Peer to Peer Networks

P2P Application Examples

- File Sharing Applications (esp.Mp3 music)
  - Napster, Gnutella, KaZaA, eDonkey, eMule, KAD, …
- Video downloads, video-on-demand (VoD), large-scale file distribution (e.g., software updates)
  - BitTorrent & variants, …
- Skype (P2P VoIP & video conferencing)
- (real-time/near-real time) video broadcasting, video streaming, …
  - pplive, qqlive, …..
- Technology Applications
  - Dynamo: Amazon storage substrate
  - Akamai Netsession: P2P streaming client
  - WebRTC for browser-browser video streaming

2

Peer to Peer Networks

- Reducing load of servers
- Faster distribution of content
- Scalable content distribution
- Low barrier to deployment
- Organic growth (self-scalable)
- Resilience to attacks and failures
- Resources grow with size of network
- Churn of peers makes network unstable

Locating the Relevant Peers

- Three main approaches
  - Central directory (Napster)
  - Query flooding (Gnutella)
  - Hierarchical overlay (Kazaa, modern Gnutella)
- Design goals
  - Scalability
  - Simplicity
  - Robustness
  - Plausible deniability

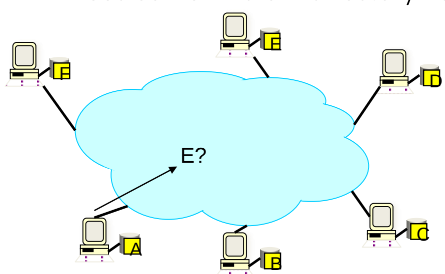
4

P2P (Application) Model

- Each user stores a subset of files (content)
- Each user has access (can download) files from all users in the system

Key Challenges in "pure" peer-to-peer model

- How to locate your peer & find what you want?
- Need some kind of "directory" or "look-up" service



- centralized
- distributed, using a hierarchal structure
- distributed, using a flat structure
- distributed, with no structure ("flooding" based)
- distributed, using a "hybrid" structured/unstructured approach

5

Other Challenges

Technical:

- Scale: up to hundred of thousands or millions of machines
- Churn: machines can come and go any time

Social, economic & legal:

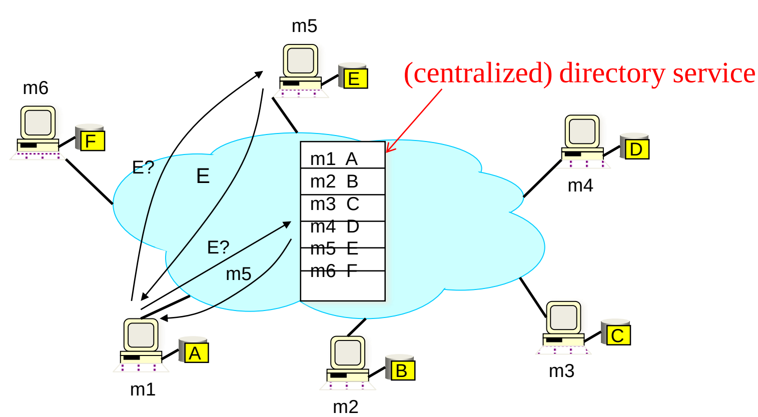
- Incentive Issues: free-rider problem
  - Vast majority of users are free-riders
    - most share no files and answer no queries
  - A few individuals contributing to the "public good"
- Copyrighted content and piracy
- Trust & security issues

