McGill

School of Computer Science

Data Management in P2P systems

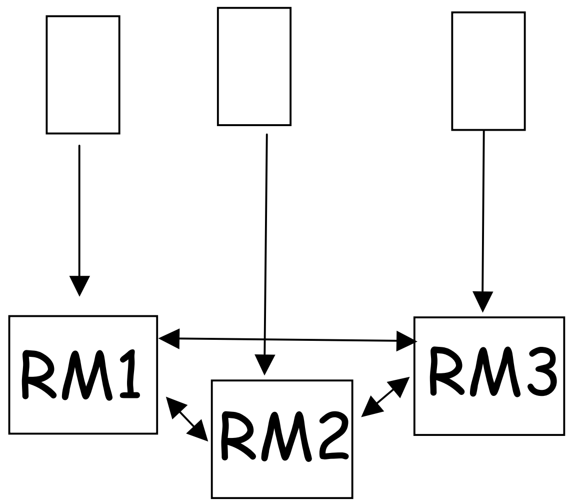
Bettina Kemme
McGill University
Montreal, Canada

Outline

- Overview
- Search Algorithms
- Architectures

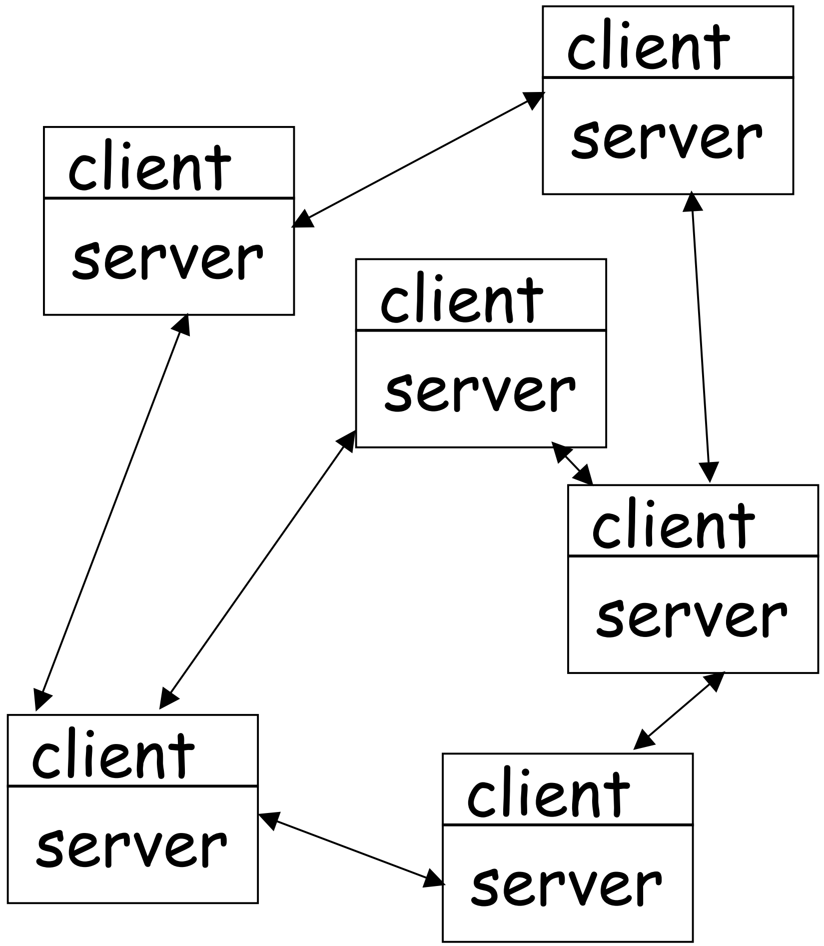
Architecture Basics

Client-
Distributed Server



Coordination among
servers

P2P



Definition of Peer-2-Peer

- Complex Definition
  - Each node/peer is client and server at the same time
  - Each peer provides content and/or resources
  - Direct exchange between peers
  - Autonomy of peers (can join and leave at their will)

Benefits and Challenges

- Data Sharing
  - How it works
    - Nodes store data; data distributed among many nodes
    - Nodes who want data download it from nodes who have data
  - Benefit
    - massive resource
    - Efficiency: distribution of expensive download operations
  - Challenges
    - Search
    - How to distribute the data
    - Load-balancing
    - Data integration
    - Availability
    - Data consistency
  - Many existing applications
    - Music, movie and other file sharing

Benefits and Challenges

- Resource Sharing
  - How it works
    - Peers are willing to perform execution on behalf of others in return to be able to use resources on other peers
  - Benefit
    - Massive resource
    - Execution possibly close to where it is needed?
  - Challenges
    - Load-balancing
    - Security
    - Finding appropriate nodes
  - Applications
    - Seti
    - For gaming?

Architecture

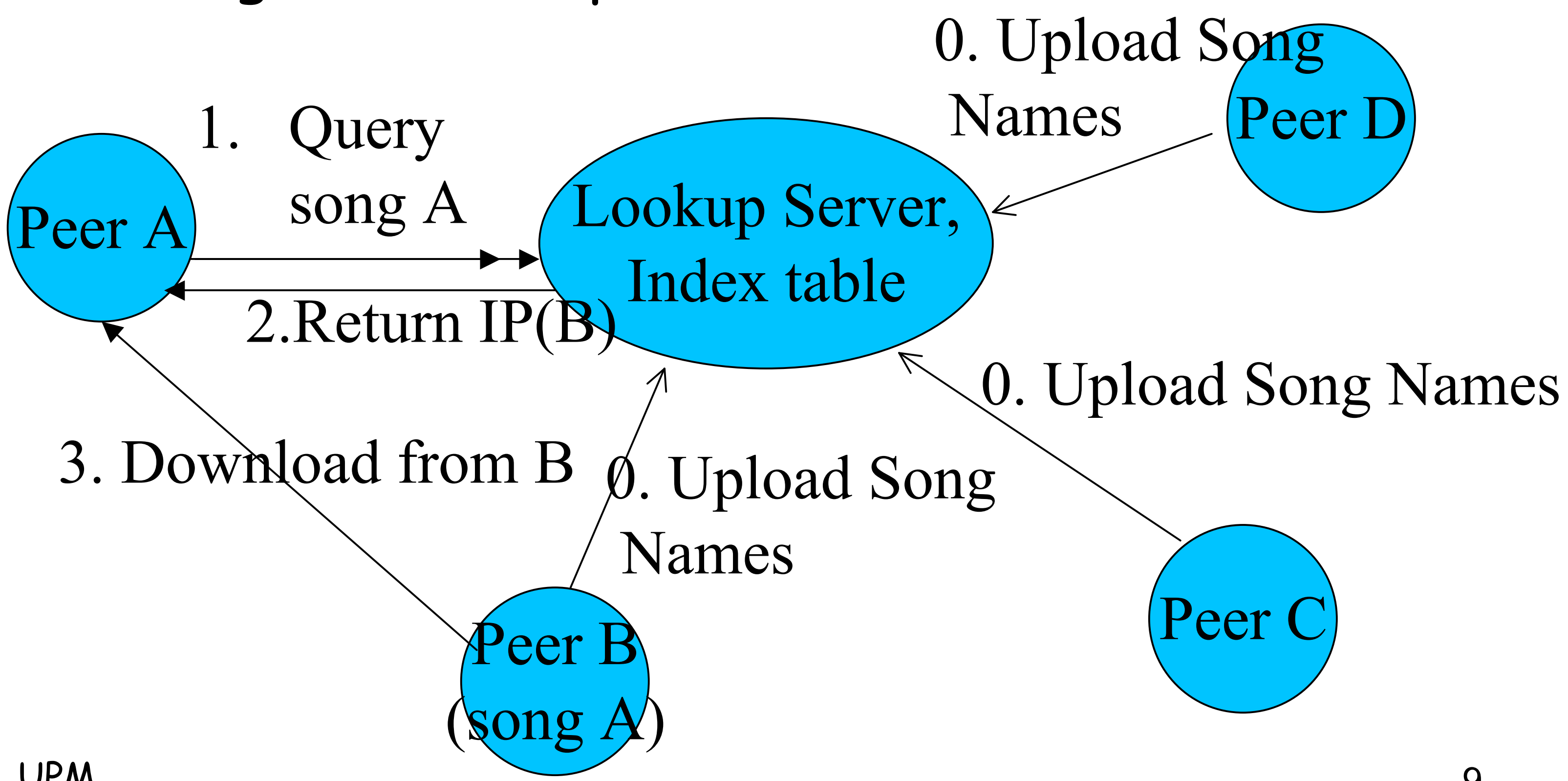
- Peer to Peer
  - Hybrid (Napster)
    - Super-peer (KaZaA)
  - Pure
    - Unstructured (Gnutella)
    - Structured (Chord)

