Peer-to-Peer Protocols

Today

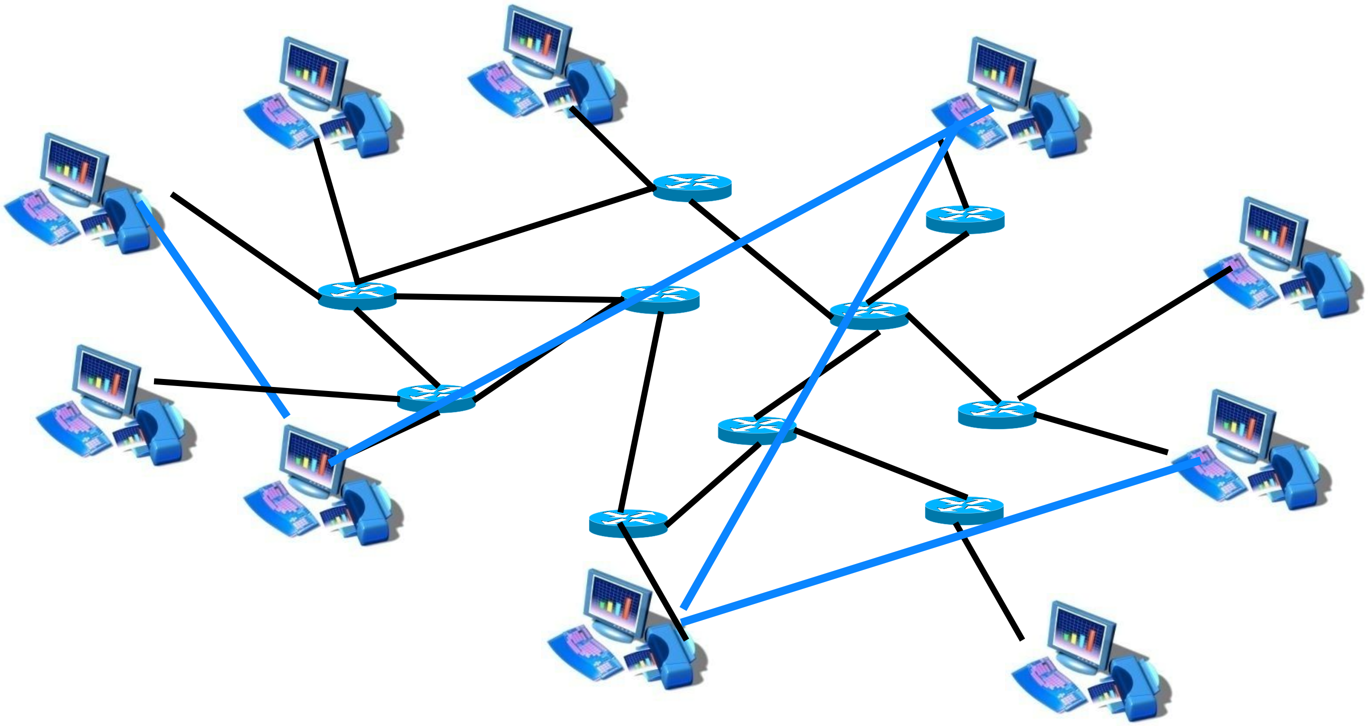
- Unstructured
- Structured or DHT

Definition

- Significant autonomy form central servers
- Exploits resources at the edges of the network
  - Storage and content – Gnutella, Kazaa, eDonkey
  - Bandwidth – BitTorrent, CoralCDN
  - CPU cycles – SETI@home, fold@home
  - People cycles :) – Wikipedia, NASA clickworkers
- Resources at the edge have intermittent connectivity – churn
- A broad definition
  - P2P file sharing, P2P communication, P2P computation, DHT and its apps …

Overlay networks

- P2P applications rely on overlay network protocols for object storage/retrieval & message routing
- Peer hosts connect to each other in arbitrary ways
  - Typically TCP connections
- Overlay must be built, maintained and refined



P2P and overlays

- Unstructured
  - Few constraints on overlay construction & data placement
  - Could support arbitrary complex queries, highly resilient to churn
  - *Restricted to inefficient, near-blind search strategies*
- Structured (DHT ~ Distributed Hash Tables)
  - Constraining overlay structure & data placement
  - Efficient object discovery
  - *Potential problems handling churn, exploiting node heterogeneity & supporting complex queries*
- Overlays are not new
  - e.g. DNS