6

Unstructured P2P Applications

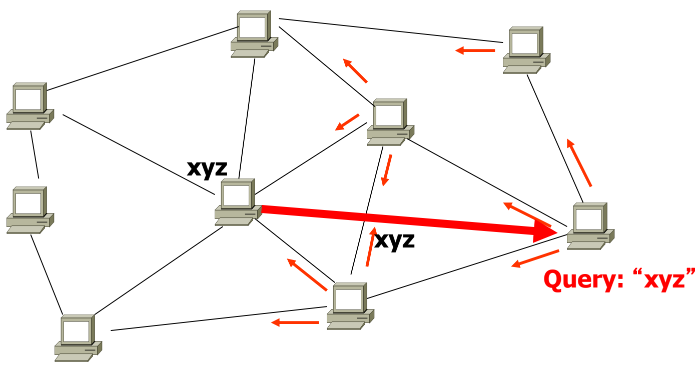
- Napster
  - a *centralized* directory service
  - peers directly download from other peers
- Gnutella
  - fully distributed directory service
  - discover & maintain neighbors, *ad hoc* topology
  - flood & forward queries to neighbors (with bounded hops)
- Skype
  - exploit heterogeneity, certain peers as "super nodes"
  - two-tier hierarchy: when join, contact a super-node
  - smart query flooding
  - peer may fetch data from multiple peers at once
- Pros and Cons of each approach?

Napster Architecture: An Illustration



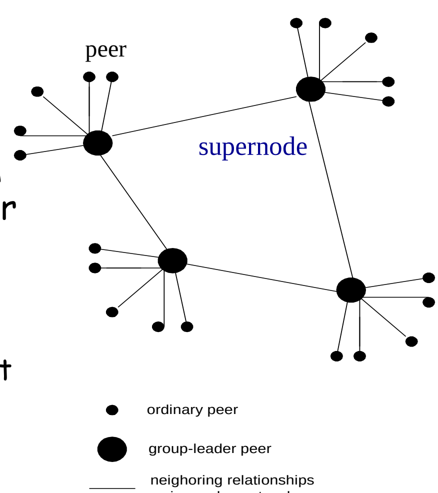
Gnutella

- Ad-hoc topology
- Queries are flooded for bounded number of hops
- No guarantees on recall



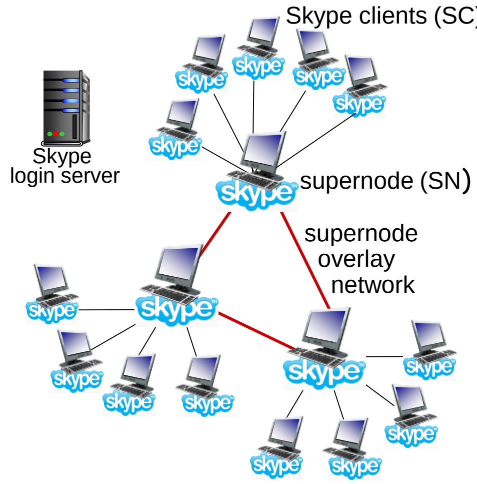
KaZaA: Exploiting Heterogeneity

- Each peer is either a group leader or assigned to a group leader (supernode)
  - TCP connection between peer and its group leader
  - TCP connections between some pairs of group leaders
- Group leader tracks the content in all its children
- Q: how to select supernodes?



Voice-over-IP: Skype

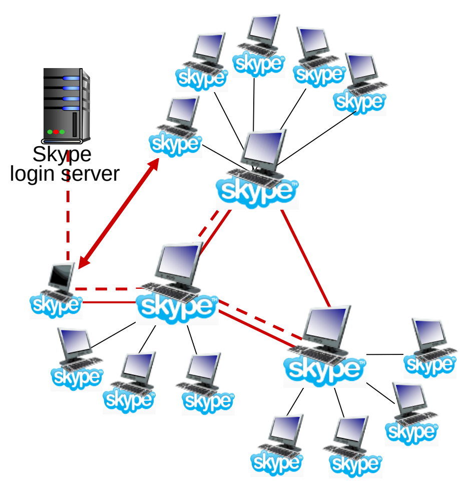
- proprietary application-layer protocol (inferred via reverse engineering)
  - encrypted msgs
- P2P components:
  - clients: Skype peers connect directly to each other for VoIP call
  - super nodes (SN): Skype peers with special functions
  - overlay network: among SNs to locate SCs
  - login server