Architecture (II)

- Hybrid
  - Some tasks via centralized component
  - Other tasks decentralized between 2 peers
- Pure
  - No special peers
  - Each peer only knows neighbors
    - Determine when joining the system (roaming for closest neighbors)
    - Set of neighbors might change during lifetime

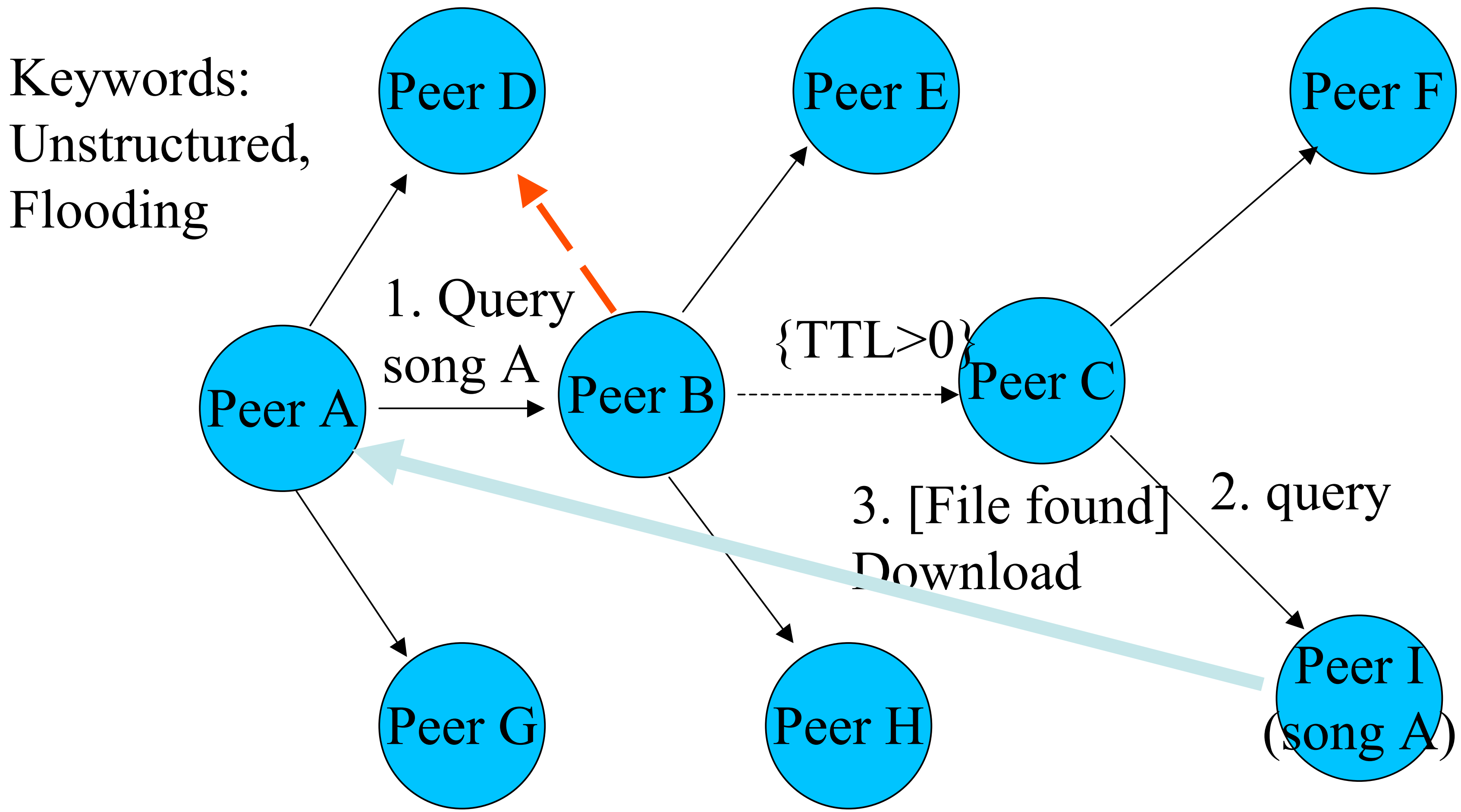
Search in Hybrid P2P

Napster [Shanning, U. Northeast, Dec 98]
- Hybrid:
  - Lookup centralized
  - Peers provide meta-information to Lookup server
  - Data exchange between peers



Search in Unstructured P2P

Gnutella [Justin Frankel, Mar 00]



