BitTorrent in Detail

Roman Dunaytsev

roman.dunaytsev@spbgut.ru

A Bit of History

- BitTorrent was developed by Bram Cohen in 2001
  - Nowadays, it is maintained by Cohen's company, BitTorrent Inc.
  - https://www.bittorrent.com/

A Bit of History (cont'd)

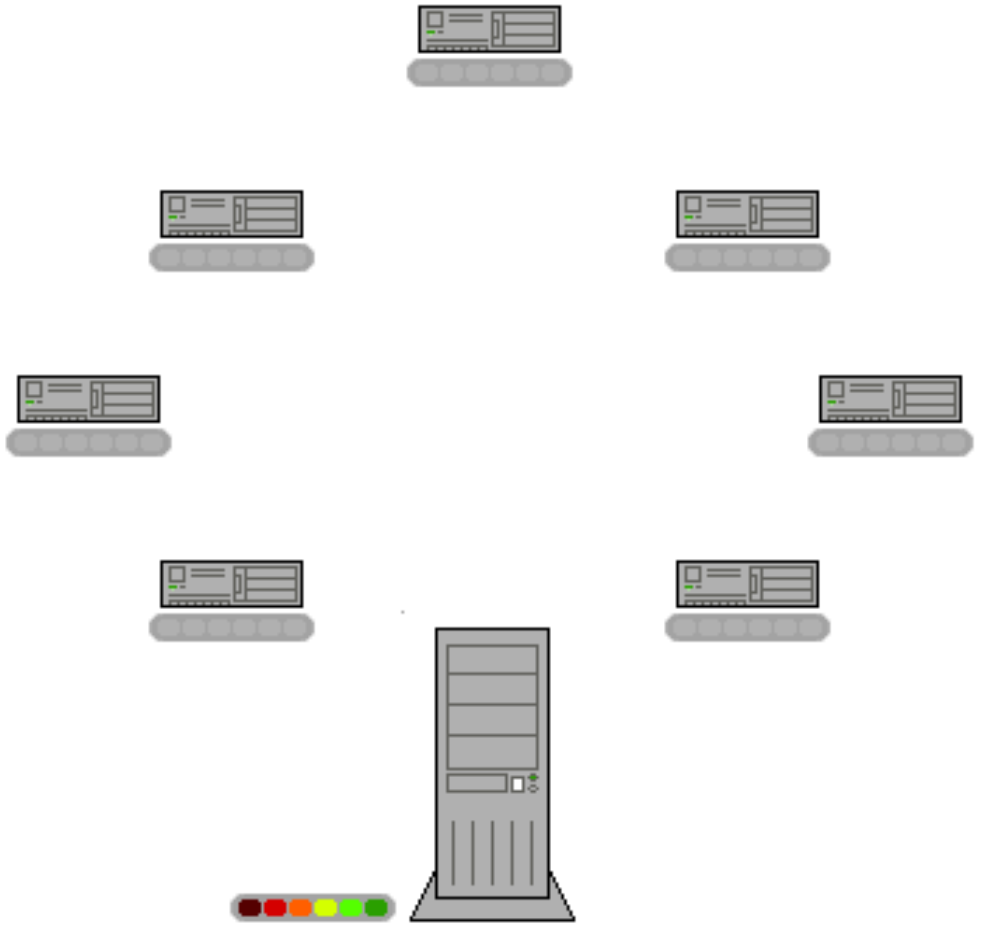
- Several possibilities for why file sharing is changing and increasing:
  - https://www.sandvine.com/blog/global-internet-phenomena-preview-file-sharing-reverses-a-downward-trend
  - More sources than ever are producing "exclusive" content available on a single streaming or broadcast service (e.g., Game of Thrones for HBO). To get access to all of these services, it gets very expensive for a consumer, so they subscribe to one or two and pirate the rest
  - Many of these exclusive series are US-based, and do not have good distribution internationally, so people download it because they have no access to the content

Content Distribution

- Problem:
  - The distribution of a large piece of static content, from a limited source, to a very large number of users, as fast as possible
  - Providing the necessary upload bandwidth at the source is expensive

- Solution:
  - Use the upload capacity of the downloaders
  - Create opportunities for data exchange between downloaders

Content Distribution (cont’d)



Content Distribution (cont'd)

- BitTorrent strategy:
  - Total download = total upload
  - To avoid free riders, try to make the download rate proportional to the upload rate for each peer
  - To provide good robustness, create a random graph between peers
  - To achieve Pareto efficiency, use tit-for-tat

