Computer Networks COL 334/672

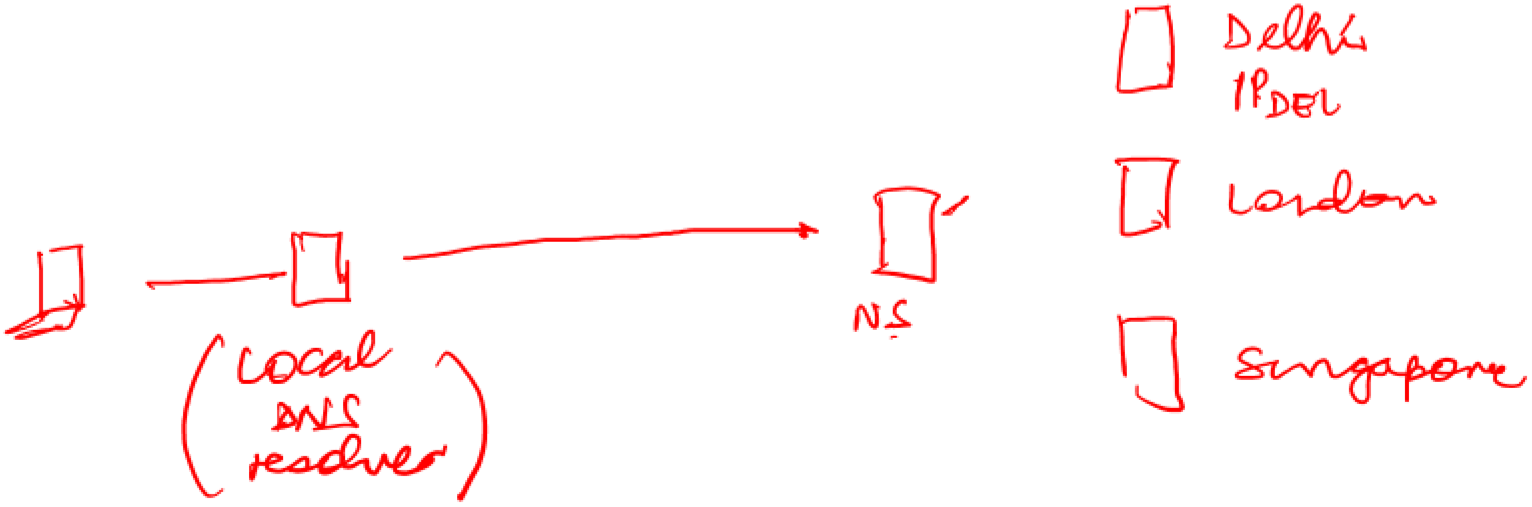
Application Layer: P2P

Slides adapted from KR

Sem 1, 2025-26

DNS and Load Balancing

IP Geolocation of the resolver



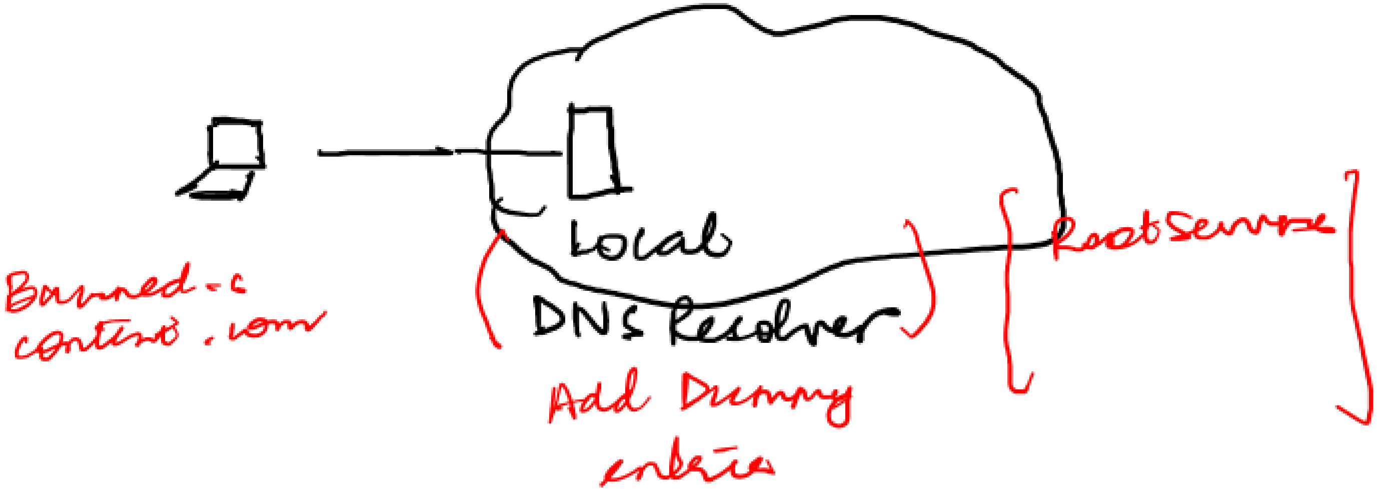
ISP's resolver

Public resolver

→

DNS, Content Regulation, Censorship

use VPN or public DNS resolver