Simple Flooding

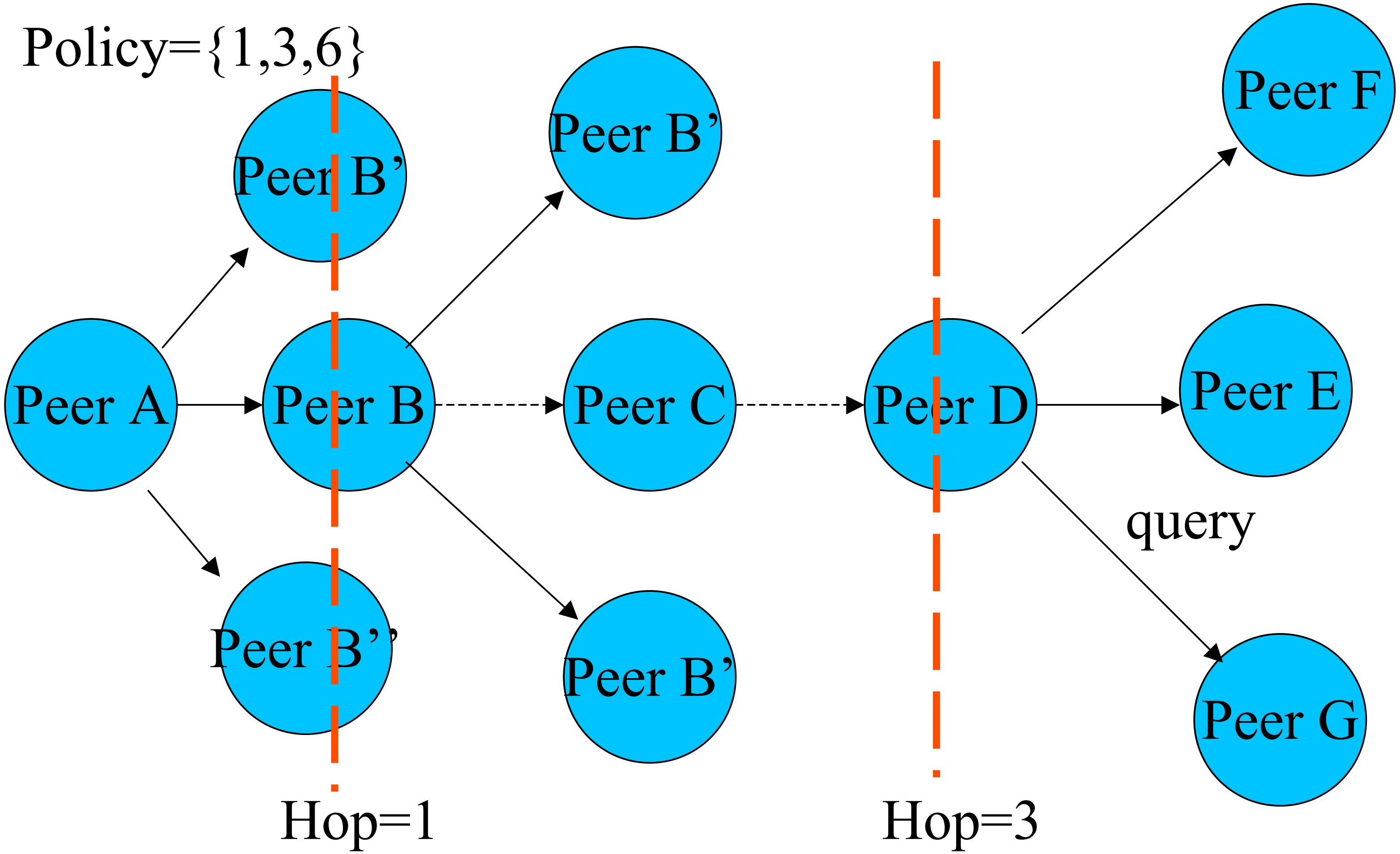
- Flooding
  - A peer sends the query to all of its neighbors
  - If the neighbor has the result, it will notify the query initiator; the query initiator can get the result directly from the neighbor.
  - Else the neighbor will decrease TTL (Time To Live) and forward the query to its neighbors as long as TTL is larger than 0.
- Note:
  - Query initiator can get redundant results or no result even if data exists in network.
  - A peer may be visited more than once (As peer D in previous slide)
- Contact between query initiator and provider of data
  - Initiator id is piggybacked on query or
  - Information that provider has data takes reverse search path

Drawbacks of Flooding

- Large amount of messages
- Duplicate queries
- Hard to choose TTL
  - TTL too high → high load in Network
  - TTL too low → No result found

Search in Unstructured P2P (II)

Iterative Deepening[Yang,ICDE02]

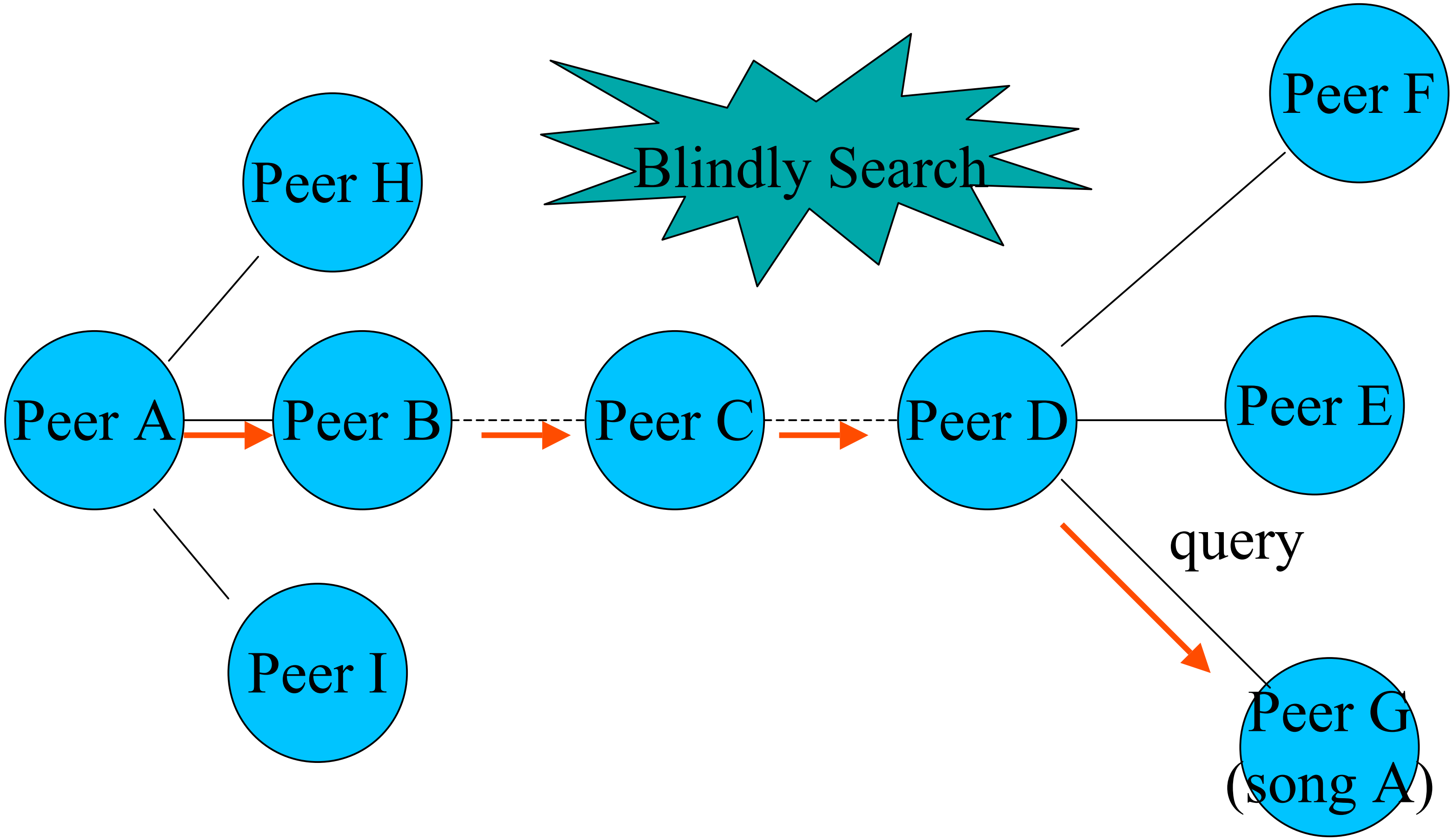


Iterative Deepening

- Idea
  - Search is started using flooding with small TTL.
  - If no result is found, new search is started with larger TTL.
  - Iterative until result is found or limit of TTL is reached
- Details
  - There is a policy array indicating for each search iteration the TTL
  - In the previous example, Peer A will first visit all of its neighbors which are 1 hop away (first number in policy array)
  - If the result is found, query stops.
  - Else, query will be forwarded via flooding to all peers that are up to 3 hops away (second number in policy array).
  - This process continue recursively until result is found or all elements in policy array have been exhausted.

