The BitTorrent Protocol

Taken from

http://www.cs.uiowa.edu/~ghosh/bittorrent.ppt

What is BitTorrent?

Efficient content distribution system using *file swarming*. Usually does not perform all the functions of a typical p2p system, like *searching*.

BitTorrent traffic

CacheLogic estimated that BitTorrent traffic accounts for roughly 35% of all traffic on the Internet.

File sharing

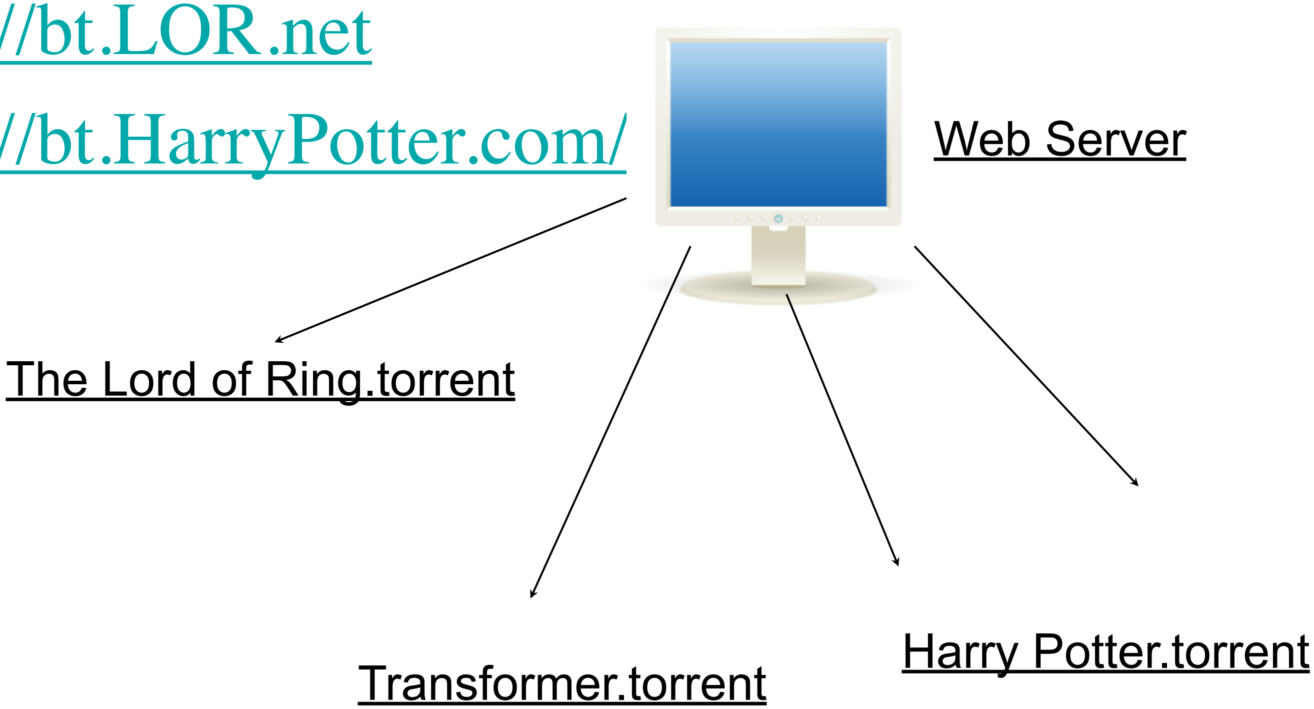
To share a file or group of files, a peer first creates a **.torrent** file, a small file that contains

(1) **metadata** about the files to be shared, and
(2) Information about the **tracker**, the computer that coordinates the file distribution.

Peers first obtain a **.torrent** file, and then connect to the specified **tracker,** which tells them from which other peers to download the pieces of the file.

BT Components

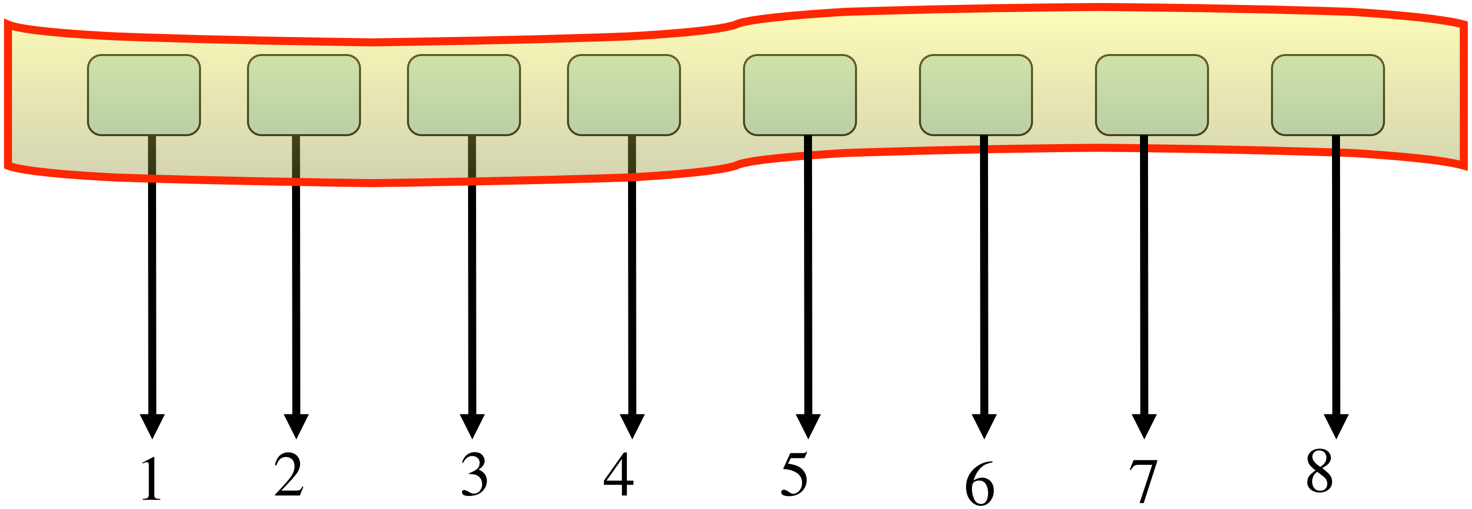
- On a public domain site, obtain .torrent file. for example:
  - http://bt.LOR.net
  - http://bt.HarryPotter.com/