- Pareto efficiency – a state of allocation of resources from which it is impossible to reallocate so as to make any one individual or preference criterion better off without making at least one individual or preference criterion worse off

Content Distribution (cont'd)

- https://en.wikipedia.org/wiki/Prisoner%27s_dilemma
  - Prisoner's dilemma
  - Iterated prisoner's dilemma
  - Tit for tat
  - Tit for tat with forgiveness

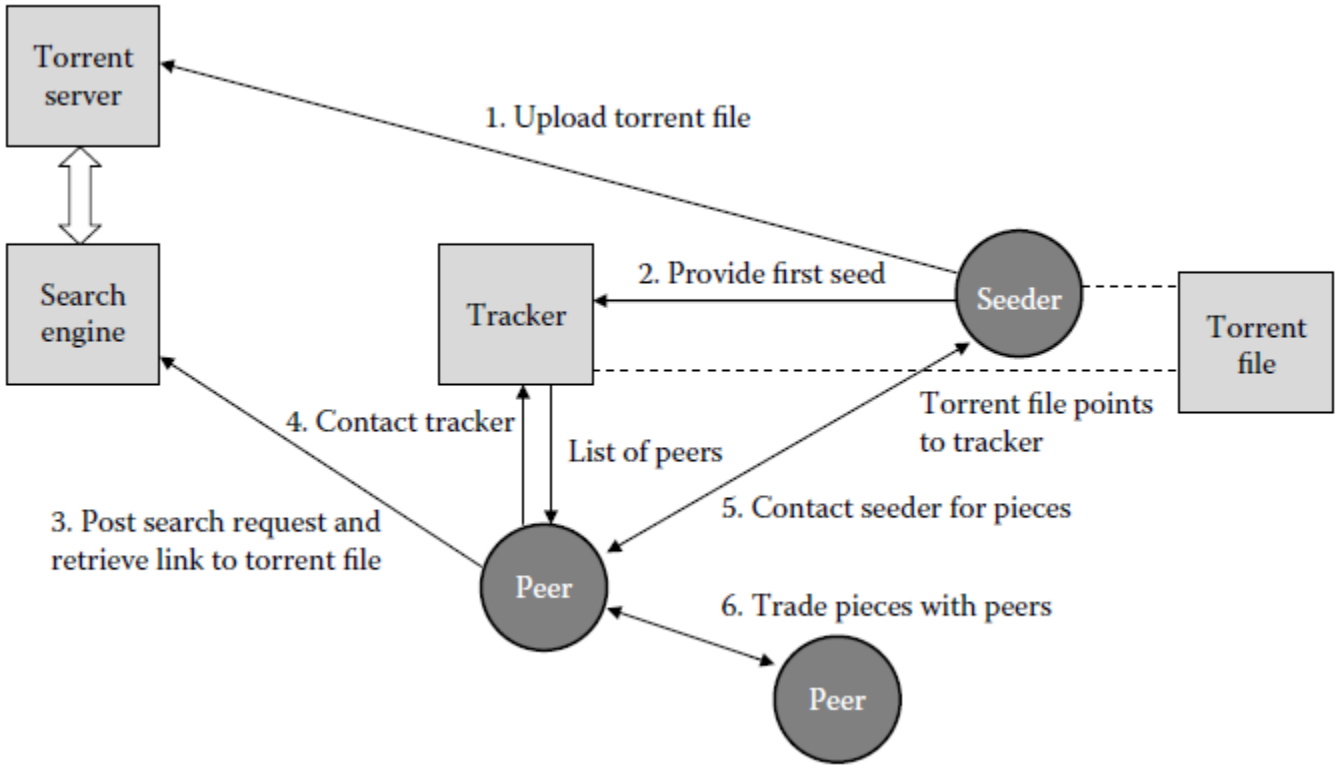
Prisoner B / Prisoner A	**Prisoner B stays silent (*cooperates*)**	**Prisoner B betrays (*defects*)**
Prisoner A stays silent (*cooperates*)	Each serves 1 year	Prisoner A: 3 years Prisoner B: goes free
Prisoner A betrays (*defects*)	Prisoner A: goes free Prisoner B: 3 years	Each serves 2 years

Content Distribution (cont'd)

- https://en.wikipedia.org/wiki/Tit_for_tat
- https://en.wikipedia.org/wiki/Nice_Guys_Finish_First