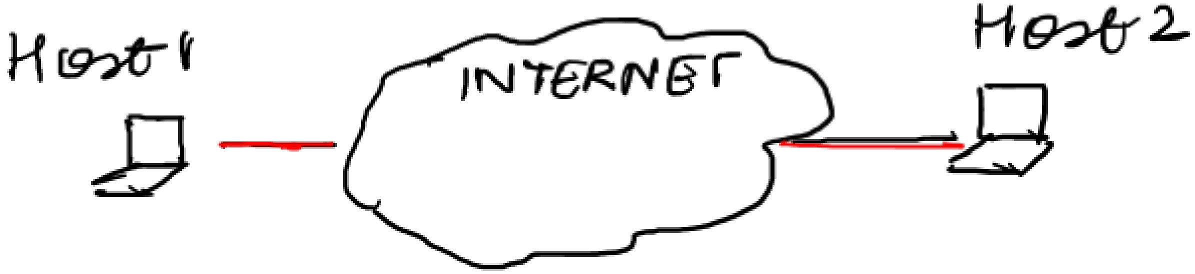


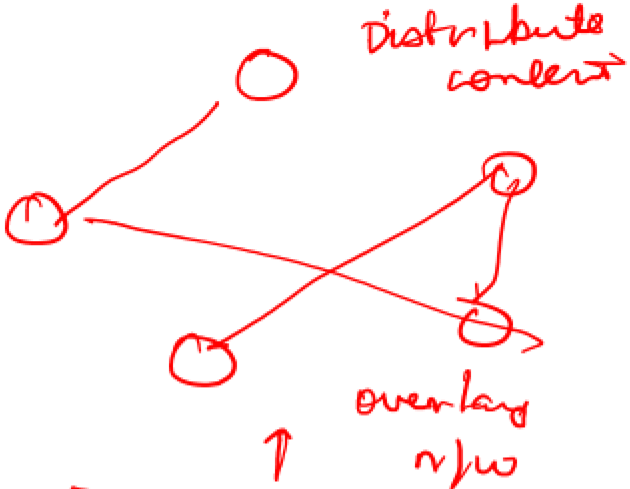
google DNS resolver

Recap: Application Layer

- HTTP
- Email
- DNS
- **P2P**
- Video streaming

What is Peer to Peer (P2P) Communication?

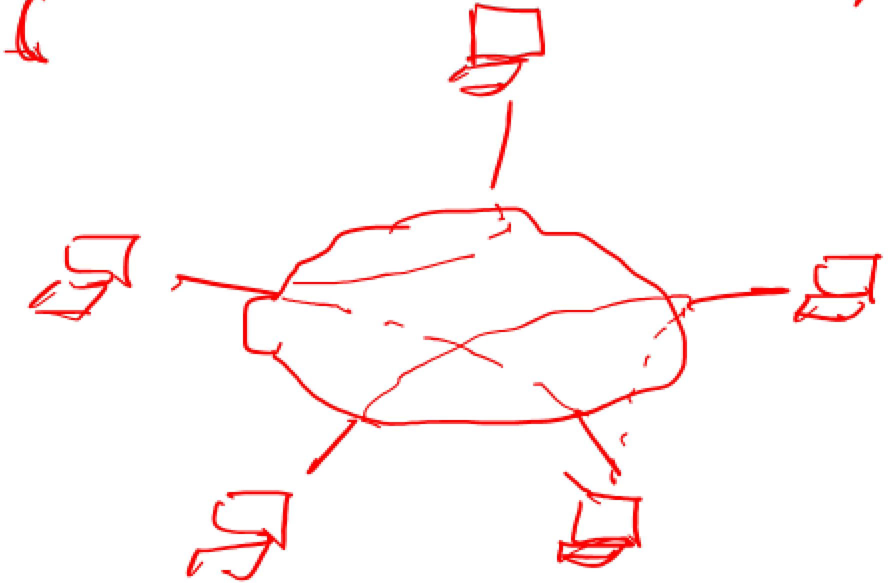




{ Whatsapp : Voice /video Telephony (

Which applications?