P2P computing – SETI@home

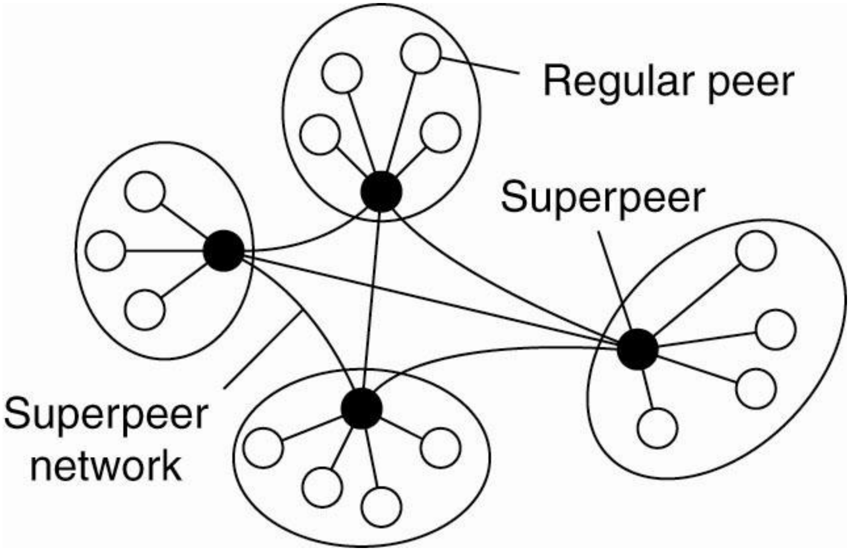
- Search for Extra-Terrestrial Intelligence
  - Search for evidence of radio transmission from ET
- Central site collects radio telescope data
- Data is split into work chunks of 300KB
- Users get client that runs in the background
  - Sets up TCP connection to central and downloads chunk
  - Peer does FFT on chunk, uploads results and get new chunk
- No really peer *to* peer but leverage resources at the edge of the network

Unstructured P2P systems

- Many unstructured P2P systems attempt to maintain a random graph:
- Basic idea – each node contacts a randomly selected other node
  - Let each peer maintain a partial view of the network, consisting of *c* other nodes
  - Each node *P* periodically selects a node *Q* from its partial view
  - *P and Q* exchange information and exchange members from their respective partial views
- An exclusive pull/push model can easily conduct to disconnected overlays
- In general, much easier to leave/join the network

Super-peers in unstructured P2P systems

- Sometimes it may help break with the symmetric nature of P2P – super/ultra-peers
- Some obvious examples
  - Transiency – pick the most stable ones
  - Search – have them keep the indexes for scalable searches
  - Organization – have them monitor the state of the network

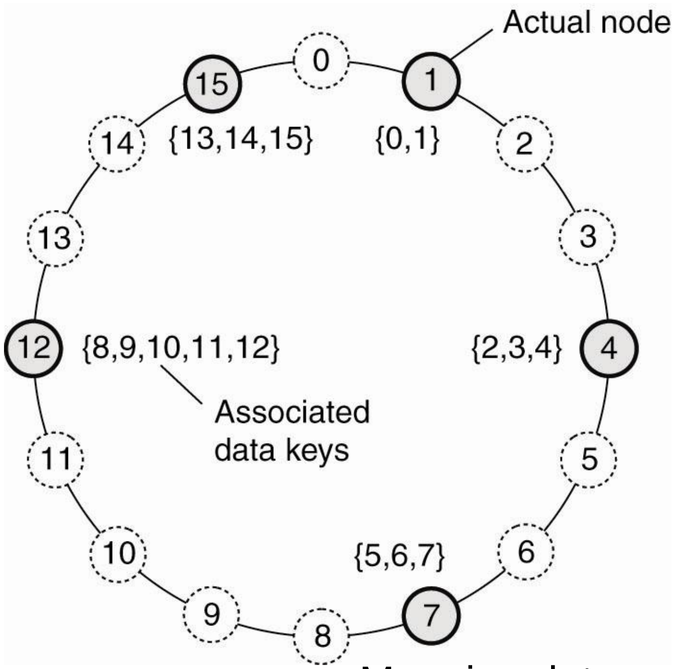


Gnutella – unstructured P2P network

- One of the three most popular P2P networks, by mid 2005, Gnutella's population was 1.8 million
- Developed in 2000, out of Nullsoft (bought by AOL)
- Peers setup random connections with other peers
  - They need a bootstrap mechanism - website
  - All peers are equal & can connect to anyone (V0.4) or
  - (weak) leaf-peers can only connect to super-peer (V0.6)
- Ping/pong & byes for control
- No constraints on placement of data objects (or pointers to)
- Flooding (ask 7, who will ask other 7, who …) or random walk for search

