Peer to Peer Systems

Topics

- **Introduction**
- **Client- server vs peer to peer**
- **Peer-to-peer networks**
  - **Routing Overlays**
  - **Structured vs unstructured**
- **Pastry**
- **BitTorrent**

Peer-to-peer - definitions

- "All nodes are equals"
  - But some nodes are *more* equal (*superpeers*)
- "P2P is a class of applications that takes advantage of resources – storage, cpu cycles, content, human presence – available at the edges of the Internet"
- P2P-test:
  - Does the system treat variable connectivity and temporary network addresses as the norm?
  - Does the system give the nodes at the edge of the network significant autonomy?
  - Does each peer has equal responsibility?

Client-Server vs. Peer-to-peer

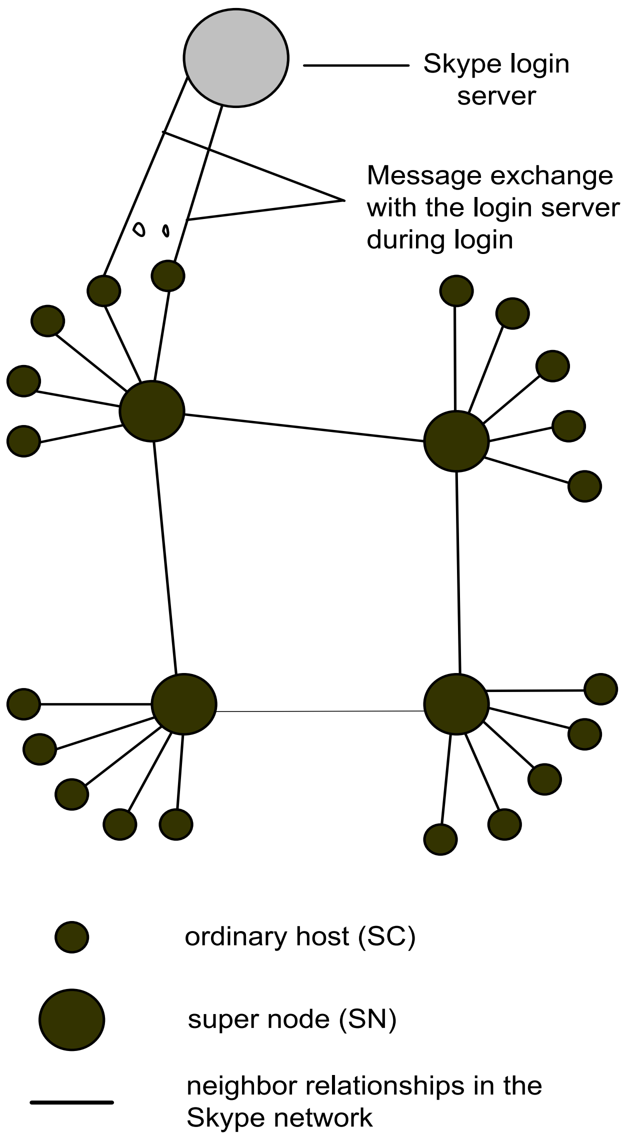
- Client-server:
  - Simple
  - Easy set up and administration
  - Security model

  - Scalability
  - Availability
  - Single point of failure

- Peer-to-peer:
  - Highly scalable
  - Failure tolerant
  - Self-organizing
  - Takes advantage of unused resources in powerful clients
  - Availability?

  - Security and Trust
  - Difficult to manage
  - Asymmetric bandwidth

- **Skype**



Skype login server
Message exchange with the login server during login
ordinary host (SC)
super node (SN)
neighbor relationships in the Skype network

Baset, S. A. ,Schulzrinne, H. G. *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol,* pp 1-11, INFOCOM 2006

P2P use case – file sharing

- You probably know a bit about these already…
- Examples include
  - Napster - 1st generation, central index, distributed data
  - Gnutella - 2nd generation, initially fully distributed index
- Good incentive to join – get access to large amounts of data
- Simplifying factors:
  - Immutability
    - File content seldom or never change
  - Non-strict availability requirements
    - Acceptable that files sometimes are unavailable

P2P – Properties

- Ensures that users contribute resources (disk, CPU cycles etc)
- The responsibilities (albeit not the performance) for each node is equal
- No Single point of failure
- Are there any *pure* P2P architectures?
  - Many have (semi-)centralized indices
  - Most use DNS

P2P - security issues

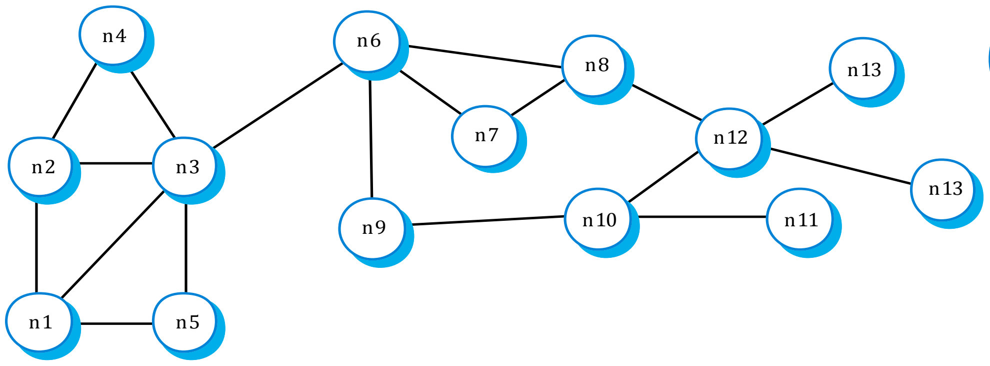
- Poisoning attacks
  - E.g. providing files whose contents are different from the description
  - Madonna's record company once did this…
- Polluting attacks
  - E.g. inserting "bad" chunks/packets into an otherwise valid file on the network
  - Both Poisoning and Polluting is a *Byzantine generals problem*
- Defection attacks
  - Users or software that make use of the network without contributing resources to it
- Insertion of viruses to carried data
  - E.g. downloaded or carried files may be infected with viruses or other malware
  - Hard to know origin of data
- Malware in the peer-to-peer network software itself
  - E.g. distributed software may contain spyware

P2P - Security issues (cont.)

