

# A guest today, and a guest Friday

- Later today, Simon Christensen from ORBIT Lab will present the lab and highlight, how you can benefit from it
- Friday, Hans-Jörg Schulz will present how visualisation can help us cope with the vast data generated by IoT

# BitTorrent



Mads Darø Kristensen  
Niels Olof Bouvin

# Overview

- **BitTorrent terms**
- **The BitTorrent protocol**
- **The life of a torrent**
- **Attacking BitTorrent**

# BitTorrent terms

- **The BitTorrent protocol operates with these important terms:**
  - Tracker: a centralised component used for peer discovery.
  - Seeds: peers that have fully downloaded the file being shared.
  - Leechers: peers that are actively downloading the file.
  - Swarm: the collection of peers participating in sharing the torrent data.
  - .torrent file: a meta data file containing information about the torrent.

# Tracker

- **The tracker is the only centralised component in BitTorrent.**
- **It is used to bootstrap the system by providing peer discovery.**
  - The tracker thus does no heavy lifting at all. It is never involved in transferring any of the data that is shared in the torrents it provides access to.
    - ... which is probably also why varying tracker sites have claimed to be innocent when faced with infringement suits ;-)
  - Peer selection is done completely at random—there is no weighing of peers or peer capabilities.

# Seeders

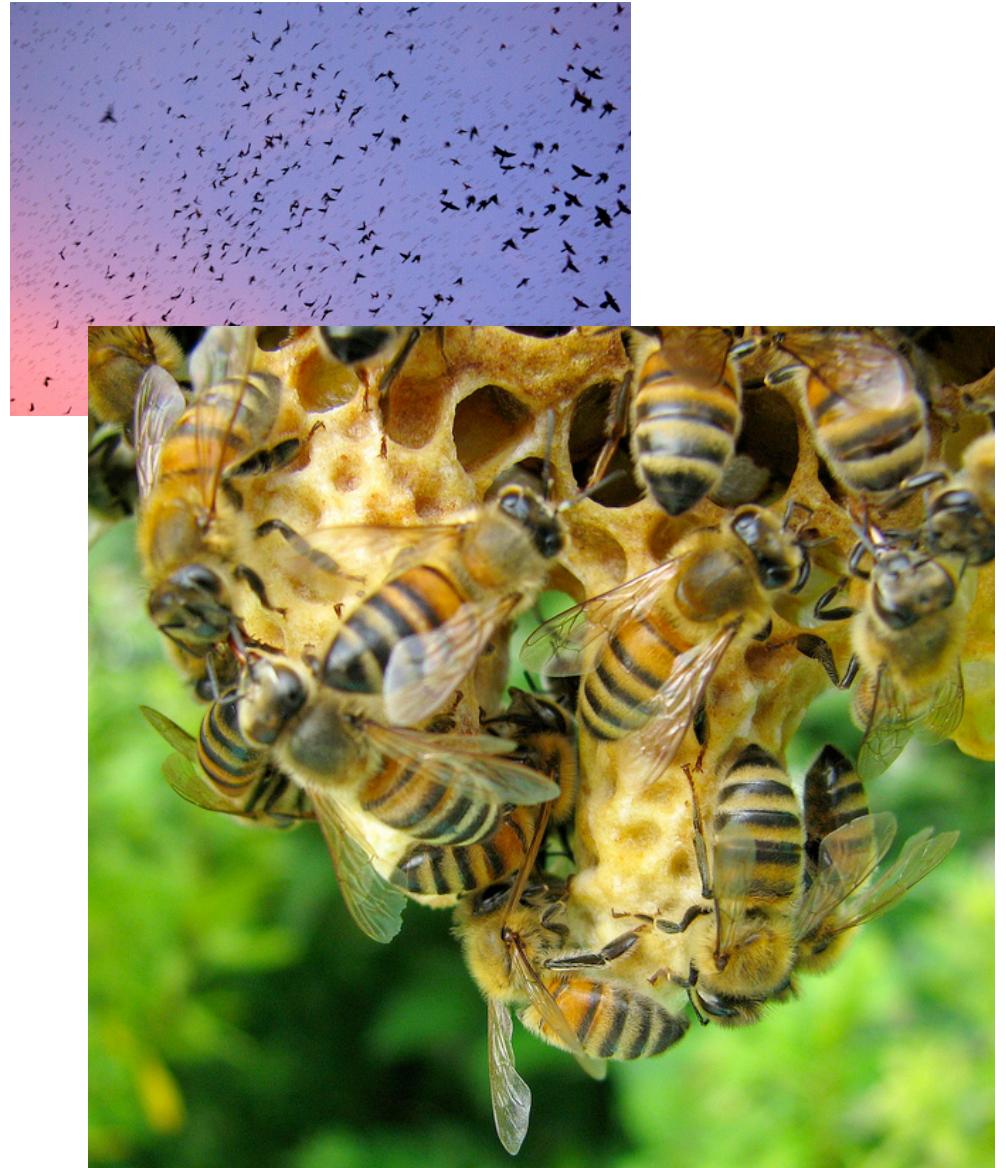
- A seeder is a peer that has the entire file being served.
- Initially, when a torrent is initiated, a single seeder connects to the tracker to make its content available.
- While the torrent swarm is active, peers will change from leechers to seeders when they finish downloading the torrent.
  - Which also means that it is good practice to leave the BitTorrent client on for a while after downloading finishes, so that you get to contribute to the swarm.

# Leechers

- A leecher is a peer that is actively downloading the torrent.
- Being a leecher does not mean that the peer contributes nothing to the swarm.
  - All leechers must serve the pieces that they have already finished to the swarm.