- Best deterministic strategy for the iterated prisoner's dilemma:
  - On the first move cooperate
  - On each succeeding move do what your opponent did the previous move
  - Be prepared to forgive after carrying out just one act of retaliation
    - This is a recovery mechanism to ensure eventual cooperation

Content Distribution (cont'd)

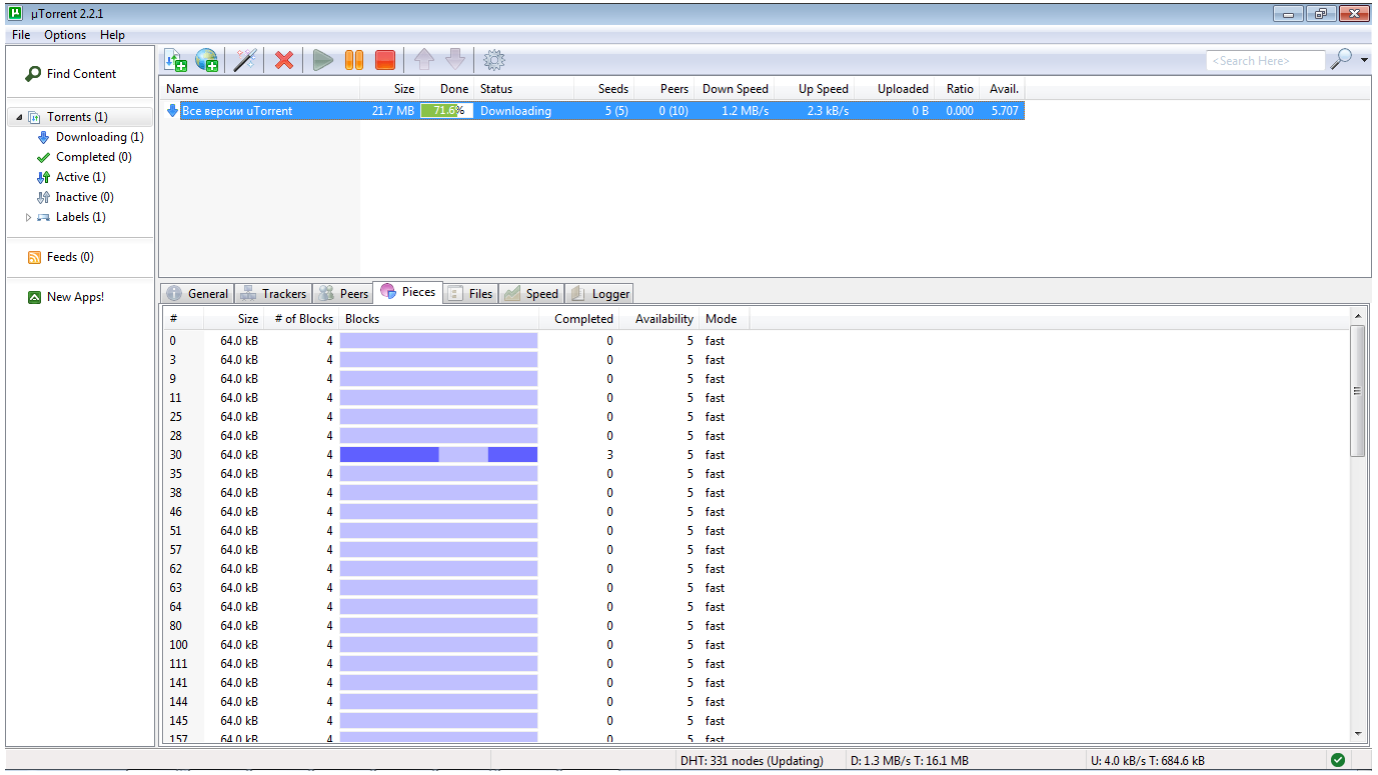


Content Distribution (cont'd)

- If a P2P network has a large proportion of **free riders** (aka **freeloaders**), it will defeat the objective to share workload
- **Public** tracker sites (e.g., Pirate Bay)
  - Allow users to search in and download from their collection of torrent files; users can typically also upload torrent files for content they wish to distribute
- **Private** tracker sites (e.g., Demonoid)
  - Operate like public ones except that they restrict access to registered users and keep track of the amount of data each user uploads and downloads, in an attempt to reduce **leeching**

