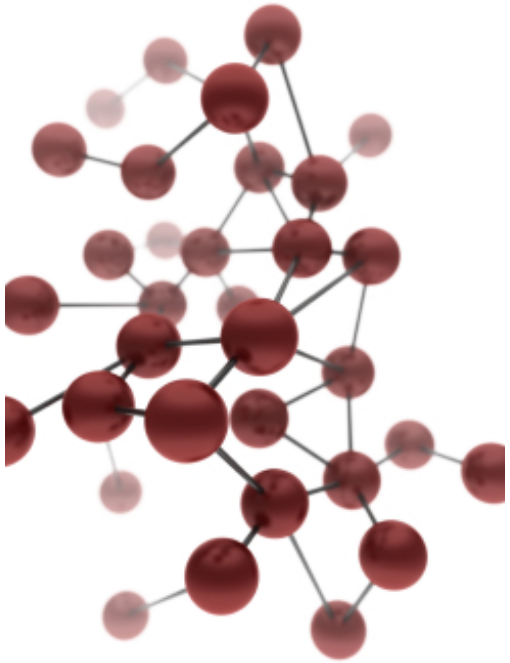


Decentralized / Peer to Peer
Lecture 15

Today: Peer-to-Peer

- Unstructured Peer-to-Peer Protocols
  - How and why do they work?
  - Napster, Gnutella, Kazaa
  - BitTorrent, Freenet, Skype

- Structured Peer-to-Peer Overlays
  - A quick overview

Traditional Internet Services Model

- Client-server
  - Many clients, 1 (or more) server(s)
  - Web servers, DNS, file downloads, video streaming

- Problems
  - Scalability: how many users can a server support?
    What happens when user traffic overload servers?
    Limited resources (bandwidth, CPU, storage)

  - Reliability: if # of servers is small, what happens when they break, fail, get disconnected, are mismanaged by humans?

  - Efficiency: if your users are spread across the entire globe, how do you make sure you answer their requests quickly?

The Alternative: Peer-to-Peer

- A simple idea
  - Users bring their own resources to the table
  - A cooperative model: clients = peers = servers

- The benefits
  - Scalability: # of "servers" grows with users
    BYOR: bring your own resources (storage, CPU, B/W)
  - Reliability: load spread across many peers, probability of them all failing is VERY low…
  - Efficiency: peers are distributed. Peers locate and get service from nearby peers (Is this always possible?)

The Peer-to-Peer Challenge

- What are the key components for leveraging P2P?
  - Communication: how do peers talk to each other
  - Service / data location: how do peers know who to talk to?

- Answers depend on the application context
  - The obvious example: file-sharing
    - Communication?
    - Service / data location?

- We'll discuss these in detail w.r.t. different protocols
  - Napster/Maze, Gnutella, Kazaa, BitTorrent, Freenet, Skype
  - Many others (Mojonation, Overnet, ICQ)

Unstructured P2P Applications

- Solves part of the problem (file location mostly)
  - Focus on locating popular objects
  - Hierarchy of super-nodes index client file collections
  - Queries sent as controlled flood amongst supernodes

