Distributed Computing Systems

Peer-to-Peer

Definition of Peer-to-Peer (P2P)

- Significant autonomy from central servers
- Exploits resources at edges of Internet
  - Storage and content
  - Multicast routing
  - CPU cycles
  - Human knowledge (e.g., recommendations, classification)
- Resources at edge may have intermittent connectivity

P2P Includes

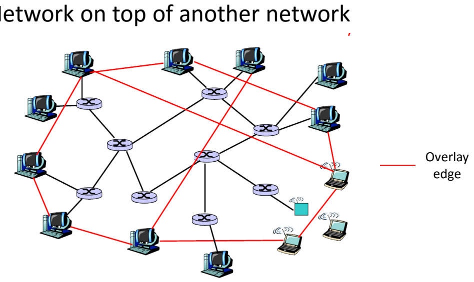
- P2P file sharing
  - Napster, gnutella, KaZaA, eDonkey …
- P2P communication
  - Instant messaging
  - Voice-over-IP (e.g. Skype)
- P2P multicast routing
  - Mbone, Yoid, Scattercast
- P2P computation
  - seti@home
- P2P apps built on overlays
  - PlanetLab

Introduction Outline

- Definition (done)
- Overlay Networks (next)
- P2P Applications

Overlay Networks

- Network on top of another network

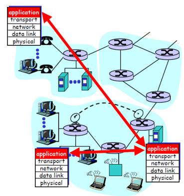


Overlay Network – Overview

- Virtual edges
  - e.g., a TCP connection
  - Simpler: an IP address (connection method not specified)
- Creation
  - May be structured (e.g., tree) or unstructured (nodes randomly choose neighbors)
  - May or may not take into account proximity of nodes
- Overlay maintenance
  - Periodically “ping” to make sure neighbor alive
  - Or, verify liveness while sending messages
  - If neighbor goes down, may establish new edge
  - New node needs to “bootstrap” in

Overlay Networks – at Application Layer

- Tremendous design flexibility
  - Topology, maintenance
  - Message types
  - Protocol
  - Underlying protocol (TCP/UDP)
- Underlying network transparent
  - But some may exploit proximity (e.g., peers in same ISP)



Examples of Overlays

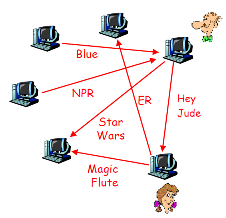
- Domain Name Service (DNS)
- Border Gateway Protocol (BGP) routers (with their peering relationships)
- Content Distribution Networks (CDNs)
- Application-level multicast (e.g., Mbone)
- And P2P applications!

Introduction Outline

- Definition (done)
- Overlay Networks (done)
- P2P Applications (next)

P2P File Sharing – General

- Alice runs P2P client on her laptop
- Intermittently connected
  - Gets new IP address each time



- Registers her content in P2P system
- Asks for "Hey Jude"
- Application displays other peers with copy
- Alice choses one, Bob
- File is copied from Bob's computer to Alice's
  → P2P
- While Alice downloads, others upload

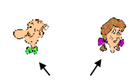
P2P File Sharing Capabilities

- Allows Alice to show directory in her file system
  - Anyone can retrieve file from it
  - Like Web server
- Allows Alice to copy files from other's
  - Like Web client
- Allows users to search nodes for content based on keyword matches
  - Like search engine (e.g., Google)

Copyright Issues (1 of 2)

Direct Infringement

- End users who download or upload copyright works



direct infringers

Indirect infringement

- An individual accountable for actions of others
- *Contributory*
  - Knew of direct infringement
  - And caused, induced or materially contributed to direct infringement
- *Vicarious*
  - Able to control direct infringers (e.g. terminate accounts)
  - And derived direct financial benefit

(Knowledge not necessary)

Copyright Issues (2 of 2)

Betamax VCR Defense
- Manufacturer not liable for contributory infringement
- "Capable of substantial, non-infringing use"
- But in Napster case, court found defense does not apply to all vicarious liability

Guidelines for P2P developers
- Total control so that can be sure there is no direct infringement

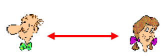
Or
- No control – no remote kill switch, no customer support
- Actively promote non-infringing use of products
- Disaggregate functions
  - Indexing, search transfer

Large P2P File Sharing Systems

- Napster
  - Disruptive, proof of concept
- Gnutella
  - Open source, non-centralized search
- KaZaA/FastTrack
  - Surpassed the Web (in terms of bytes)
- eDonkey/Overnet
  - Distributed Hash Table (distributed content)
- Is success due to massive number of servers (i.e., P2P aspect) or simply because content is "free"?