File sharing

Large files are broken into pieces of size between

64 KB and 1 MB

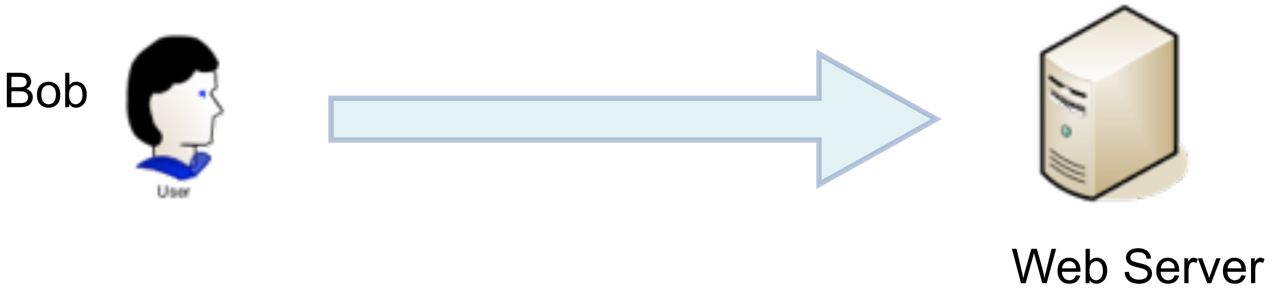


BT: publishing a file

Bob

Web Server

BT: publishing a file



Bob

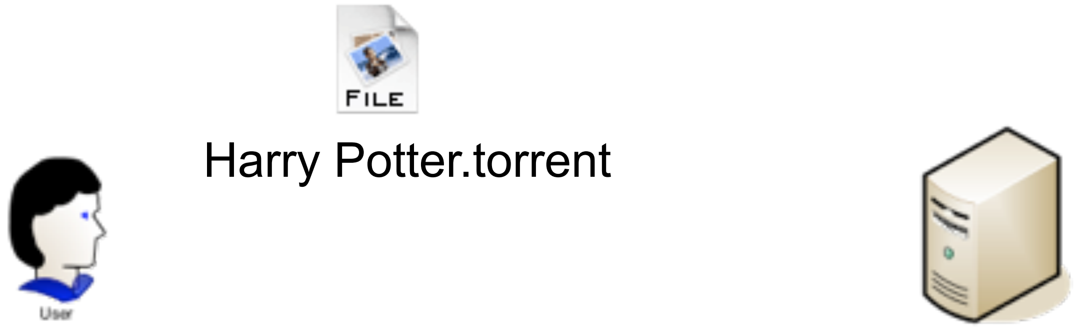
Web Server

BT: publishing a file

Bob

Web Server

BT: publishing a file

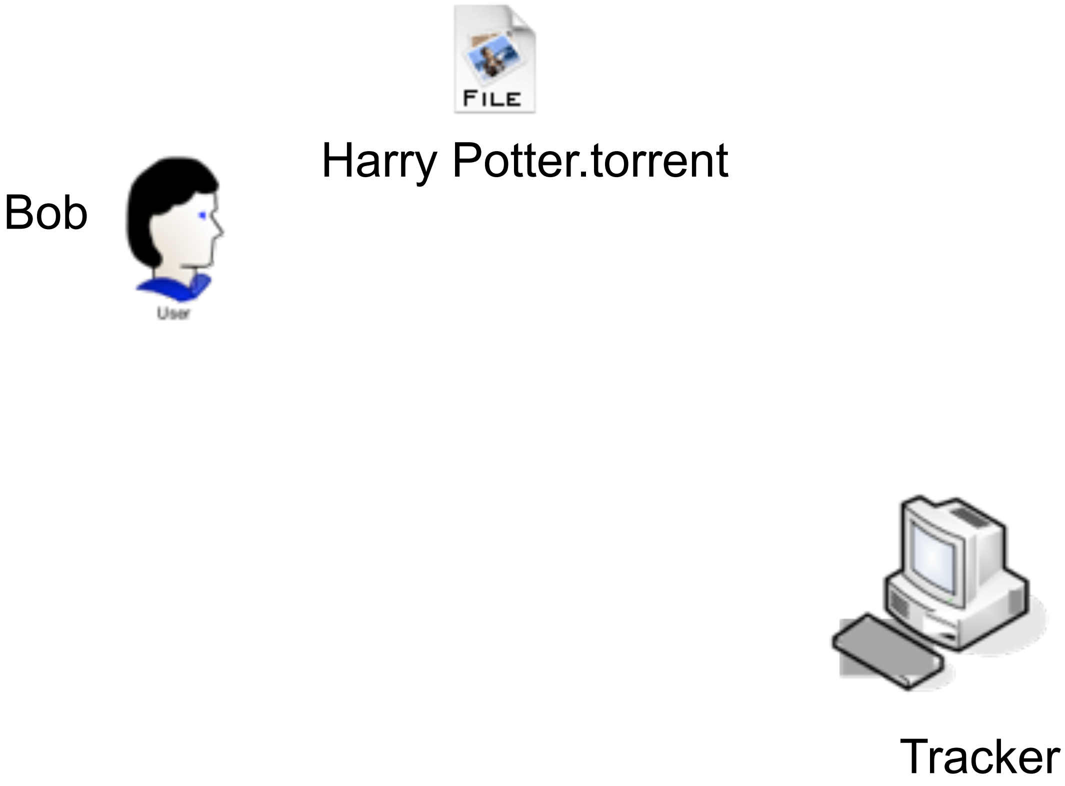


Harry Potter.torrent

Bob

Web Server

BT: publishing a file

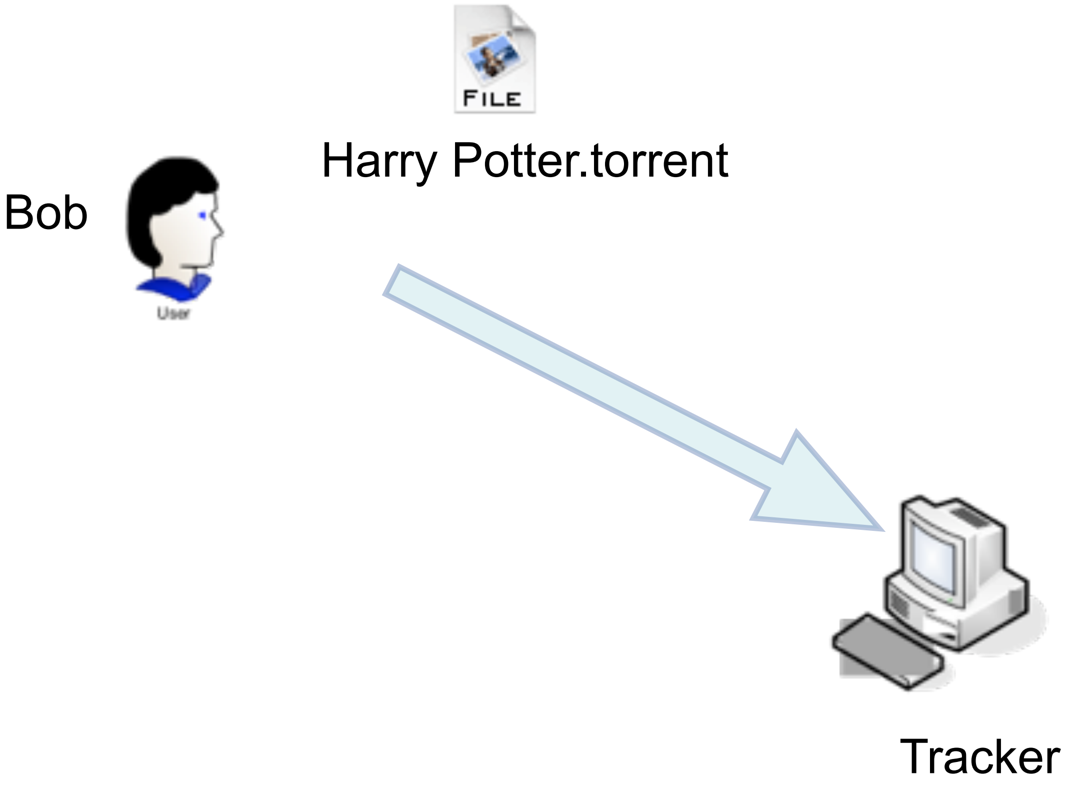


Harry Potter.torrent

Bob

Tracker

BT: publishing a file

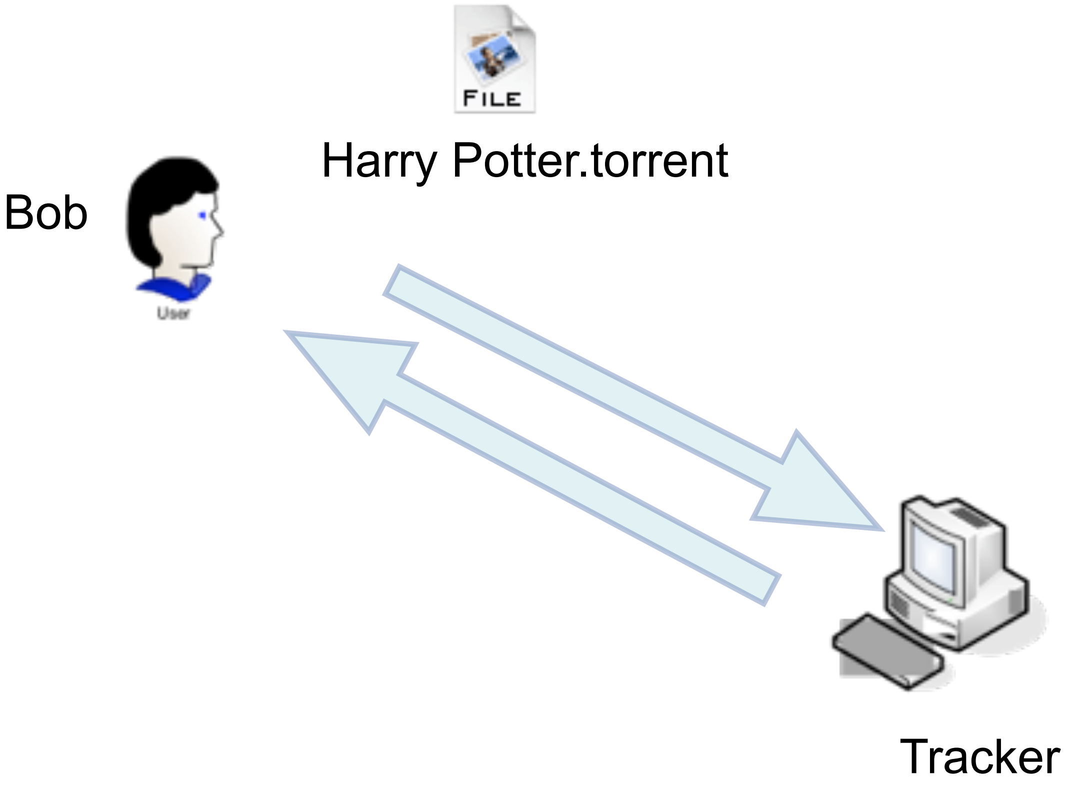


Harry Potter.torrent

Bob

Tracker

BT: publishing a file

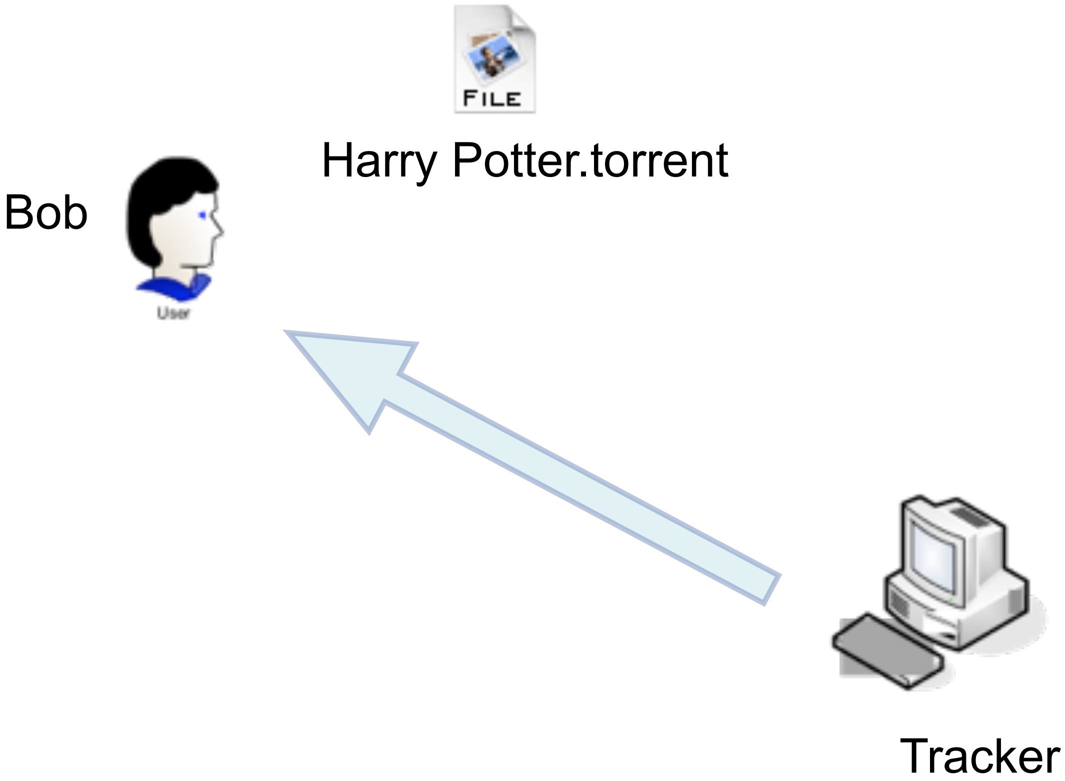


Harry Potter.torrent

Bob

Tracker

BT: publishing a file

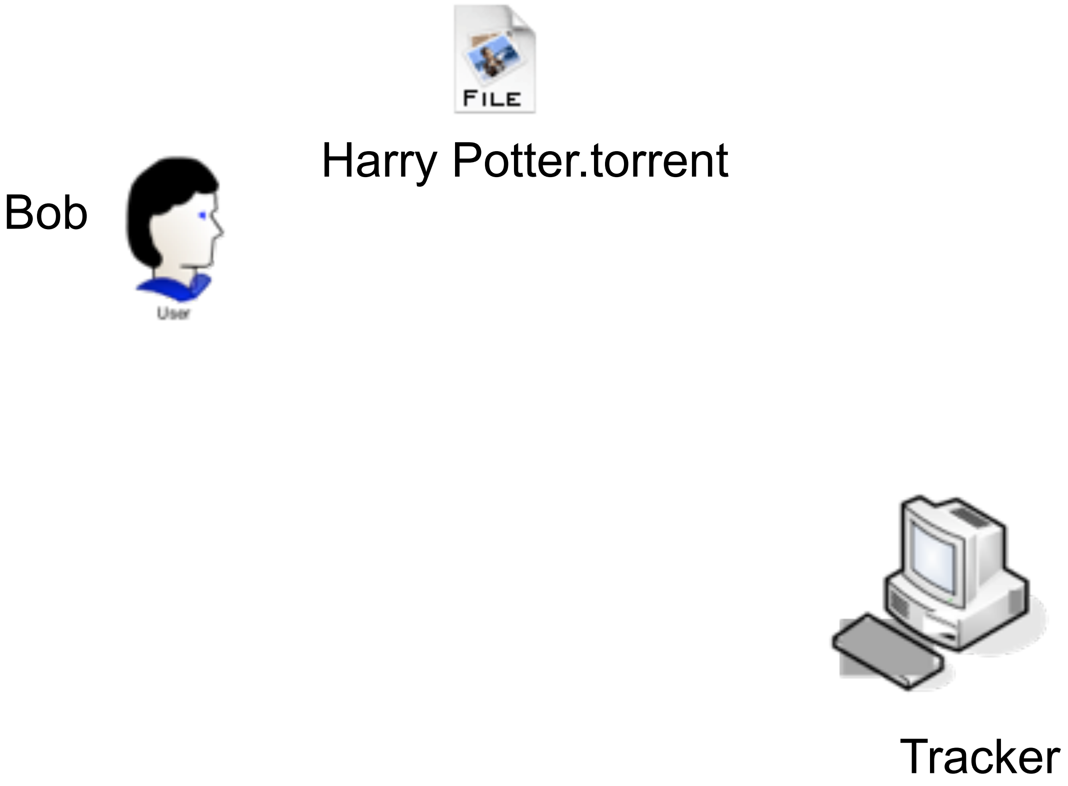


Harry Potter.torrent

