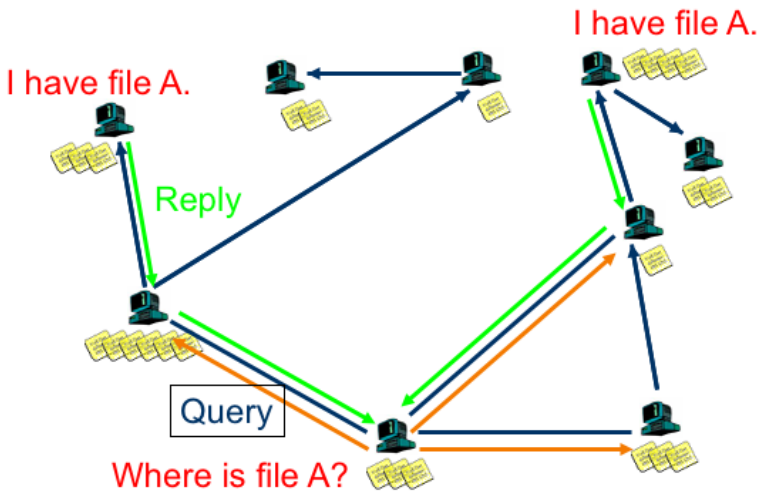
20

“Old” Gnutella: Overview

- Query Flooding:
  - **Join**: on startup, client contacts a few other nodes; these become its “neighbors”
  - **Publish**: no need
  - **Search**: ask neighbors, who ask their neighbors, and so on... when/if found, reply to sender.
    - TTL limits propagation
  - **Fetch**: get the file directly from peer

21

Gnutella: Search



Gnutella: Discussion

- Pros:
  - Fully de-centralized
  - Search cost distributed
  - Processing @ each node permits powerful search semantics
- Cons:
  - Search scope is O(*N*)
  - Search time is O(???)
  - Nodes leave often, network unstable
- TTL-limited search works well for haystacks.
  - For scalability, does NOT search every node. May have to re-issue query later

Flooding: Gnutella, Kazaa

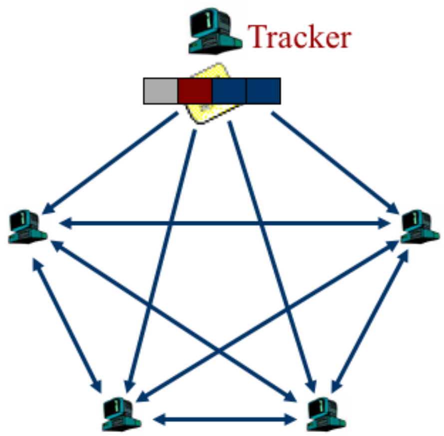
- Modifies the Gnutella protocol into two-level hierarchy
  - Hybrid of Gnutella and Napster
- Supernodes
  - Nodes that have better connection to Internet
  - Act as temporary indexing servers for other nodes
  - Help improve the stability of the network
- Standard nodes
  - Connect to supernodes and report list of files
  - Allows slower nodes to participate
- Search
  - Broadcast (Gnutella-style) search across supernodes
- Disadvantages
  - Kept a centralized registration → allowed for law suits ☹

24

BitTorrent: Overview

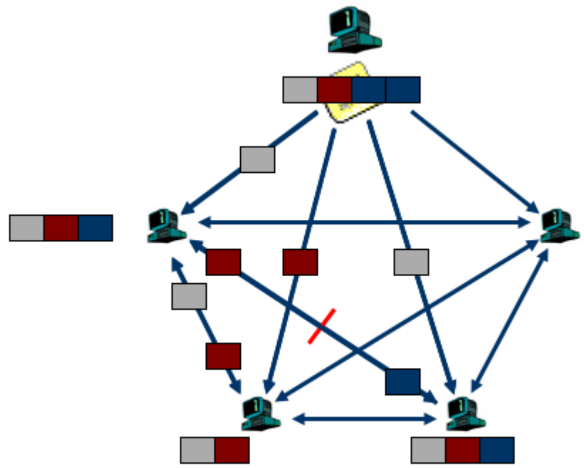
- Swarming:
  - **Join**: contact centralized “tracker” server, get a list of peers.
  - **Publish**: Run a tracker server.
  - **Search**: Out-of-band. E.g., use Google to find a tracker for the file you want.
  - **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
- Big differences from Napster:
  - Chunk based downloading
  - “few large files” focus
  - Anti-freeloading mechanisms

BitTorrent: Publish/Join



26

BitTorrent: Fetch



27

BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
- Goal: Pareto Efficiency
  - Game Theory: “No change can make anyone better off without making others worse off”
  - Does it work? (not perfectly, but perhaps good enough?)