Search in Unstructured P2P (III)

Random Walks [Lv, ICS02]

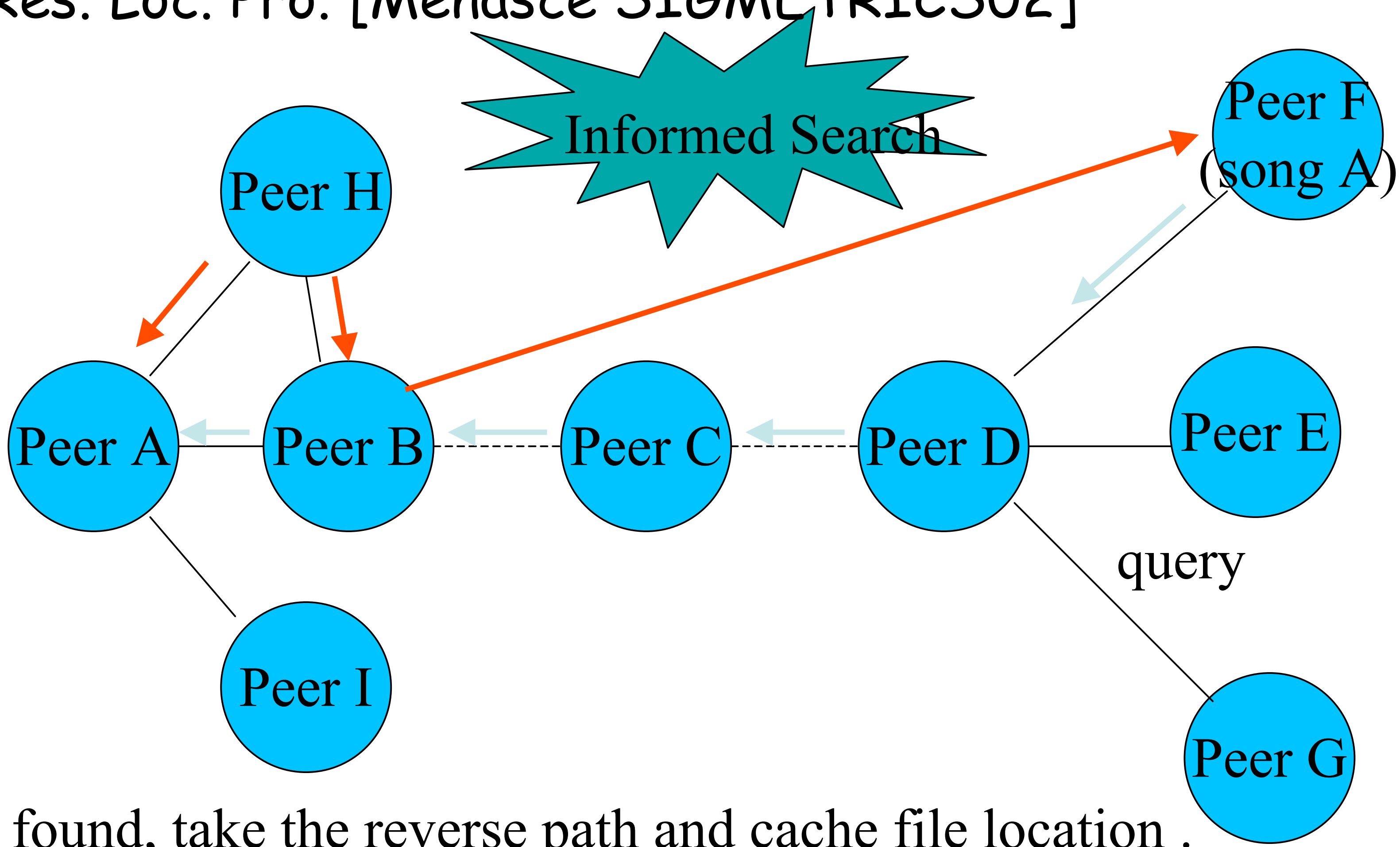


Blind Search

- Query initiator selects only one neighbor to send the query.
- If the neighbor doesn't have the result, it will select one of its neighbors to send the query.
- The process will repeat until result is found or TTL is met.

Search in Unstructured P2P (IV)

Intelligent-BFS [Kalogeraki, CIKM02]
Dist. Res. Loc. Pro. [Menasce SIGMETRICS02]



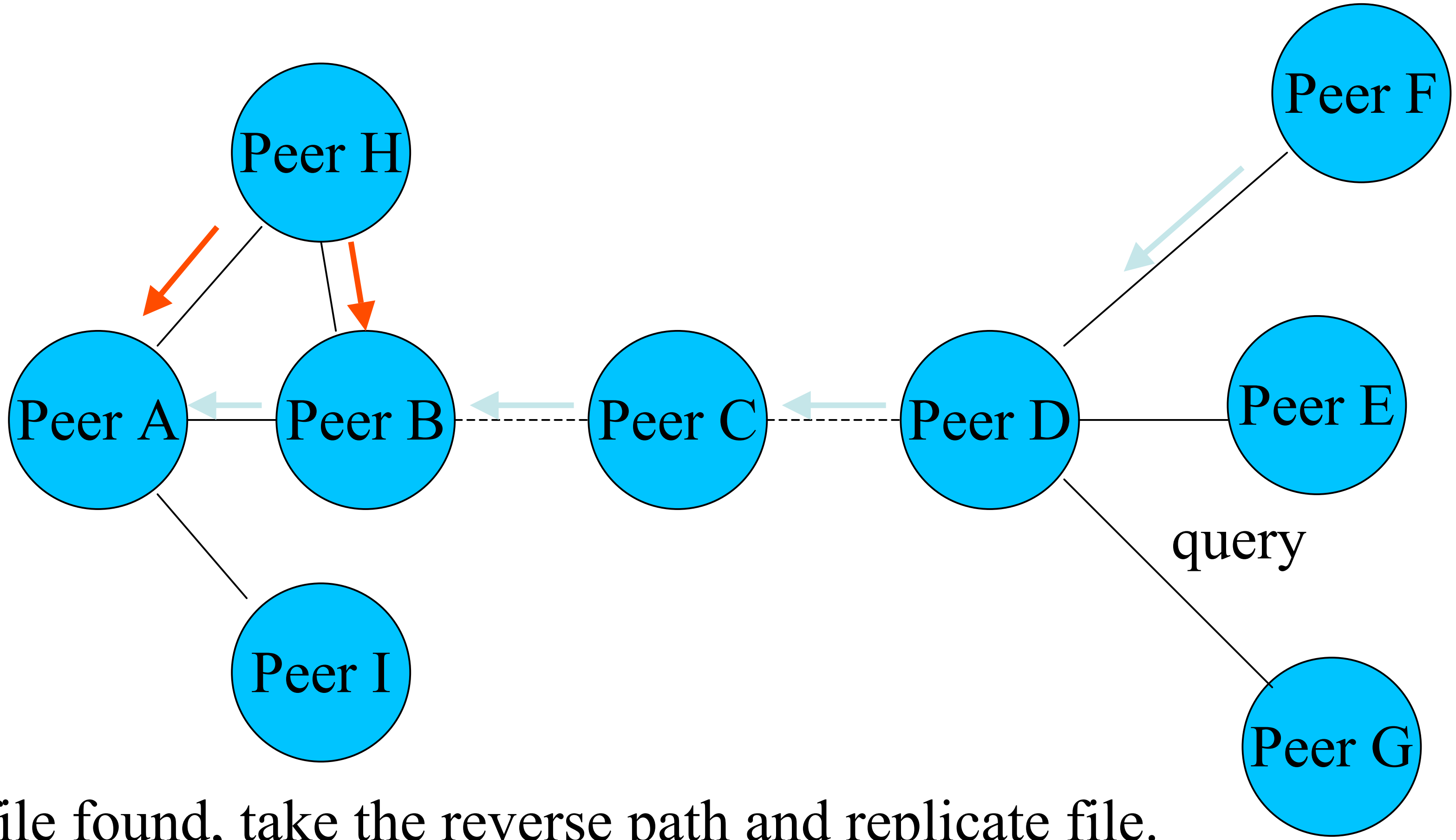
If file found, take the reverse path and cache file location .