Bob

Tracker

BT: publishing a file

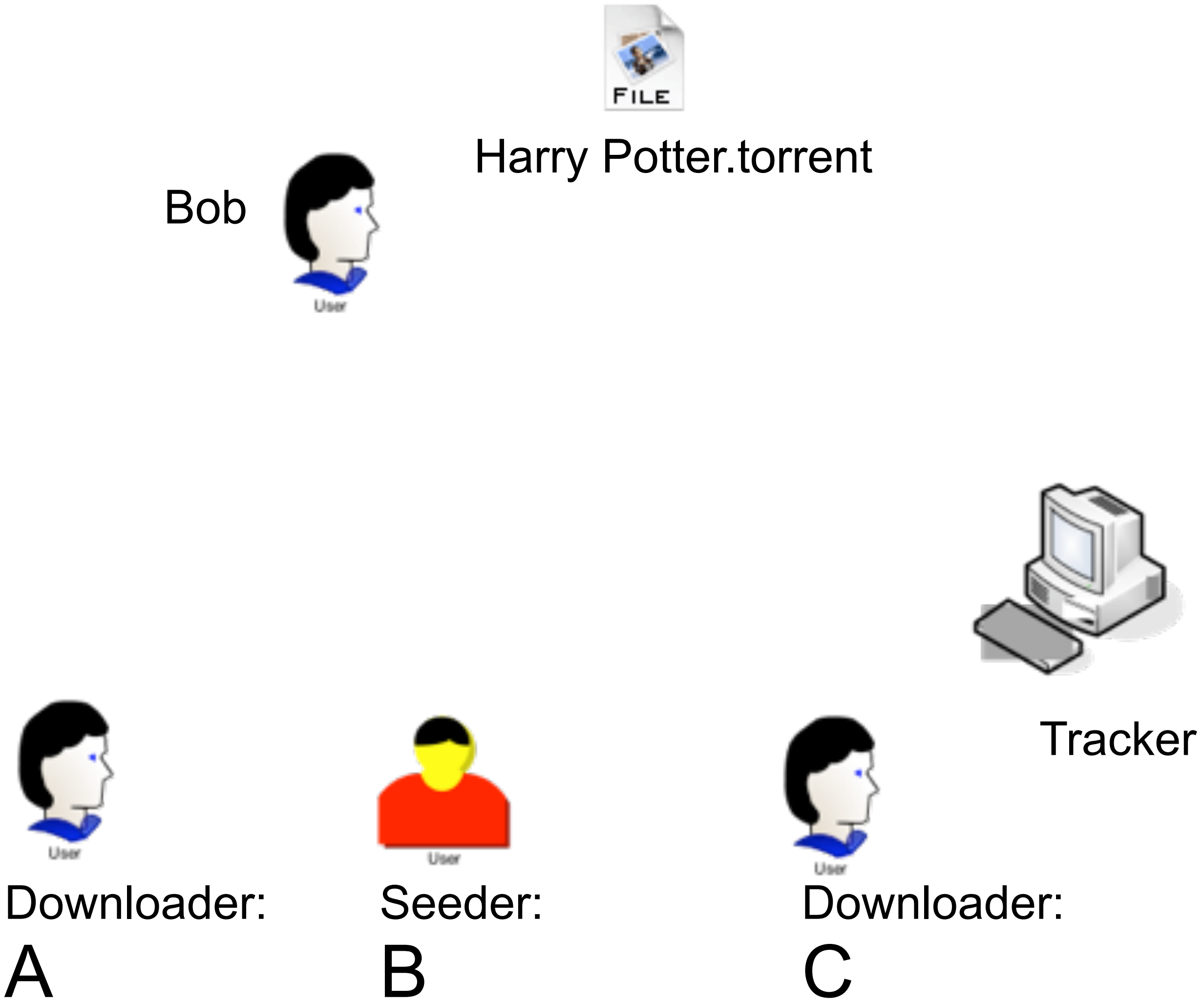


Harry Potter.torrent

Bob

Tracker

BT: publishing a file



Harry Potter.torrent

Bob

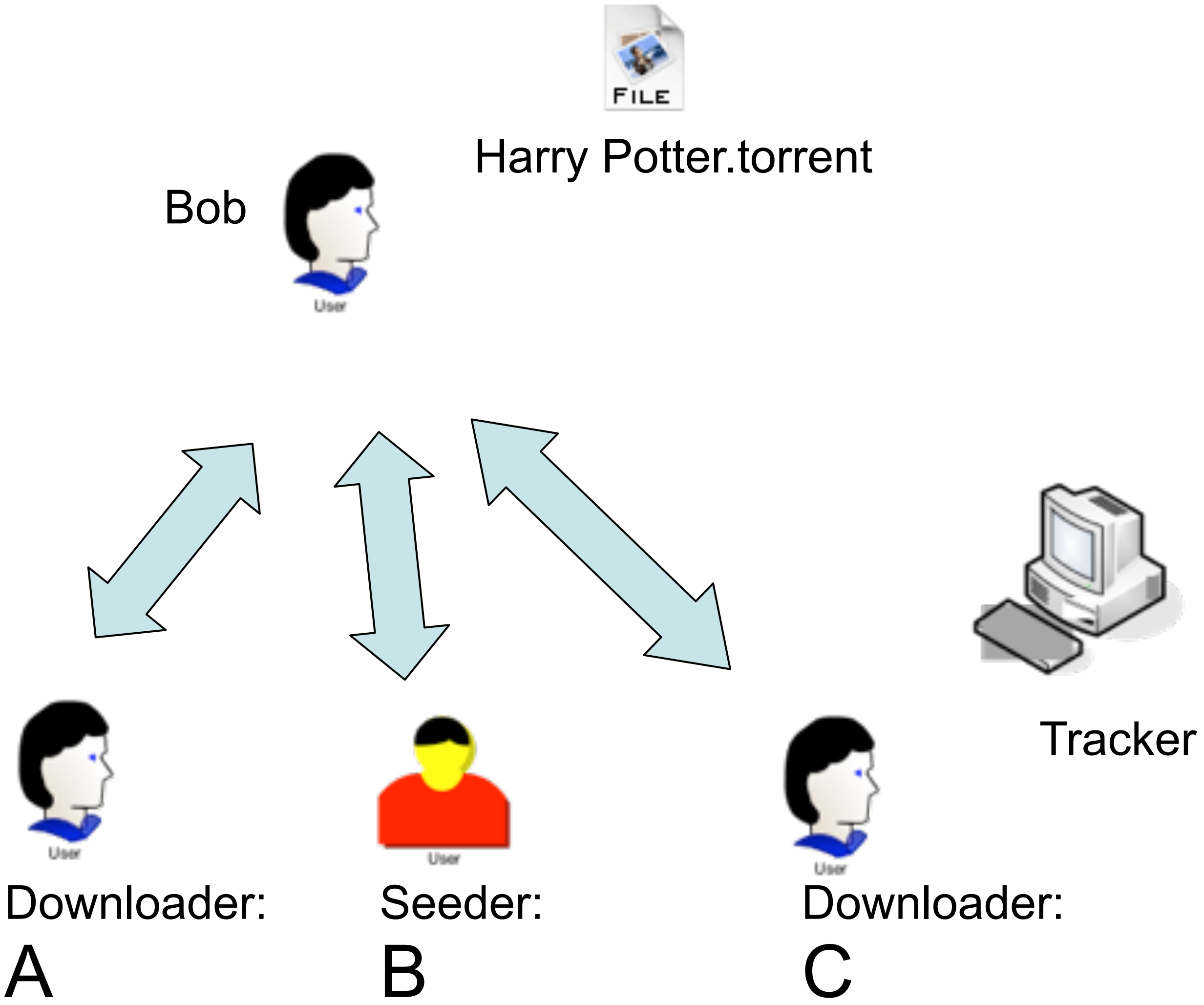
Tracker

Downloader:
A

Seeder:
B

Downloader:
C

BT: publishing a file



Harry Potter.torrent

Bob

Tracker

Downloader:
A

Seeder:
B

Downloader:
C

The .torrent file

- The URL of the tracker
- Pieces <hash1,hash 2,….hash n>
- Piece length
- Name
- Length of the file

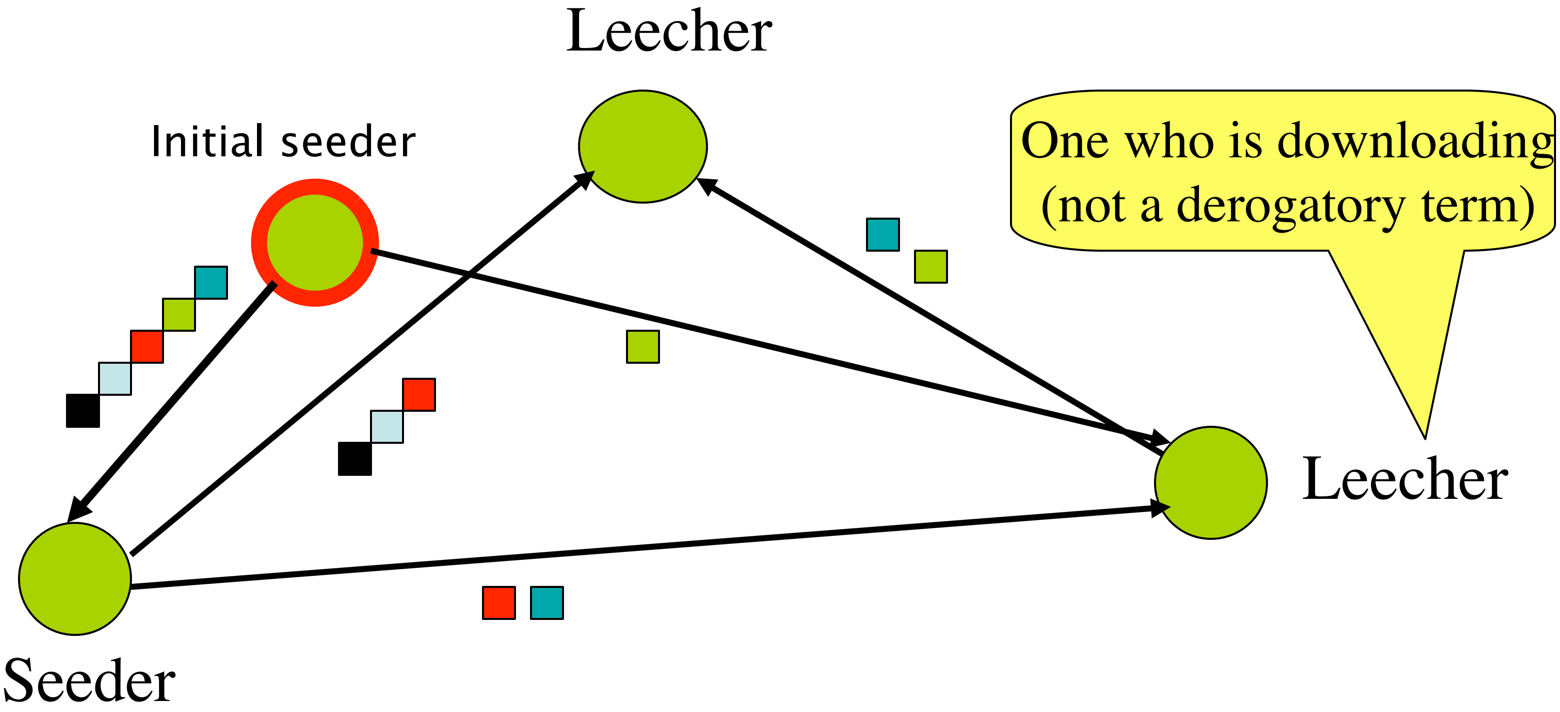
The Tracker

- IP address, port, peer id
- State information (Completed or Downloading)
- Returns a random list of peers

BitTorrent Lingo

Seeder = a peer that provides the complete file.

Initial seeder = a peer that provides the initial copy.