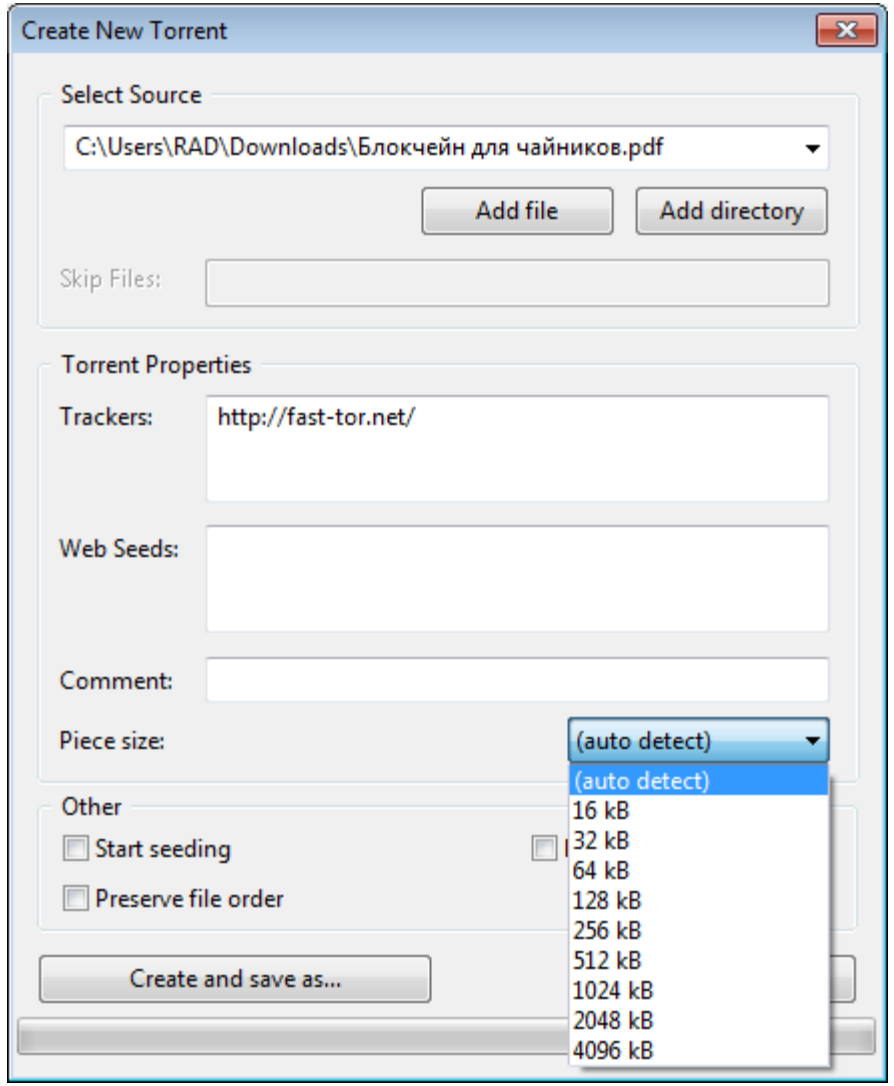
Creating and Publishing Torrents

- Content is split into **pieces** and pieces are split into **blocks** (16 kB)



Creating and Publishing Torrents (cont'd)

- File > Create New Torrent…
  - Блокчейн для чайников.pdf = 114 MB
  - Piece size 16 kB > .torrent = 144 kB
  - Piece size 4096 kB > .torrent = 4 kB
  - Piece size auto detect > .torrent = 20 kB



Creating and Publishing Torrents (cont'd)

- BitTorrent uses the SHA-1 hash function to determine which pieces of the file are good and which are bad

- Piece size
  - Selecting too small a piece size for large files creates a needlessly large torrent file and extreme protocol overhead
  - On the other hand, using too large a piece size means more wasted bandwidth for users who often experience hashfails, since they would have to redownload entire pieces for each hashfail that occurs
  - **Rule of thumb**: Avoid very large piece sizes for small files; likewise avoid small piece sizes for very large files

Creating and Publishing Torrents (cont’d)

- Split content into pieces, compute hashes for each piece, and create a metadata torrent file
- Register the torrent with a tracker
- Start the BitTorrent client acting as **seed**
- Publish the torrent file on a Web server

Creating and Publishing Torrents (cont'd)

- Torrent files are encoded using **bencoding**
  - Pronounced like "B-encode"
  - http://bittorrent.org/beps/bep_0003.html