Informed Search

- Each peer has a lookup index storing file locations which have been searched previously.
  - Lookup index neither complete nor accurate
- If a peer finds the location for a file in the index, it will directly contact the file holder and get the file.
- Otherwise, it uses flooding for search
- Once the file is found, the reverse path of query path is used to inform the query initiator about the location
  - Hence, peers on the query path can update their indices speeding up next queries
- Various other types of lookup indices for more directed search

Search in Unstructured P2P (V)

Optimization: Replication [Lv,ICS02]



If file found, take the reverse path and replicate file.

Add Replication

- Replication Search
  - The difference between this and Informed Search is that instead of caching the file location the file will be cached along the reversed query path.

Structured P2P

- Chord [Stoica, ACM Tran NW 03]
- Pastry [Rowstron, Middleware 2001]
- Tapestry [Zhao, Berkeley 01]
- CAN [Ratnasamy SIGCOMM, 01]

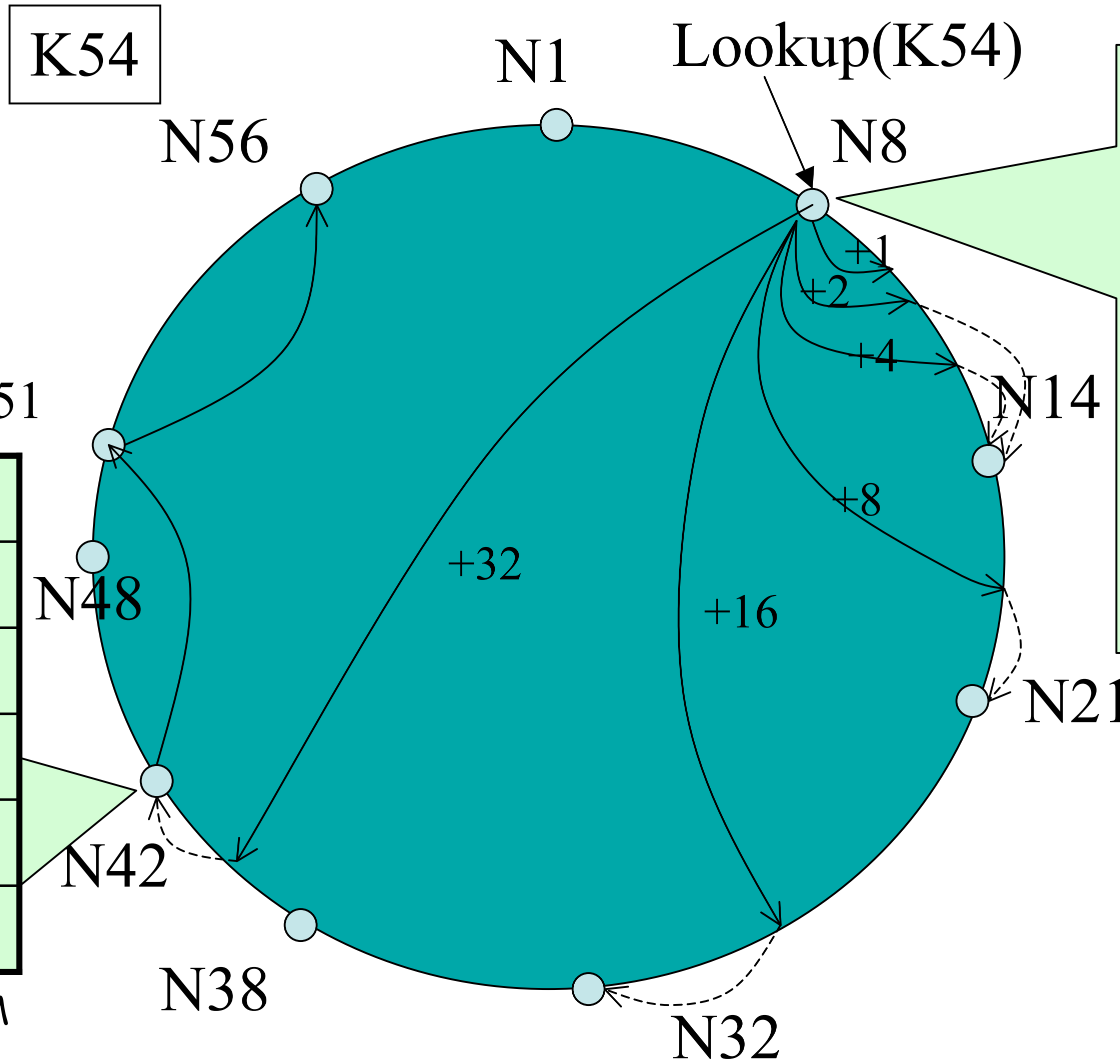
Distributed Hash Table (DHT)

Structured P2P (II)

- Two Versions
  - Data items are distributed over peers according to an algorithm
    - Peers don't choose their data items by "free will"
    - Needs additional replication mechanism for availability
  - Lookup information (location information) is distributed over peers according to an algorithm
- Assignment mechanism
  - Each node has a unique identifier (Hash of IP)
  - Each data item (e.g. file) has a key (Hash of title, author etc)
  - Each node is responsible for storing files or location of files that have a key that is similar to the node identifier
  - Given a key, a node efficiently routes the query to the node with a ID closet to the key.

Search in Structured P2P

Chord [Stoica, ACM TRAN NW 03] Id has max of 6 bits: 2^6 = 64



N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

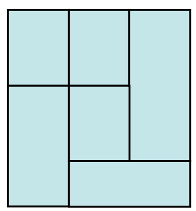
N42+1	N48
N42+2	N48
N42+4	N48
N42+8	N51
N42+16	N1
N42+32	N14

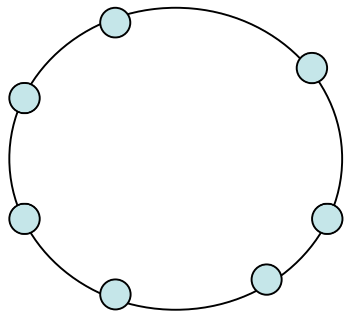
Key Search in Structured P2P

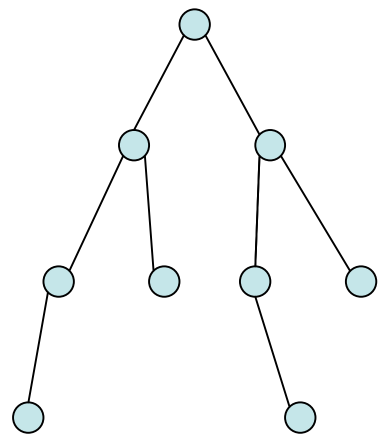
- Consistent hashing
  - Assign keys to chord nodes
  - Requires load balancing such that each node receives roughly the same number of keys
  - Requires little movement of keys when nodes join and leave the system
- Key Queries
  - Structure requires query to contain the file identifier

Topologies

- Network spaces can have many different shapes:
  - Multi-dimensional spaces (CAN, ProBe)
  - Ring-like spaces (Chord, Mercury, Oscar)
  - Tree-based spaces (Baton, P-Grid)

