Introduction to BitTorrent

Arvid Norberg
arvid@cs.umu.se
http://libtorrent.net

Distributed systems C, TDBC85, Umeå University, Fall 2006

Bittorrent

- Introduction
- Efficiency & Reliability
- The incentive mechanism
- Trackerless with DHT

Introduction

- Bittorrent is a system for efficient and scalable replication of large amounts of **static** data
  - Scalable - the throughput increases with the number of downloaders
  - Efficient - it utilises a large amount of available network bandwidth

Introduction

- The file to be distributed is split up in *pieces* and an SHA-1 hash is calculated for each piece

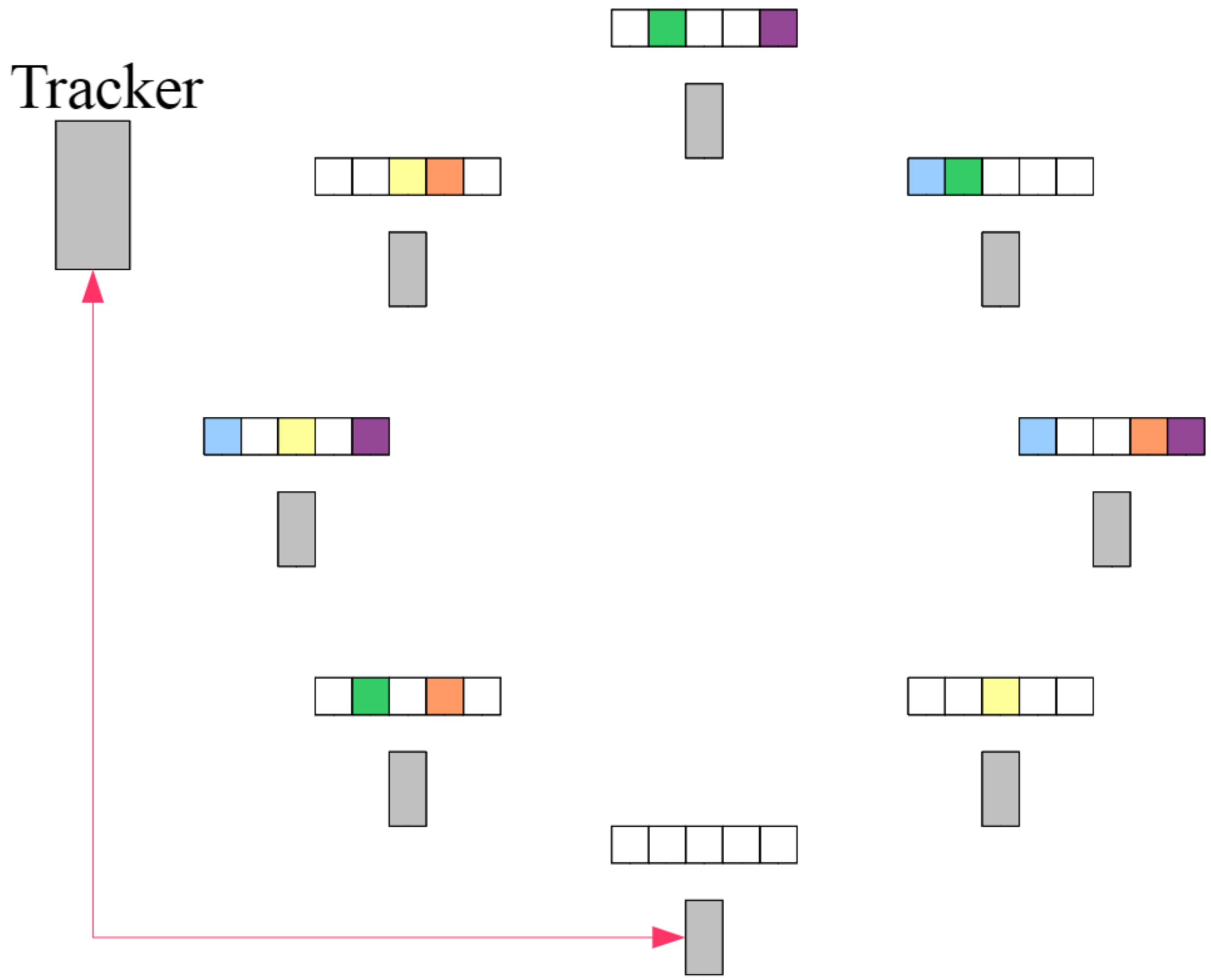
Introduction

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

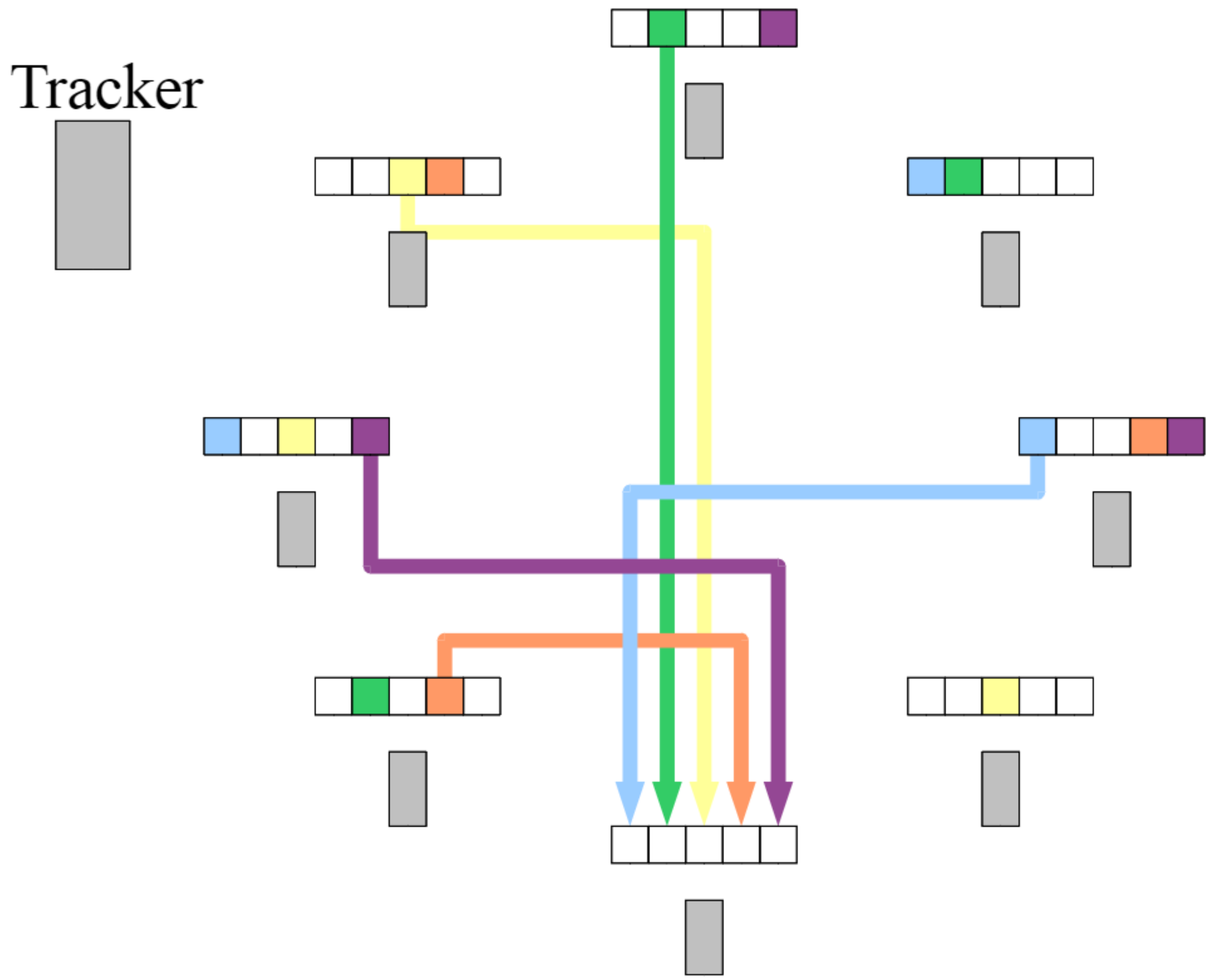
Introduction

- The tracker is a central server keeping a list of all peers participating in the *swarm*
- A swarm is the set of peers that are participating in distributing the same files
- A peer joins a swarm by asking the tracker for a peer list and connects to those peers