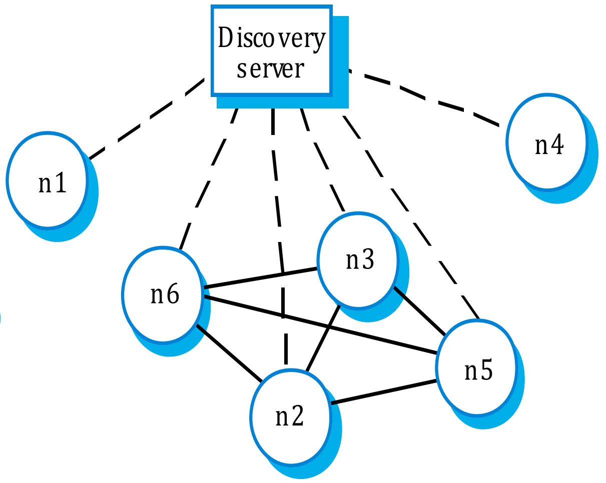
- Denial of service attacks
  - What differs from DoS against client-server systems?
- Filtering
  - Network operators may attempt to prevent peer-to-peer network data from being carried
  - Firewalls
- Identity attacks
  - E.g. tracking down the users of the network and harassing or legally attacking them
    - Pirate bay
- Spamming
  - E.g. sending unsolicited information across the network- not necessarily as a denial of service attack

P2P architectures

Decentralized architectures



Semi-centralized architectures



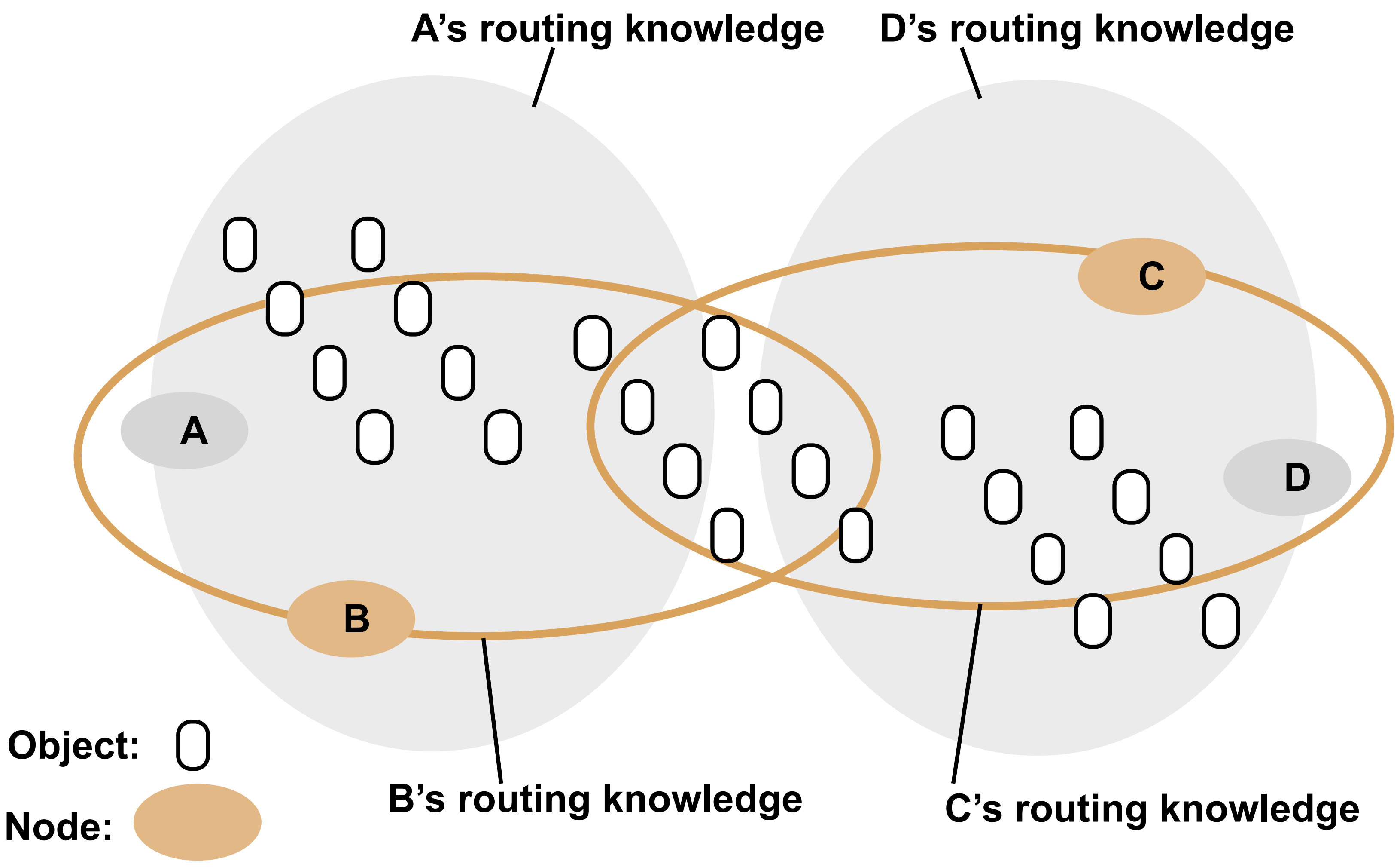
Peer-to-peer requirements

- Global scalability
- Load balancing
- Optimization for local interactions between neighbouring peers
- Accommodating to highly dynamic host availability
- Security of data
  - Integrity
  - Privacy
- Anonymity, deniability, resistance to censorship