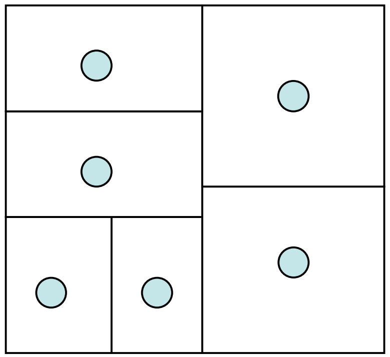






Range Queries

- Range Queries
  - documents are described by several attributes
  - Search by giving values for attributes (point query) or ranges for values of attributes
    - where a= 2 and b = 10
    - where 10 <= a <= 20 and 1000 < b < 2000

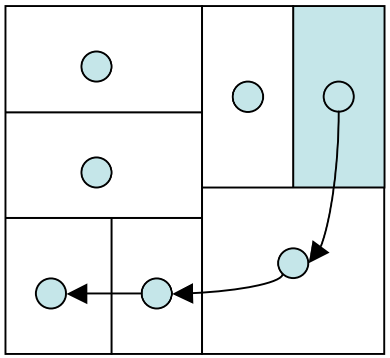


- Two-dimensional
  - each node is assigned documents for which the attribute values lie in a certain range

Routing Range and Point Queries

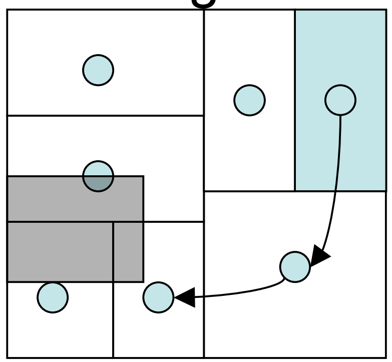
- keep track of the neighbors responsible for neighbor ranges
- forward queries to neighbors whose ranges are closer to the requested

Point Query:

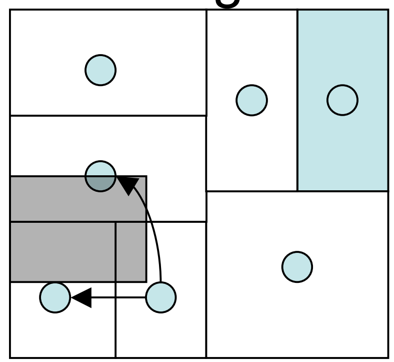


Range Query:

Routing



Flooding

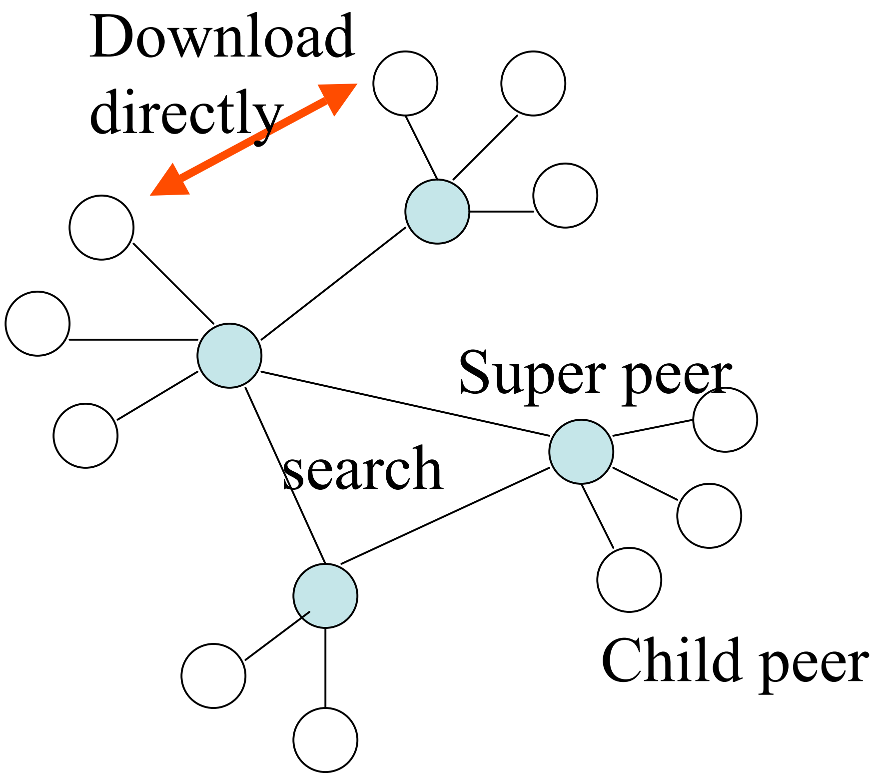


Challenges

- Joins
  - distribute space
  - more nodes -- longer routing
- leaves
  - make sure not to loose important information (data, index information)
  - replication necessary

Super-Peer

[Yang and Garcia-Molina, ICDE 03]

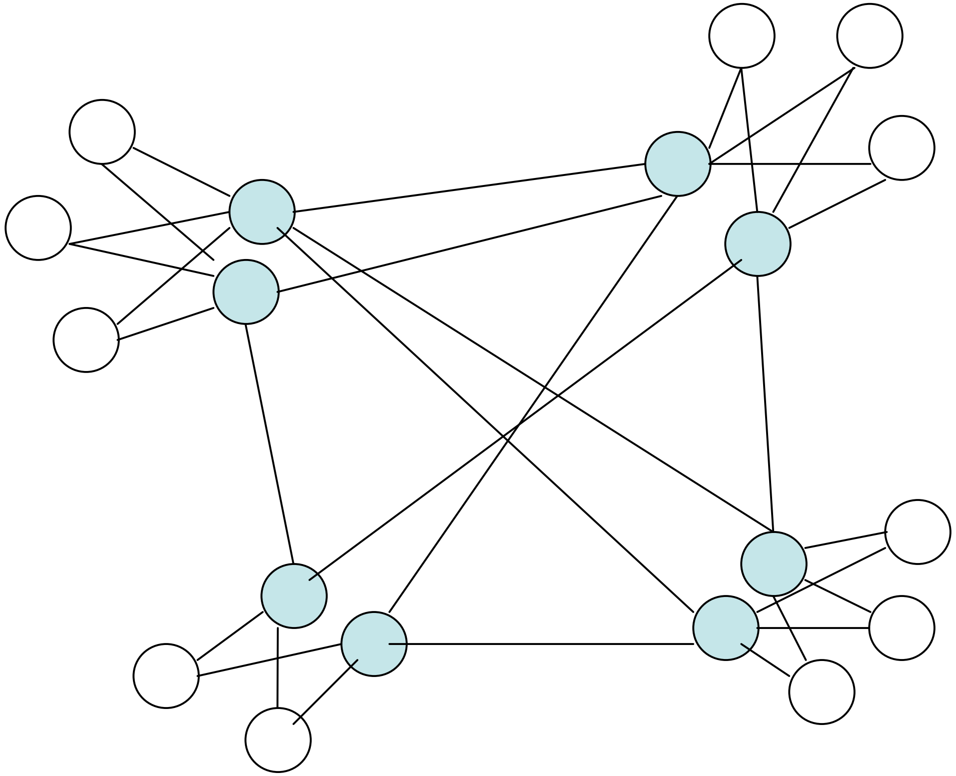


- A super peer keeps an index over its leaf nodes
- Super peer performs queries on behalf of leaves
    - faster routing since less nodes
- Direct exchange between any peer

Challenges of Super-Peer

- What is good ratio of leaves to super peer?
- Search more efficient?
- How should super-peers connect to each other? Unstructured v.s structured?
- System more reliable? K-redundant