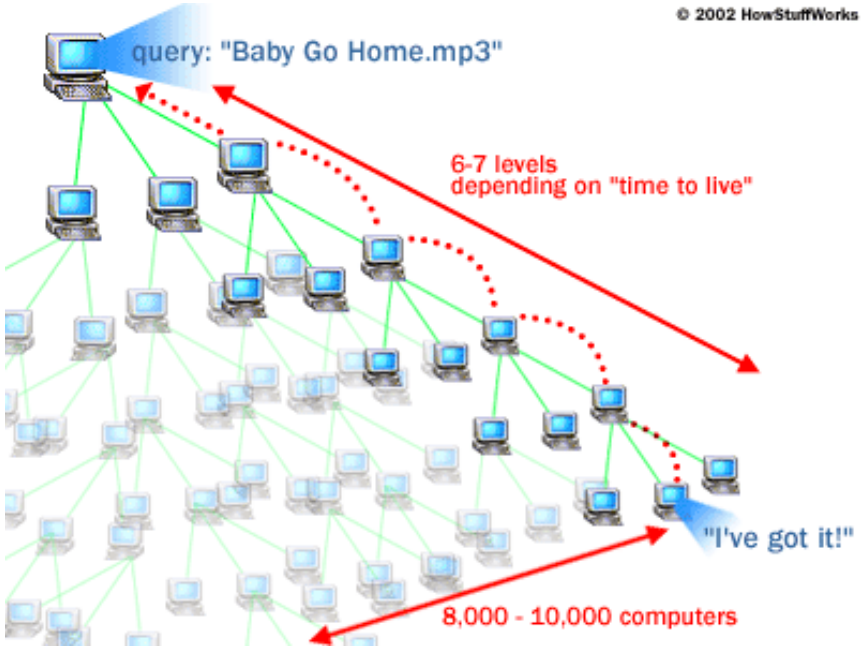
- Limitations
  - File search
    - Does not support generic any-to-any node communication
  - Probabilistic
    - Can only search a small portion of files in system
      otherwise would flood network with query traffic (e. g. Gnutella)
    - Uncommon files (in a file system) are easily lost
      the more unique your file is, more likely it will be inaccessible

Centralized Approach

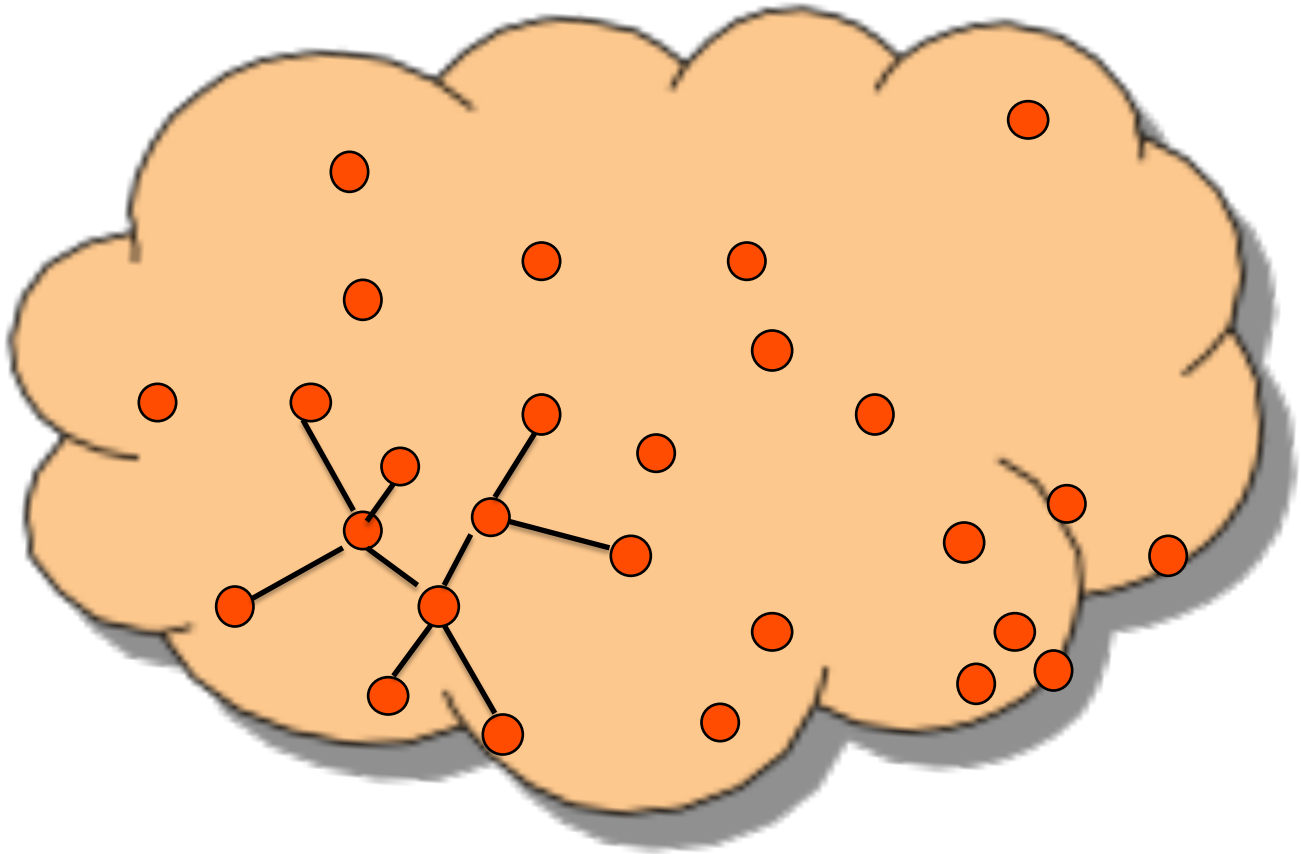
- The original: Napster
- Centralized index server(s)
  - Supported all queries
- What caused its downfall?
  - Not scalability
  - Centralization of liability