Routing overlays

- A routing overlay is a distributed algorithm for a middleware layer responsible for routing requests from any client to a host that holds the object to which the request is addressed.
- Any node can access any object by routing each request through a sequence of nodes, exploiting knowledge at each of theme to locate the destination object.
- Nodes know how to route message to a subset of the network
- Overlapping subsets allow messages to be forwarded correctly
- Forms a *logical layer* on top of existing routing network (IP)

Distribution of information in a routing overlay



Unstructured routing overlays

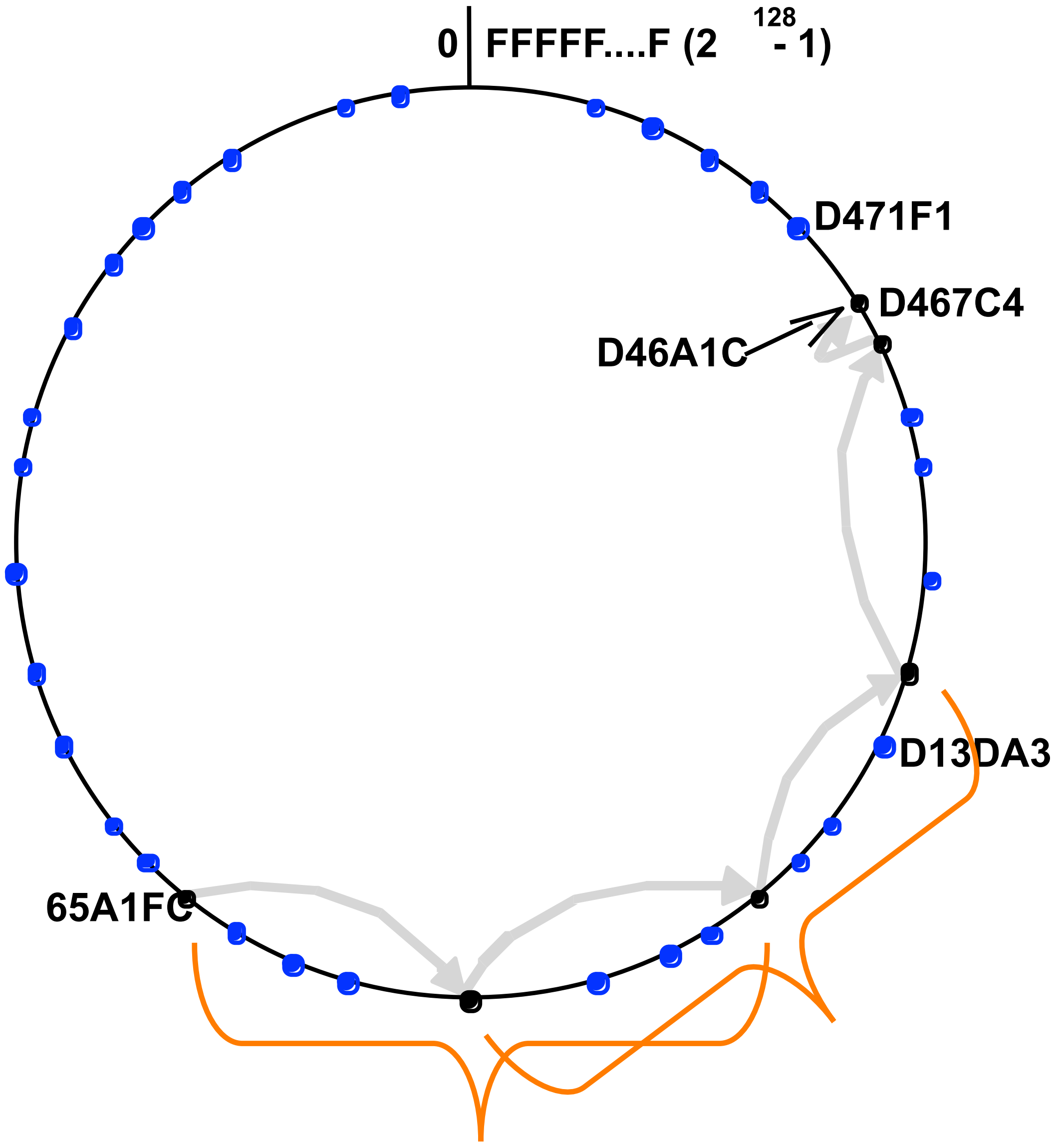
- Random establishment of links
- Easy to join new nodes to network
  - Copy links of existing nodes, set up own links after time
- No special action required when node leaves
- E.g. BitTorrent, Gnutella, Napster, Freenet,EDonkey
- Main disadvantage – searching
  - Queries must be flooded across network
  - Popular content probably replicated
  - Rare content hard to find
  - Huge amount of overhead traffic
  -