Simple example

{1,2,3,4,5,6,7,8,9,10}

Seeder: A

Simple example

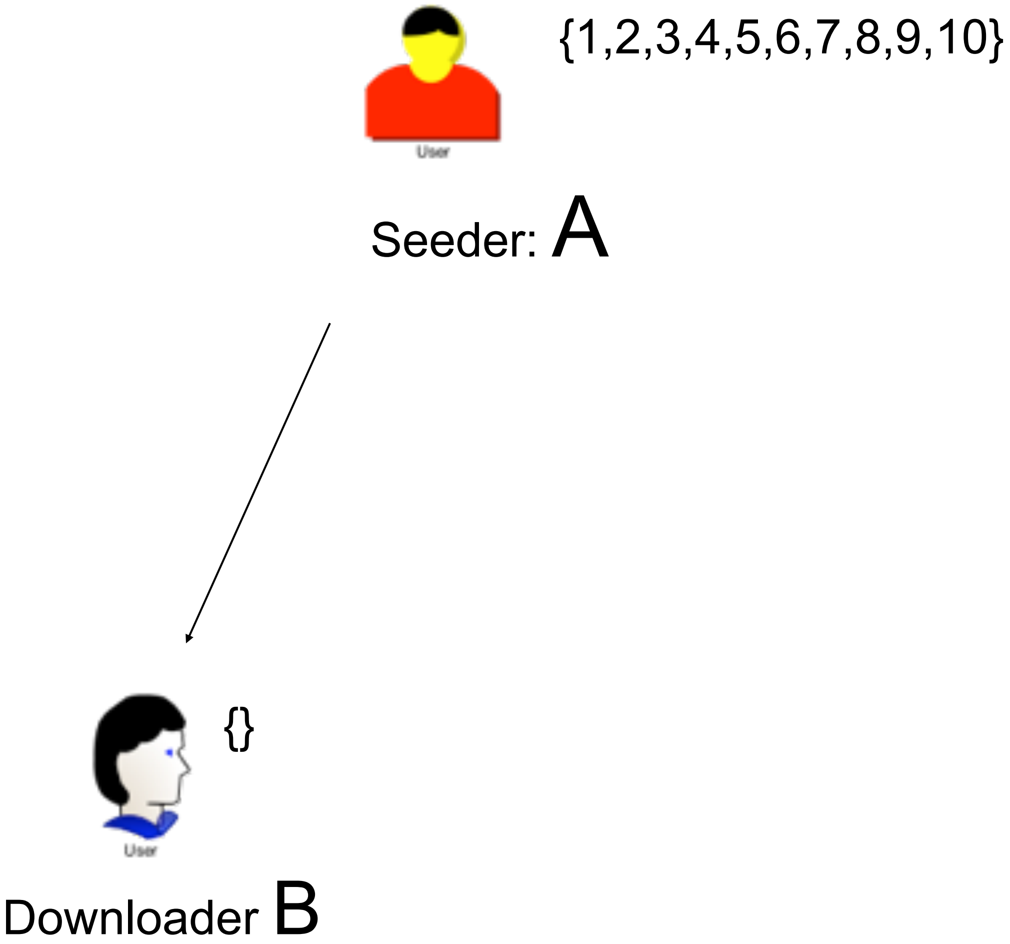
{1,2,3,4,5,6,7,8,9,10}

Seeder: A

{}

Downloader B

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

{}

Downloader B

Simple example

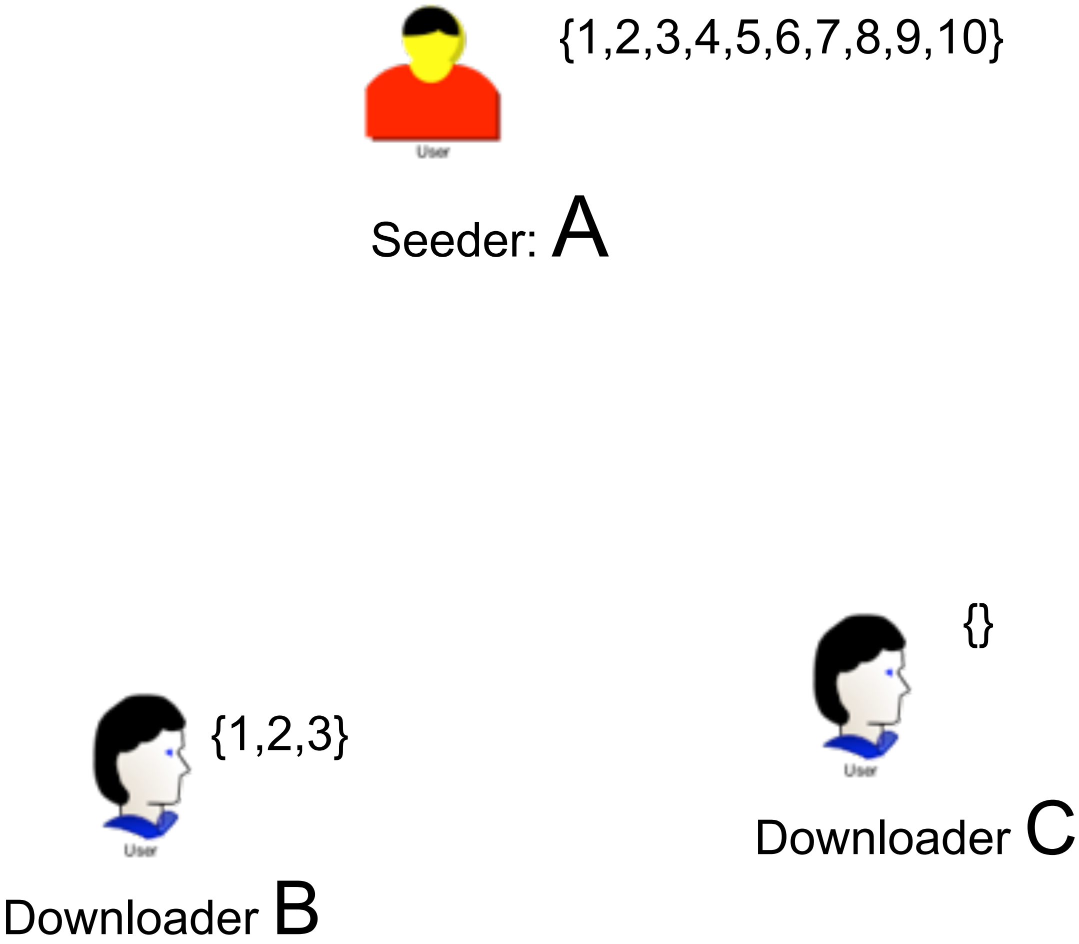
{1,2,3,4,5,6,7,8,9,10}

Seeder: A

{1,2,3}

Downloader B

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

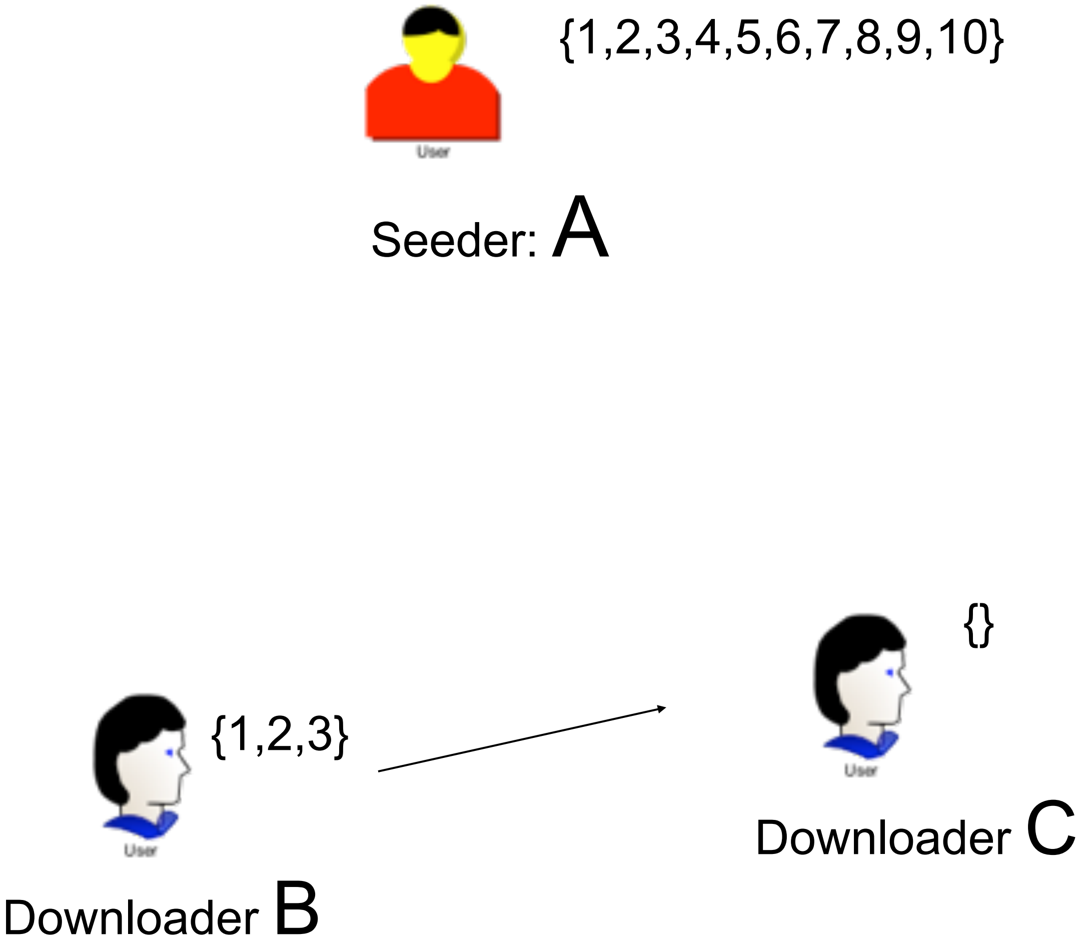
{1,2,3}

Downloader B

{}

Downloader C

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

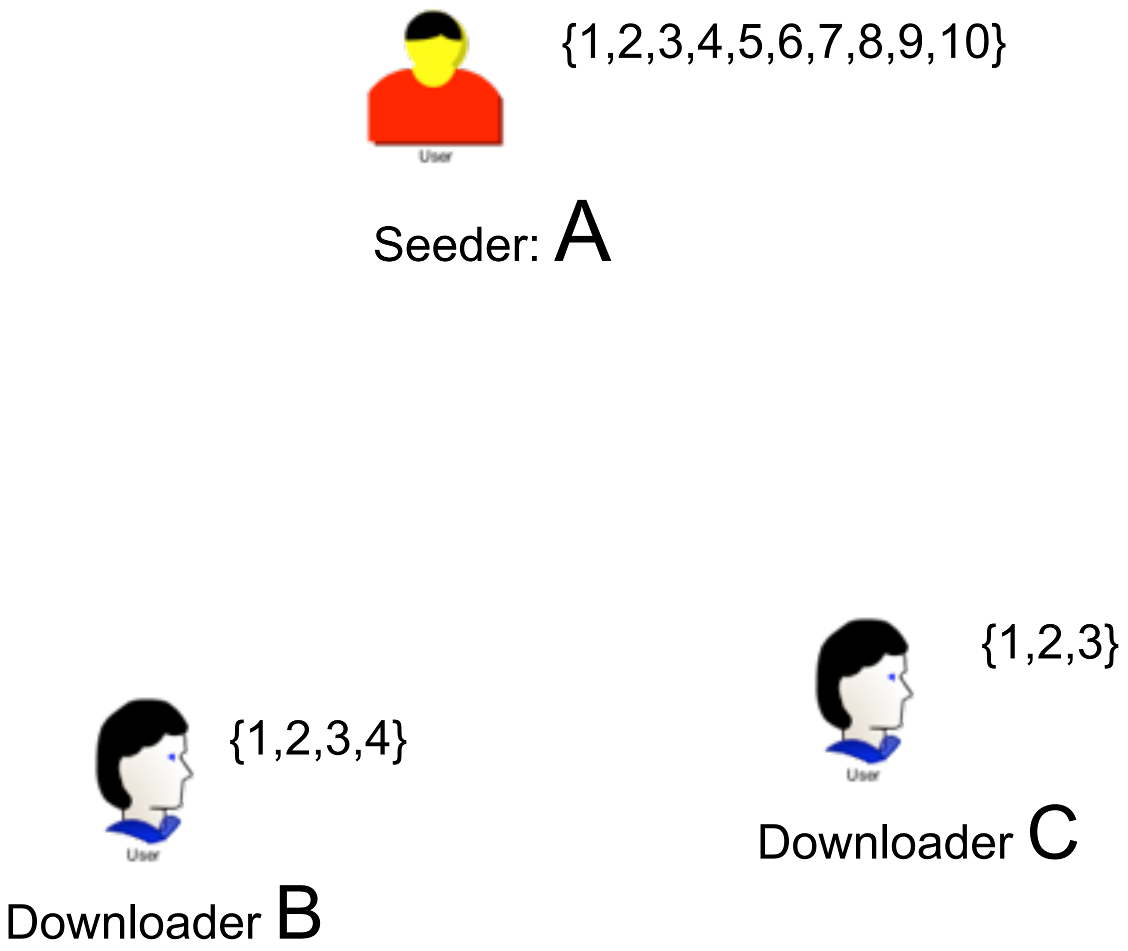
{1,2,3}

Downloader B

{}

Downloader C

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

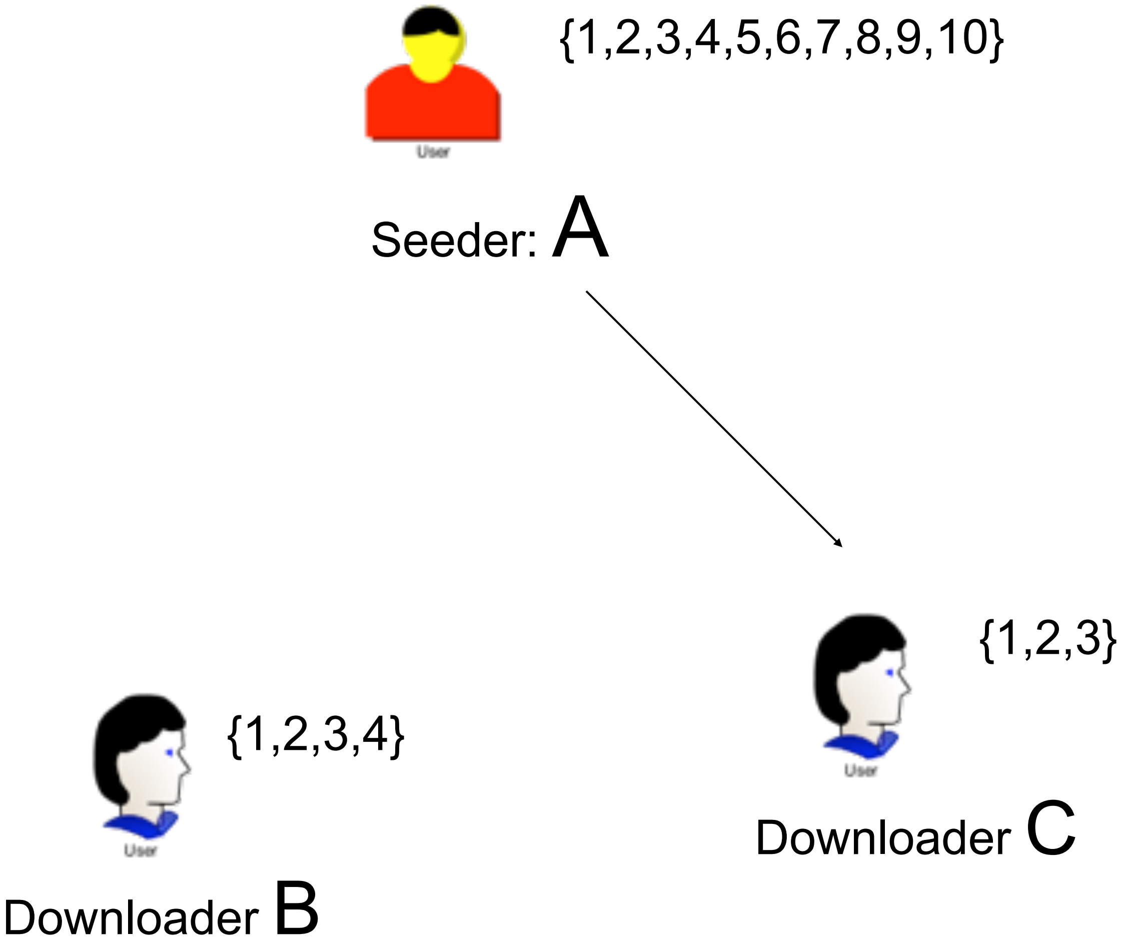
{1,2,3}

Downloader C

{1,2,3,4}

Downloader B

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

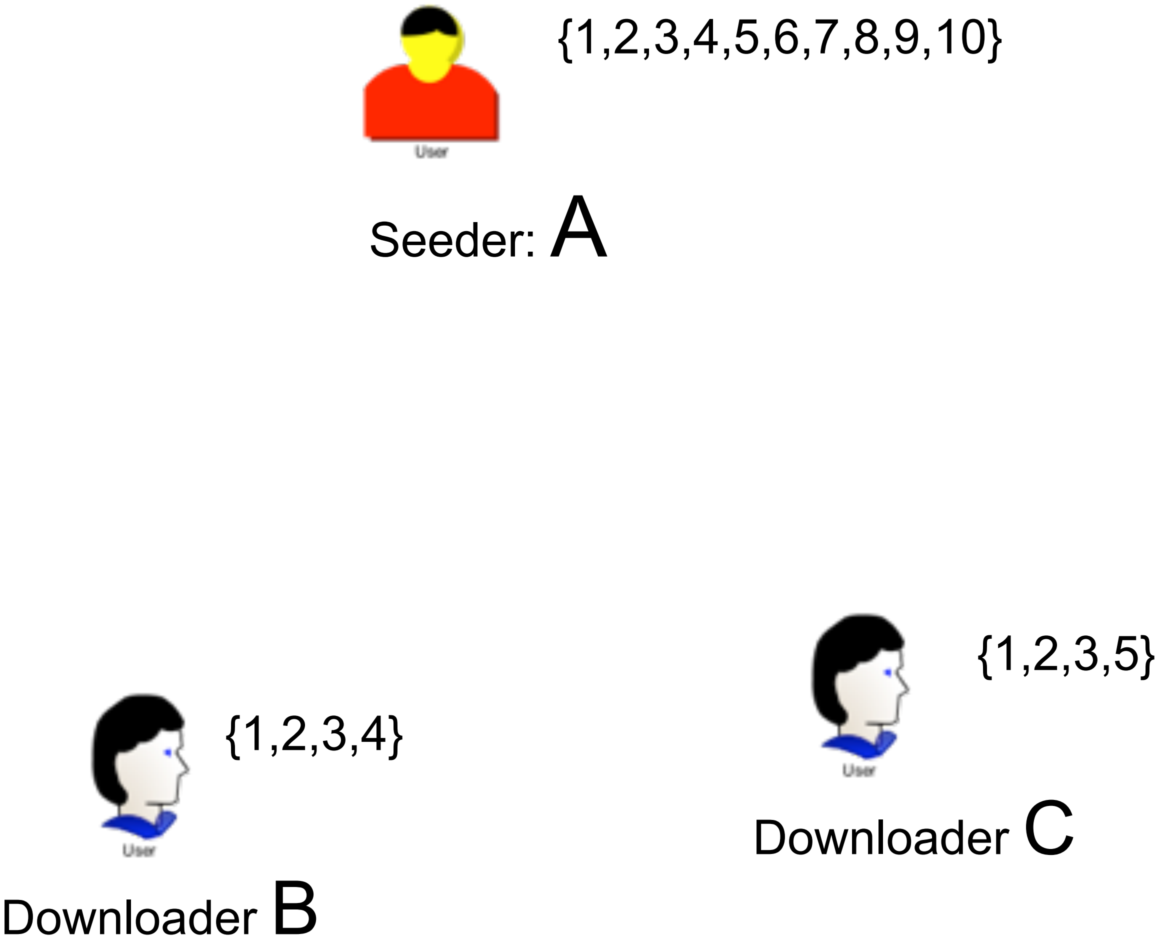
{1,2,3}

Downloader C

{1,2,3,4}

Downloader B

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

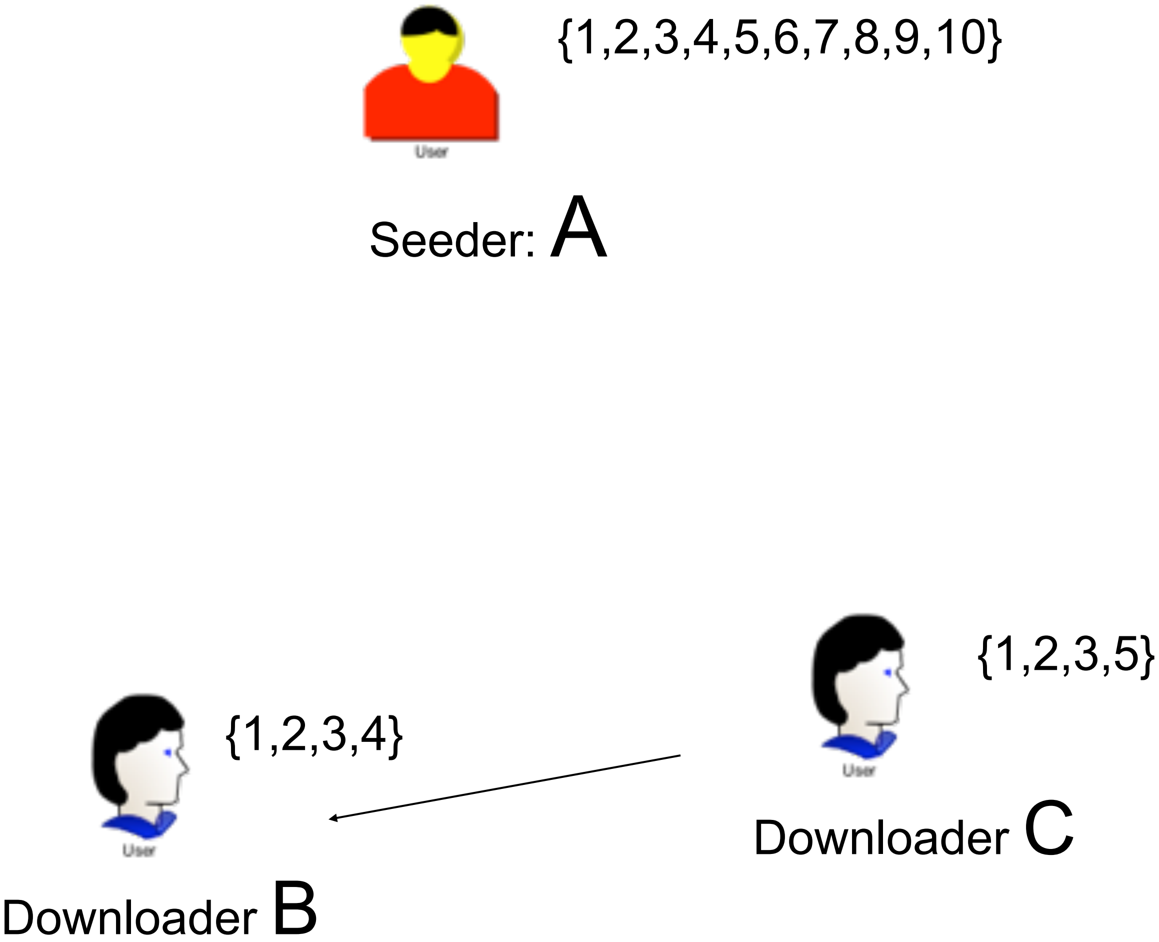
{1,2,3,4}

Downloader B

{1,2,3,5}

Downloader C

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

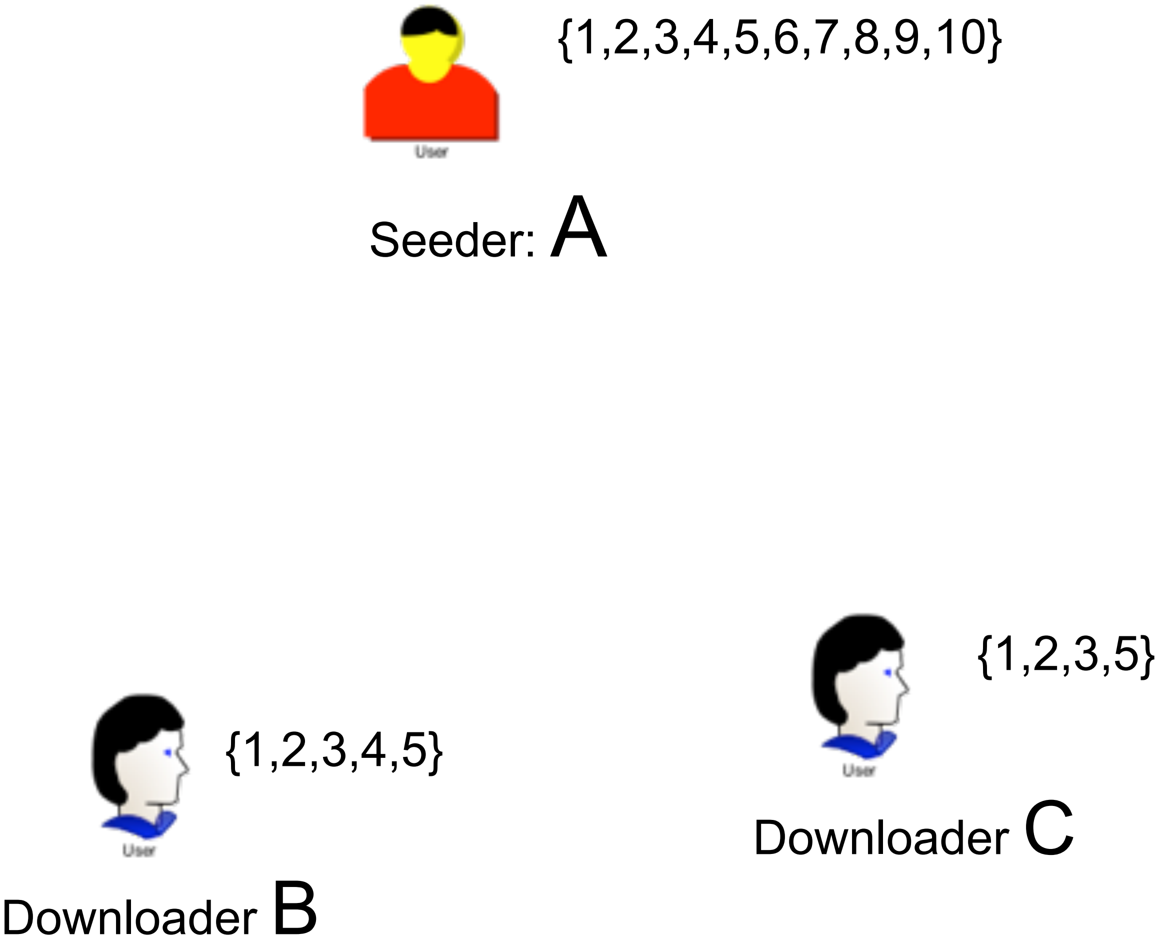
{1,2,3,5}

Downloader C

{1,2,3,4}

Downloader B

Simple example



{1,2,3,4,5,6,7,8,9,10}

Seeder: A

{1,2,3,4,5}

Downloader B

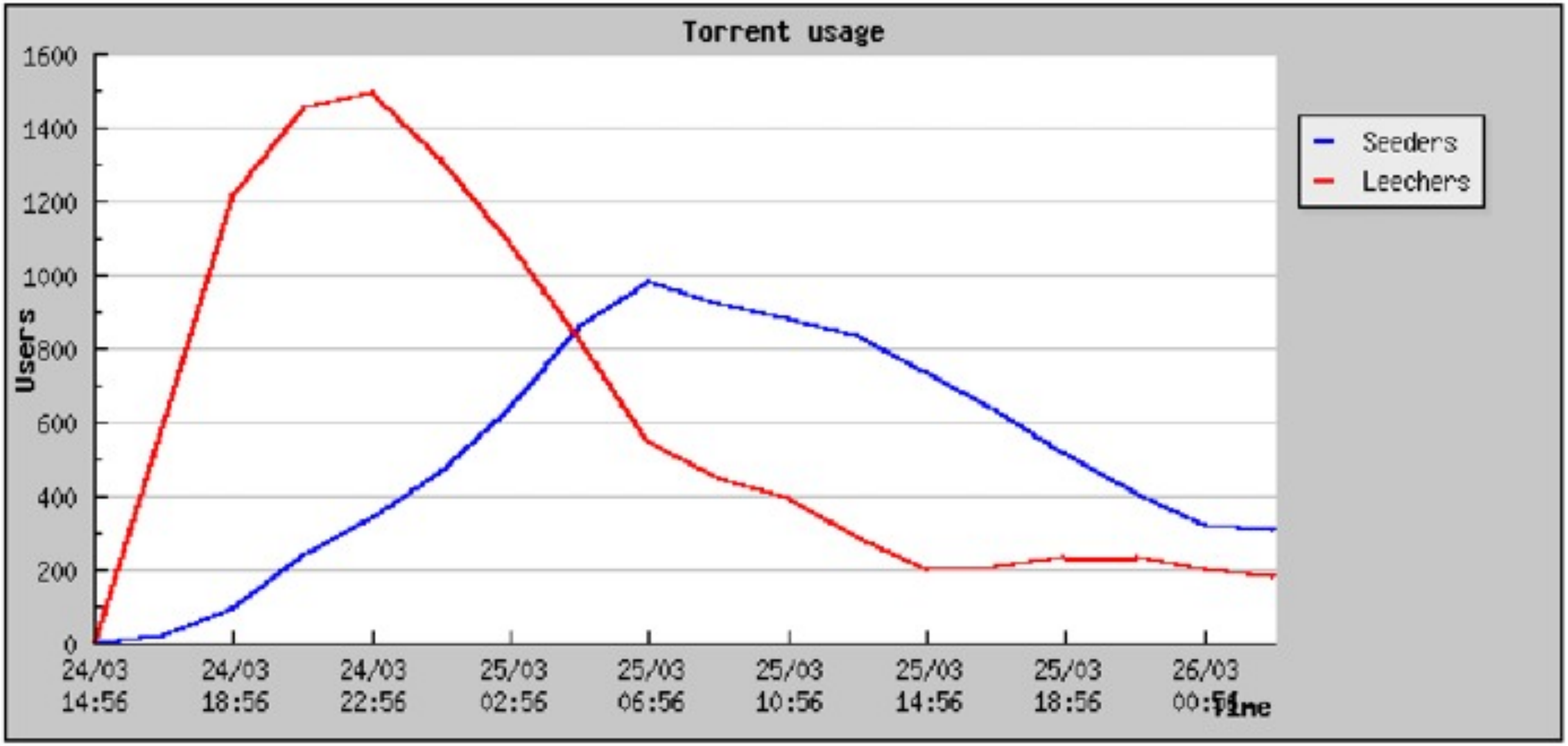
{1,2,3,5}

Downloader C

Basic Idea

- Initial seeder chops file into many pieces.
- Leecher first locates the **.torrent** file that directs it to a **tracker**, which tells which other peers are downloading that file. As a leecher downloads pieces of the file, replicas of the pieces are created. ***More downloads mean more replicas available***
- As soon as a leecher has a complete piece, it can potentially share it with other downloaders. Eventually each leecher becomes a seeder by obtaining all the pieces, and assembles the file. Verifies the checksum.