Torrent : Content Distribution



Distribute content
using this P2P
n/w

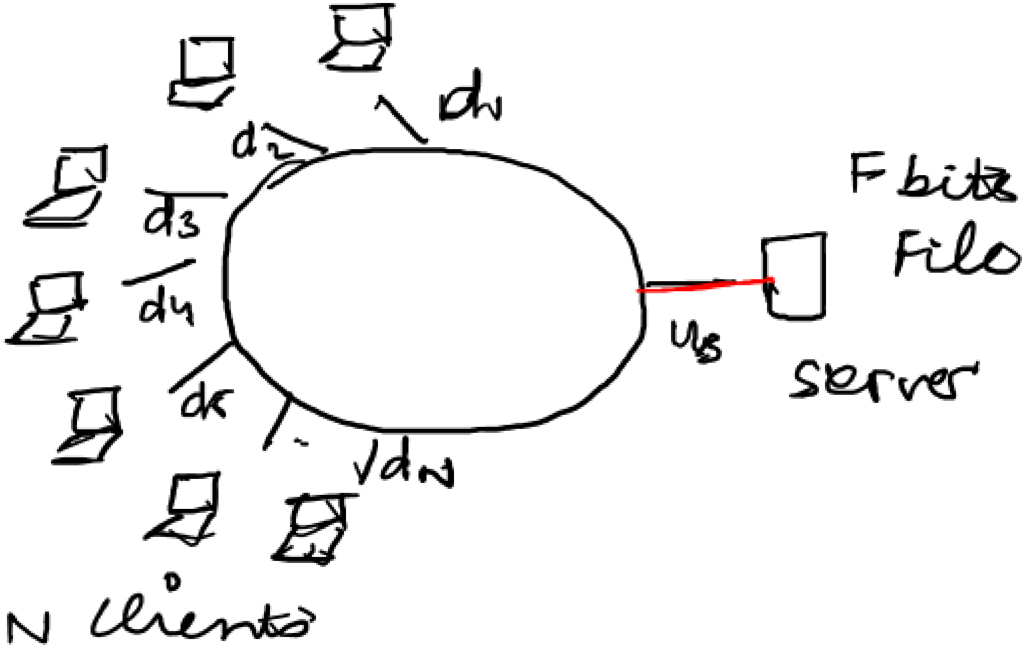
Why P2P for content distribution?