# Swarm

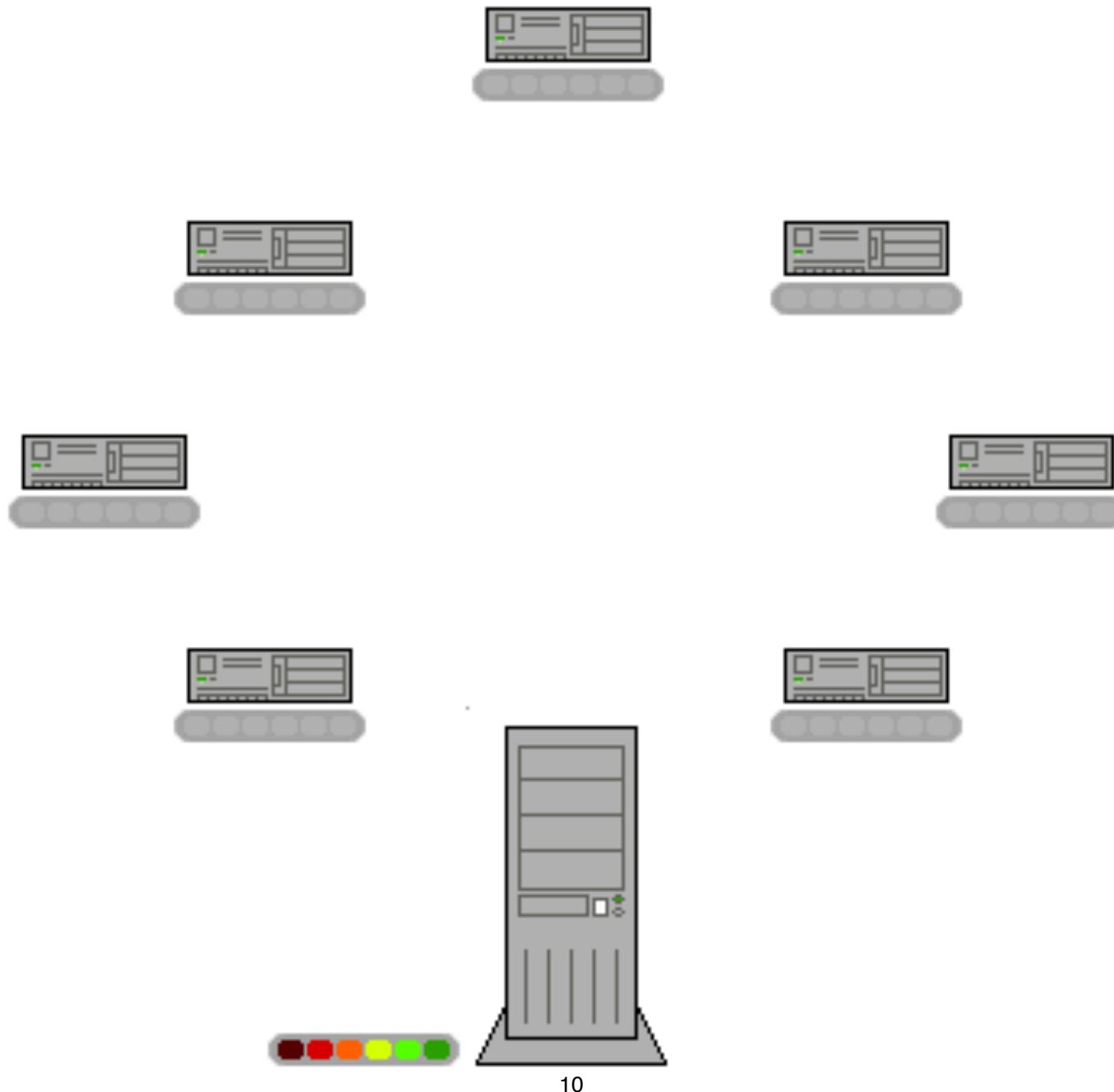
- The swarm is all of the peers currently participating in the torrent
  - The swarm may be huge, so most peers only deal with a small subset of the swarm—their personal *peer set*.



# .torrent files

- The .torrent file describes a given torrent.
- It contains information about the tracker(s) coordinating the torrent, as well as some meta information about the file being shared.
- The .torrent file is distributed “offline” (i.e., outside of the BitTorrent system).
  - Typically it is hosted on a webpage (or send around to peers in an email).

# A BitTorrent animation



# Overview

- BitTorrent terms
- **The BitTorrent protocol**
- The life of a torrent
- Attacking BitTorrent

# The BitTorrent protocol

- In the following I will explain the basics of the BitTorrent protocol.
  - For a more in-depth introduction to the nitty gritty details see
    - [http://bittorrent.org/beps/bep\\_0003.html](http://bittorrent.org/beps/bep_0003.html)
    - <http://wiki.theory.org/BitTorrentSpecification>

# The contents of a .torrent file

- When a peer wishes to download a file, it retrieves the .torrent file.
- A .torrent file is a *bencoded* Python dictionary containing (at least) the keys announce and info.
  - **announce** is the URL of the tracker.
  - Where **info** is another dict containing the following keys:
    - **name**: the suggested file (or directory) name of the shared file.
    - **piece length**: the length in bytes of the individual pieces.
    - **pieces**: one big string containing the SHA1 hashes of all pieces.
    - **length**: the total length of the file being shared.

# Sharing directories

- It is also possible to share an entire directory using BitTorrent.
- In this case the length field is exchanged for a files field containing a list of files with information about the length and path of each file.
- For the purposes of the other keys, the multi-file case is treated as only having a single file by concatenating the files in the order they appear in the files list.

# An example .torrent file

- This .torrent was retrieved from Ubuntu's homepage.  
It has been parsed—the native format is bencoded.



A screenshot of a Mac OS X TextEdit window titled "parsed\_torrent.txt". The window contains the following bencoded data:

```
comment -> Ubuntu CD releases.ubuntu.com
announce -> http://torrent.ubuntu.com:6969/announce
info ->
    length -> 726827008
    piece length -> 524288
    name -> ubuntu-10.10-desktop-i386.iso
    pieces -> 'SHA1 hashes of pieces...'
creation date -> 1286702723
announce-list ->
    'http://torrent.ubuntu.com:6969/announce'
    'http://ipv6.torrent.ubuntu.com:6969/announce'
```

The status bar at the bottom of the window shows "U:--- parsed\_torrent.txt All (12,0) SVN-747 (Text Fly yas WordWrap)" and "Beginning of buffer".

# Working with the tracker

- After retrieving the .torrent file, the peer contacts the tracker listed in that file.
- The tracker responds by returning a list of (~50) randomly chosen peers in the swarm.
- After that point in time the tracker is only rarely contacted:
  - Once every 30 minutes to show that the peer is active,
  - if running low on peers in the peer set,
  - and when leaving the swarm.

# The peer protocol

- After receiving a list of ~50 peers, the new peer proceeds to establish a TCP connection to ~30 of these peers.
- The peer thus enters into a neighbourhood of peers and starts adhering to the *peer protocol*.

# Spreading information about available pieces

- Initially, when a peer enters a new neighbourhood of the swarm (i.e., when it gets new neighbours) it sends a **bitfield** message to the new neighbours.
  - The bitfield message contains a space efficient representation of the pieces that the peer holds (a bitmap)
    - If the peer has the piece at index  $x$  the  $x$ 'th bit is set to one
    - ... and if it hasn't got it the bit is set to zero
- When a peer finished downloading a piece (and the SHA1 sum matches) it sends a **have** message to all its neighbours, telling them that the new piece has been fetched.

# Downloading

- Peers may then start downloading pieces from each other.
  - They know which peers have got pieces that they are interested in...
- But peers are not allowed to download pieces willy nilly. BitTorrent is a tit-for-tat protocol, meaning that you have to give in order to receive.
- Once a peer is allowed to fetch a given piece it does so by sending the piece message with the index of the piece as an argument.