Gnutella

- The original Gnutella
  - Recent incarnations look very much like Kazaa
- Flat network
  - Join via bootstrap node
  - Connect to random set of existing hosts
  - Resolve queries by localized flooding
    - Time to live fields limit hops
- Problem
  - High bandwidth costs in control msgs
  - Flood of queries took up all available bandwidth

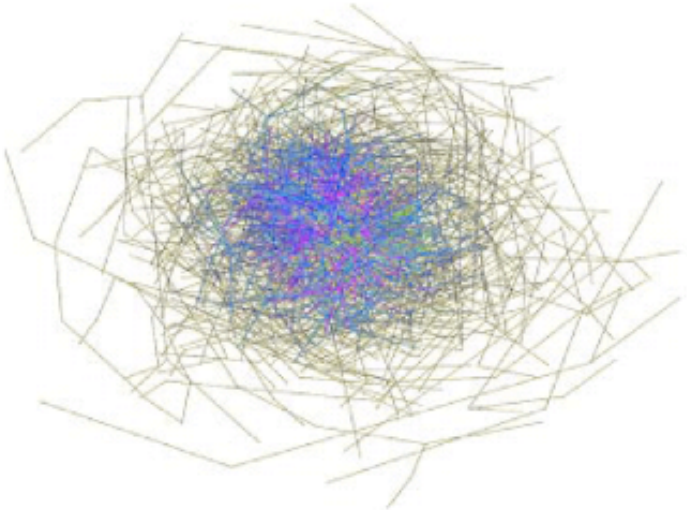


File Search via Flooding in Gnutella

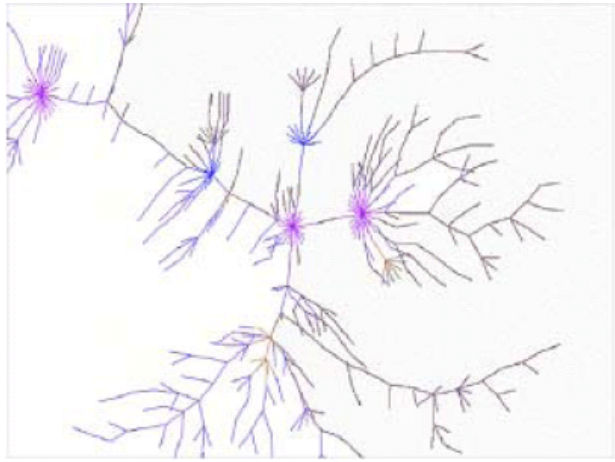


Resilience to Failures and Attacks

- Previous studies (Barabasi) show interesting dichotomy of resilience for "scale-free networks"
  - Resilient to random failures, but not attacks
- Here's what it looks like for Gnutella