- A torrent file contains:
  - File name
  - Its length in bytes
  - SHA-1 hashes for all pieces
  - URL of the tracker
  - Optional fields

Creating and Publishing Torrents (cont’d)

- <?xml version="1.0" encoding="UTF-8"?>
- <tor:TORRENT
- xmlns:tor=http://azureus.sourceforge.net/files
- xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
- xsi:schemaLocation="http://azureus.sourceforge.net/files http://azureus.sourceforge.net/files/torrent.xsd">
- <ANNOUNCE_URL>http://torrent.opera.com:6969/announce</ANNOUNCE_URL>
- <CREATION_DATE>1281694453</CREATION_DATE>
- <TORRENT_HASH>2EEAB52F0242C42AC788D17924CCC197DC3D2F0C</TORRENT_HASH>
- <INFO>
- <NAME encoding="utf8">Opera_1061_en_Setup.exe</NAME>
- <PIECE_LENGTH>262144</PIECE_LENGTH>
- <LENGTH>10809256</LENGTH>
- <PIECES>
- <BYTES>54E6CCEC9EB3BEA12711FB9F383890023650D5AF</BYTES>
- <BYTES>D134F302346EFB60E57BA240F8F38BD27A408C35</BYTES>
- <BYTES>73811C5D1313F5A16A18BFF77DCAD18CADFB3F97</BYTES>
- .
- <BYTES>6AE8353393A66804FF95C98624331824A6F61D62</BYTES>
- </PIECES>
- </INFO>
- </tor:TORRENT>

Joining a Swarm

- Downloaders find the torrent file
- A BitTorrent download is started by opening the torrent file in the BitTorrent client
- The tracker keeps a log of peers that are currently downloading a file, and helps them find each other
- The tracker is not directly involved in the transfer of data and does not have a copy of the file
  - The tracker and the downloading users exchange information over HTTP
  - Peers communicate with each other over TCP or uTP (LEDBAT)

Joining a Swarm (cont'd)

- First, the user gives information to the tracker about which file it is downloading, ports it is listening on, etc.
- Peers interested for the same torrent form an independent mesh **overlay network** where each peer work together for a common target to complete the file download as fast as possible
  - This group of peers that all share the same torrent represents a **swarm**
- The tracker keeps a log of peers that are currently downloading a file, and helps them find each other
- Once a peer has the entire file, it may (selfishly) leave or (altruistically) remain as a seed

Joining a Swarm (cont'd)

- Each peer has a **neighbor set** of other peers
  - Initially retrieved from the tracker
  - Maximum size of the neighbor set is typically 80
- At all times, a peer uploads data to no more than 4 neighbor peers, it is an **active neighbor set**
- After establishing a connection, peers shake hands and exchange their piece **bitfields**
  - After the bitfield exchange, both peers know what pieces the other peer has

Joining a Swarm (cont'd)

- Peer A is **interested** in peer B, if peer B has pieces that peer A does not have
- Peer A is **not interested** in peer B, if peer B has a subset of the pieces that peer A has
- When a peer acquires a new piece, it tells all its neighbors by sending them a **have** message

BitTorrent Algorithms

- In many other P2P file sharing protocols the swapping of files happens one-to-one, meaning that the user himself chooses another peer to download from
- The concept of BitTorrent is to be able to download from many other peers simultaneously
- This requires a way of knowing which peers to download what pieces of the file from, with the goal of receiving the complete file as quickly as possible
- Plus it needs to happen without the involvement of the end user

BitTorrent Algorithms (cont'd)

- Choosing peers to connect to is a two-sided problem
- First, you need a way of finding the best sequence of downloading the pieces
  - This is determined by the **piece selection algorithm**
- Second, a peer, who has the piece you want, might not let you download it
  - Strategies for peers not allowing other peers to download from them is known as choking, and concerns resource allocation
  - This is determined by the **choking algorithm**

Piece Selection Algorithm

- How BitTorrent selects what pieces of the file to download have great impacts on the performance of the protocol

- The goal is to replicate different pieces on different peers as soon as possible
  - This will increase the download speed, and also make sure that all pieces of a file is somewhere in the network if the seeder leaves
  - In case of TCP, it is thus crucial to always transfer data or else the transfer rate will drop because of the **slow start** mechanism

Piece Selection Algorithm (cont'd)

