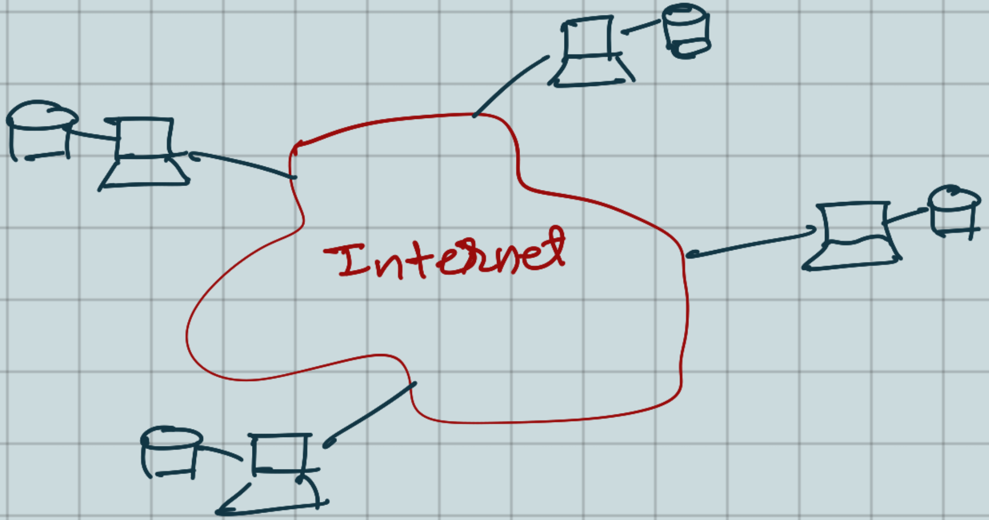
How did it start?

→ A killer application: Napster
  - free music over the Internet

Key idea: share the content, storage and bandwidth of individual users.



Model: Each user stores a subset of files
Each user has access to files from all users in the system.

Main Challenge:

→ Find where a particular file is saved?

→ **Scale:** Up to hundred of thousands or millions of machines

→ **Dynamicity:** machines can come and go any time

→ Assume a centralized index system that maps files to machines that are alive

→ How to find a file
- Query the index system → Return a machine that stores the required file
  - Ideally this is the closest/least loaded machine

→ Advantages:
- Simplicity, easy to implement sophisticated search engines on top of the index system

→ Disadvantages:

- Robustness, Scalability.

Gnutella

- Distribute file location
- Idea: flood the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machine that has the file receives the request. and sends back the answer.

- 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 (time to live)

→ Ad-hoc topology.
→ Queries are flooded for bounded # hops
→ No guarantees on recall.

Distributed Hash tables: (DHT)

- Abstraction: a distributed hash table data structure

  - Insert (id, item)
  - item = query (id)
  - Note: an item can be anything: a data object, document, file, pointer to a file

- Proposals:
  - CAN, Chord, Kademila, Pastry, Tapestry, etc.

DHT design goals:

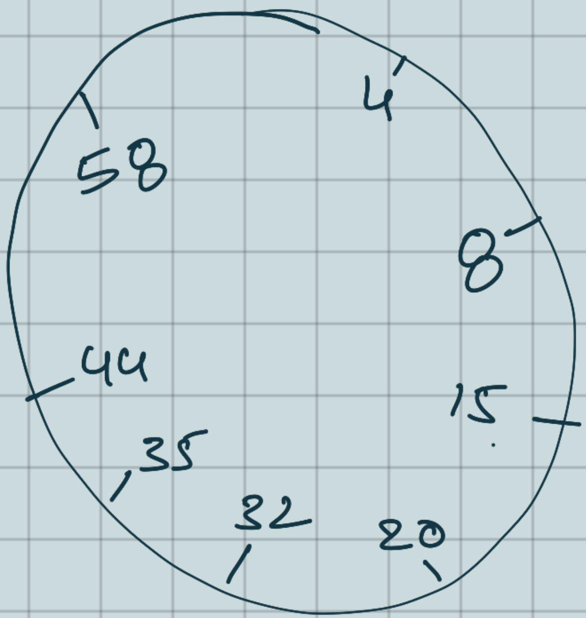
- Make sure that an item identified is always found.
- Scales to hundreds of thousands of nodes
- Handles rapid arrival / failure of nodes.

Chord:

- Associate to each node and item a unique id in an uni-dimensional space $0 \ldots 2^{m}-1$

- Key design decision:
  - Decouple correctness from efficiency.

- Properties
  - Routing table size $O(\lg N)$, where N is the total number of nodes
  - Guarantees that a file is found in $O(\lg N)$ steps.



8 : [5, 8)
15 : [9, 15]
20 : [16, 20

- Each node maintains a pointer to its successor.

Lookup:

- → Each node maintains its successor
- → Route packet (ID, data) to the node responsible for ID using successor pointers.

Joining operation:

- Each node A periodically sends a stabalize() message to its successor B.

- Upon receiving a stabilize() message, node B
  - returns its predecessor B' = pred(B) to A by sending a notify(B') message

- Upon receiving notify(B') from B.
  - if B' is between A and B, A updates its successor to B'
  - A doesn't do anything, otherwise.

Joining operation:

- Node with id=50 joins the ring.

- Node 50 needs to know atleast one node already in the system

  - Assume known node is 15

- Node 50 sends join(50) to Node 15
- Node 44: returns node 58
- Node 50 updates it successor to 58

- Node 50 sends stabalize to Node 58
- Node 58:
  - update predecessor to 50
  - Send notify back.

- Node 44 sends stabilize to Node 58
- Node 58 reply with a notify message
- Node 44 updates its successor to Node 50.

- Node 44 sends stabilize to Node 50
- Node 50 sets its predecessor to Node 44.

- Joining Complete:

Achieving Efficiency: finger tables

Finger table at 80

i	ft[i]	
0	96	96+1
1	96	96+2
2	96	96+4
3	96	96+8
4	96	96+16
5	112	←
6	20	

Say $m = 7$.

$N = \#$ node

- i^{th} entry at peer with id n is first peer with

$$id >= n + 2^i \pmod{2^m}$$