28

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relative weak condition
  - Central tracker server needed to bootstrap swarm
    - Alternate tracker designs exist (e.g. DHT based)

Outline

- Content Distribution Networks
- P2P Lookup Overview
- Centralized/Flooded Lookups
- Routed Lookups – Chord

30

DHT: Overview (1)

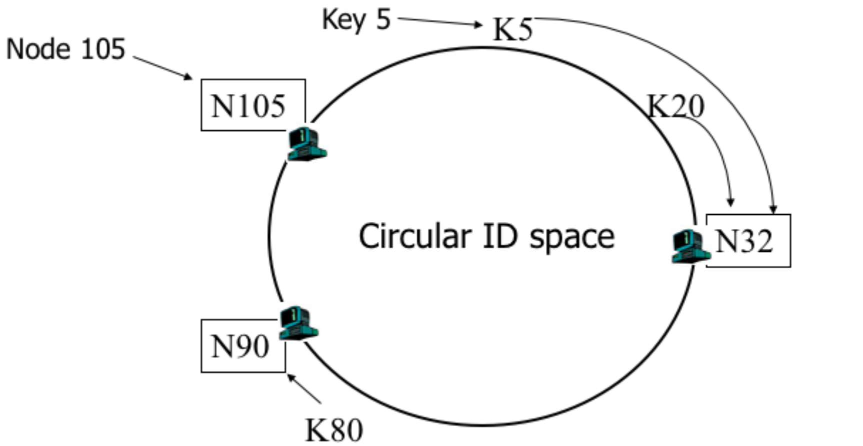
- Goal: make sure that an item (file) identified is always found in a reasonable # of steps
- Abstraction: a distributed hash-table (DHT) data structure
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…
- Implementation: nodes in system form a distributed data structure
  - Can be Ring, Tree, Hypercube, Skip List, Butterfly Network, ...

DHT: Overview (2)

- Structured Overlay Routing:
  - **Join**: On startup, contact a “bootstrap” node and integrate yourself into the distributed data structure; get a *node id*
  - **Publish**: Route publication for *file id* toward a close *node id* along the data structure
  - **Search**: Route a query for file id toward a close node id. Data structure guarantees that query will meet the publication.
  - **Fetch**: Two options:
    - Publication contains actual file ➔ fetch from where query stops
    - Publication says “I have file X” ➔ query tells you 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