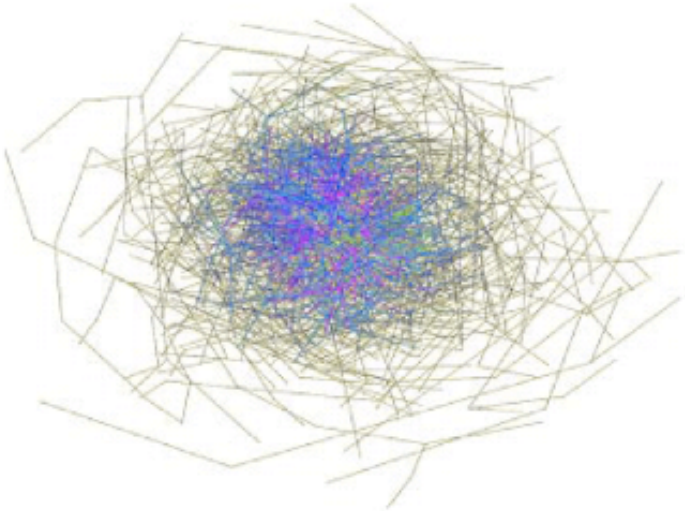
1771 Peers in Feb, 2001



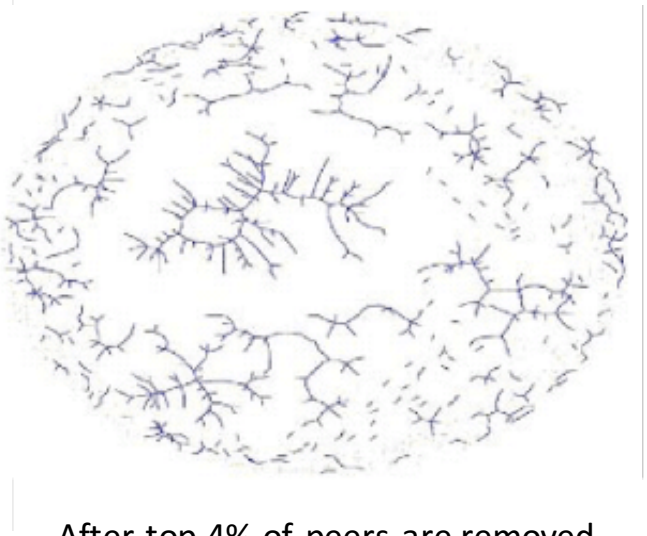
After random 30% of peers removed

Resilience to Failures and Attacks

- Previous studies (Barabasi) show interesting dichotomy of resilience for "scale-free networks"
  - Resilient to random failures, but not attacks
- Here's what it looks like for Gnutella