P2P Communication

- Alice runs IM client on PC
- Intermittently connects to Internet
  - Gets new IP address each time
- Registers herself with "system"
- Learns from "system" that Bob (in her buddy list) is active
- Alice initiates direct TCP connection with Bob
  →P2P
- Alice and Bob chat



- Can also be voice, video and text (e.g., Skype)

P2P Distributed Computing: seti@home

- Search for extra terrestrial (ET) intelligence
- Central site collects radio telescope data
- Data divided into work chunks (300 Kbytes)
- User obtains client
  - Runs in background (when screensaver on)
- Peer sets up TCP connection to central computer
- Downloads chunk
- Peer does FFT on chunk, uploads results, gets new chunk
- Not Peer-to-peer, but exploits resource at network edge

Outline

- Introduction (done)
- P2P file sharing techniques (next)
- Uses of P2P
- Challenges

P2P Common Primitives

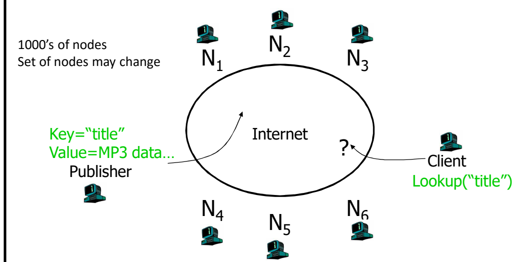
- **Join**: how to I begin participating?
- **Publish**: how do I advertise my file?
- **Search**: how to I find a file?
- **Fetch**: how to I retrieve a file?

Top 2, relatively easy
Bottom 2, more of challenge

P2P Challenges

- Challenge 1 – Search
  - Human's goals → find file
  - Given keywords or human description, find a specific file
- Challenge 2 – Fetch
  - One file determined → get bits
  - Computer goal, obtain content

Example: Searching



- Needles versus Haystacks
  Searching for top 40 pop song? Or obscure punk track '81 nobody's heard of?
- Search expressiveness
  Whole word? Regular expressions? File names? Attributes? Whole-text search?

What's out there?

	Central	Flood	Super-node flood	Route
Whole File	Napster	Gnutella		Freenet
Chunk Based	BitTorrent		KaZaA (bytes, not chunks)	(DHTs) eDonkey2k New BT

Next Topic...

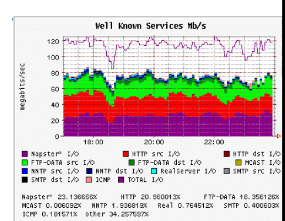
- **Centralized Database**
  - Napster
- Query Flooding
  - Gnutella
- Intelligent Query Flooding
  - KaZaA
- Swarming
  - BitTorrent
- Structured Overlay Routing
  - Distributed Hash Tables

Napster Legacy

- Paradigm shift
- Not the first (probably Eternity, from Ross Anderson in Cambridge)
  - http://www.cl.cam.ac.uk/~rja14/eternity/eternity.html
- But instructive for what it got right
- And wrong…
- Also had an economic message…
- And legal…

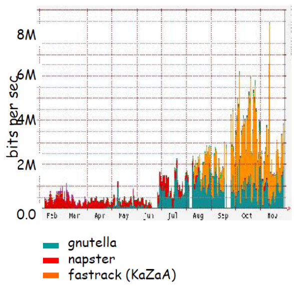
Napster: History

- **May '99**: Shawn Fanning (freshman at Northeastern) founds Napster
- **Dec '99**: first lawsuit
- **Mar '00**: 25% UWisc traffic Napster
- **Apr '01**: US Circuit Court of Appeals: "Napster knew users violating copyright laws"
- **Jul '01**: # simultaneous online users
  - Napster: 160K, Gnutella: 40K, Morpheus (KaZaA): 300K



Napster: History

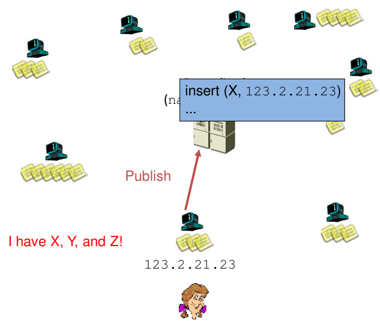
- **Jul ’01**: Napster shuts down
  - Judge orders Napster to pull plug…
- But other file sharing apps take over!