Operation


Torrent usage
Users
1600
1400
1200
1000
800
600
400
200
0
24/03 14:56
24/03 18:56
24/03 22:56
25/03 02:56
25/03 06:56
25/03 10:56
25/03 14:56
25/03 18:56
26/03 00:56
Time
Seeders
Leechers

Pieces and Sub-Pieces

- A piece is broken into sub-pieces ... typically 16KB in size
- Until a piece is assembled, only download the sub-pieces of that piece only
- This policy lets pieces assemble quickly

Pipelining

- When transferring data over TCP, always have several requests pending at once, to avoid a delay between pieces being sent. At any point in time, some number, typically 5, are requested simultaneously.
- Every time a piece or a sub-piece arrives, a new request is sent out.

Piece Selection

- The order in which pieces are selected by different peers is critical for good performance
- If an inefficient policy is used, then peers may end up in a situation where each has all identical set of easily available pieces, and none of the missing ones.
- If the original seed is prematurely taken down, then the file cannot be completely downloaded! What are "good policies?"

BT: internal Chunk Selection mechanisms

- **Strict Priority**
  - First Priority
- **Rarest First**
  - General rule
- **Random First Piece**
  - Special case, at the beginning
- **Endgame Mode**
  - Special case

Random First Piece

- Initially, a peer has nothing to trade
- Important to get a complete piece ASAP
- Select a random piece of the file and download it

Rarest Piece First

- Determine the pieces that are most rare among your peers, and download those first.
- This ensures that the most commonly available pieces are left till the end to download.

Endgame Mode