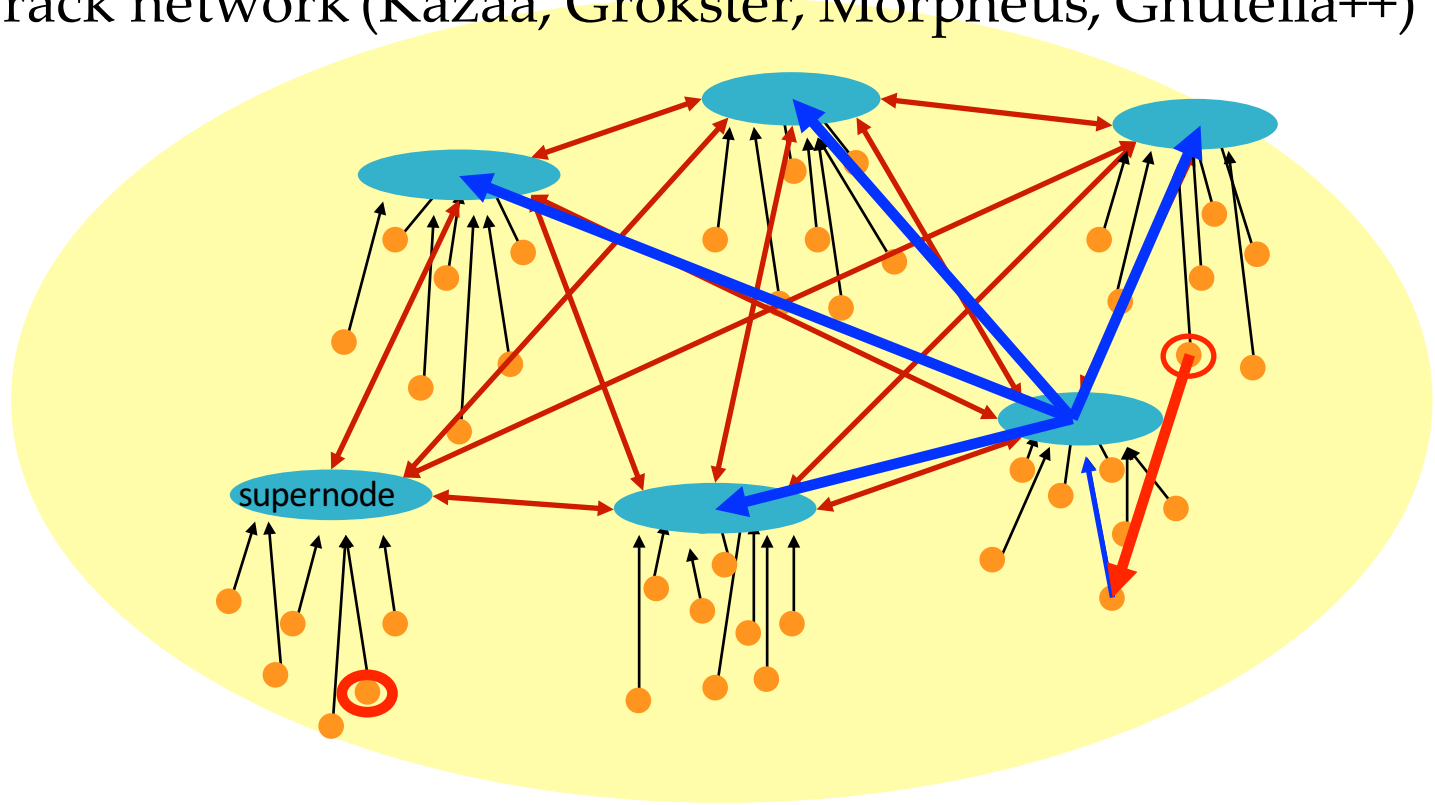
1771 Peers in Feb, 2001



After top 4% of peers are removed

Hierarchical P2P Networks

- FastTrack network (Kazaa, Grokster, Morpheus, Gnutella++)



Kazaa

- Very popular from its inception
  - Hierarchical flooding helps improve scale
  - Large shift to broadband helped quite a bit as well
  - Based in Europe, more relaxed copyright laws

- Susceptible to poison attacks
  - Mainly used by RIAA-like organizations
  - Get online and distribute large # of popular files
    - Insert static or silence in songs to make them useless
  - Quite effective in dissuading downloaders

Onwards to Bigger Things…

- Kazaa works well for audio
  - User churn is still a problem
  - High enough to disrupt downloads of large files

- BitTorrent
  - Ideal for larger files, movies, dvds, large collections…
  - Leverages simultaneous downloads by many users
    - A "seed" has a complete copy, forwards different blocks to different users
    - Users then share blocks amongst themselves
    - Typically 256KB blocks, verified using SHA-1 checksums
  - Key is incentive model for downloads: Tit-for-Tat
    - Send blocks to those who you are downloading from
    - Optimistic unchoke to bring in new users

BitTorrent Operation

- Centralized boot-strap mechanism
  - Need a .torrent file containing length, name, hashes, URL of tracker
  - Easily locatable via HTTP servers (e.g. piratebay)

- The Tracker
  - Coordinator that matches up leechers with seeds & each other
  - Maintains info about all currently active clients

- What's new?
  - Trackerless BitTorrent clients (Azureus etc.)
  - Now use structured overlays (DHTs) instead

Outline

- Unstructured Peer-to-Peer Protocols
  - How and why do they work?
  - Napster, Gnutella, Kazaa
  - BitTorrent, Freenet, Skype

- Structured Peer-to-Peer Overlays
  - Quick overview of protocols
  - Applications

Why Do We Need Structure?