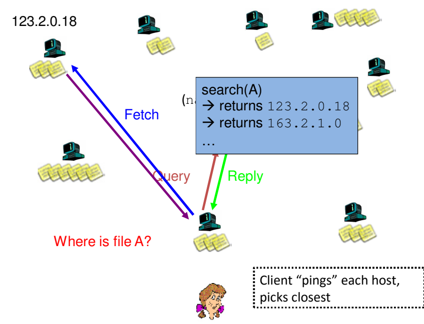
Napster: Overiew

- Application-level, client-server protocol over TCP
- 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
    - Selected as best based on “pings”
  - **Fetch**: get the file directly from peer

Napster: Publish



Napster: Search



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
  - (Napster’s server farm had difficult time keeping up with traffic)

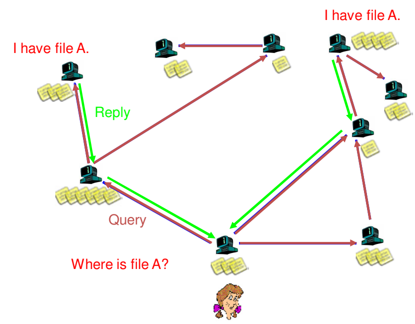
Next Topic...

- Centralized Database (Searching)
  - Napster
- **Query Flooding (Searching)**
  - Gnutella
- Supernode Query Flooding (Searching)
  - KaZaA
- Swarming (Fetching)
  - BitTorrent
- Structured Overlay Routing (Both, but mostly search)
  - Distributed Hash Tables (DHT)

Query Flooding Overview

- Decentralized method of searching
  - Central server directory no longer bottleneck
  - More difficult to "pull the plug"
- Each peer:
  - Stores selected files
  - Routes queries to and from neighbors
  - Responds to queries if files stored locally
  - Serves files
- **Join**: on startup, client contacts a few other nodes
  - these become its "neighbors"
- **Publish**: no need
- **Search**: ask neighbors (*Gnutella uses 7*), who ask their neighbors, and so on... when/if found, reply to sender.
  - TTL limits propagation (*Gnutella uses 10*)
  - Reverse path forwards responses (not files)
- **Fetch**: get the file directly from peer

Query Flooding



Flooding 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(???) – depends upon "height" of tree
  - 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
- Example: Gnutella

Gnutella: History

- **Mar '00**: J. Frankel and T. Pepper from released Gnutella (AOL)
  - Immediately withdrawn, but Slashdot-ted
- Became open source
  - Good, since initial design was poor
- Soon many other clients: Bearshare, Morpheus, LimeWire, etc.
- **'01**: many protocol enhancements including "ultrapeers" http://www.computing2010.com/expansions.php?id=07

Gnutella: Discussion

- Researchers like it because it is open source
  - But is it representative for research?
- Users' anecdotal comments: "It stinks"
  - Tough to find anything
  - Downloads don't complete
- Fixes
  - Hierarchy of nodes
  - Queue management
  - Parallel downloads
  - ...

Next Topic...

- Centralized Database (Searching)
  - Napster
- Query Flooding (Searching)
  - Gnutella
- **Supernode Query Flooding (Searching)**
  - KaZaA
- Swarming (Fetching)
  - BitTorrent
- Structured Overlay Routing (Both, but mostly search)
  - Distributed Hash Tables (DHT)

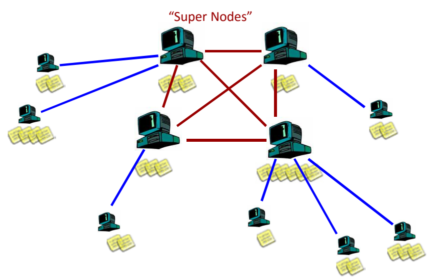
Flooding with Supernodes

- Some nodes better connected, longer connected than others
  - Use them more heavily
- Architecture
  - Hierarchical
  - Cross between Napster and Gnutella
- "Smart" Query Flooding
- **Join**: on startup, client contacts a "supernode"
  - may at some point become one itself
- **Publish**: send list of files to supernode
- **Search**: send query to supernode, supernodes flood query amongst themselves.
- **Fetch**: get the file directly from peer(s)
  - can fetch simultaneously from multiple peers