Introduction



Introduction



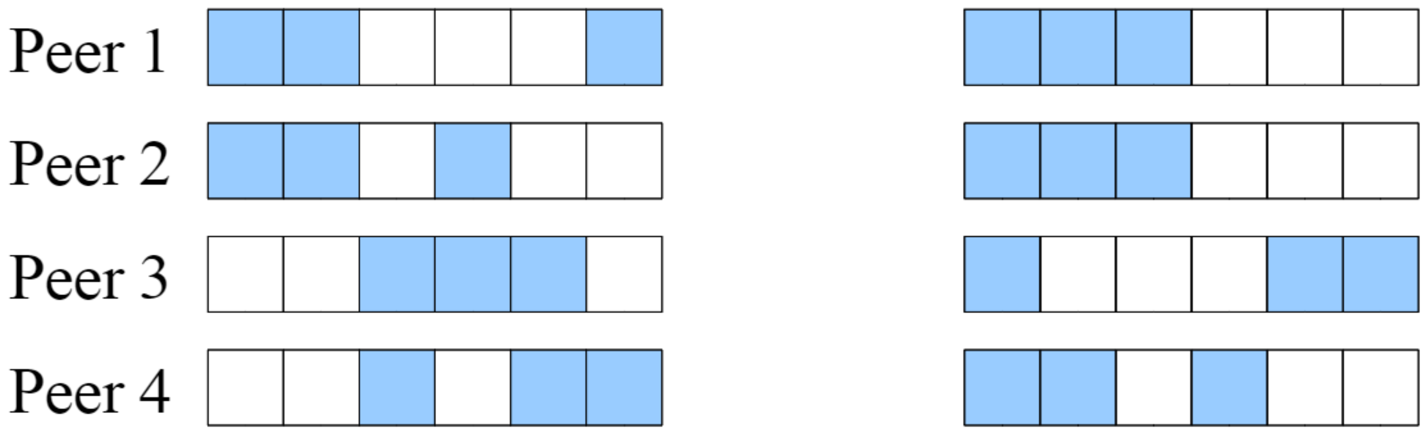
Goals

- Efficiency
  - Fast downloads
- Reliability
  - Tolerant to dropping peers
  - Ability to verify data integrity (SHA-1 hashes)

Efficiency

- Ability to download from many peers yields fast downloads
- Minimise piece overlap among peers to allow each peer to exchange pieces with as many other peers as possible

Piece overlap



- Small overlap
  - Every peer can exchange pieces with all other peers
  - The bandwidth can be well utilised
- Big overlap
  - Only a few peers can exchange pieces
  - The bandwidth is under utilised