# Downloading

- **Each peer in a peer's neighbour list has two state bits:**
  - **interested/uninterested:** this bit tells us whether the neighbour is interested in the pieces we have got.
  - **choked/unchoked:** this bit states whether we are currently choking the neighbour.
- **Choking a peer means disallowing it to download pieces at this point in time.**
- **Peers send choke, unchoke, interested, and not interested messages to each other in the peer protocol.**

# Choking

- **Choking works on a tit-for-tat basis:**
  - If we are currently downloading from a peer, we will unchoke that peer so that it may also download from us.
    - This means, that when selecting a peer to download from, we should prefer peers that are interested in us.
  - If a peer does not contribute (i.e., we are not able to download from it) we can choke it again.
- **Optimistic unchoke:**
  - One or more peers will be optimistically unchoked at all time. This role rotates every 30 seconds.
  - If an optimistically unchoked peer start contributing, it may stay unchoked.

# Choking

- Choked/unchoked state of neighbours is reconsidered every 10 seconds.
- At any point in time a peer should have a number of unchoked neighbours.
  - This is of course implementation specific...
    - Some implementations have a static value of 4, whereas others use the square root of the upload capacity in KB/s
- Replacing contributing peers
  - If an optimistic unchoke results in a peer that is performing better (yielding faster download rates), one of the currently unchoked peers will be replaced.

# Choking and seeders

- When seeding, tit-for-tat stops making sense
- A seeder works for the general good of the swarm
  - It wants to upload as much as possible to the swarm.
  - It thus prefers to unchoke peers to which it has a **high upload rate**.

# Piece distribution

- **Piece selection strategies are in use in BitTorrent to ensure that the swarm stays alive.**
- **A client may choose to simply select pieces at *random***
  - This means, that the different peers will (with high probability) possess different pieces of the file, meaning that they have something to contribute to the swarm
- **Another selection strategy is the *rarest first* strategy**
  - In this strategy peers request the pieces that are least distributed within their peer set.
  - This decreases the likelihood of the the torrent “breaking” when a peer leaves.
    - ... no peers will be holding “the only copy” of a piece for very long.

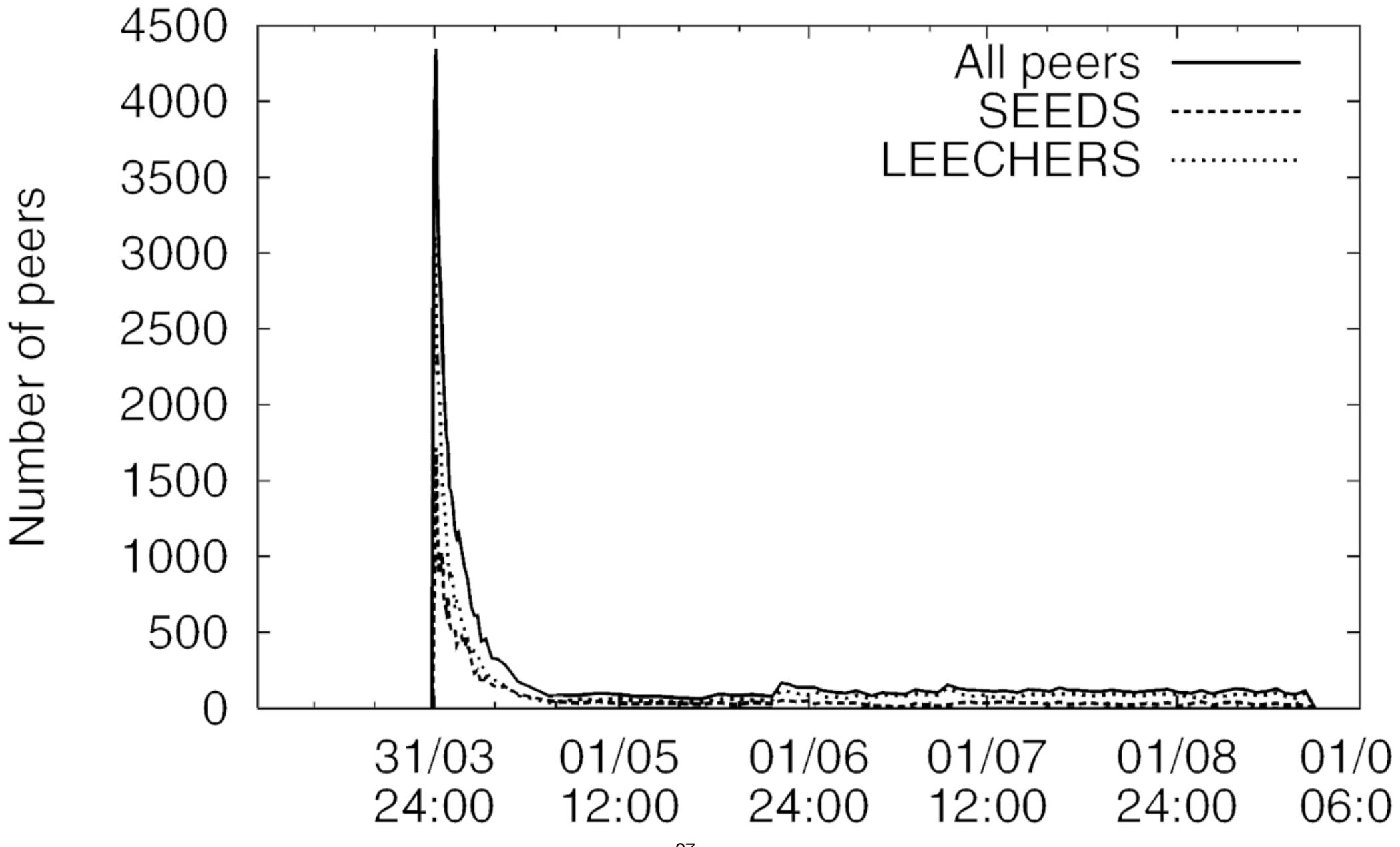
# Rarest first

- **Initially, a peer will request a randomly chosen piece.**
  - This is done in order to get started—the rarest pieces will be slightly harder to get at, since many peers are interested in them.
- **Then it will start adhering to the *rarest first* strategy:**
  - By looking at its bitfields it will calculate a set of the  $n$  rarest pieces and at random choose some pieces to download from that set.
    - This randomisation is done to balance the load so that all peers do not jump on the same least common piece.
- **In the end, when the peer only misses a few pieces, it may start downloading all of them in parallel.**
  - It is even allowed to download the same piece from two sources, but it is good form to notify the slowest of the two when download has succeeded from another source.