P2P voice-over-IP: Skype

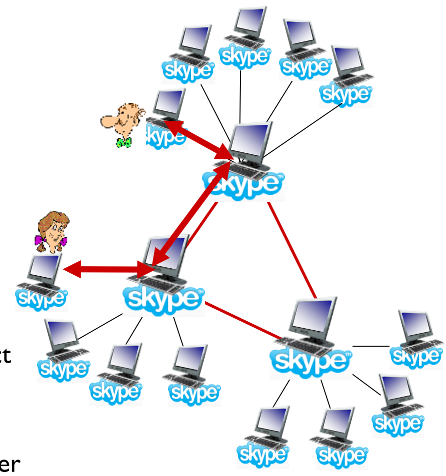
Skype client operation:

1. joins Skype network by contacting SN (IP address cached) using TCP
2. logs-in (username, password) to centralized Skype login server
3. obtains IP address for callee from SN, SN overlay
   - or client buddy list
4. initiate call directly to callee



Skype: peers as relays

- *problem:* both Alice, Bob are behind "NATs"
  - NAT prevents outside peer from initiating connection to insider peer
  - inside peer *can* initiate connection to outside
- relay solution: Alice, Bob maintain open connection to their SNs
  - Alice signals her SN to connect to Bob
  - Alice's SN connects to Bob's SN
  - Bob's SN connects to Bob over open connection Bob initially initiated to his SN

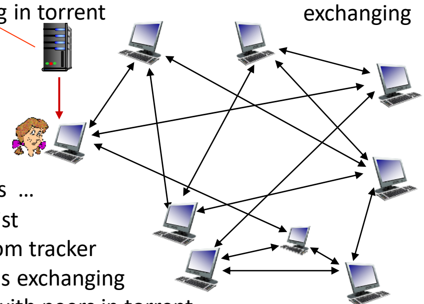


P2P file distribution: BitTorrent

- file divided into 256Kb chunks
- peers in torrent send/receive file chunks

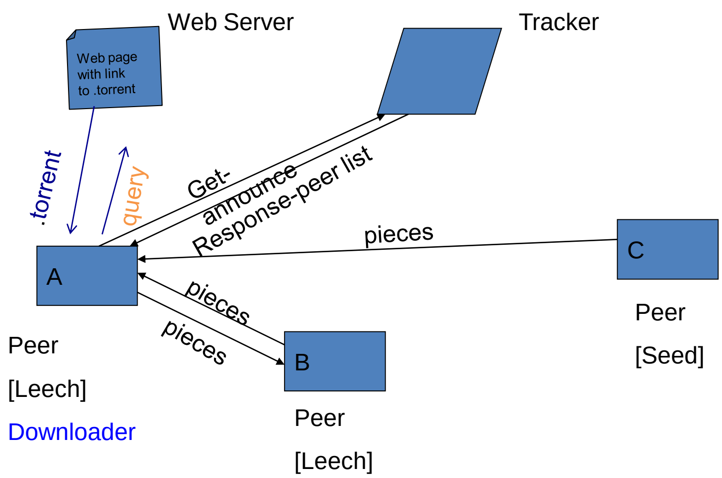
tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

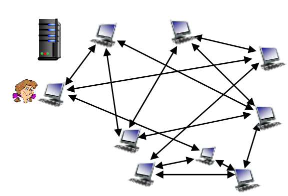


Alice arrives …
… obtains list
of peers from tracker
… and begins exchanging
file chunks with peers in torrent

BitTorrent: Overall Architecture



P2P file distribution: BitTorrent



- peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers
- peer may change peers with whom it exchanges chunks
- *churn:* peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

BitTorrent: requesting, sending file chunks

Requesting chunks:

- at any given time, different peers have different subsets of file chunks
- periodically, Alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first