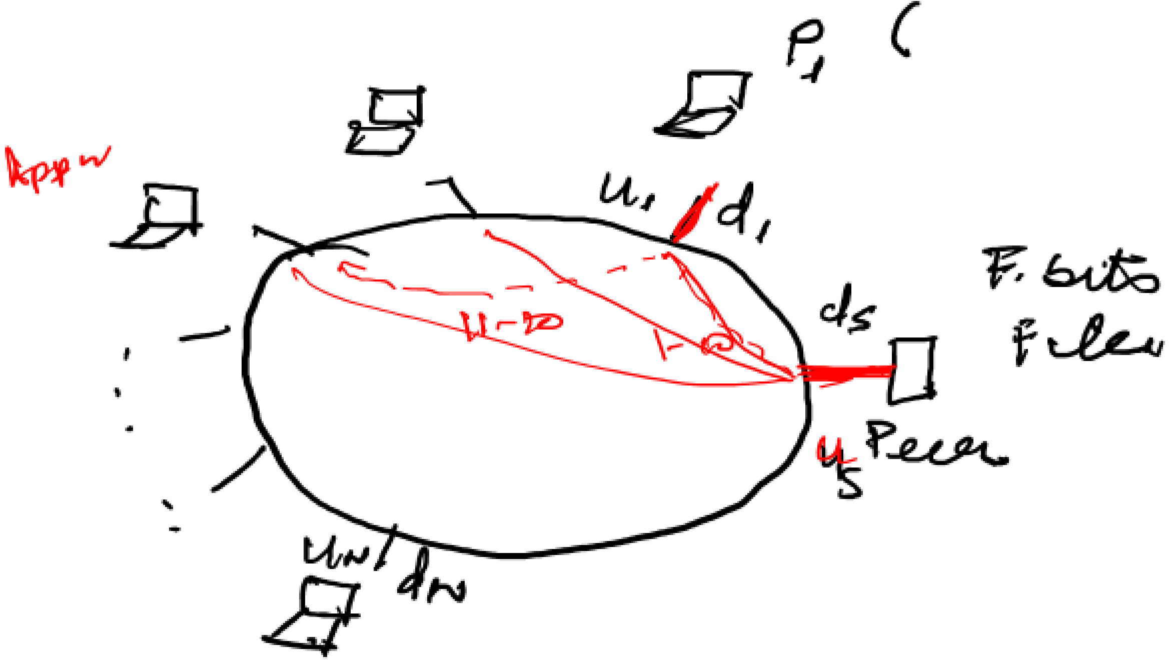
→ PROS: Self scalable

CONS: security issue, Peers can come & go

- Scales better than client-server architecture **Why?**

Client-server architecture





$$\max \left\{ \frac{NF}{u_s}, \frac{F}{d_{min}} \right\}$$

$$\max \left\{ \frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{\sum u_i + u_s} \right\}$$

as N scales, this does not scale

[unless you have more servers]

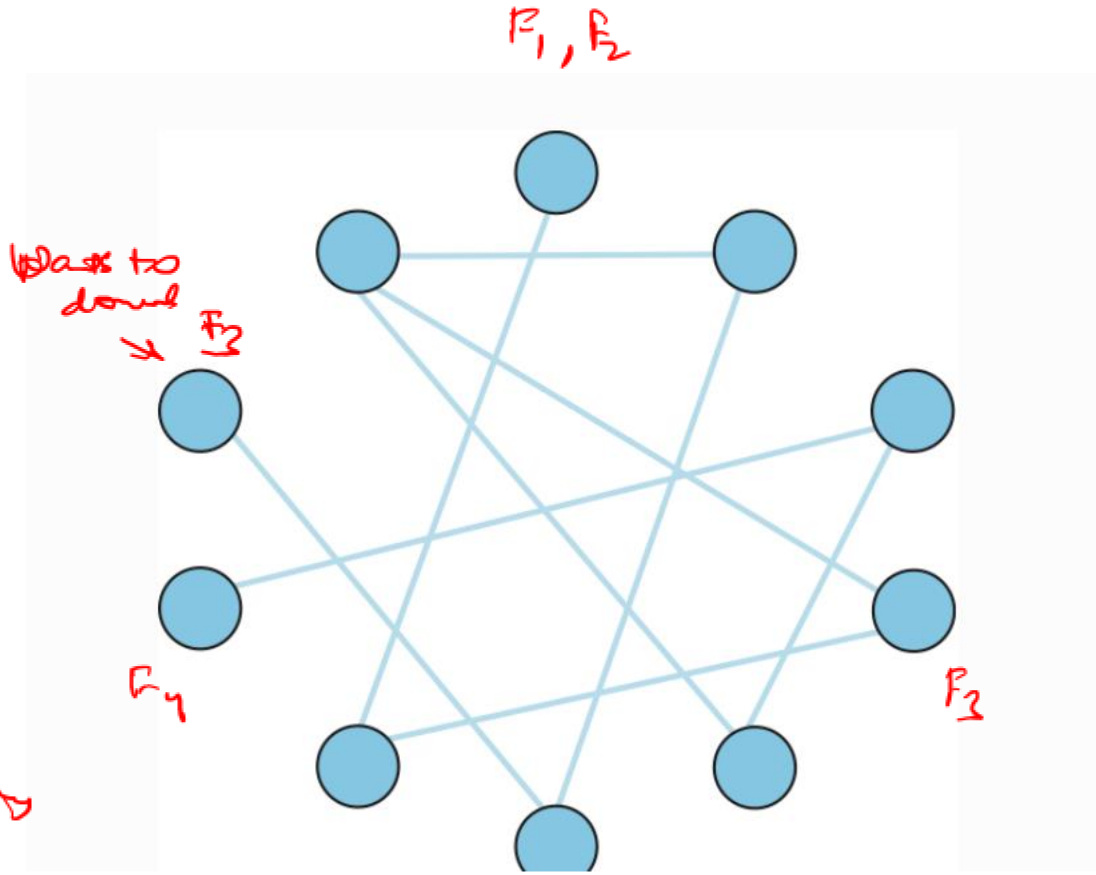
Particularly popular in the early 2000s

Two Interesting Questions for Content Distribution

- How to find content?
- How to download content?