- Without structure, the search problem is harder
  - Anything can be anywhere
  - Need to look through many (or all) nodes to have confidence in finding object X
- So how do we add structure? Ans: Add rules
  - Give everyone (every machine) a name
  - Give every object a name
  - Match up all object names to machine names
    - If you are looking for X, mapping(X)→Y, talk to node Y
- Challenges
  - This mapping must be easy, embedded into the network
  - It must be consistent as the machines in network change
    - One new machine should not disrupt everything

Structured Overlays

- Application infrastructure for large-scale apps

- Two key pieces of functionality
  - Scalable any to any communication in network of size N
    - Much larger scale compared to distance vector/link state
    - Each machine only needs to know Log(N) other nodes
    - Routing from A->B takes at most Log(N) steps or hops
  - Deterministic key-node mapping
    - Allows peer rendezvous using common name
    - This is the object to node mapping we need
    - We call it Key-Based Routing

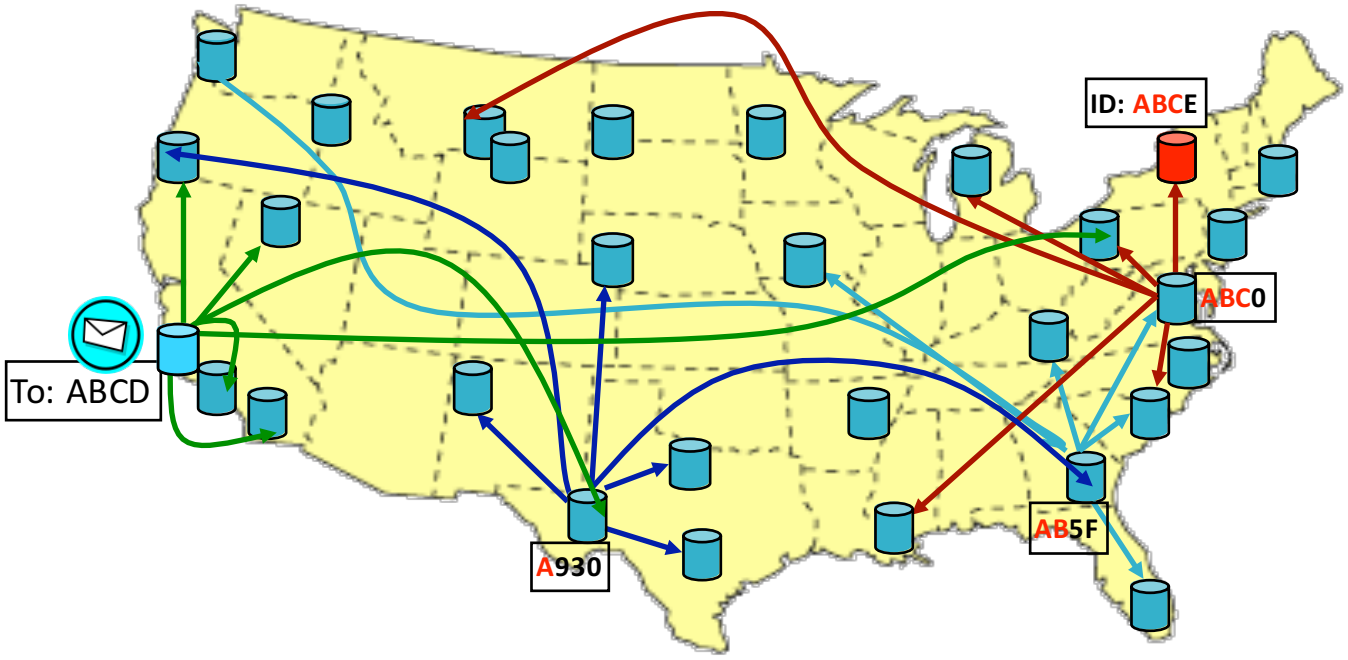
Peer-to-Peer Protocols

- Unstructured Peer to Peer Approaches
  - Napster, Gnutella, KaZaa
  - ❖probabilistic search (optimized for the hay, not the needle)
  - ❖locality-agnostic routing (resulting in high network b/w costs)