Sending chunks: tit-for-tat

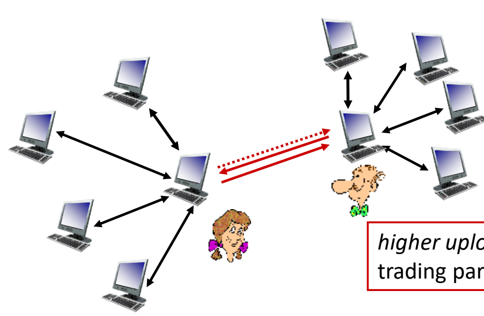
- Alice sends chunks to those four peers currently sending her chunks *at highest rate*
  - other peers are choked by Alice (do not receive chunks from her)
  - re-evaluate top 4 every10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

BitTorrent: tit-for-tat

(1) Alice "optimistically unchokes" Bob

(2) Alice becomes one of Bob's top-four providers; Bob reciprocates

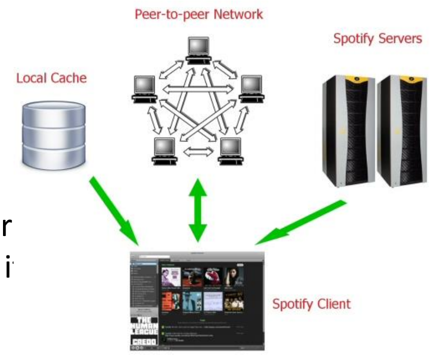
(3) Bob becomes one of Alice's top-four providers



higher upload rate: find better trading partners, get file faster !

Spotify

- Uses BT as basic protocol
  - Uses server for first 15s
  - Tries to find peers and download from them
  - Only 8.8% of bytes come from servers
- When 30s left
  - Starts searching for next tr
  - Uses sever with 10s to go i no peers found



BitTorrent & Video Distribution

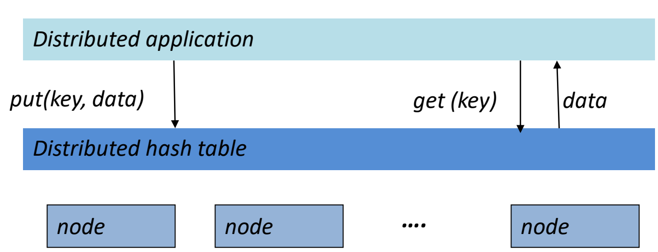
- Designed for large file (e.g., video) downloads
  - esp. for popular content, e.g. flash crowds
- Focused on efficient *fetching*, not search
  - Distribute same file to many peers
  - Single publisher, many downloaders
- Divide large file into many pieces
  - Replicate different pieces on different peers
  - A peer with a complete piece can trade with other peers
- Allows simultaneous downloading
  - Retrieving different parts of the file from different peers at the same time
  - Fetch rarest piece first
- Also includes mechanisms for preventing “free riding”

Structured P2P Networks

- Introduce a structured logical topology
- Abstraction: a distributed hash table data structure
  - put(key, object); get (key)
  - key: identifier of an object
  - *object* can be anything: a data item, a node (host), a document/file, pointer to a file, …
- Design Goals: guarantee on recall
  - i.e., ensure that an item (file) identified is always found
  - Also scale to hundreds of thousands of (or more) nodes
  - handle rapid join/leave and failure of nodes
- Proposals
  - Chord, CAN, Kademlia, Pastry, Tapestry, etc

21

Distributed hash table



DHT provides the information look up service for P2P applications.

- Nodes uniformly distributed across *key* space
- Nodes form an *overlay* network
- Nodes maintain list of neighbors in routing table
- Decoupled from physical network topology

22

Key Ideas (Concepts & Features)

- Keys and node ids map to the same "flat" id space
  - node ids are thus also (special) keys!
- Management (organization, storage, lookup, etc) of keys using consistent hashing
  - distributed, maintained by all nodes in the network
- (Logical) distance defined on the id space: structured!
  - different DHTs use different distances/structures
- Look-up/Routing Tables ("finger table" in Chord)
  - each node typically maintains O(log n) routing entries
  - organizing using structured id space: more information about nodes closer-by; less about nodes farther away
- Bootstrap, handling node joins/leaves or failures
  - when node joins: needs to know at least one node in the system
- Robustness/resiliency through redundancy