Structured P2P systems

- Organize the nodes in a structured overlay network such as a logical ring, and make specific nodes responsible for services based only on their ID
- The system provides an operation LOOKUP(key) to route the lookup request to the associated node
- Node join is straightforward
  - Generate a random id
  - Do a lookup on id, getting the succ(id)
  - Contact succ(id), and its predecessor, to insert itself in the ring
  - Transfer data items from succ(id) to new node



Mapping data items onto Chord nodes

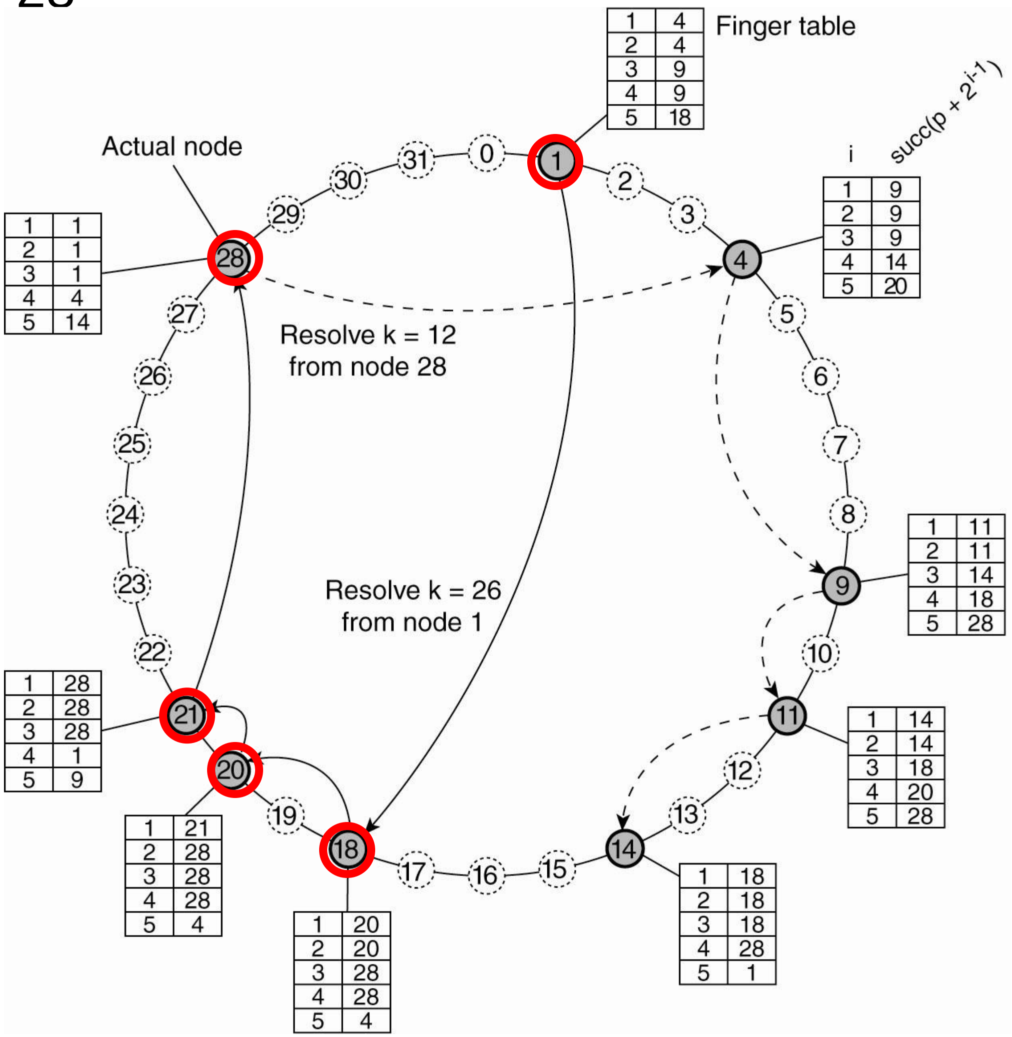
Distributed Hash Tables (DHT) - Chord

- Consider the organization of nodes into a logical ring
  - Each node is assigned a random *m-bit identifier.*
  - Every entity is assigned a unique *m-bit key.*
  - Entity with key *k falls under jurisdiction of node* with smallest *id* $\geq k$ *(called its successor)*
- Non-solution – linear search along the ring
- Finger tables – each node *p* maintains a finger table $FT_p[]$ with at most *m* entries: $FT_p[i] = succ(p + 2^{i-1})$
  - Basically shortcuts to nodes in the identifier space
- $FT_p[i]$ points to first node succeeding *p* by at least 2^{i-1}.
  - To look up key *k*, *p* forwards the request to node with index *j* satisfying $q = FT_p[j] \leq k < FT_p[j + 1]$
  - If $p < k < FT_p[1]$, the request is also forwarded to $FT_p[1]$

Chord DHT – resolving keys

Resolving key 26 from node 1
and key 12 from node 28

Entity with key *k falls under jurisdiction of node* with smallest *id* ≥ *k (called its successor)*