How to Find Content?

- **Scenario:**
  - Network of peers who may each have a different set of content
  - Each peer is connected directly to some other peers, not necessarily all (why?)
- **Problem Statement**: How does a peer find who has file F_i?



Broadcast → TTL
Seq #s

unnecessary
overhead
↓
latency

Centralized index table

File Name	Who has File
	(IP address)

→ Scalability
→ SPF
→ accountability

Finding a file in a P2P network

- Intuition: Some indexing is useful for a faster lookup.
- Challenge: But can’t have a centralized hash table
- Solution: Use a distributed hash table

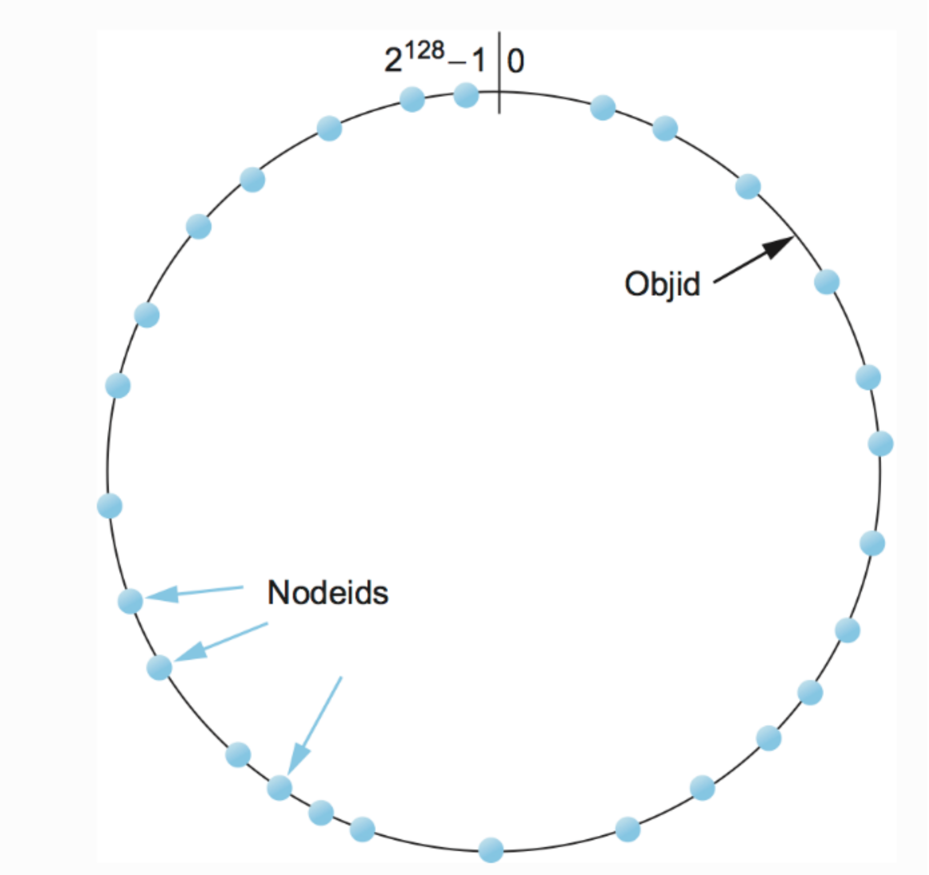
Distribute the hash table

(DHTs)

(PASTRY) → DHT

Idea: H(filenames) H(Name a IP)

- Map the objects and the nodes to a common virtual space
- Store the object information in a node that is closest to it in the virtual space



PASTRY Example

{0, . . . , 7}

Node: 2, 4, 6

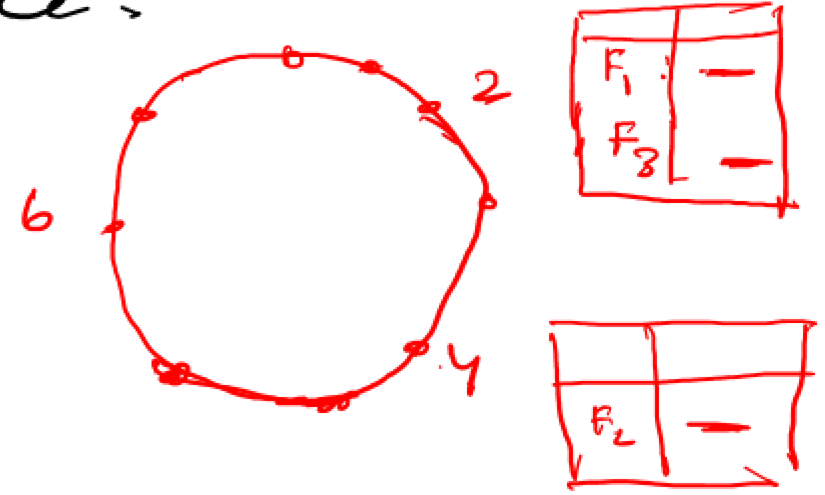
Files: $\overset{F_1}{2}$, $\overset{F_2}{3}$, $\overset{F_3}{7}$