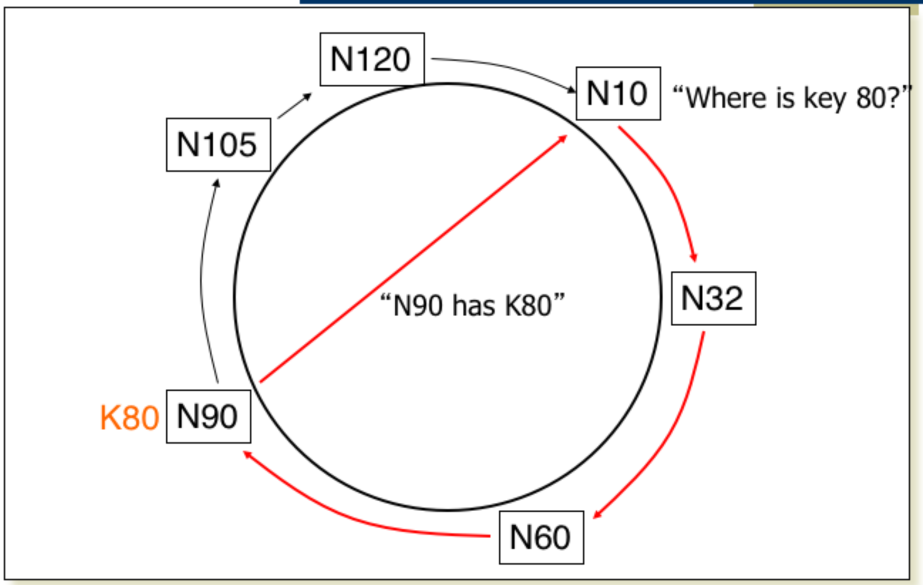
32

DHT: Consistent Hashing



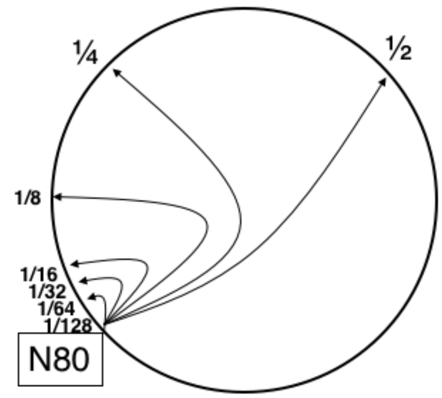
A key is stored at its successor: node with next higher ID

Routing: Chord Basic Lookup



34

Routing: Finger table - Faster Lookups



35

DHT: Chord Summary

- Routing table size?
  - Log *N* fingers
- Routing time?
  - Each hop expects to 1/2 the distance to the desired id => expect O(log *N*) hops.

DHT: Example - Chord

- Associate to each node and file a unique *id* in an *uni*-dimensional space (a Ring)
  - E.g., pick from the range $[0...2^m]$
  - Usually the hash of the file or IP address
- Properties:
  - Routing table size is O(log *N*) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log *N*) hops

from MIT in 2001

37

Routing: Chord

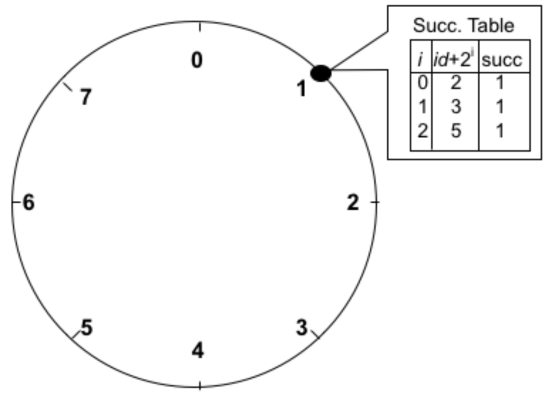
- Associate to each node and item a unique *id* in an *uni*-dimensional space
- Properties
  - Routing table size O(log(*N*)) , where *N* is the total number of nodes
  - Guarantees that a file is found in O(log(*N*)) steps

Routing: Chord Summary

- Assume identifier space is $0 \ldots 2^m$
- Each node maintains
  - Finger table
    - Entry *i* in the finger table of *n* is the first node that succeeds or equals $n + 2^i$
  - Predecessor node
- An item identified by *id* is stored on the successor node of *id*

Routing: Chord Example

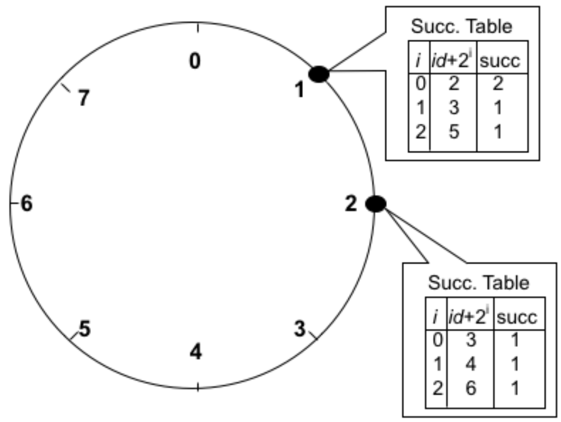
- Assume an identifier space 0..7
- Node n1:(1) joins→all entries in its finger table are initialized to itself