- Structured Peer to Peer Overlays
  - the first protocols (2001): Tapestry, Pastry, Chord, CAN
  - then: Kademlia, SkipNet, Viceroy, Symphony, Koorde, Ulysseus…
  - ❖distinction: how to choose your neighbors
    - ❖Tapestry/Chimera, Pastry: latency-optimized routing mesh
  - ❖distinction: application interface
    - ❖distributed hash table: put (key, data); data = get (key);
    - ❖Tapestry/Chimera: decentralized object location and routing

Structured Overlays at 10000 ft

- Node IDs and keys from randomized namespace (SHA-1)
  - incremental routing towards destination ID
  - each node has small set of outgoing routes, e.g. prefix routing
  - log (n) neighbors per node, log (n) hops between any node pair

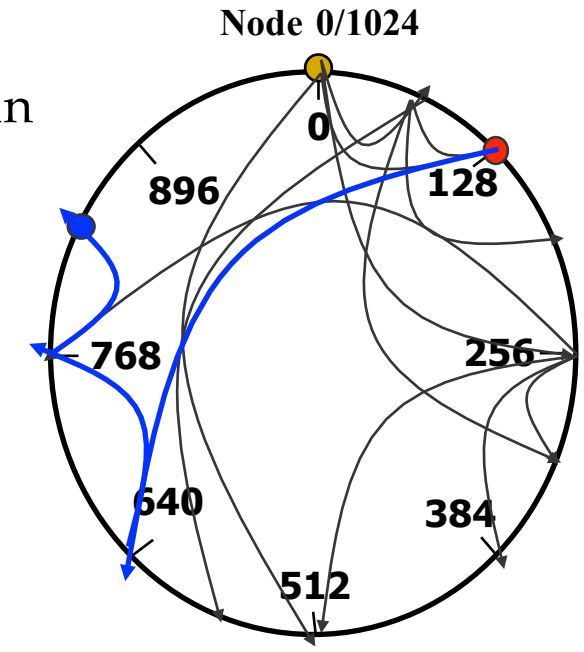


The Common Denominator

- Key-based Routing layer (Tier 0)
  - Large sparse Id space N (160 bits: $0 - 2^{160}$ represented as base b)
  - Nodes in overlay network have nodeIds $\in$ N
  - Given k $\in$ N, overlay deterministically maps k to its root node (a live node in the network)
- Intuition
  - Take set of all possible names
  - Divide space "evenly" into "buckets" stored at live nodes in network
  - Provide pointers between nodes so nodes can find desired bucket
- Main call: route (key, msg)

Chord

- NodeIDs are numbers on ring
- Closeness defined by numerical proximity
- Finger table
  - keep routes for next node 2^i away in namespace
  - routing table size: $\log_2 n$
  - n = total # of nodes
- Routing
  - iterative hops from source
  - at most $\log_2 n$ hops