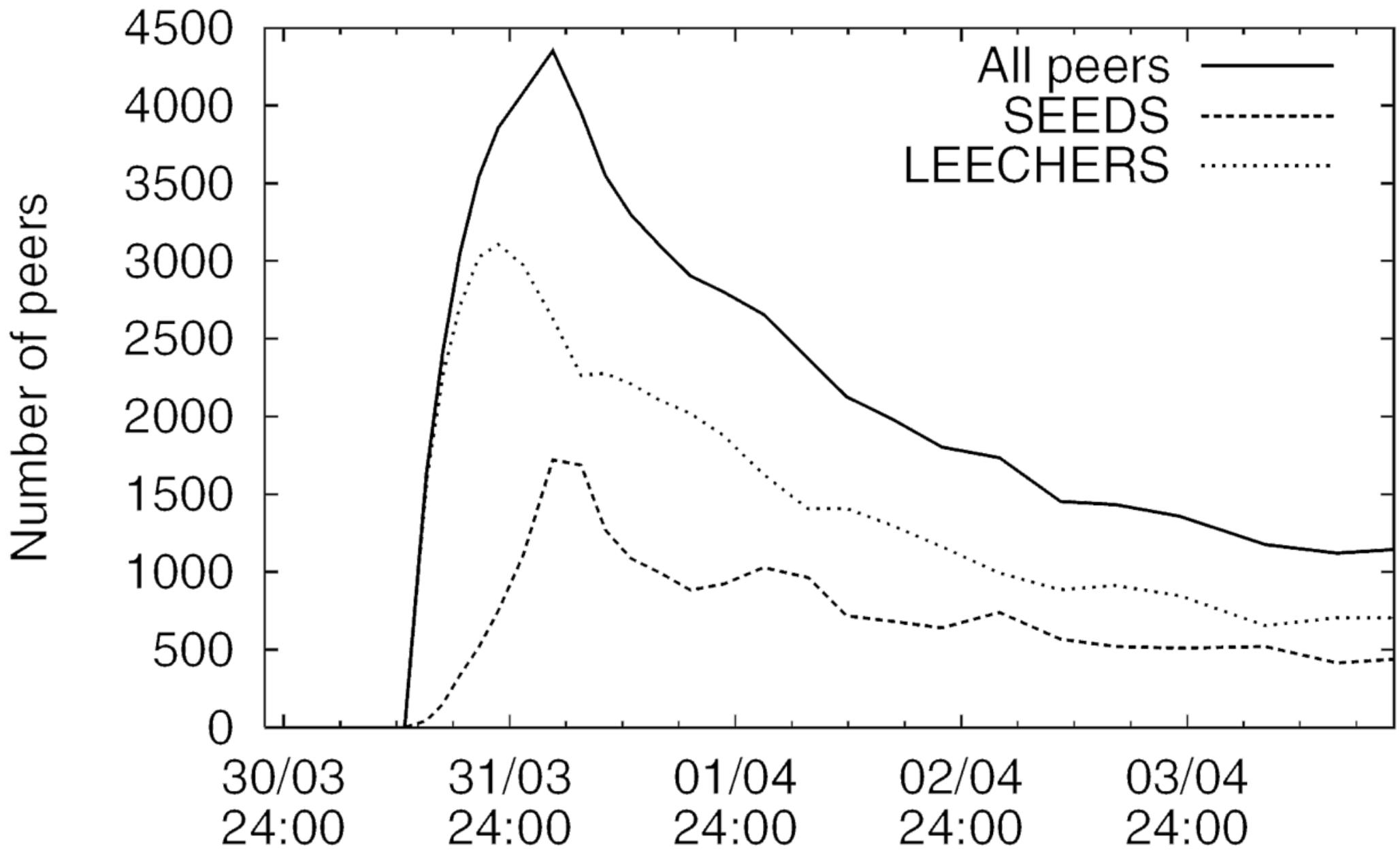
# Overview

- BitTorrent terms
- The BitTorrent protocol
- **The life of a torrent**
- Attacking BitTorrent