23

DHT identifiers

- assign integer identifier to each peer in range $[0,2^n-1]$ for some n.
  - each identifier represented by n bits.
- require each key to be an integer in same range
- to get integer key, hash original key
  - e.g., key = hash("Led Zeppelin IV")
  - this is why its is referred to as a *distributed "hash" table*

Consistent Hashing

- Every key is hashed to generate a number in a circular range
- Every node/replica assigned an ID in the same space

- A key is stored at the first N nodes which succeed the hash of the key in the circular ring
  - Called "preference list" of the key
- First node on the list is the coordinator for the key
  - Get/put operations at all nodes managed by coordinator

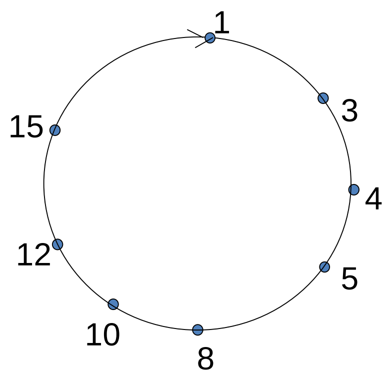
- For better load balancing, every node is treated as multiple virtual nodes, assigned many positions on list
  - Preference list will contain N distinct physical nodes

Consistent Hashing Properties

- Load balance: all nodes receive roughly the same number of keys

- For *N* nodes and *K* keys, with high probability
  - each node holds at most $(1+\varepsilon)K/N$ keys
  - (provided that K is large enough compared to N)

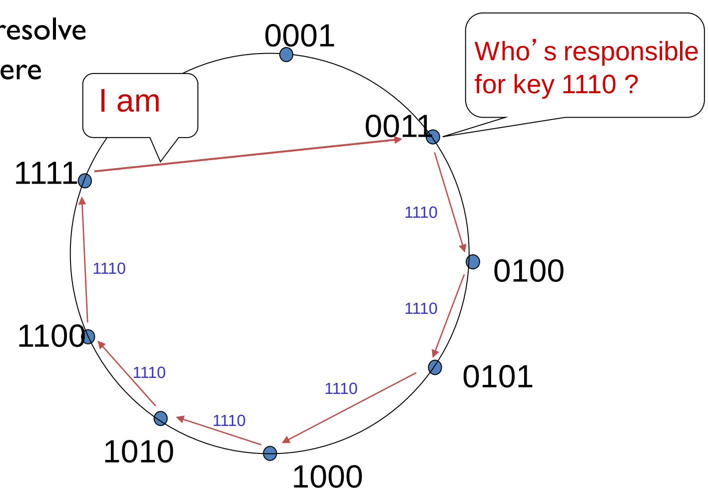
Circular DHT (I)



- each peer *only* aware of immediate successor and predecessor.
- "overlay network"

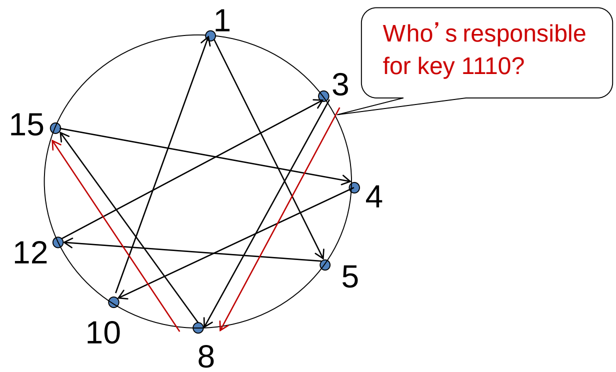
Circular DHT (I)

O(N) messages on average to resolve query, when there are *N* peers



Define closest as closest successor

Circular DHT with finger table



- 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