- **Policy 1: Strict Policy**
- All pieces are further broken into sub-pieces (blocks)
- Once a block has been requested, the remaining blocks for that particular piece are requested before blocks from any other piece
- This helps getting a complete piece as quickly as possible

- Moreover, BitTorrent always keeps some number, typically 5, requests for blocks pipelined at once (aka **pipelining**)
- Every time a block arrives a new request for another block is sent

Piece Selection Algorithm (cont'd)

- **Policy 2: Rarest First**
- This means that when a peer selects the next piece to download, it selects the piece which the fewest of their peers have

- Spreading the seed:
  - In the beginning, the seed will be a bottleneck since it is the only one with any piece of the file
  - A downloader can see what pieces their peers have, and the "rarest first"-policy will result in that the pieces fetched from the seed are pieces which have not already been uploaded by others

Piece Selection Algorithm (cont'd)

- Increased download speed:
  - The more peers that have the piece, the faster the download can happen, as it is possible to download sub-pieces from different places
  - We want to replicate rare pieces so they can be downloaded faster

- Enabling uploading:
  - A rare piece is most wanted by other peers, and by getting a rare piece others will be interested in uploading from you

Piece Selection Algorithm (cont'd)

- Most common last:
  - It is sensible to leave the most common pieces to the end of the download
  - Since many peers have it, the probability of being able to download them later is much larger than that of the rare pieces

- Prevent rarest piece missing:
  - When the seed is taken down, it is important that all the different pieces of the file are distributed somewhere among the remaining peers
  - By replicating the rarest pieces first, we reduce the risk of missing one or more pieces of a file when the seeder leaves

Piece Selection Algorithm (cont'd)

- **Policy 3: Random First Piece**
  - Once you start downloading, you don't have anything to upload
  - It is important to get the first piece as fast as possible, and this means that the "rarest first"-policy is not the most efficient
  - Rare pieces tend to be downloaded slower, because you can download its blocks from only one (or maybe a few) other peers
  - The policy is then to select the first piece randomly
  - When the first piece is complete, change to "rarest first"

Piece Selection Algorithm (cont'd)

- **Policy 4: Endgame Mode**
- When a download is almost complete, there is a tendency for the last few blocks to trickle in slowly
- To speed this up, the client sends requests for all of its missing blocks to all of its peers
- The client should send a cancel to everyone else every time a block arrives
- When to enter end game mode is an area of discussion

Choking Algorithm

- No centralized resource allocation exists in BitTorrent
- Every peer is responsible for maximizing its download rate
- A peer will try to download from whoever they can
- To decide which peers to upload to, a peer uses a variant of the “tit-for-tat” strategy

Choking Algorithm (cont'd)

- **Choking** – a temporary refusal to upload to another peer, but you can still download from him/her
- To cooperate peers allow uploading, and to not cooperate they "choke" the connection to their peers
- The idea is to upload to peers who have uploaded to you recently
- The goal is to have several bidirectional connections at any time, and achieve "Pareto efficiency"

Choking Algorithm (cont'd)

- The big question is how to determine which peers to choke and which to unchoke
- A peer always unchokes a fixed number of its peers (the default is 4)
- Deciding which peers to unchoke is determined only by the current download rates
- The result is that any peer will upload to peers which provide the best download rates
- This also means that you can get a higher download rate if you have many uploads

Choking Algorithm (cont'd)

- **Optimistic Unchoking**
- Simply uploading to the peers which provide the best download rates would suffer from having no method of discovering if currently unused connections are better than the ones being used
- To fix this, at any one time there is a single peer which is unchoked regardless of its upload rate
  - Which peer is the optimistic unchoke is rotated every 30 seconds
- If this new connection turns out to be better than one of the existing unchoked connections, it will replace it

Choking Algorithm (cont'd)

- **Anti-snubbing**
- What happens if a client suddenly is choked by all peers it was downloading from?
  - It then has to find new peers, but the optimistic unchoking mechanism only checks 1 unused connection every 30 seconds
- If a client hasn't got anything from a particular peer for 60 seconds, it presumes that is has been "snubbed" and doesn't upload to it
- It will then increase the number of optimistic unchokes in order to try to find new connections quicker

Choking Algorithm (cont'd)

- **Upload Only**
- Once a peer is done downloading, it no longer has useful download rates to decide which peers to upload to
- Then the question is, which nodes to upload to?
- When a download is completed, a peer uses a new choking algorithm which unchokes the peers with the highest upload rates
- This ensures that pieces get uploaded faster and that they get replicated faster