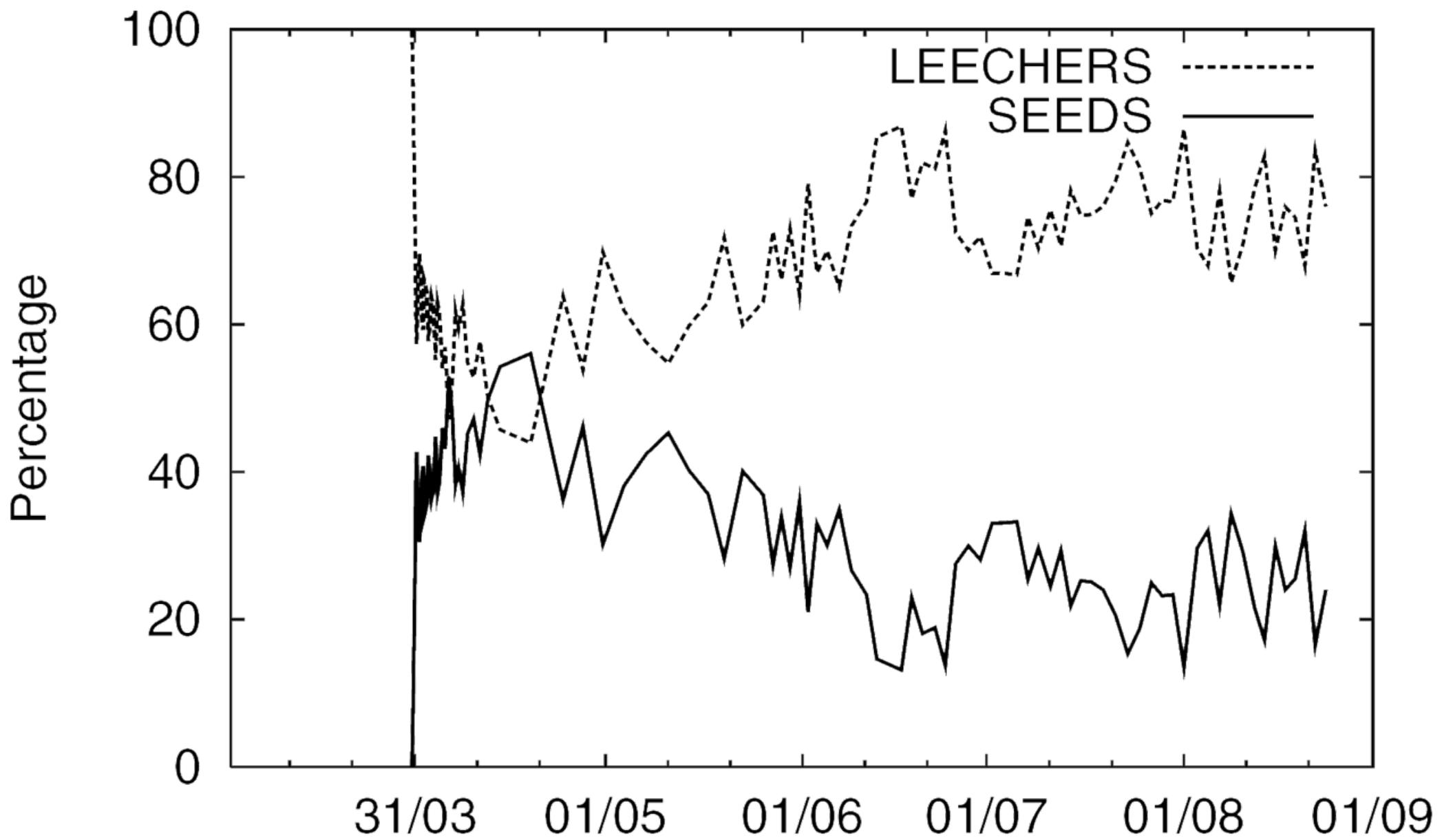
# The life of a (legal) torrent



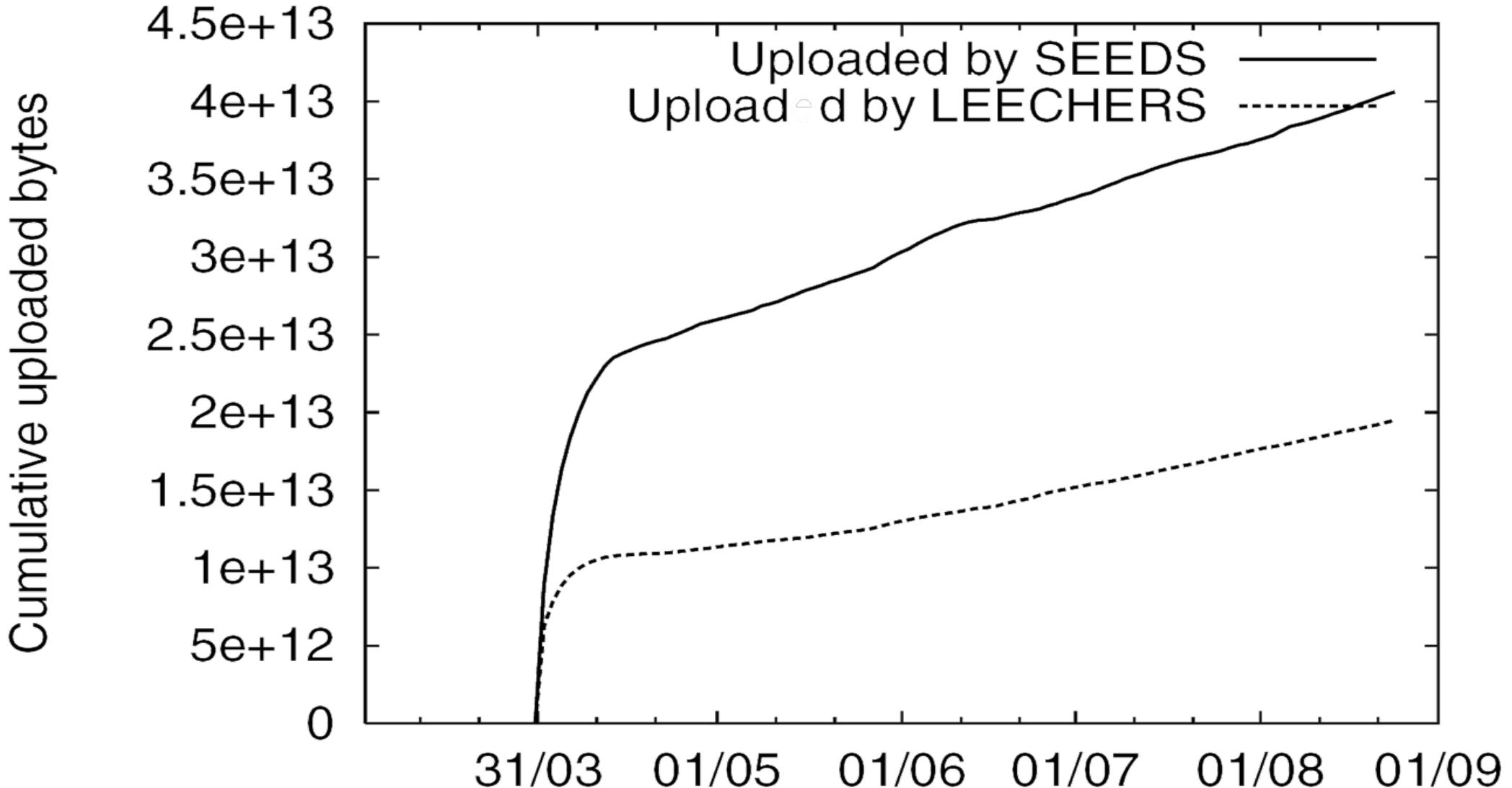
# The first few days



# Seeders vs. leechers



# Contributions by seeders and leechers



# Overview

- BitTorrent terms
- The BitTorrent protocol
- The life of a torrent
- **Attacking BitTorrent**

# Collaboration?

- **BitTorrent is great for collaborating peers.**
  - But can the protocol be subverted by malicious peers?
- **An “attack” on a BitTorrent may take on two forms:**
  - Harming the swarm; i.e., making it difficult for other peers to download the file.
  - Taking advantage of the swarm; i.e., (mis)using the protocol to ones own advantage.