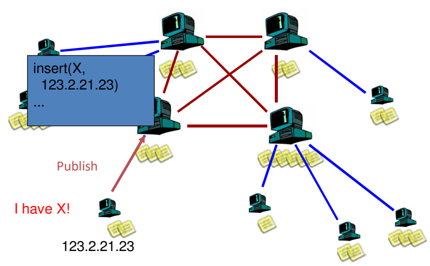
Stability and Superpeers

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating query to sub-node would take more b/w than answering it yourself
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you've been on good predictor of how long you'll be around

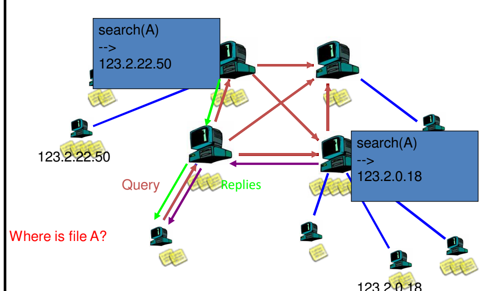
Supernodes Network Design



Supernodes: Publish



Supernodes: Search



Supernodes: Fetch (And use of hashes...)

- More than one node may have requested file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B

Supernode Flooding Discussion

- Pros:
  - Tries to take into account node heterogeneity
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
  - Scales better
- Cons:
  - Still no real guarantees on search scope or search time
- Similar behavior to plain flooding, but better
- Example: KaZaA

KaZaA: History

- In 2001, KaZaA created by Dutch company Kazaa BV
- Single network called FastTrack used by other clients as well
  - Morpheus, giFT, etc.
- Eventually protocol changed so other clients could no longer talk to it
- Most popular file sharing network in 2005 with >10 million users, sharing over 10,000 terabytes of content (numbers vary)
  - More popular than Napster
  - Over 50% of Internet traffic at one time
- MP3s & entire albums, videos, games

KaZaA: The Service

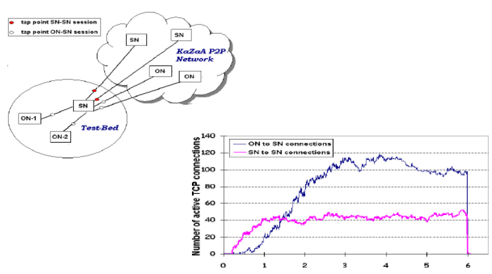
- Optional parallel downloading of files
  - User can configure max down and max up
- Automatically switches to new download server when current server unavailable
- Provides estimated download times
- Queue management at server and client
  - Frequent uploaders can get priority
- Keyword search
- Responses to keyword queries come in waves: stops when x responses are found
- From user's perspective, resembles Google, but provides links to mp3s and videos rather than Web

KaZaA: Technology

- Software
  - Proprietary
  - Control data encrypted
  - Everything in HTTP request and response messages
- Each peer a supernode or assigned to a supernode
  - Each SN has about 100-150 children
  - Each SN has TCP connections with 30-50 other supernodes

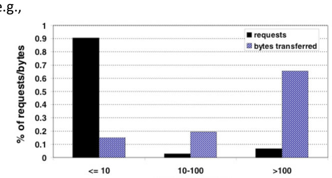
(Measurement study next slide)

KaZaA Measurement Study



KaZaA Measurement Study

- KaZaA is more than one workload!
  - Many files < 10MB (e.g., Audio Files)
  - Many files > 100MB (e.g., Movies)



from Gummadi *et al.*, *SOSP* 2003

KaZaA: Architecture

- Nodes with more connection bandwidth and more availability → supernodes (SN)
- Each supernode acts as a min-Napster hub
  - Tracking content of children (ordinary nodes, ON)
  - Maintaining IP addresses of children
- When ON connects to SN, upload metadata
  - File name
  - File size
  - Content hash – used for fetch request, rather than name
  - File descriptions – used for keyword matches

KaZaA: Overlay Maintenance

- List of potential supernodes when download software
- New peer goes through list until finds operational supernode
  - Connects, contains up-to-date list (200 entries)
  - Nodes in list are "close" to ON
  - Node pings 5 nodes, connects to one
- If supernode goes down, node obtains updated list and choses new supernode

KaZaA: Queries

- Node first sends query to supernode
  - Supernode responds with matches
  - If x matches found, done
- Otherwise, supernode forwards query to subset of supernodes
  - If total of x matches found, done
- Otherwise, query further forwarded
  - By original supernode

KaZaA-lite

- Hacked version of KaZaA client
- No spyware, no pop-up windows
- Everyone is rated as a priority user
- Supernode hopping
  - After receiving replies from SN, ON often connects to a new SN an re-sends query
  - SN does not cache hopped-out ON's metadata