Structured routing overlays
- Distributed hash tables

- Non-random links
- Each node (and object) has a GUID
  - GUID calculated from hash values
- An object is stored at the node(s) with the GUID closest to that of the object
- Routing: Forward requests to the neighbour that is numerically closest to the target

- Efficient use of bandwidth
- Higher probability to find content/Read is fast
- More complex for nodes to join/leave
- Insert and delete are very expensive

Routing in a DHT (Pastry) - Basic idea



Each node knows 8 *logical* neighbours (4 on each side)

Example: Route a message from node 65A1FC to find object D46A1C

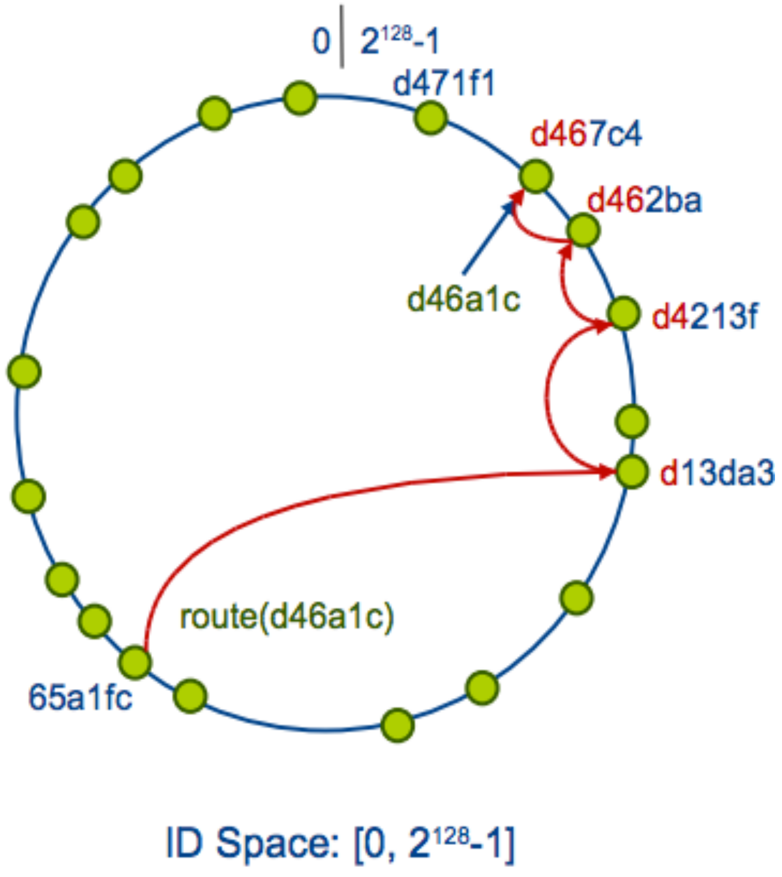
This requires O(N/8) hops!

Real routing in Pastry - Routing table

- m-number of rows
- b-number of entries per row
- E.g. b = 16, m = 32 and search for nodeID/ GUID = D46A1C

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1																6F
		D13DA3	D7													

Real routing in Pastry



Each node knows 8 *logical* neighbours (4 on each side)

Make use of routing table in addition to leaf sets

Example: Route a message from node 65A1FC to find object D46A1C

Longest common prefix (routing table)