i	*id*+2[i]	succ
0	2	1
1	3	1
2	5	1

Routing: Chord Example

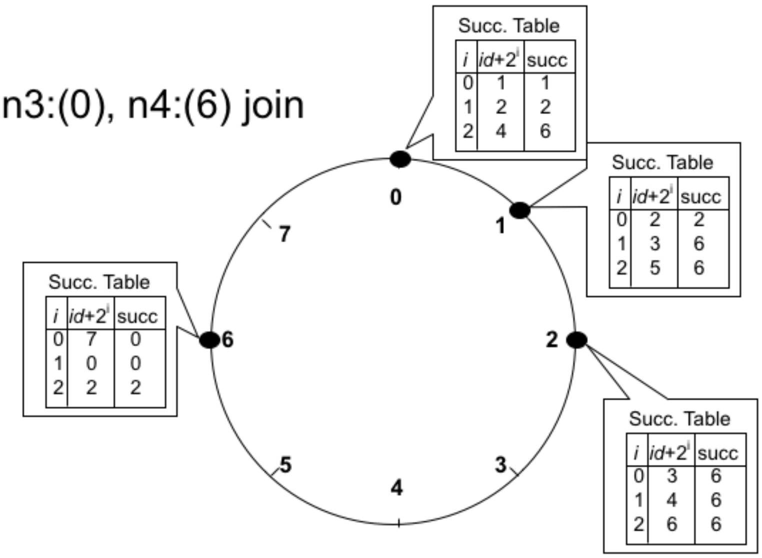
- Node n2:(2) joins



41

Routing: Chord Example

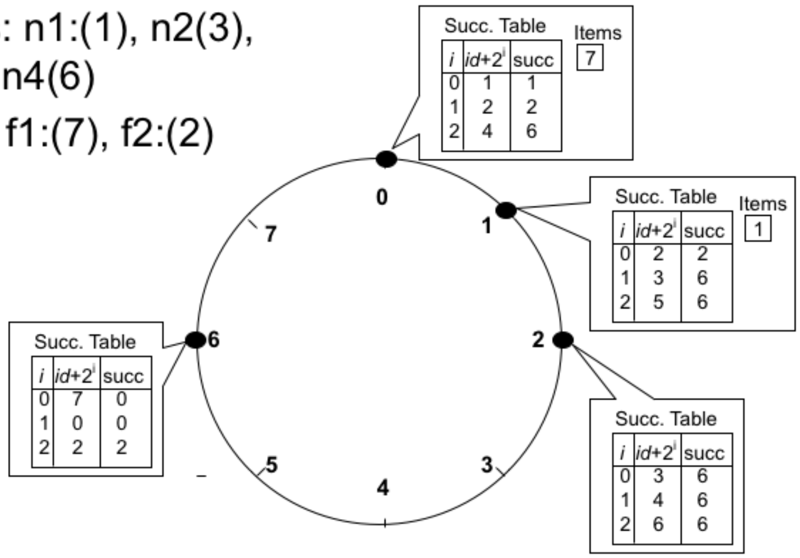
- Nodes n3:(0), n4:(6) join



42

Routing: Chord Examples

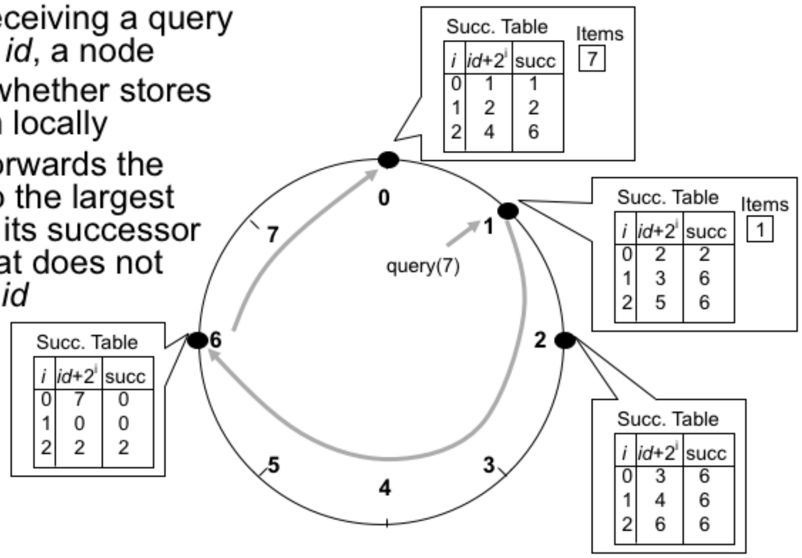
- Nodes: n1:(1), n2(3), n3(0), n4(6)
- Items: f1:(7), f2:(2)