If node with same value as $H(F_i)$ present ∴

store the <k, v> in that node.

Else

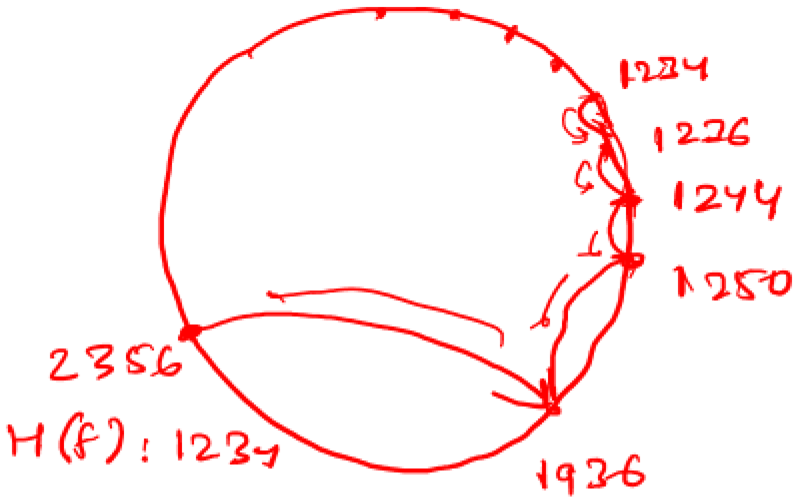
store in successor $(H(F_i))$



Distibuted Hash Table

PASTRY: Searching Closest Node

Idea: *To search for a file f, route query messages closer to H(f) in the virtual space until you find the node containing information about f*



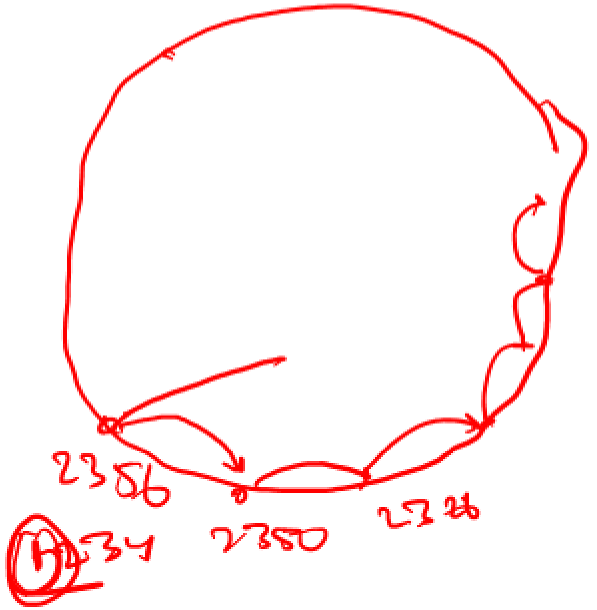
Challenge: *How do we ensure that we can always go to a closer node?*

Put some structure on the network

PASTRY: Searching Closest Node

Randomized algorithm

CHORD : Puts more structure



Idea: *To search for a file f, route query messages closer to H(f) in the virtual space until you find the node containing information about f*

Challenge: *How do we ensure that we can always go to a closer node?*

Reach the neighbor in log(N)

Solution: *Each node should store L nodes (L/2 successors, L/2 predecessors) and log(N) nodes distributed randomly in the virtual space*

→ 3, 4

→ 2

→ just linear search is too slow

→ let's make bigger jumps

Physically closer

Distributed Hash Table

- You should think about the following:
  - How the neighbors are maintained in the first place
  - What happens when a neighbor disconnects?
  - How is a new neighbor added?

- Various optimizations exist for DHTs

- Used in other domains such as distributed file system, web caching etc.

Next question: How to download content?