- Near the end, missing pieces are requested from every peer containing them. When the piece arrives, the pending requests for that piece are cancelled.
- This ensures that a download is not prevented from completion due to a single peer with a slow transfer rate.
- Some bandwidth is wasted, but in practice, this is not too much.

BT: internal mechanism

- Built-in **incentive** mechanism (where all the magic happens):
  - Choking Algorithm
  - Optimistic Unchoking

Choking

- **Choking** is a *temporary refusal* to upload. It is one of BitTorrent's most powerful idea to deal with **free riders (those who only download but never upload)**.
- *Tit-for-tat strategy* is based on game-theoretic concepts.

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm
caps the number of simultaneous
uploads for good TCP performance.
Avoids choking and unchoking
too quickly, (known as fibrillation)..

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm
caps the number of simultaneous
uploads for good TCP performance.
Avoids choking and unchoking
too quickly, (known as fibrillation)..

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

Alice

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm
caps the number of simultaneous
uploads for good TCP performance.
Avoids choking and unchoking
too quickly, (known as fibrillation)..

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

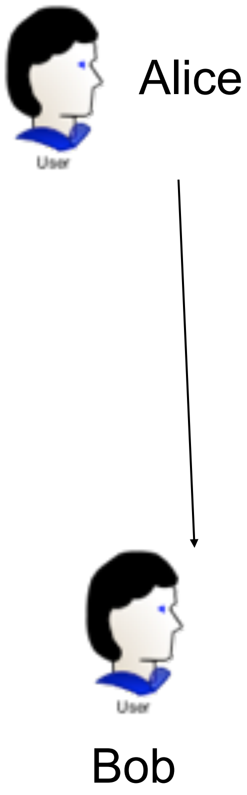
A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

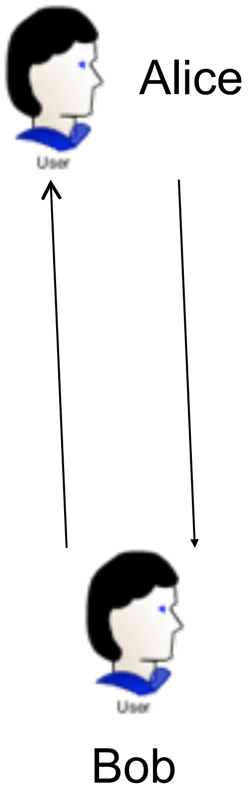


Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

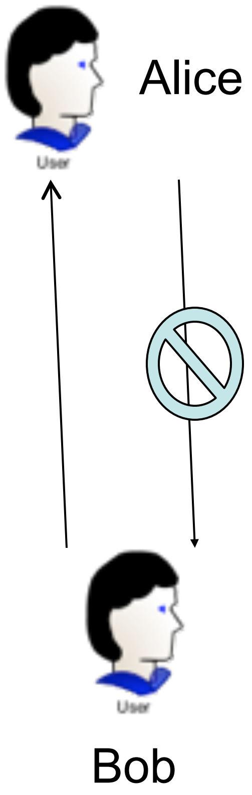


Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm
caps the number of simultaneous
uploads for good TCP performance.
Avoids choking and unchoking
too quickly, (known as fibrillation)..