Piece overlap

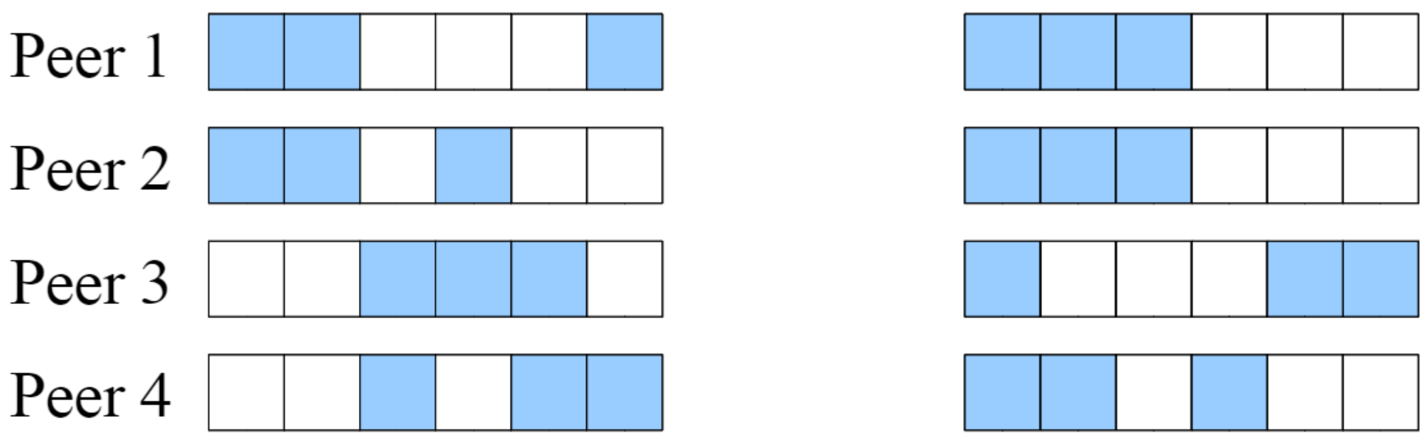
- To minimise piece overlap:
  - Download random pieces
  - Prioritise the rarest pieces, aiming towards uniform piece distribution

Reliability

- Be tolerant against dropping peers
  - Each dropped peer means decreased piece availability
- Maximise piece redundancy
  - Maximise the number of *distributed copies*

Distributed copies

- The number of distributed copies is the number of copies of the **rarest** piece e.g.



Distributed copies = 2   Distributed copies = 1

Distributed copies

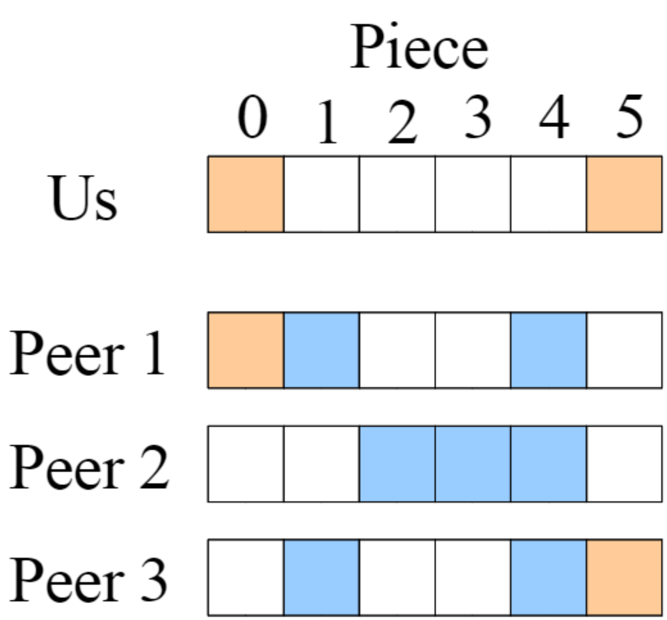
- To maximise the distributed copies, maximise the availability of the rarest pieces
- To increase the availability of a piece, download it
- To maximise the distributed copies:
  - Download the rarest pieces first

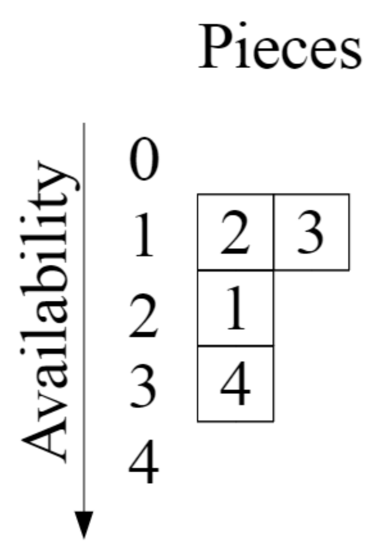
Rarest first

- The piece picking algorithm used in Bittorrent is called *rarest first*
- Picks a **random** piece from the set of **rarest** pieces
- No peer has global knowledge of piece availability, it is approximated by the availibility among neighbours