This only requires O(log N) hops

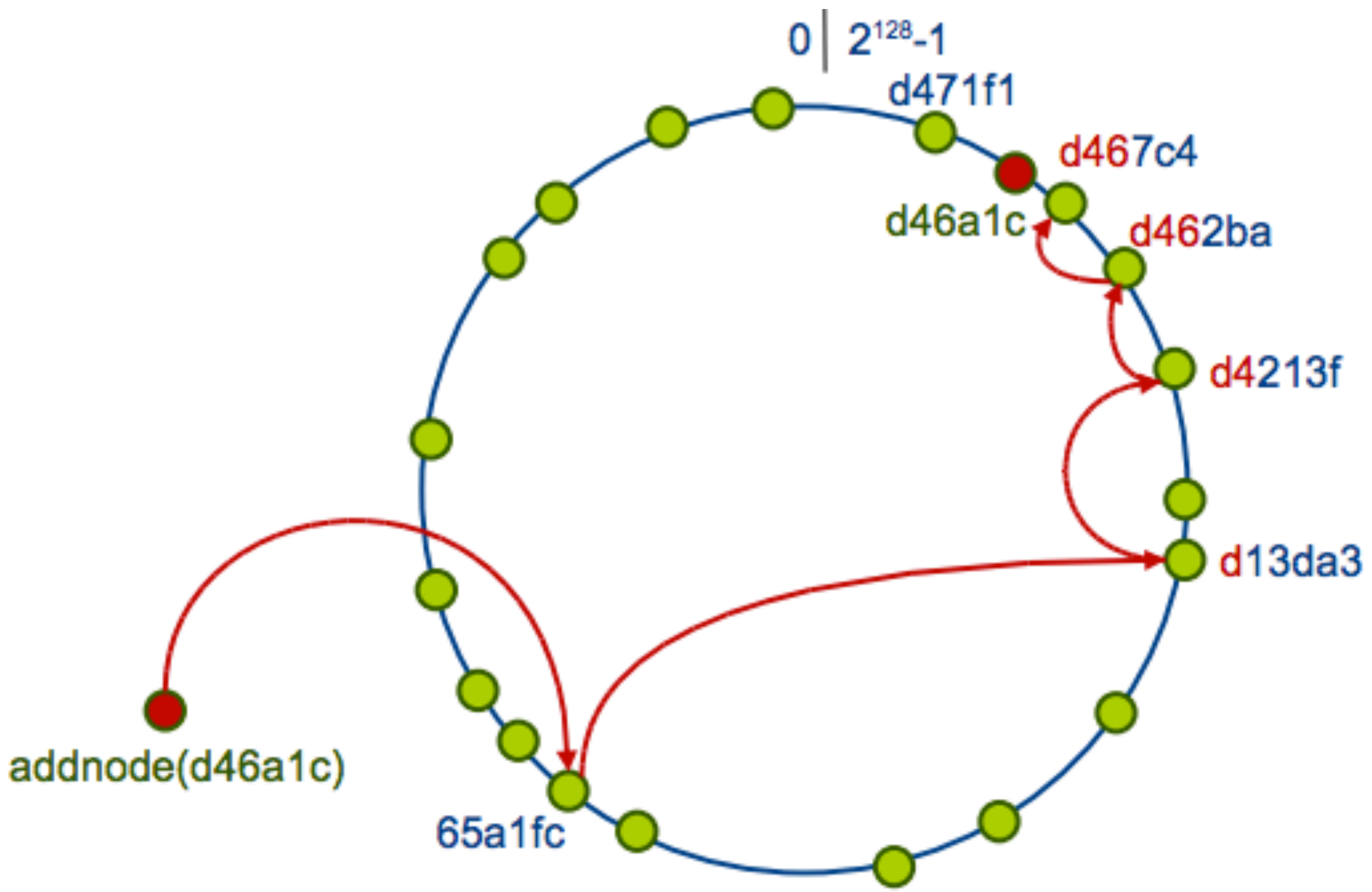
Basic DHT programming API

- put(GUID, data)
  - Store data (N replicas) at nodes with identities closest to GUID
- remove(GUID)
  - Delete all (up to N) occurrences of data identified by GUID
- value = get(GUID)
  - Retrieve data associated with GUID from some nodes holding it

Adding new hosts (Pastry DHT)

1. Compute GUID of new node:
   1. X = hash(public key of new node)
2. Contact “nearby” node A
   1. Is this pure P2P?
3. Send *join request* to A, specifying X as destination.
4. Pastry routes join message to Z (node with GUID closest to A)
5. Join message will pass nodes A, B, C, … Z
6. Nodes A, B, C, … Z sends relevant information (neighbor lists, routing tables) to X
7. X constructs its own neighbor list and routing table.
   1. Neighbour list in X almost identical to that in node Z
8. X contacts all nodes in its neighbour list so that they can add X to theirs

Adding new hosts(pastry)



Host departure (Pastry DHT)

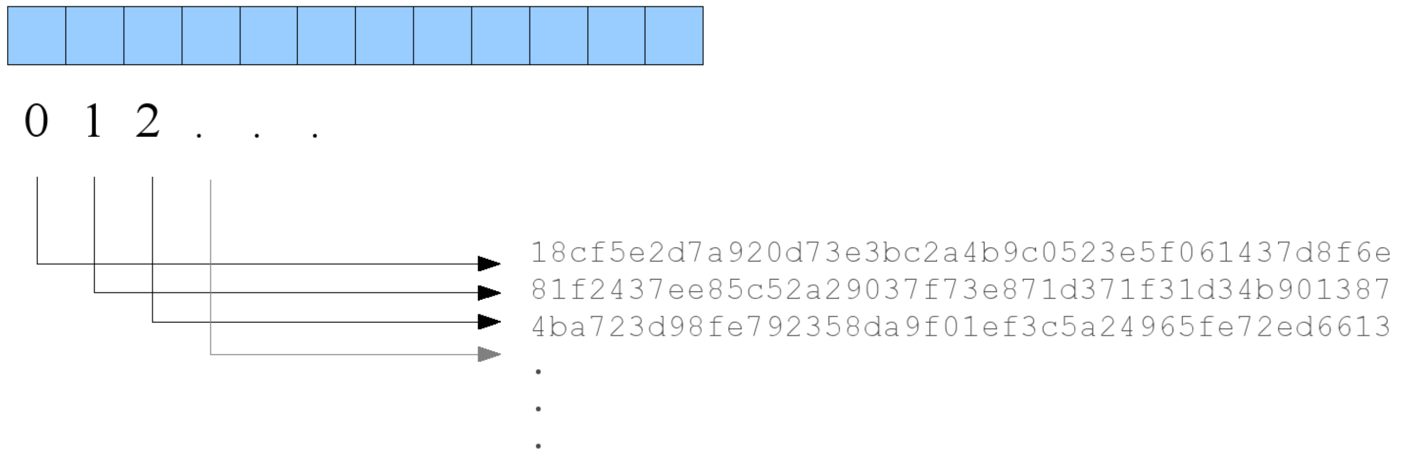
- Hosts may depart or fail at anytime
- Node Failure := when the nearest neighbours can not contact node
- Repair neighbour list of node close to failed node X:
  1. Get copy of neighbour list from node close to X
  2. Exchange X with appropriate node
  3. Inform other neighbouring nodes so they can repeat the procedure

BitTorrent – Introduction

- Efficient
- Scalable
- Suited for static data
- Terminology:
  - Peer
  - Torrent file
  - Tracker
  - Swarm
- Incentive to share:
  - Download speed related to upload speed
  - Peers are interested in exchanging data