Super Peer, K-redundant



K=2

Search in P2P Comparison

Category	Hybrid	Unstructured		Structured
		Blindly	Informed	
Example	Napster	Gnutella	Int. BFS	Chord
Structure	One Central.	Random	Random	Fixed
Search Method	Centralized Index Server	Flood	Opt Flood	DHT
Info. about Data Loc.	deterministic	Nothing	Partial	High prob.
Data Loc.	Anywhere	Anywhere	Anywhere	Anywhere
Self Org.	Bottleneck	Good	Good	Bad

Replication in P2P

- Improves Search
  - Find match faster
- Improves Availability
  - Loss of node does not lead to unavailability of data
- Improves Load Balancing
  - If highly demanded documents are replicated, download processing can be distributed among replicas

How to replicate

- No proactive replication
  - Hosts store and serve only what they requested: called *owner replication*
    - Naturally proportional replication
      - A document is replicated according to its popularity
- Proactive replication of meta-information (Keys and pointers) for search efficiency (FastTrack, DHT)
- Proactive replication of copies of data-item (for search efficiency, load-balancing and availability)

Challenges in Replication

- How many replicas?
- When to create replicas?
- Where to place replicas?
- How to route queries to different replicas?

Replication in P2P

- Adaptive Replication in Peer-to-Peer Systems. Gopalakrishnan, Silaghi, Bhattacharjee, Keleher, ICDCS 2004
- Path-replication (app-cache)
  - works well for skewed query distribution
  - but cannot handle load properly (nodes around hot-spot data more heavily loaded)
- adaptive replication (LAR)
  - server load should be balanced
  - choose specific replication points
    - similar to owner replication!
  - enhance routing process with hints to find new replicas
  - works with structured and unstructured P2P

LAR step 1: Measure local load

- high-load and low-load thresholds
- keep track of the load due to each object
- Specify processing capacity and queue length for each server

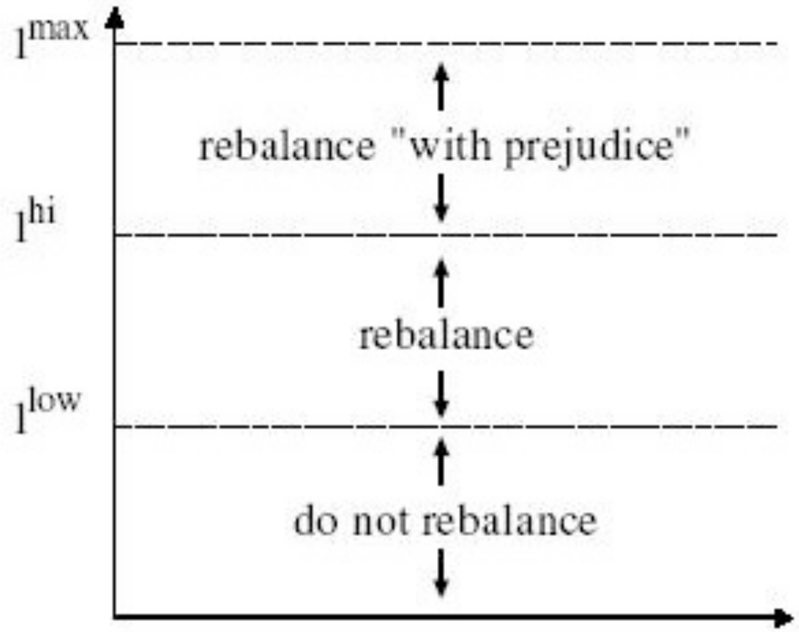


Figure 3. The server capacity is l^{max}. Load is sometimes re-balanced if greater than l^{low}, and always if greater than l^{hi}.

LAR step 2: Make replication decision

- A message: Sj -> Si
- Cur(si) > high(si), Si is overloaded.
  - If Cur(si) - Cur(sj) > K
    - Si then ask Sj to create replicas of the n most highly loaded items of Si to make the load balanced between Si and Sj.
- If Low(si) < Cur(si) < Max(si)
  - If Cur(si) - Cur(sj) > low(si)
    - then distribute

LAR step 3: Propagate routing hints.

- Each server maintains cache of routing hints
- A hint consists of
  - a data item label,
  - its home node,
  - and a max. set of known replica locations
  - Hint entries are replaced using LRU
- Hints are cached on the path from source to destination
- Cache incomplete and might be stale
- Hint dissemination: piggybacked on other messages