Rarest first

- Pick a **random** piece from the set of **rarest** pieces {2, 3}
- Ignore pieces that we already have





The incentive to share

- All peer connections are symmetric
- Both peers have an interest of exchanging data
- Peers may prefer to upload to peers from whom they can download
  - Leads to slow starts
  - Fixed in a recent extension

The incentive to share

- There is a loose connection between upload and download speed
- Each peer has an incentive to upload

Trackerless torrents

- Common problems with trackers
  - Single point of failure
  - Bandwidth bottleneck for publishers
- Solutions
  - Multiple trackers
  - UDP trackers
  - DHT tracker

DHT distributed hash table

- Works as a hash table with sha1-hashes as keys
- The key is the *info-hash*, the hash of the metadata. It uniquely identifies a torrent
- The data is a peer list of the peers in the swarm

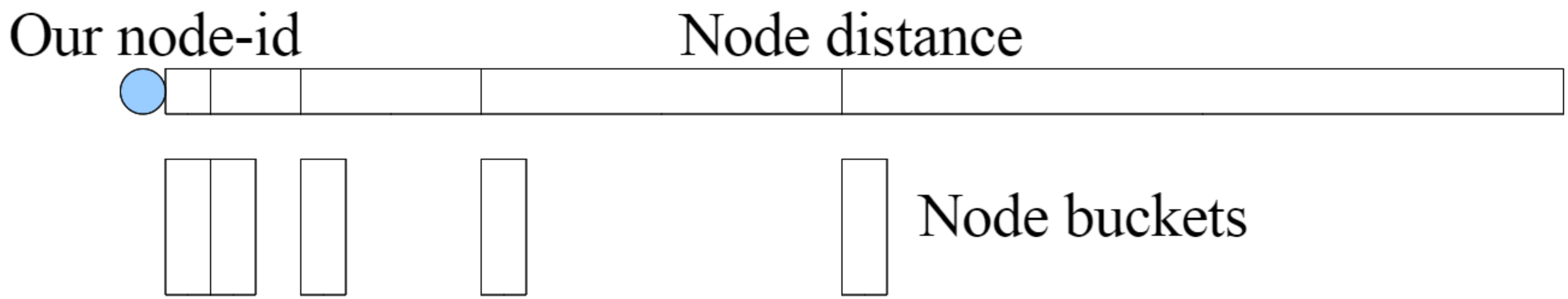
DHT distributed hash table

- Each node is assigned an ID
  - in the key space (160 bit numbers)
- Nodes order themselves in a defined topography
  - Makes it possible to search for Ids by traversing the node topography
- Bittorrent uses *kademlia* as DHT

Kademlia bootstrap

- Each node bootstraps by looking for its own ID
  - The search is done recursively until no closer nodes can be found
  - The nodes passed on the way are stored in the routing table
  - The routing table have more room for *close* nodes than distant nodes

Kademlia routing table



- Each node knows much more about close nodes than distant nodes
  - The key space each bucket represents is growing with the power of 2 with the distance
  - Querying a node for a specific ID will on average halve the distance to the target ID each step