BitTorrent – Introduction

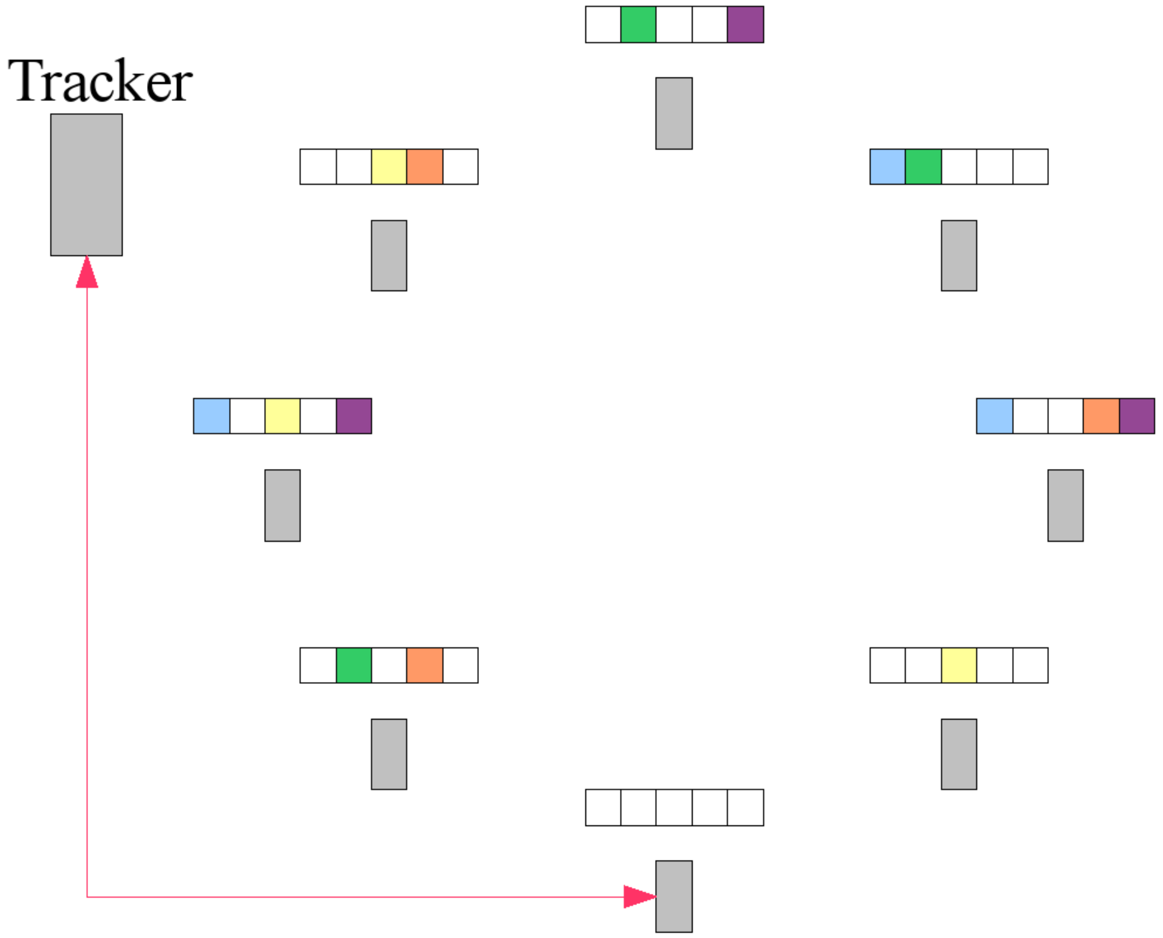
- Files are split up in pieces, and an SHA-1 hash is calculated for each piece.


0 1 2 . . .
18cf5e2d7a920d73e3bc2a4b9c0523e5f061437d8f6e
81f2437ee85c52a29037f73e871d371f31d34b901387
4ba723d98fe792358da9f01ef3c5a24965fe72ed6613
.
.
.