Implementation over CHORD

- replicated data item: Finger-list

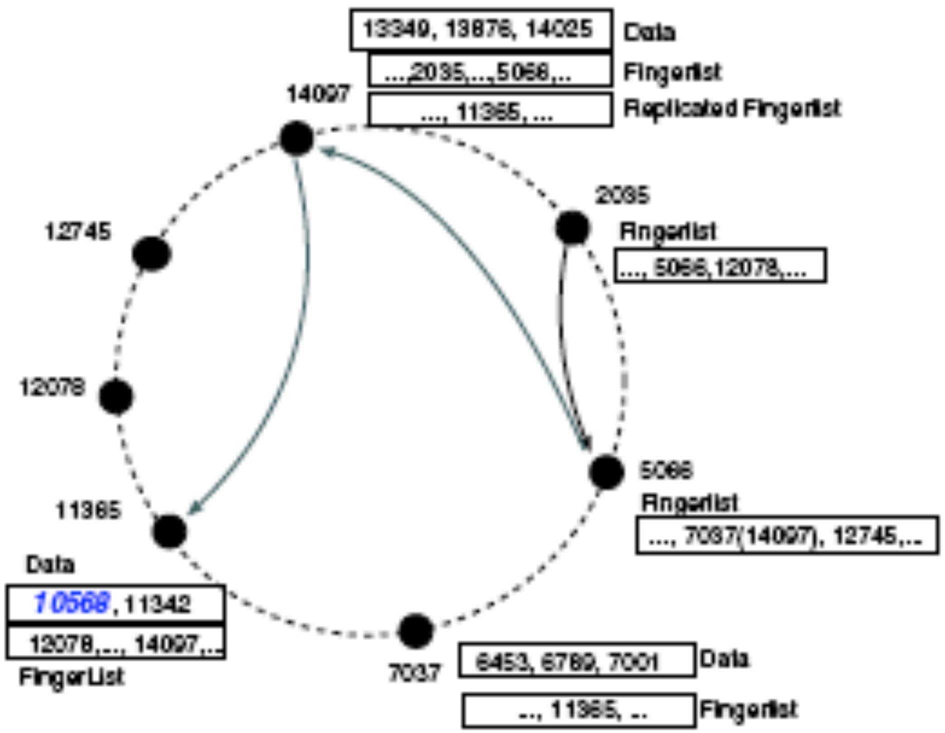


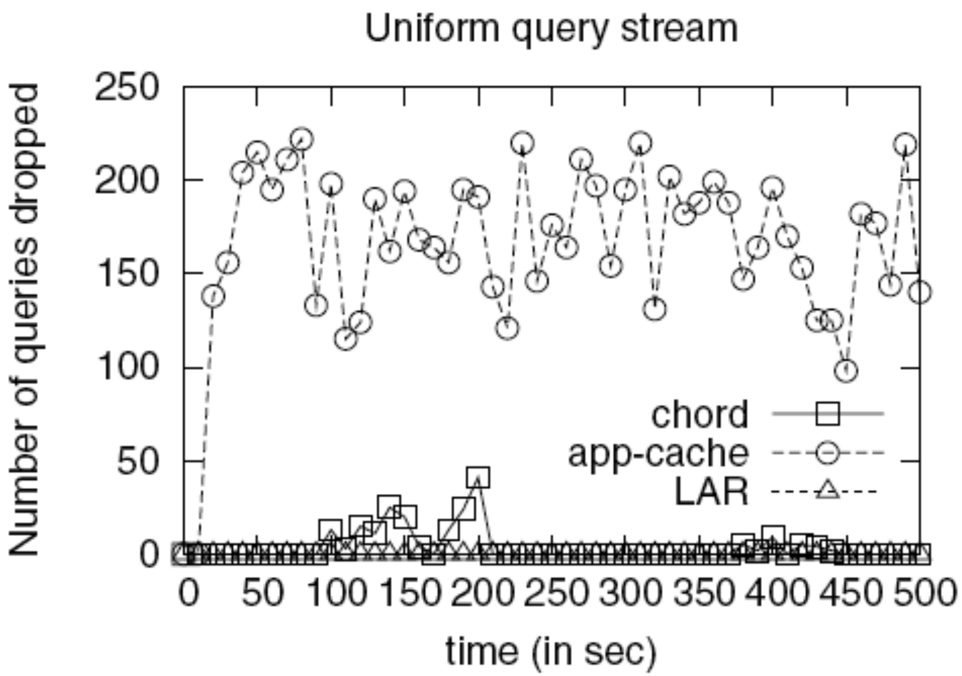
Figure 5. LAR routing and replication in Chord

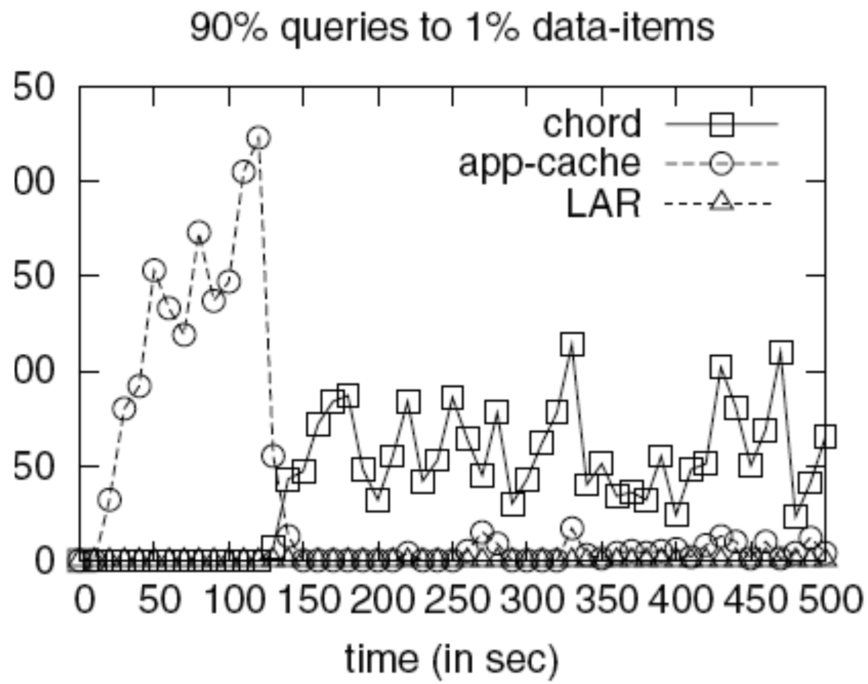
Evaluation

- Simulation based
- 1000 servers
- 35.000 items
- server capacity: 10 per second
- servers drop messages if too many requests in queue
- load is triggered that "average" load on server would be 25%

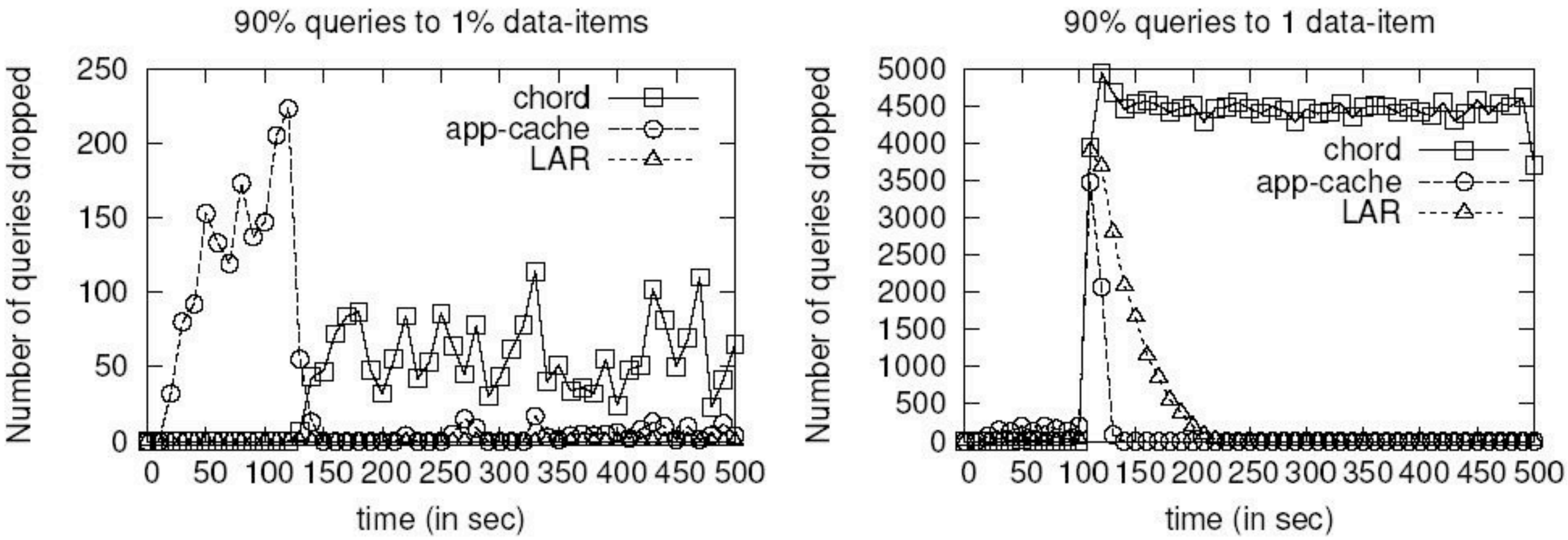
Simulation Results 1: Chord, App-Cache, LAR

- skewed query distribution starts at 100 ms





Simulation Results 2: Chord, App-Cache, LAR



Simulation Results 2: Chord, App-Cache, LAR

Input dist.	Scheme	# q served (250K max)	# replicas creat.	# replicas evict.	# hints creat.	# hints evict.
Unif.	Chord	249.9K	-	-	-	-
	AC	242.4K	1.13M	1.09M	-	-
	LAR	249.9K	5K	0	10.8K	5.9K
90% → 10%	Chord	249.9K	-	-	-	-
	AC	245.7K	994K	962K	-	-
	LAR	249.9K	6.6K	0	12.5K	7.5K
90% → 1%	Chord	248.2K	-	-	-	-
	AC	248.1K	691K	660K	-	-
	LAR	249.9K	10.3K	0	17.3K	12.3K
90% → 1	Chord	72.1K	-	-	-	-
	AC	244.1K	328K	296K	-	-
	LAR	233.4K	2.6K	0	7.2K	2.4K

Table 1. Protocol Overhead of Chord, LAR, and app-cache(AC).