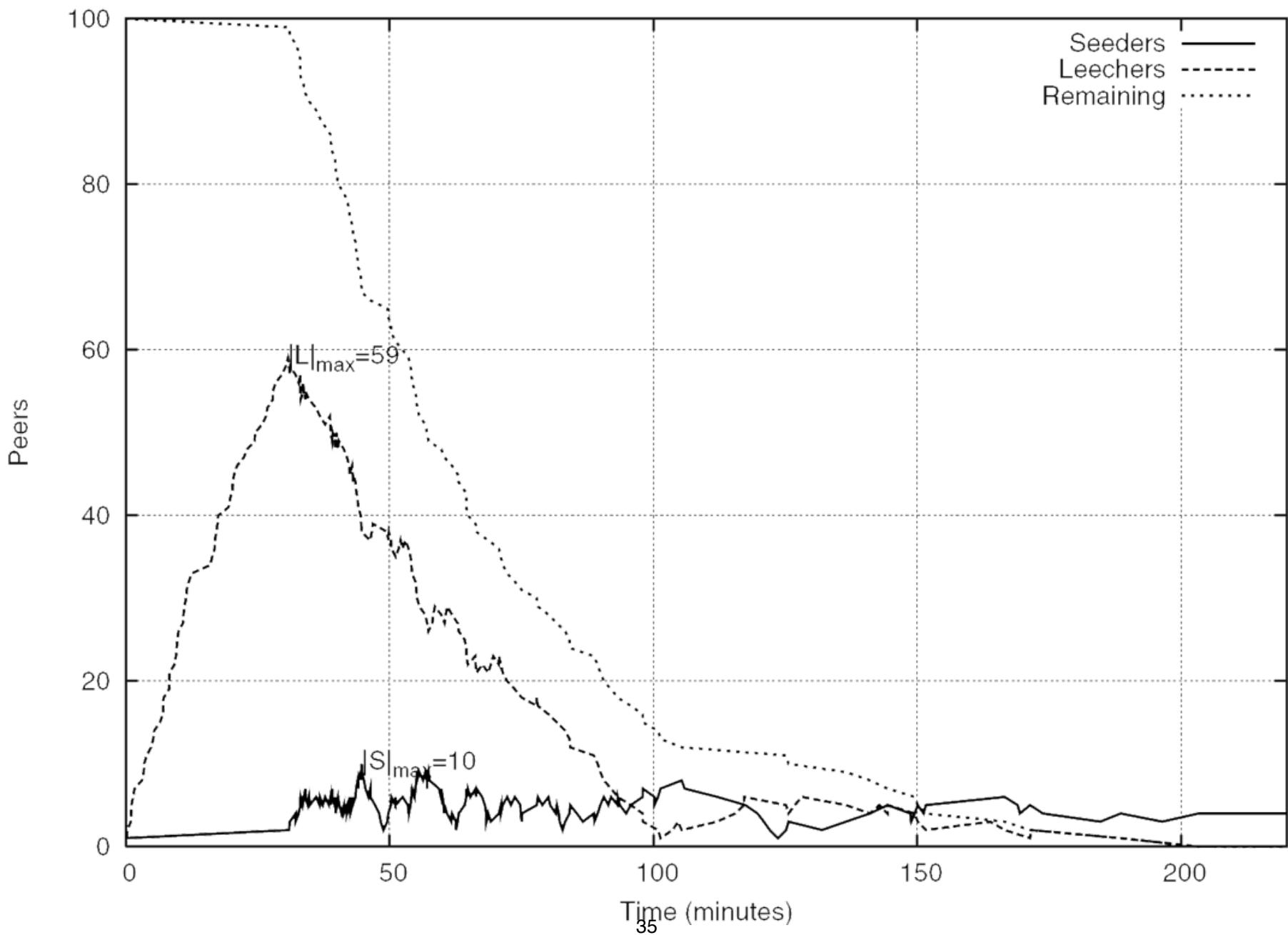
# Harming the swarm

- “**Attacking a Swarm with a Band of Liars: evaluating the impact of attacks on BitTorrent**” explores methods to poison a swarm.
  - ... and provide an excellent overview and analysis of BitTorrent.
- They mention two Sybil attacks on BitTorrent:
  - Piece lying
  - Eclipse attacks

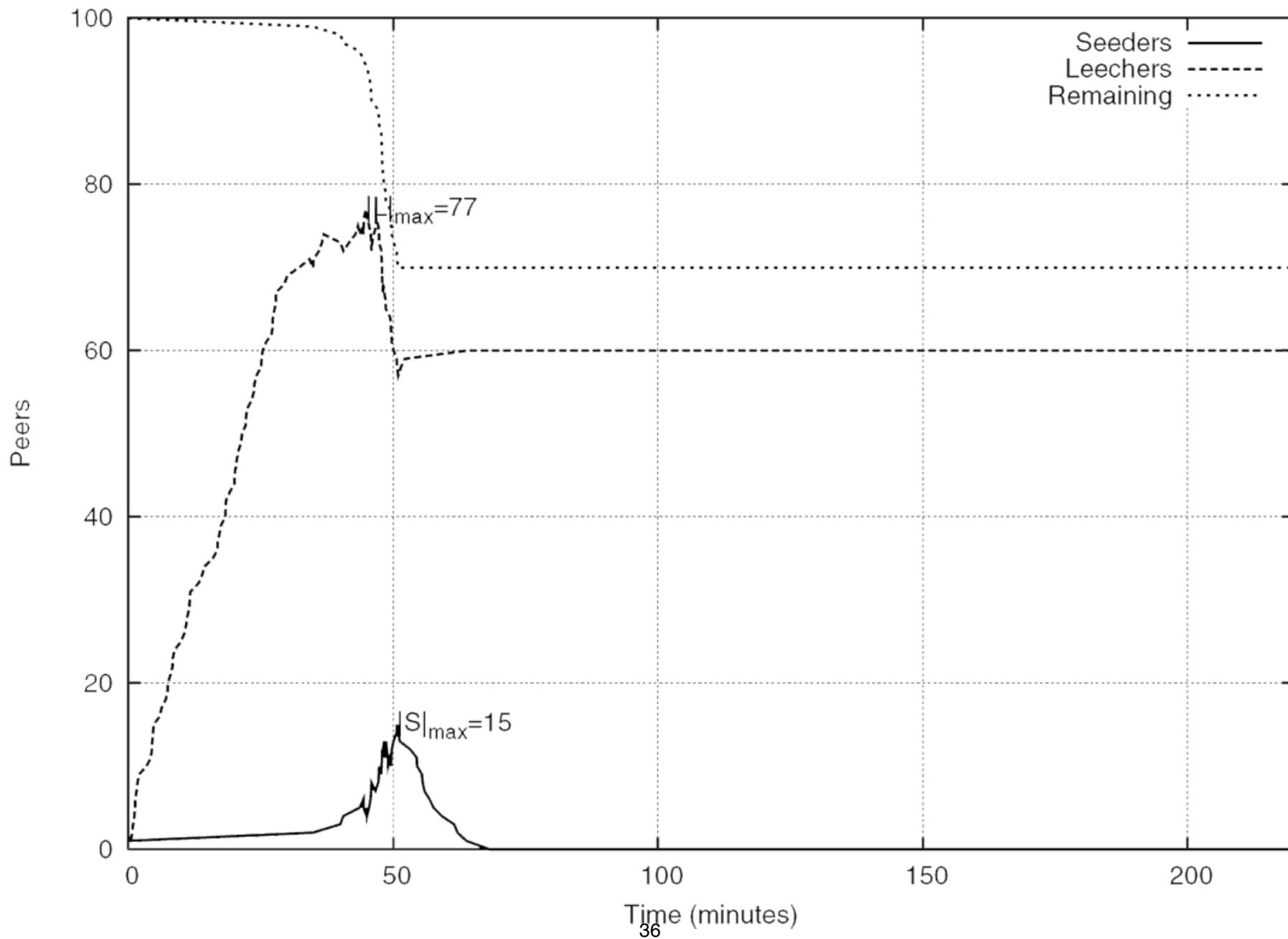
# Piece lying

- A Sybil attack on a P2P network is an attack using multiple, pseudonymous peers (Sybils)
  - This could be multiple peers spawned on the same physical machine.
- In the piece lying attack the attacker(s) take advantage of the *rarest first* piece selection scheme.
  - The attackers work in collusion lying about a set of pieces.
  - By having a large number of peers that claim to hold that set of pieces, the rare pieces *appear* common, and thus nobody specifically requests them
    - If a peer should request nonetheless randomly request one of the pieces, the lying peers will simply **choke** the requesting peer.
- Once the last true seed has left, the swarm has *failed*