KaZaA: Downloading & Recovery

- If file is found in multiple nodes, user can select parallel downloading
  - Identical copies identified by ContentHash
- HTTP byte-range header used to request different portions of the file from different nodes
- Automatic recovery when server peer stops sending file
  - ContentHash

KaZaA: Summary

- KaZaA provides powerful file search and transfer service without server infrastructure
- Exploit heterogeneity
- Provide automatic recovery for interrupted downloads
- Powerful, intuitive user interface
- Copyright infringement
  - International cat-and-mouse game
  - With distributed, serverless architecture, can the plug be pulled?
  - Prosecute users?
  - Launch DoS attack on supernodes?
  - Pollute? (Copyright holder intentionally puts bad copies out)

Searching & Fetching

- Query flooding finds:
  - An object (filename or hash)
  - A host that serves that object
- In Query flooding systems, download from host that answered query
- Generally uses only one source
- Can we do better?

Next Topic...

- Centralized Database (Searching)
  - Napster
- Query Flooding (Searching)
  - Gnutella
- Supernode Query Flooding (Searching)
  - KaZaA
- **Swarming (Fetching)**
  - BitTorrent
- Structured Overlay Routing (Both, but mostly search)
  - Distributed Hash Tables (DHT)

Fetching in Parallel and Swarming

- When you have an object ID,
- Get list of peers serving that ID
  - Easier than the keyword lookup
  - Queries are structured
- Download in parallel from multiple peers
- "Swarming"
  - Download from others downloading same object at same time
- Example: BitTorrent

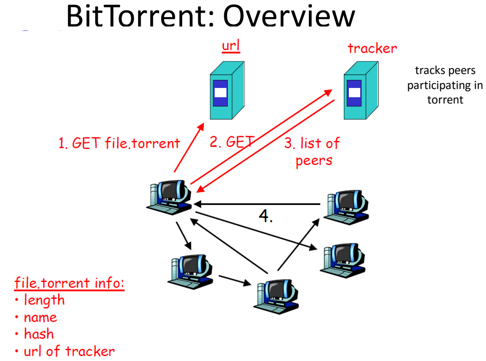
BitTorrent: Swarming

- 2001, BramCohen debuted BitTorrent
  - Was open source, now not (µTorrent)
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient *Fetching*, not *Searching*
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some "real" publishers:
  - Blizzard Entertainment has used it to distribute beta of games

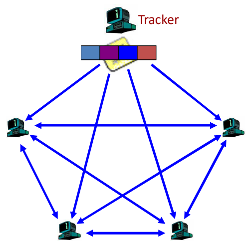
BitTorrent: Overview

- 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

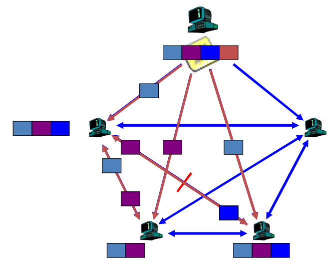
BitTorrent: Overview



BitTorrent: Publish/Join



BitTorrent: Fetch



BitTorrent: Sharing Strategy

- File is broken into pieces
  - Typically 256 Kbytes
  - Upload pieces while download
- Piece selection
  - Select rarest piece for request
  - Except at beginning, select random pieces
- 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

BitTorrent: Summary

- Pros
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons
  - Central tracker server needed to bootstrap swarm
  - Tracker is a design choice, not a requirement
    - Newer BT variants use a "distributed tracker" - a Distributed Hash Table

Next Topic...

- Centralized Database (Searching)
  - Napster
- Query Flooding (Searching)
  - Gnutella
- Supernode Query Flooding (Searching)
  - KaZaA
- Swarming (Fetching)
  - BitTorrent
- **Structured Overlay Routing (Both, but mostly search)**
  - Distributed Hash Tables (DHT)

Directed Searches

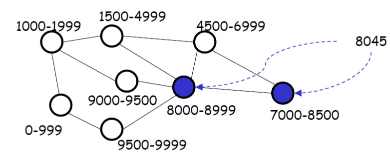
- Idea:
  - Assign particular nodes to hold particular content (or pointers to it)
  - When node wants that content, go to node that is supposed to have or know about it
- Challenges:
  - Distributed: want to distribute responsibilities among existing nodes in overlay
  - Adaptive: nodes join and leave P2P overlay
    - Distribute knowledge responsibility to joining nodes
    - Redistribute responsibility knowledge from leaving nodes