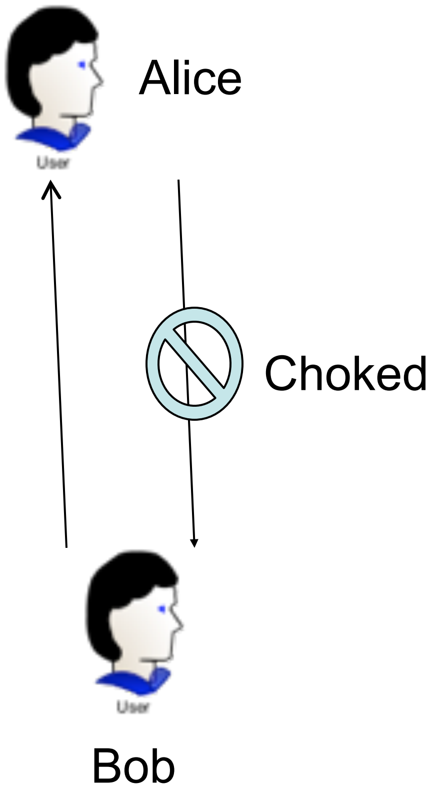


Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

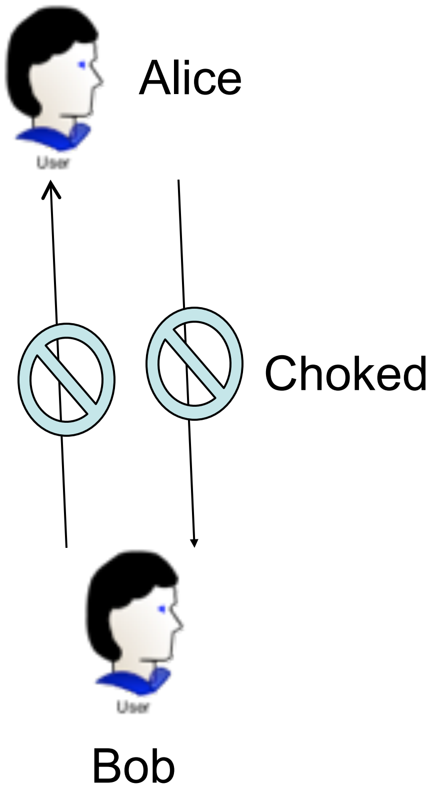


Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..

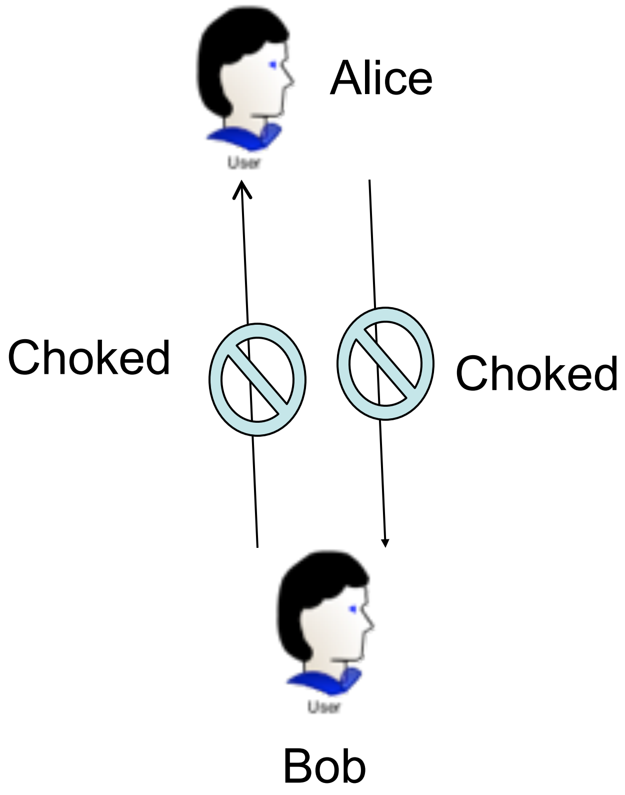


Choking

Reasons for choking:

- Avoid free riders
- Network congestion

A good choking algorithm caps the number of simultaneous uploads for good TCP performance. Avoids choking and unchoking too quickly, (known as fibrillation)..



More on Choking

Peers try out unused connections once in a while to find out if they might be better than the current ones (optimistic unchoking).

Optimistic unchoking

Optimistic unchoking

- A BitTorrent peer has a single “optimistic unchoke” to which it uploads regardless of the current download rate from it. This peer rotates every 30s

Optimistic unchoking

- A BitTorrent peer has a single "optimistic unchoke" to which it uploads regardless of the current download rate from it. This peer rotates every 30s

Optimistic unchoking

- A BitTorrent peer has a single “optimistic unchoke” to which it uploads regardless of the current download rate from it. This peer rotates every 30s

- Reasons:

Optimistic unchoking

- A BitTorrent peer has a single "optimistic unchoke" to which it uploads regardless of the current download rate from it. This peer rotates every 30s

- Reasons:
  - To discover currently unused connections are better than the ones being used

Optimistic unchoking

- A BitTorrent peer has a single "optimistic unchoke" to which it uploads regardless of the current download rate from it. This peer rotates every 30s

- Reasons:
  - To discover currently unused connections are better than the ones being used
  - To provide minimal service to new peers

Upload-Only mode

- Once download is complete, a peer has no download rates to use for comparison nor has any need to use them. The question is, which nodes to upload to?

- Policy: Upload to those with the best upload rate. This ensures that pieces get replicated faster, and new seeders are created fast

Questions about BT

Questions about BT

- Which features contribute to the efficiency of BitTorrent?

Questions about BT

- Which features contribute to the efficiency of BitTorrent?
- What is the effect of bandwidth constraints?

Questions about BT

- Which features contribute to the efficiency of BitTorrent?
- What is the effect of bandwidth constraints?
- Is the Rarest First policy really necessary?

Questions about BT

- Which features contribute to the efficiency of BitTorrent?
- What is the effect of bandwidth constraints?
- Is the Rarest First policy really necessary?
- Must nodes perform seeding after downloading is complete?

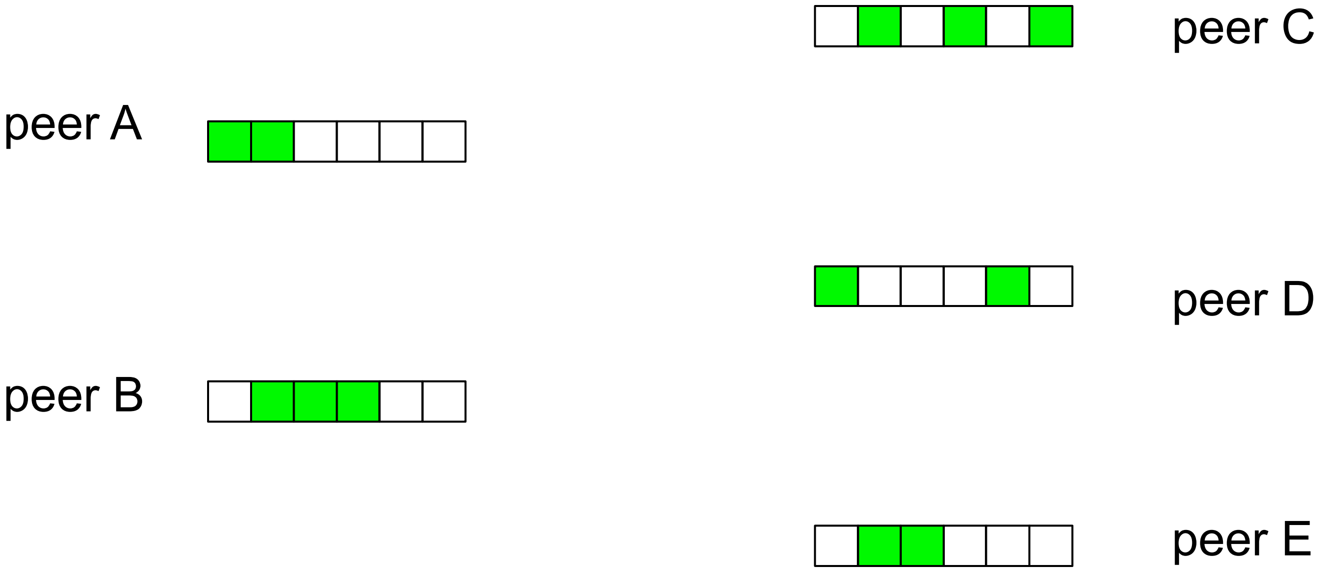
Questions about BT

- Which features contribute to the efficiency of BitTorrent?
- What is the effect of bandwidth constraints?
- Is the Rarest First policy really necessary?
- Must nodes perform seeding after downloading is complete?
- How serious is the Last Piece Problem?

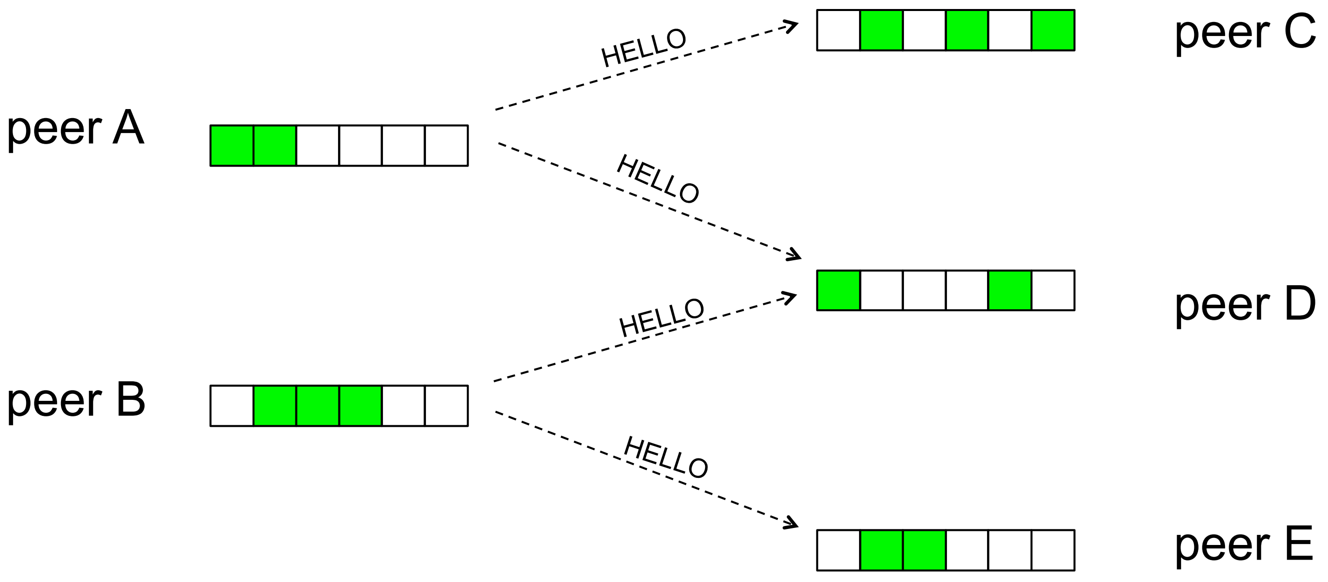
Questions about BT

- Which features contribute to the efficiency of BitTorrent?
- What is the effect of bandwidth constraints?
- Is the Rarest First policy really necessary?
- Must nodes perform seeding after downloading is complete?
- How serious is the Last Piece Problem?
- Does the incentive mechanism affect the performance much?