# A honest swarm



# A swarm with piece lying (25 liars)



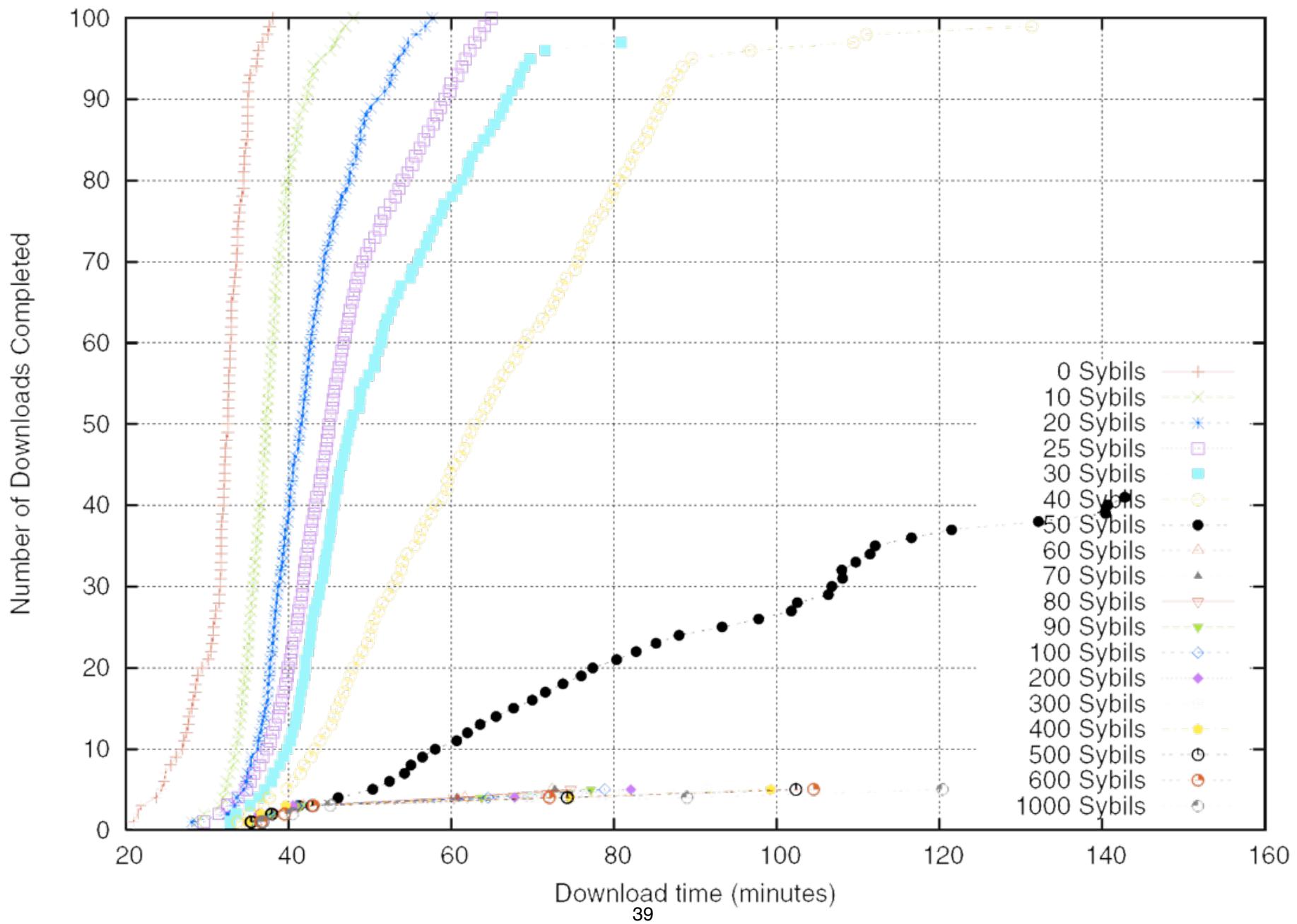
# Effectiveness of piece lying

- As the evaluation (the previous two graphs) show, piece lying can be detrimental to swarm health.
- The effectiveness is tied to
  - 1) the number of sybils in the attack
  - 2) the size of the swarm
  - 3) peer behaviour—if e.g., all peers keep seeding for a long time, the attack will be less effective.