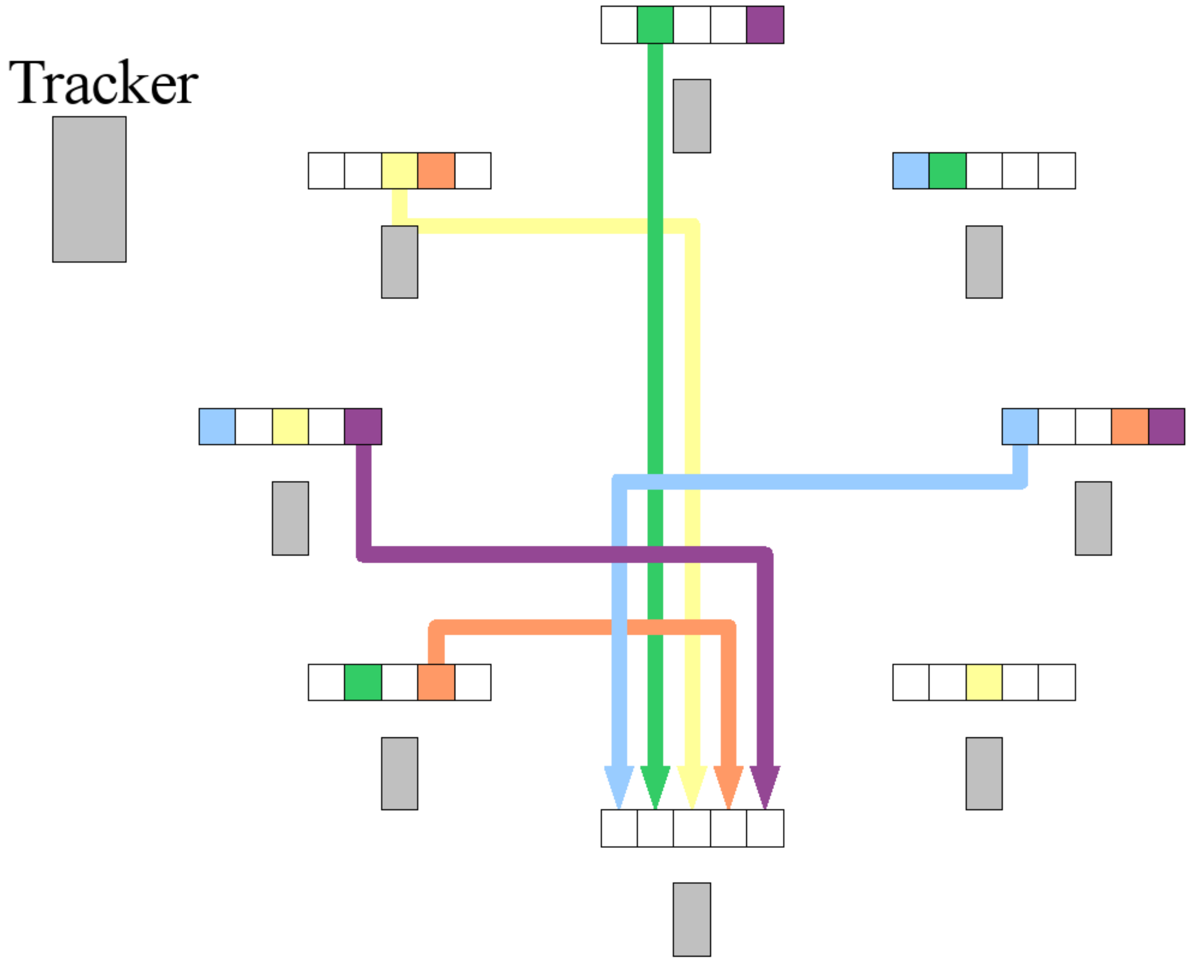
BitTorrent – Introduction

- The torrent file is distributed to all peers
  - Usually via HTTP
- The torrent file contains:
  - The SHA-1 hashes of all pieces
  - A mapping of the pieces to files
  - A tracker reference

BitTorrent – Introduction



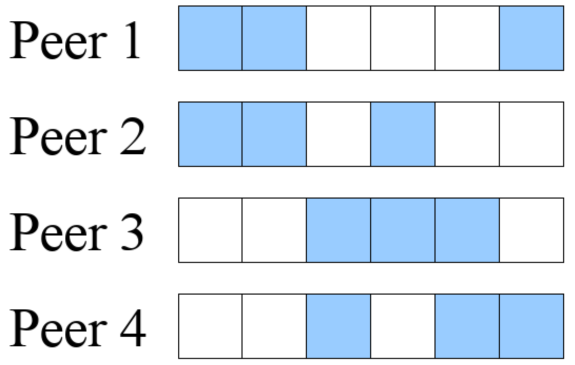
BitTorrent – Introduction

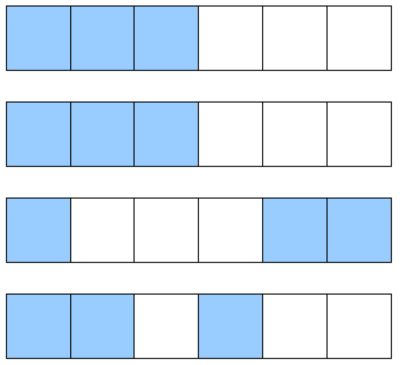


BitTorrent – Efficiency

- Fast downloads by enabling downloads from many different peers
- Minimize piece overlap -> peers can exchange pieces with many other peers

BitTorrent – Efficiency





- Small overlap
  - Many possible exchanges
  - Bandwidth well utilized

- Big overlap
  - Only a few possible exchanges
  - Bandwidth under utilized

BitTorrent – Efficiency

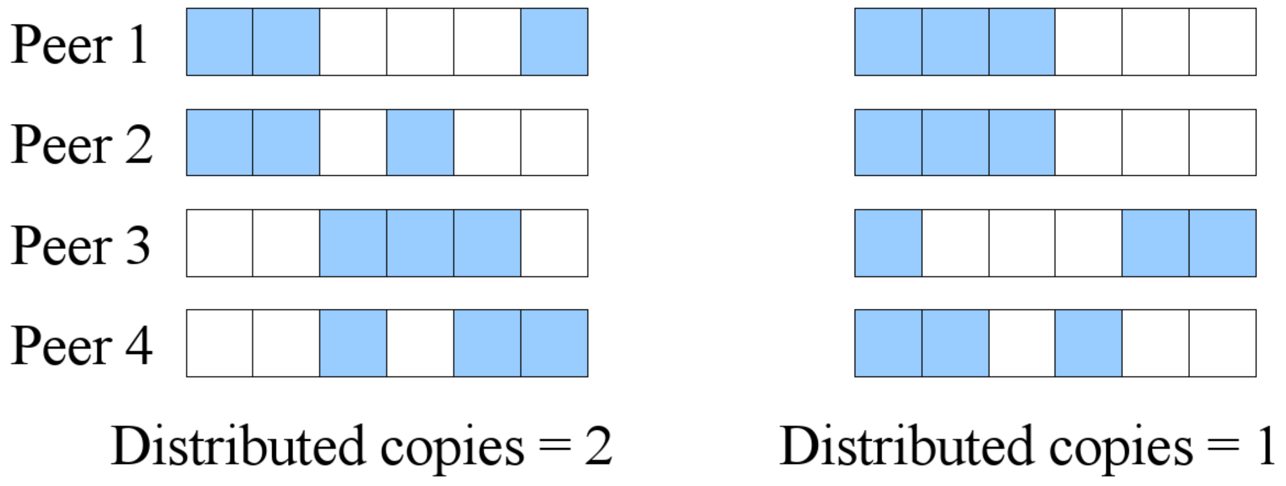
- To minimize overlap:
  - Download random pieces
  - Prioritize the rarest pieces