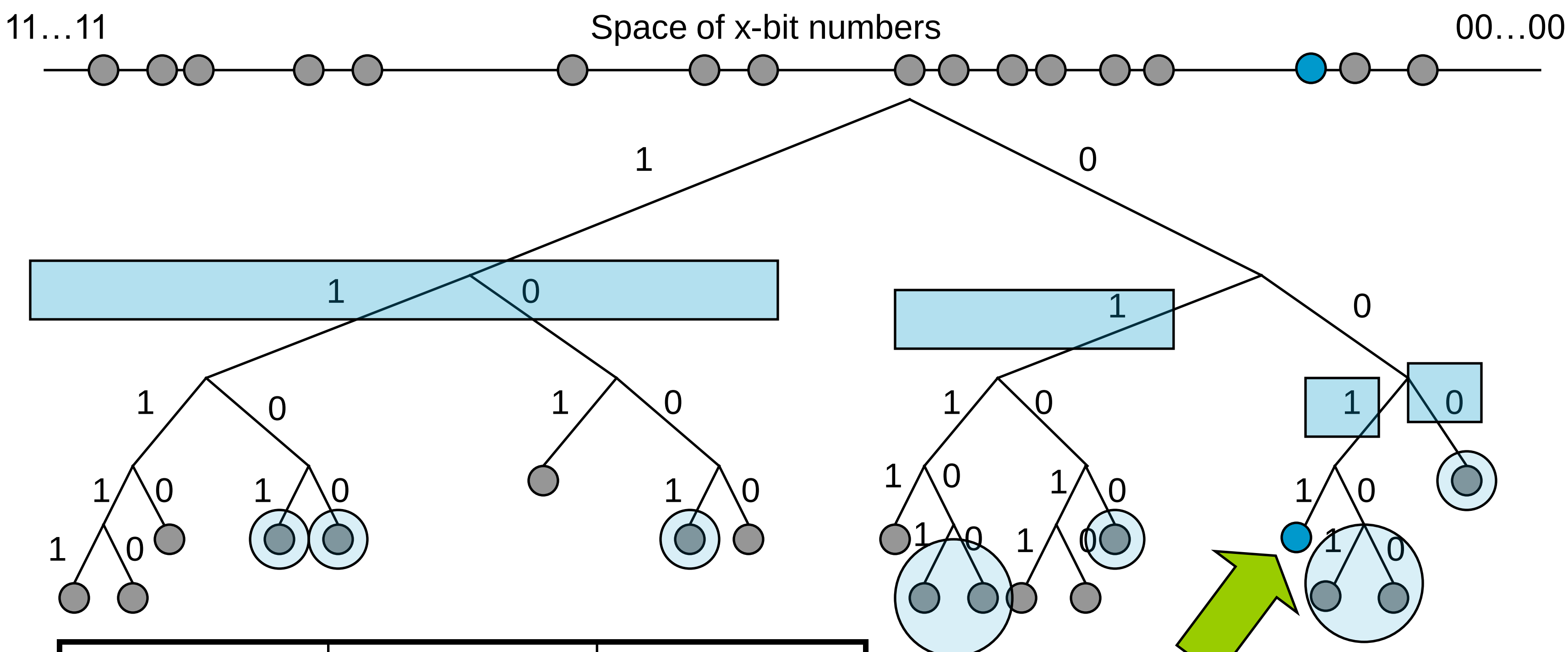
Kademlia

- Each peer & each object has a unique hash ID
  - 160-bit
- <key,value> pairs stored on nodes with IDs “close” to the key
  - distance (x, y) = x XOR y
- XOR is a good metric for a number of reasons
  - d(x,x) = 0, d(x,y) > 0 if x!=y and d(x,y) + d(y,z) ≥ d(x,z) **and** symetric d(x,y) = d(y,x)
  - XOR is unidirectional – i.e. for any given point x and distance D > 0, there’s only one point y such that d(x,y) = D (path convergence)
- Peer’s routing table has list of k-buckets; $bucket_i$ with IDs of peers sharing an i-bit long prefix
- List is kept sorted by time last seen

Kademlia

- Joining is easy
  - A peer n contact an already participant node m
  - Inserts m into the appropriate k-bucket
  - Perform a peer lookup for its own peer ID, thus populating its own k-buckets and inserting itself in others' k-buckets
- Iterative lookup, reply with k closest nodes to key, from the appropriate bucket: lookup upper-bound is *O(log(n))*
- When a node receives any msg, it updates its k-bucket
  - If node's there, move it to the tail (most recently seen)
  - If not there and fewer than k entries, add it to tail
  - If not there but bucket's full, ping the head node (least recently seen) and if alive, move to head, otherwise replace
    - Never delete an old node! Lifespan distribution of nodes says is good for you

Routing in Kademlia