43

Routing: Query

- Upon receiving a query for item *id*, a node
- Check whether stores the item locally
- If not, forwards the query to the largest node in its successor table that does not exceed *id*



DHT: Discussion

- Pros:
  - Guaranteed Lookup
  - O(log *N*) per node state and search scope
- Cons:
  - Supporting non-exact match search is hard

What can DHTs do for us?

- Distributed BitTorrent tracket
- Distributed object lookup
  - Based on object ID
- De-centralized file systems
  - CFS, PAST, Ivy
- Application Layer Multicast
  - Scribe, Bayeux, Splitstream
- Databases
  - PIER

P2P: Summary

- Many different styles; remember pros and cons of each
  - centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are very bad
  - Flooding messages to everyone is bad
  - Not all nodes are equal
  - Need incentives to discourage freeloading
  - Structure can provide theoretical bounds and guarantees
  - Underlying network topology is important
  - Privacy and security are important

•47

When are p2p / DHTs useful?

- Caching and “soft-state” data
  - Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot data
- Finding read-only data
  - Limited flooding finds hay
  - DHTs find needles

- BUT…

48

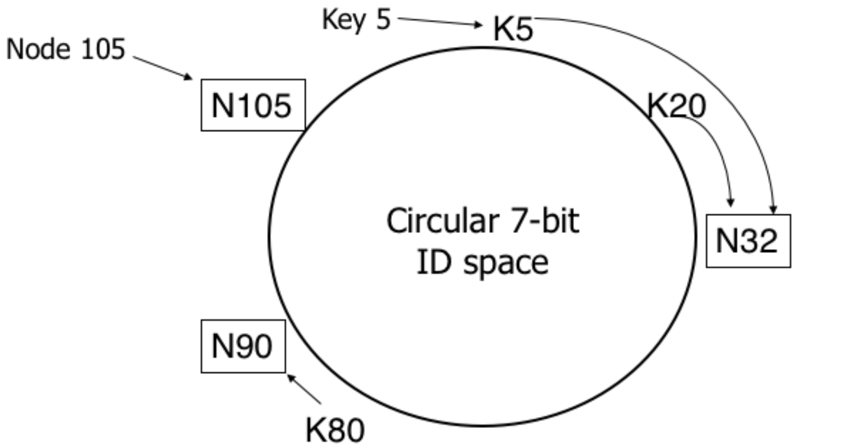
A Peer-to-peer Google?

- Complex intersection queries (“the” + “who”)
  - Billions of hits for each term alone
- Sophisticated ranking
  - Must compare many results before returning a subset to user
- Very, very hard for a DHT / p2p system
  - Need high inter-node bandwidth
  - (This is exactly what Google does - massive clusters)

Writable, persistent p2p

- Do you trust your data to 100,000 monkeys?
- Node availability hurts
  - Ex: Store 5 copies of data on different nodes
  - When someone goes away, you must replicate the data they held
  - Hard drives are *huge*, but home upload bandwidth is tiny - perhaps 50 Gbytes/day
  - Takes many days to upload contents of 1TB hard drive. Very expensive leave/replication situation!