Distributed Hash Table (DHT): Overview

- **Abstraction**: a distributed "hash-table" data structure:
  - put(id, item);
  - item = get(id);
- **Implementation**: nodes in system form distributed data structure
  - Can be Ring, Tree, Hypercube, …

DHT: Step 1 – The Hash

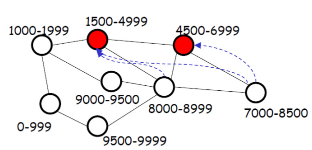
- Introduce a hash function to map the object being searched for to a unique identifier:
  - e.g., h("Led Zepplin") → 8045
- Distribute range of hash function among all nodes in network



- Each node must "know about" at least one copy of each object that hashes within its range (when one exists)

DHT: "Knowing About Objects"

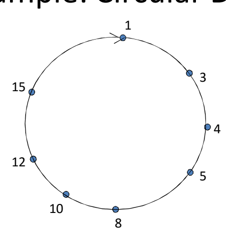
- Two alternatives
  - Node can cache each (existing) object that hashes within its range
  - Pointer-based: level of indirection – node caches pointer to location(s) of object



DHT: Step 2 – Routing

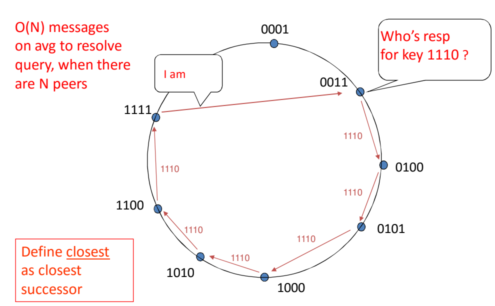
- For each object, node(s) whose range(s) cover that object must be reachable via "short" path
  - by querier node (assumed can be chosen arbitrarily)
  - by nodes that have copies of object (when pointer-based approach is used)
- Different approaches (CAN, Chord, Pastry, Tapestry) differ fundamentally only in routing approach
  - Any "good" random hash function will suffice

Example: Circular DHT (1)

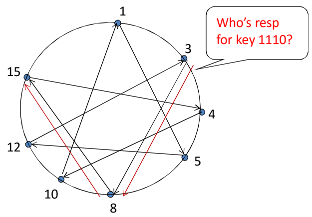


- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

Example: Circle DHT (2)

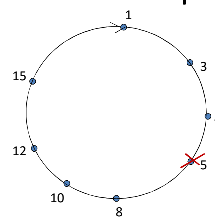


Example: Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of predecessor, successor, short cuts.
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Example: Peer Churn



- To handle peer churn, require each peer to know IP address of its two successors.
- Each peer periodically pings its two successors to see if still alive

- Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.
- What if peer 13 wants to join?

DHT: Operations

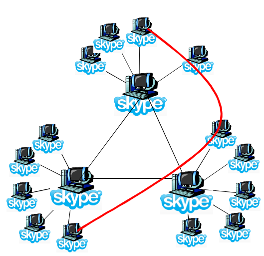
- **Join**: On startup, contact "bootstrap" node and integrate into distributed data structure → get a *node id*
- **Publish**: Route publication for *file id* toward close *node id* along data structure (e.g. next in ring)
- **Search**: Route query for file id toward a close node id
- **Fetch**: Two options:
  - Publication contains actual file → fetch from where query stops
  - Publication says "I have file X" → query says 128.2.1.3 has X, use IP routing to get X from 128.2.1.3

DHT: Discussion

- Pros:
  - Guaranteed lookup
  - O(log *N*) per node state and search scope
- Cons:
  - Not used
    - Academically popular since 2k, but in practice, not so much
  - Supporting non-exact match search is hard
  - Churn hard

Peers as Relays

- Problem when both Alice and Bob are behind "NATs".
  - NAT prevents outside peer from initiating call to insider peer (e.g. can't be server)
- Solution:
  - Relay is chosen (based on connectivity to both)
  - Each peer initiates session with relay
  - Peers can now communicate through NATs via relay
- (Used by Skype)



When 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"

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 subset to user
- Very, very hard for 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 data they held
  - Hard drives are *huge*, but upload bandwidths relatively tiny
    - Takes many hours to upload contents of hard drive
    - Very expensive leave/replication situation!

P2P: Summary

- Many different styles
  - Centralized, flooding, swarming, unstructured and structured routing
- Lessons learned:
  - Single points of failure are bad
  - Flooding messages to everyone is bad
  - Underlying network topology is important
  - Not all nodes are equal
  - Need incentives to discourage freeloading