Bucket ID	Common prefix length	Bucket entries
B_0	**0**	**1001..., 1100…, 1101…**
B_1	**1**	**0110…, 0100…**
B_2	**2**	**000…**
B_3	**3**	**0010…**

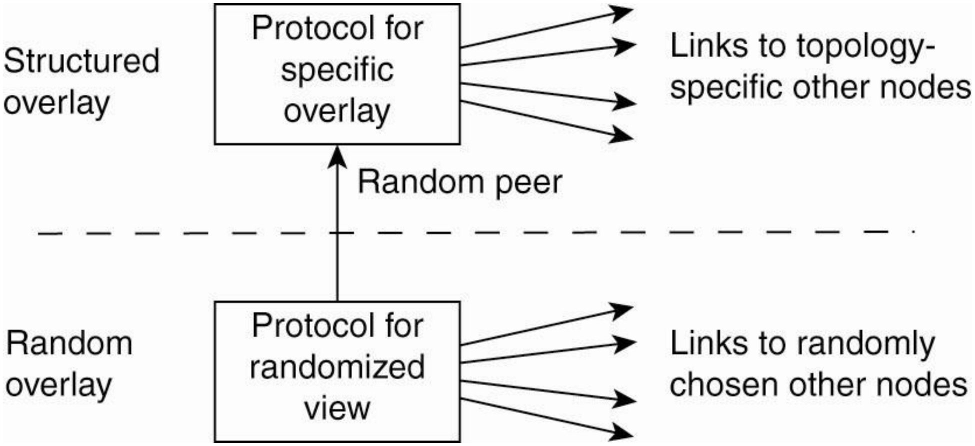
Routing table for 0011…

Exploiting network proximity

- The logical organization of nodes in the overlay may lead to erratic message transfers in the underlying
  - Topology-aware node assignment – When assigning an ID to a node, make sure that nodes close in the ID space are also close in the network. Can be very difficult.
  - Proximity routing – Maintain more than one possible successor, and forward to the closest.
    - Example: in Chord $FT_p[i]$ points to first node in $INT = [p + 2^{i-1}, p + 2^{i-1}]$. Node *p* can also store pointers to other nodes in *INT*.
  - Proximity neighbor selection – When there is a choice of selecting who your neighbor will be (not in Chord), pick the closest one.

Combining structured and unstructured

- Distinguish two layers: (1) maintain random partial views in lowest layer; (2) be selective on who you keep in higher-layer partial view
- Lower layer feeds upper layer with random nodes; upper layer is selective when it comes to keeping references
  - Instead of simple random, ranking peers based on some simple function (latency, semantic) may help



Question 2

- *Discuss the tradeoffs between efficient use of the underlying network by P2P overlay networks and measurement overhead*