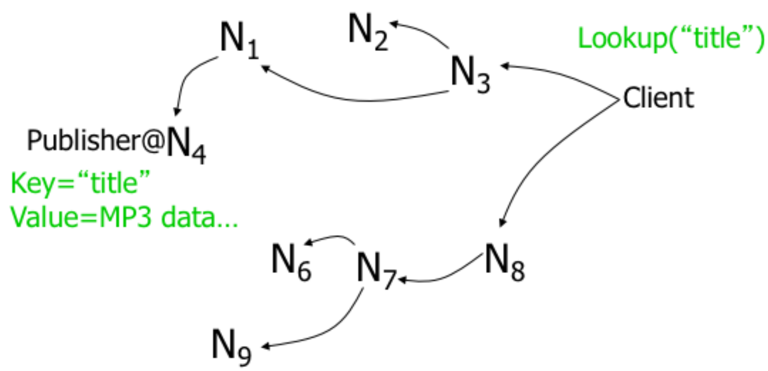
Aside: Consistent Hashing [Karger 97]



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

51

Flooded Queries (Gnutella)



Robust, but worst case O(N) messages per lookup

52

Flooding: Old Gnutella

- On startup, client contacts any servent (**serv**er + cli**ent**) in network
  - Servent interconnection used to forward control (queries, hits, etc)
- Idea: broadcast the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively forward the request
  - Eventually a machine that has the file receives the request, and it sends back the answer
  - Transfers are done with HTTP between peers

Flooding: Old Gnutella

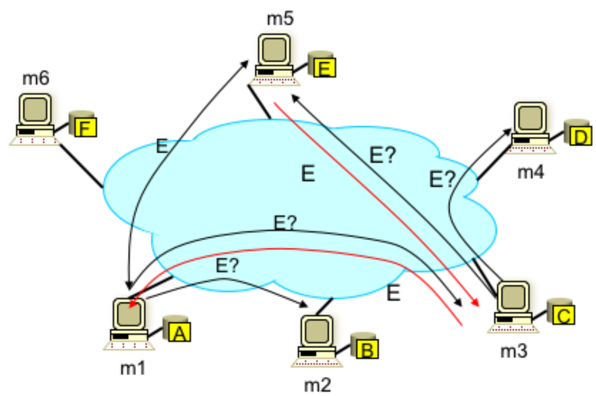
- Advantages:
  - Totally decentralized, highly robust
- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)
  - Especially hard on slow clients
    - At some point broadcast traffic on Gnutella exceeded 56kbps – what happened?
    - Modem users were effectively cut off!

Flooding: Old Gnutella Details

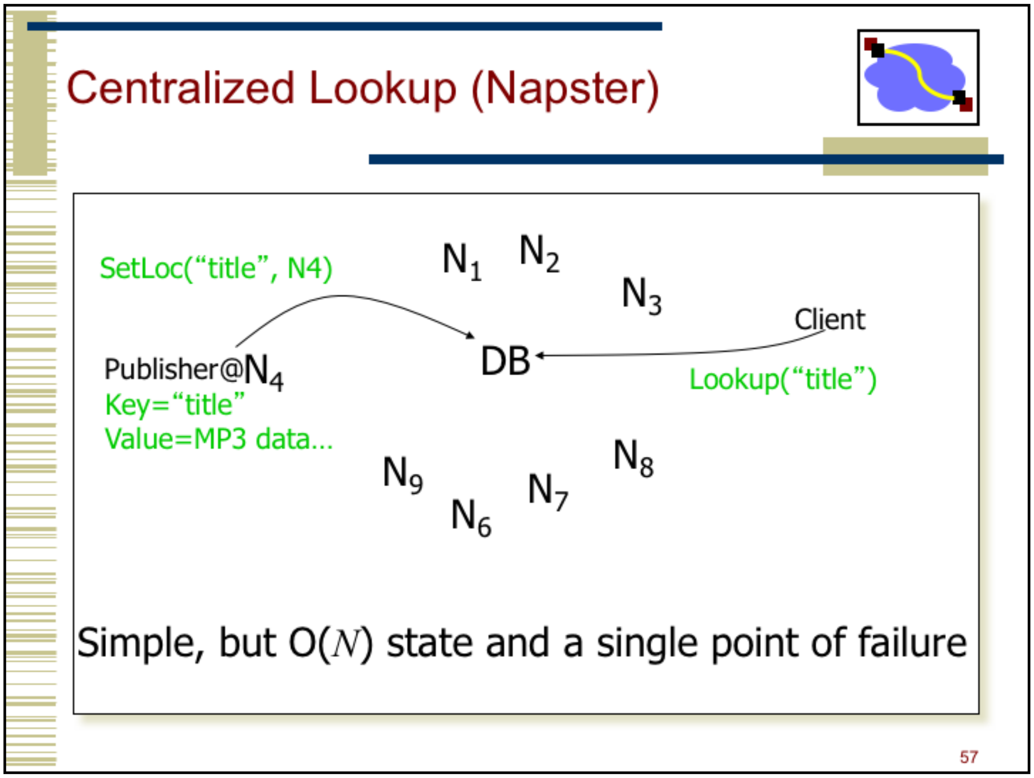
- Basic message header
  - Unique ID, TTL, Hops
- Message types
  - Ping – probes network for other servents
  - Pong – response to ping, contains IP addr, # of files, # of Kbytes shared
  - Query – search criteria + speed requirement of servent
  - QueryHit – successful response to Query, contains addr + port to transfer from, speed of servent, number of hits, hit results, servent ID
  - Push – request to servent ID to initiate connection, used to traverse firewalls
- Ping, Queries are flooded
- QueryHit, Pong, Push reverse path of previous message