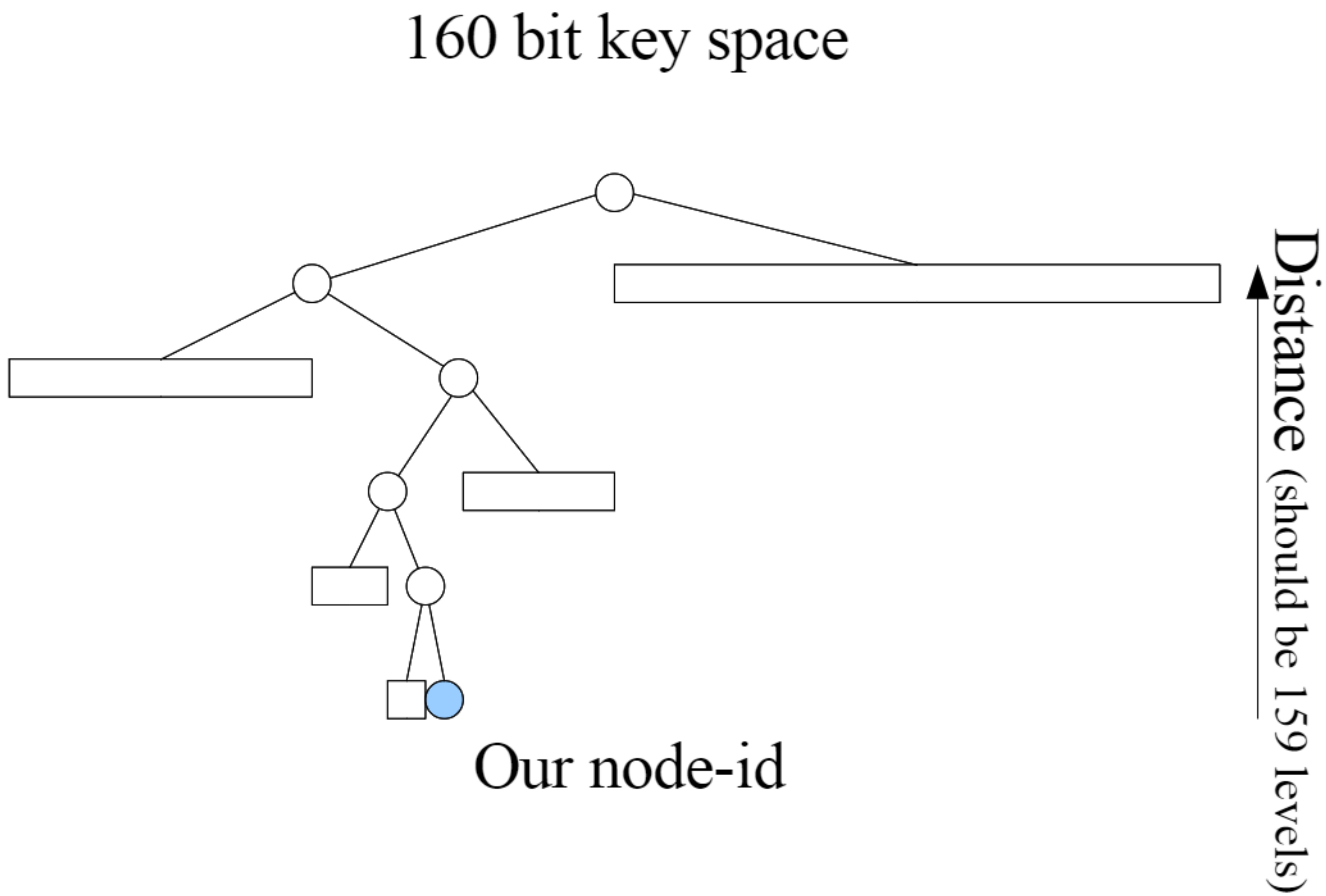
Kademlia routing table

- The distance metric is defined as XOR
  - In practice, the distance is 2 to the power of the inverse of the size of the common bit prefix

```
10011011001 1101010110001
10011011001 0101110101100
```

Common prefix = 11 Distance $\geq 2^{13}$

Kademlia routing table



Kademlia search

- Each search step increases the common bit prefix by at least one
  - Search complexity: $O(\log n)$

Kademlia distributed tracker

- Each peer *announces* itself with the distributed tracker
  - by looking up the 8 nodes closest to the info-hash of the torrent
  - And send an announce message to them
  - Those 8 nodes will then add the announcing peer to the peer list stored at that info-hash

Kademlia distributed tracker

- A peer joins a torrent by looking up the peer list at a specific info-hash
  - Like a search but nodes return the peer list if they have it

Kademlia distributed tracker

- 8 nodes is considered enough to minimise the probability that all of them will drop from the network within the announce interval
  - Each announce looks up new nodes, in case nodes have joined the network with Ids closer to the info-hash than a previous node