BitTorrent – Reliability

- Tolerant against dropping peers
- Ability to verify data integrity (SHA-1 hashes)
- Maximize the number of *distributed copies*

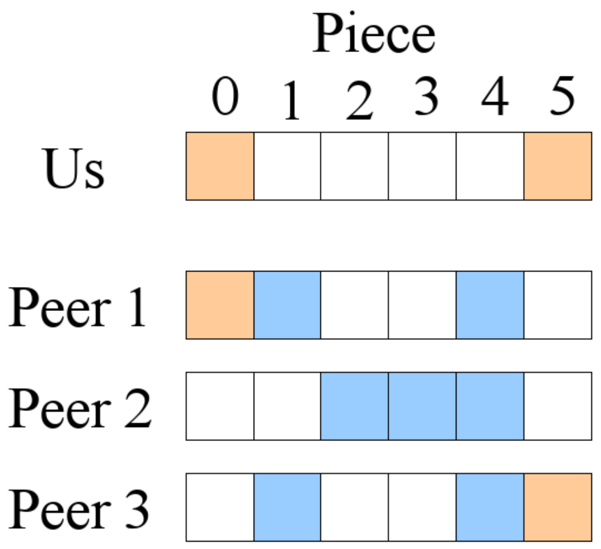
BitTorrent – Reliability

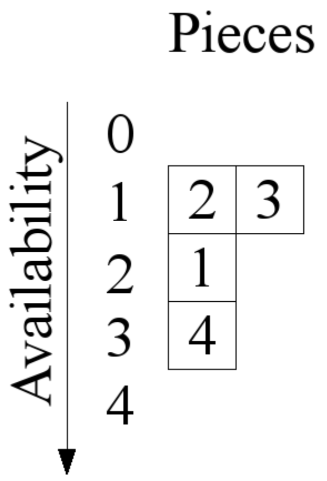
- Distributed copies
  - Number of copies of the rarest piece



BitTorrent – Reliability

- Rarest first
  - Pick a **random** piece from the set of **rarest** pieces.
  - Ignore pieces we already have





BitTorrent – Trackerless torrents

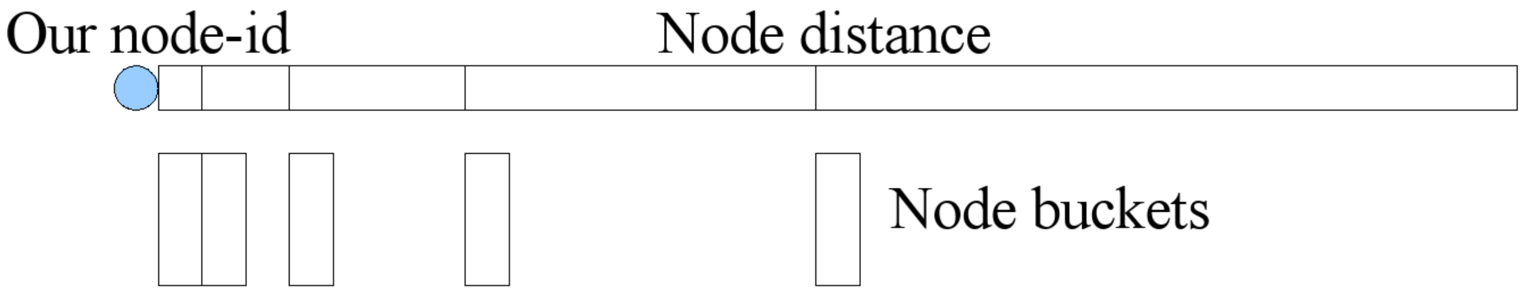
- Common problem with trackers:
  - Single point of failure
  - Bandwidth bottleneck
  - Legal issues
- Solutions:
  - Multiple trackers
  - UDP trackers
  - DHT tracker

BitTorrent – Trackerless torrents

- DHT
  - Kademlia as DHT
  - The key is the *info-hash*, a hash of the meta data.
  - The data is not the file, but a list of peers in the swarm
  - Each node is assigned an ID, and nodes order themselves in a defined topography

BitTorrent – Trackerless torrents

- Each node knows much more about close nodes than distant nodes, similarly to Pastry.
- Querying a node will on average halve the distance, making a search O(log N).



BitTorrent – Trackerless torrents

- Each peer announces itself with the distributed tracker
  - Looking up the 8 nodes closes to the info-hash of the torrent
  - Sending announce messages to them
  - Those 8 nodes will then add the announcing peer to the peer list stored at that info-hash
  - 8 nodes is considered enough to minimize the probability that all of them will drop from the network within the announce interval.

References

1) Eng Keong Lua; Crowcroft, J.; Pias, M.; Sharma, R.; Lim, S., "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE* , vol.7, no.2, pp.72,93, Second Quarter 2005.
2) Antony I. T. Rowstron and Peter Druschel. 2001. “Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”. Middleware '01,, 329-350.