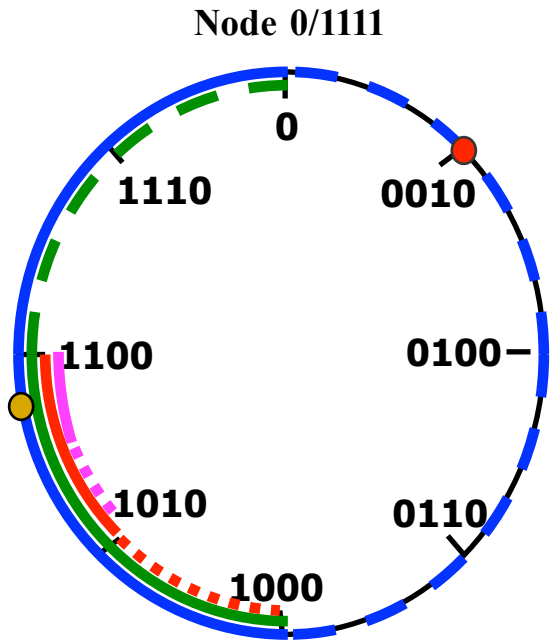


Questions About Chord

- Why is it Log_2N hops?
- How does a node join the network?
- What is the "region" each node is responsible for?
- What happens when nodes fail?

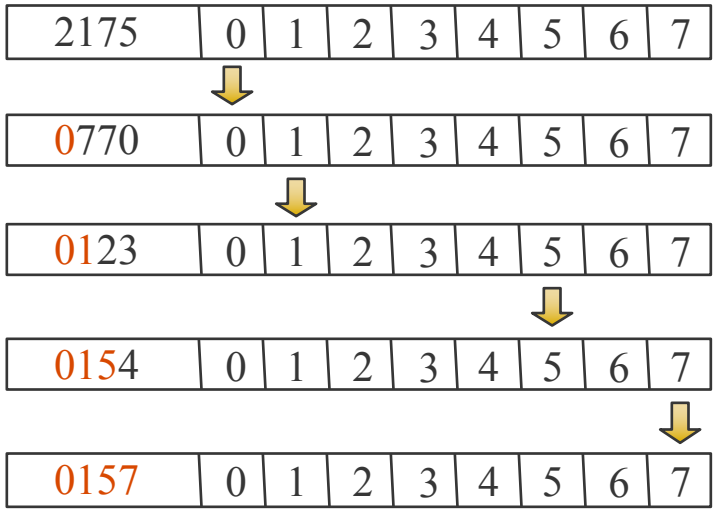
Tapestry / Pastry / Chimera

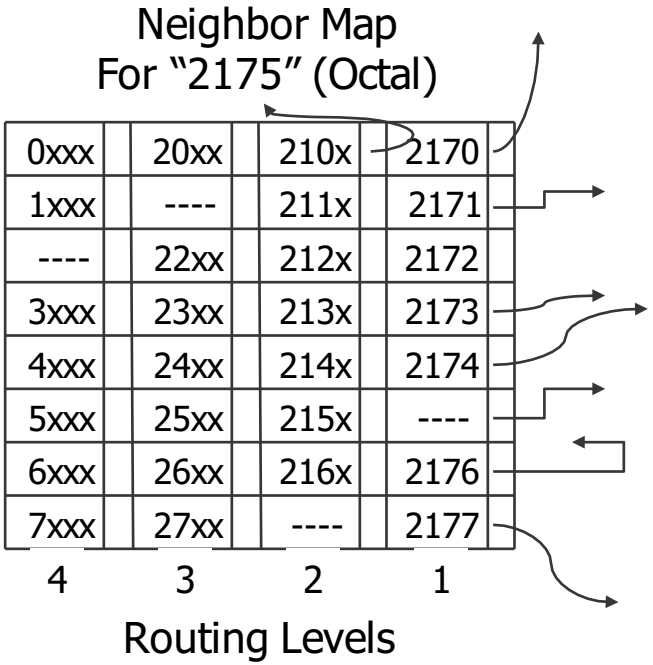
- incremental prefix routing
  - 0010→1XXX→10XX→101X→1011
- routing table
  - keep nodes matching at least i digits to destination
  - table size: b * $\log_b n$
- routing
  - recursive routing from source
  - at most $\log_b n$ hops



Routing in Detail

Example: Octal digits, 2^{12} namespace, 2175 → 0157





Today: Peer-to-Peer

- Unstructured Peer-to-Peer Protocols
  - How and why do they work?
  - Napster, Gnutella, Kazaa
  - BitTorrent, Freenet, Skype

- Structured Peer-to-Peer Overlays
  - A quick overview