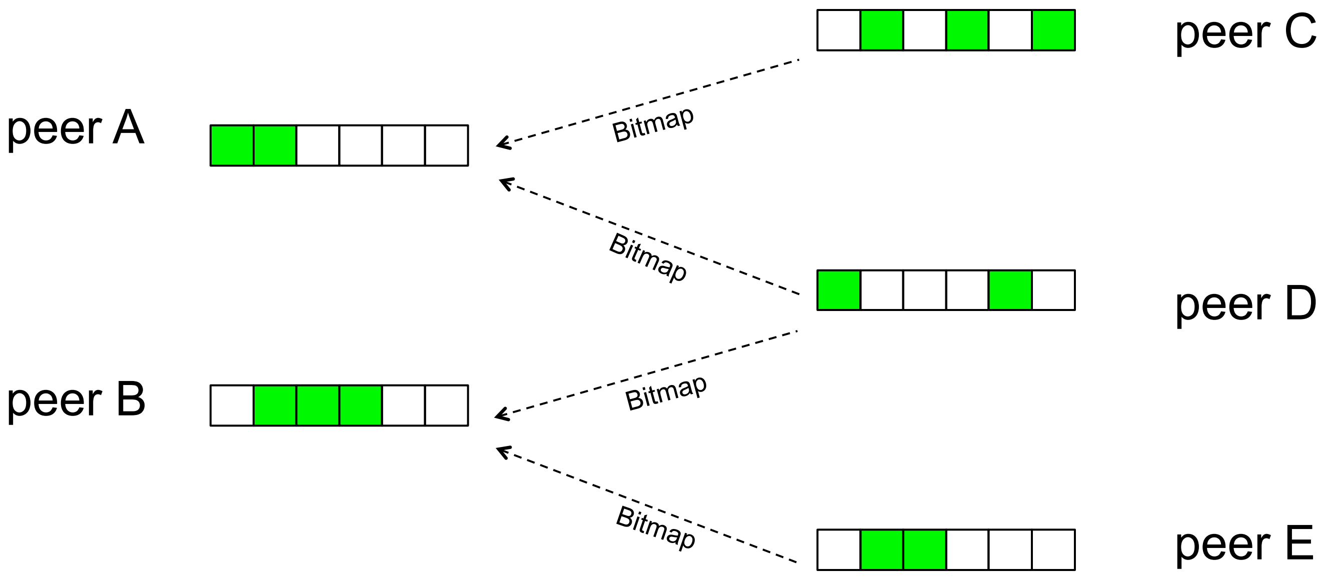
One more example



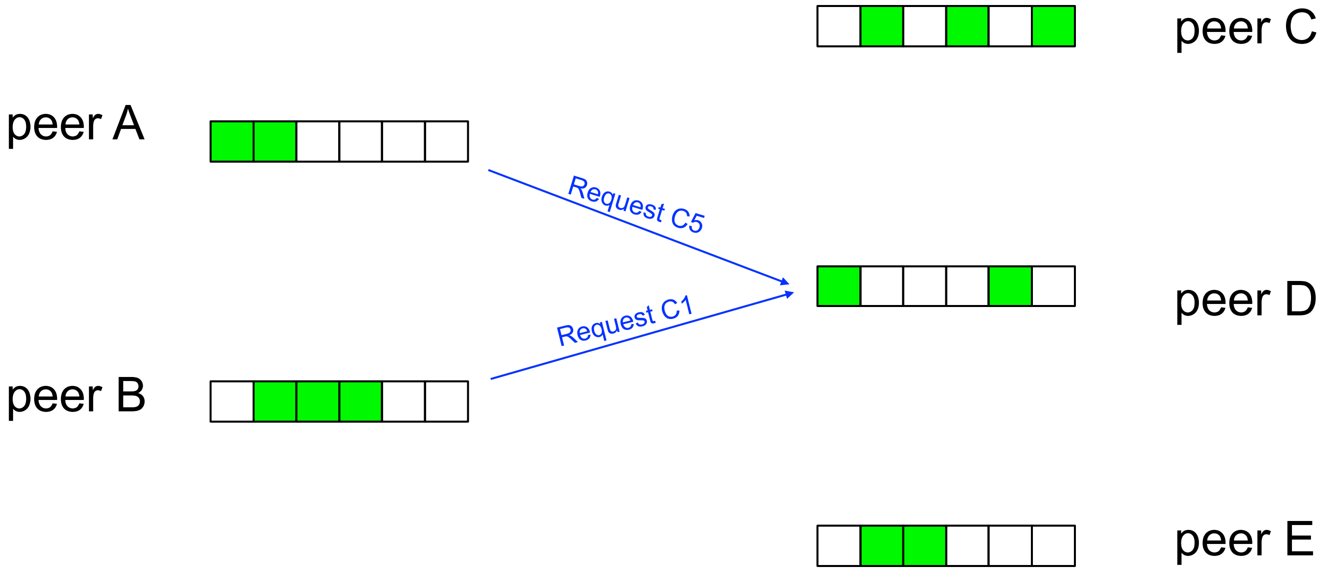
One more example



One more example

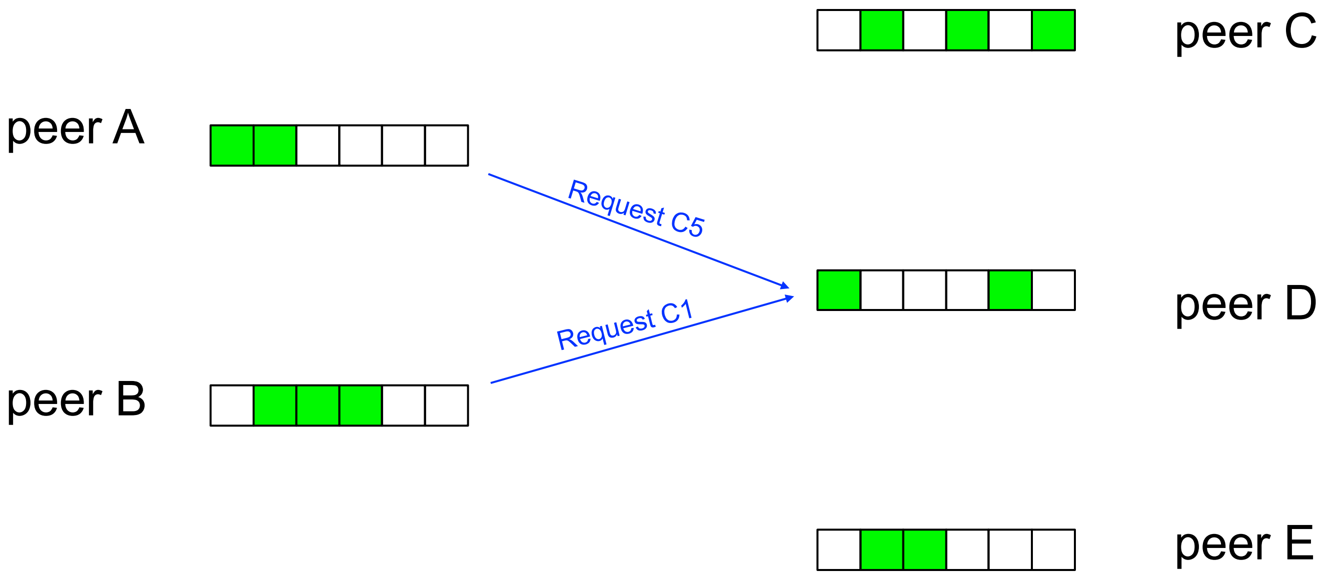


One more example



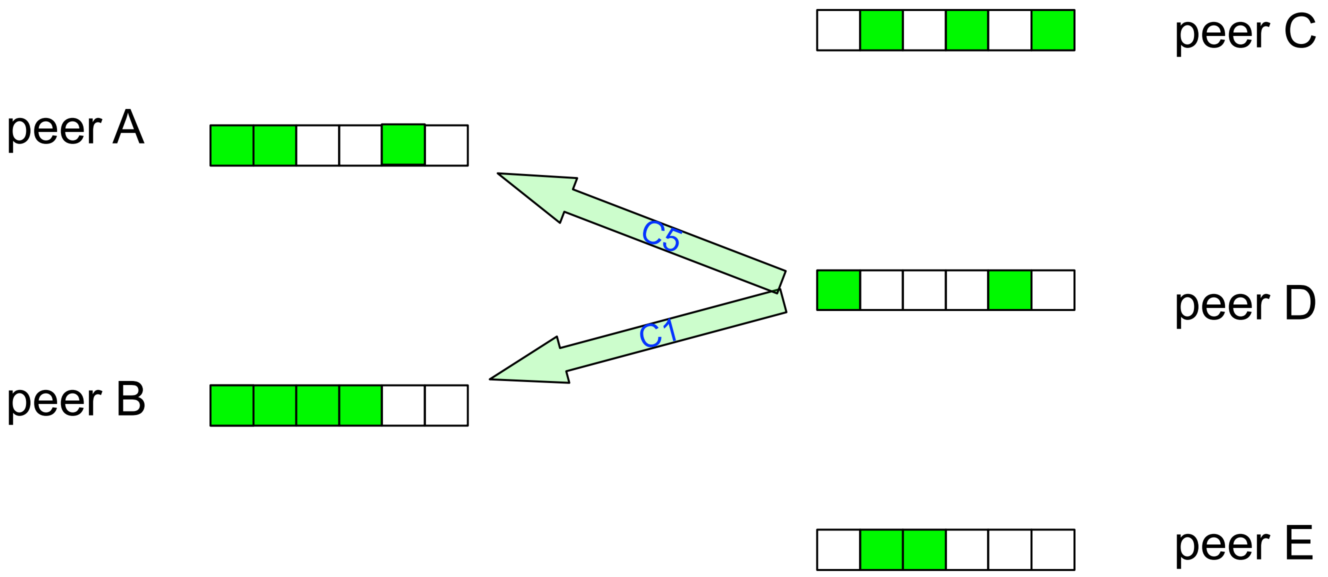
One more example

Without upload constraint



One more example

Without upload constraint



peer C

peer A

peer D

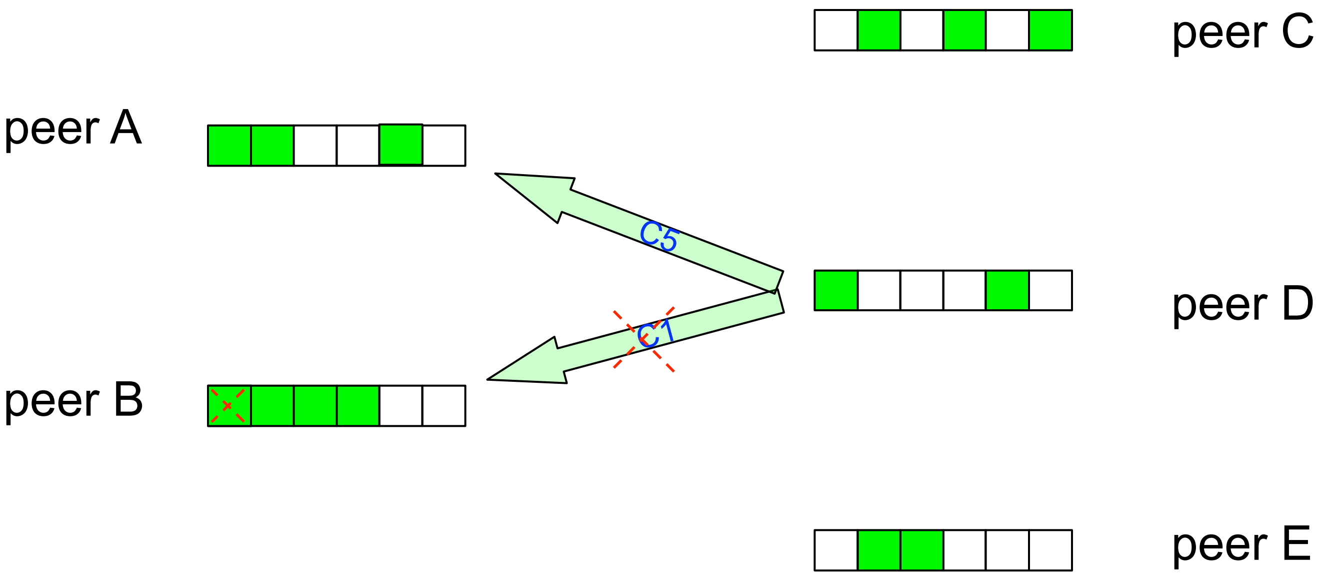
peer B

peer E

One more example

Without upload constraint

With upload constraint



peer C

peer A

peer D

peer B

peer E

Trackerless torrents

BitTorrent also supports "trackerless" torrents, featuring a DHT implementation that allows the client to download torrents that have been created without using a BitTorrent tracker.