Flooding: Old Gnutella Example

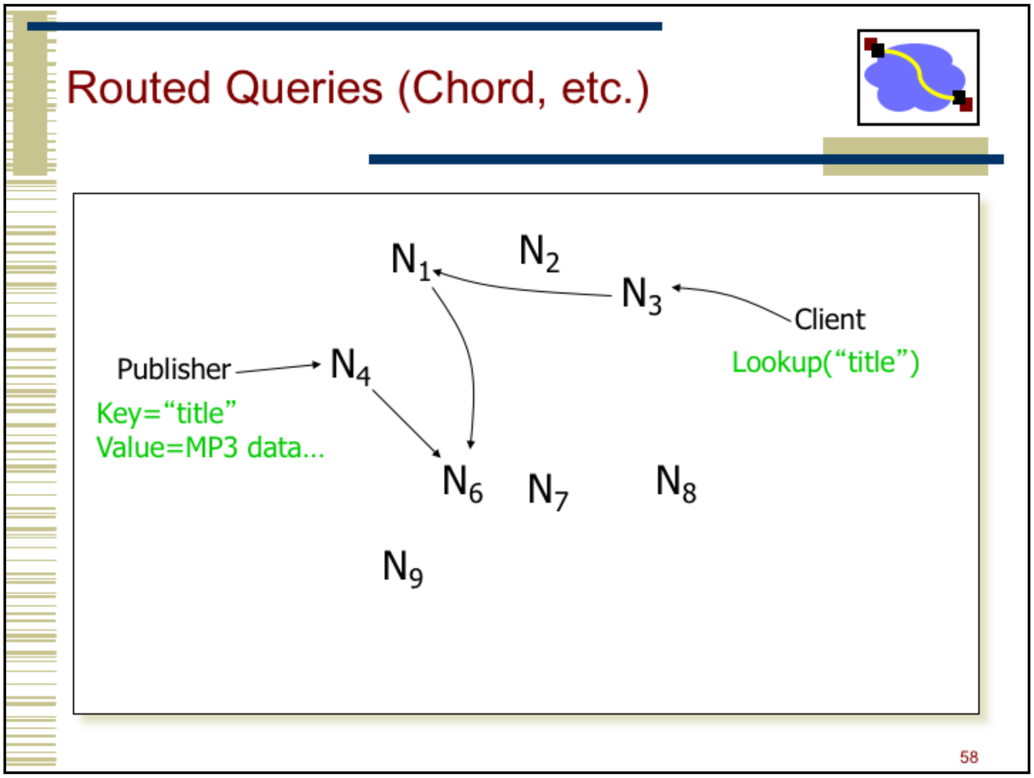
Assume: m1's neighbors are m2 and m3;
m3's neighbors are m4 and m5;…



56



O(N) state means its hard to keep the state up to date.



Challenge: can we make it robust? Small state? Actually find stuff in a changing system?

Consistent rendezvous point, between publisher and client.