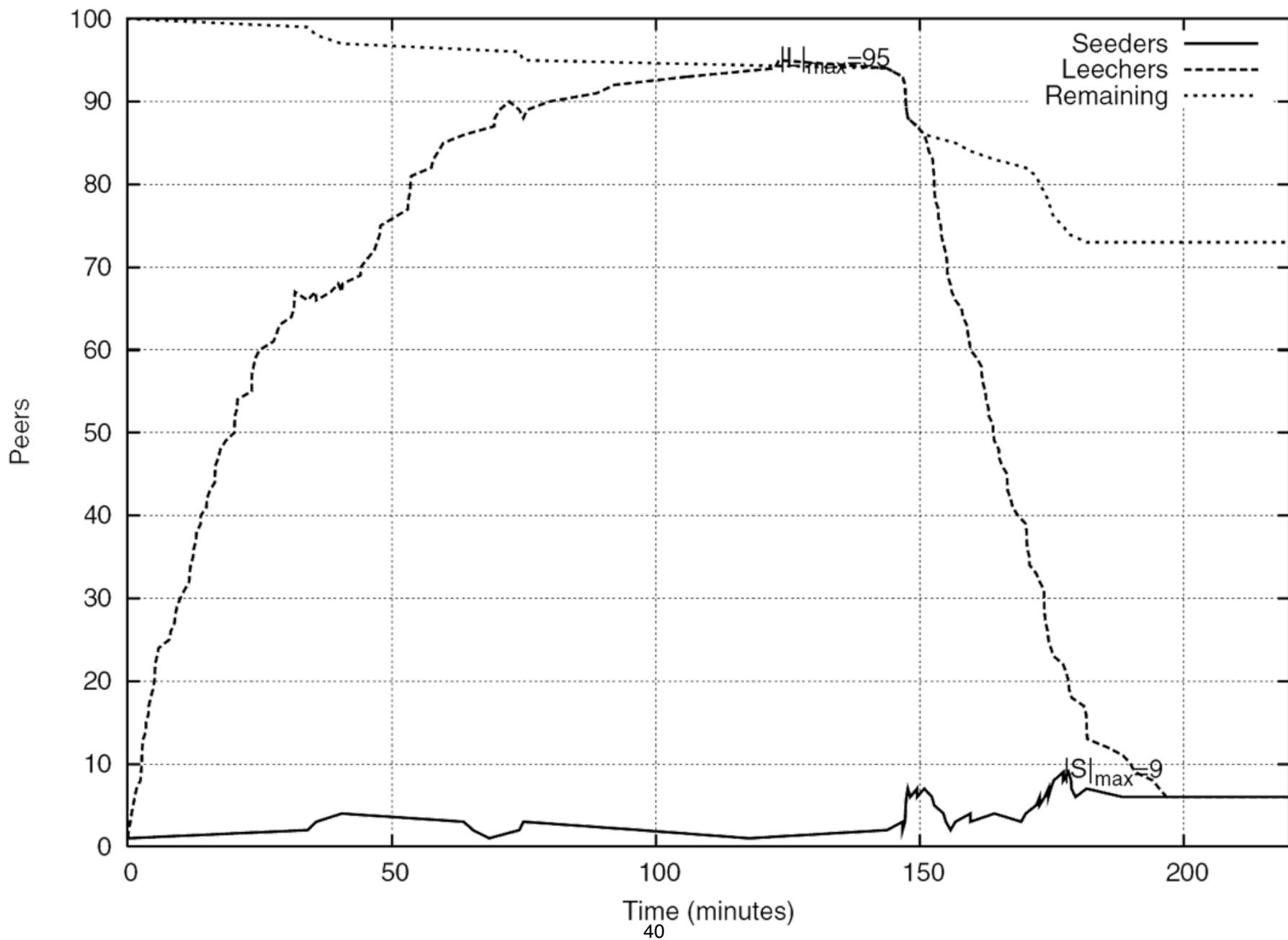
# Eclipsing correct peers

- The idea behind an eclipse attack is to eclipse the regular peers by making sure that they only (or at least to a very high degree) connect to malicious peers.
- In BitTorrent, this is done by adding a large number of malicious peers to the swarm.
  - These peers will try to connect to as many peers as possible to spread their influence in the network.
  - When a correct peer connects to a malicious peer the malicious peer will notify other malicious peers of this.
    - ... these will then try to connect to the correct peer also.

# How many Sybils do you need to poison the swarm?



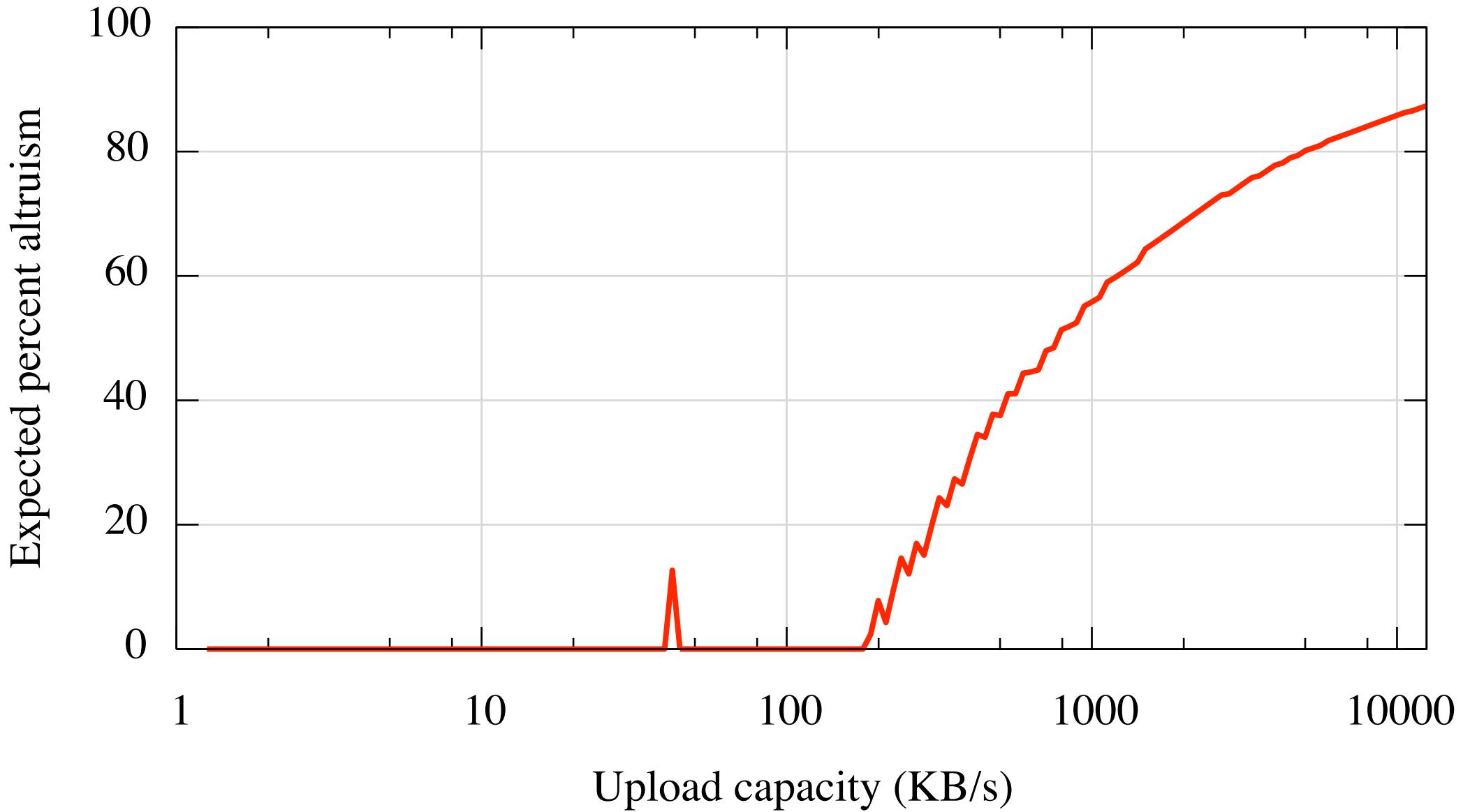
# Peer eclipsing (10 Sybils lie about 32 pieces)



# Taking advantage of the swarm

- **What if an attacker's intention is selfishness?**
  - The aim of such attacks is increasing one's own benefits,
  - and not as such to harm the swarm—but of course the swarm is hurt in the process, when some peers start to “free ride” the system.
- **BitTorrent has an incentive mechanism (tit-for-tat) that *should* provide incentive to contribute, but this can be circumvented.**
- **The BitTyrant system is an example of a *strategic* client that takes advantage of the BitTorrent protocol**
  - insight: you want to do the **minimal needed to stay unchoked**

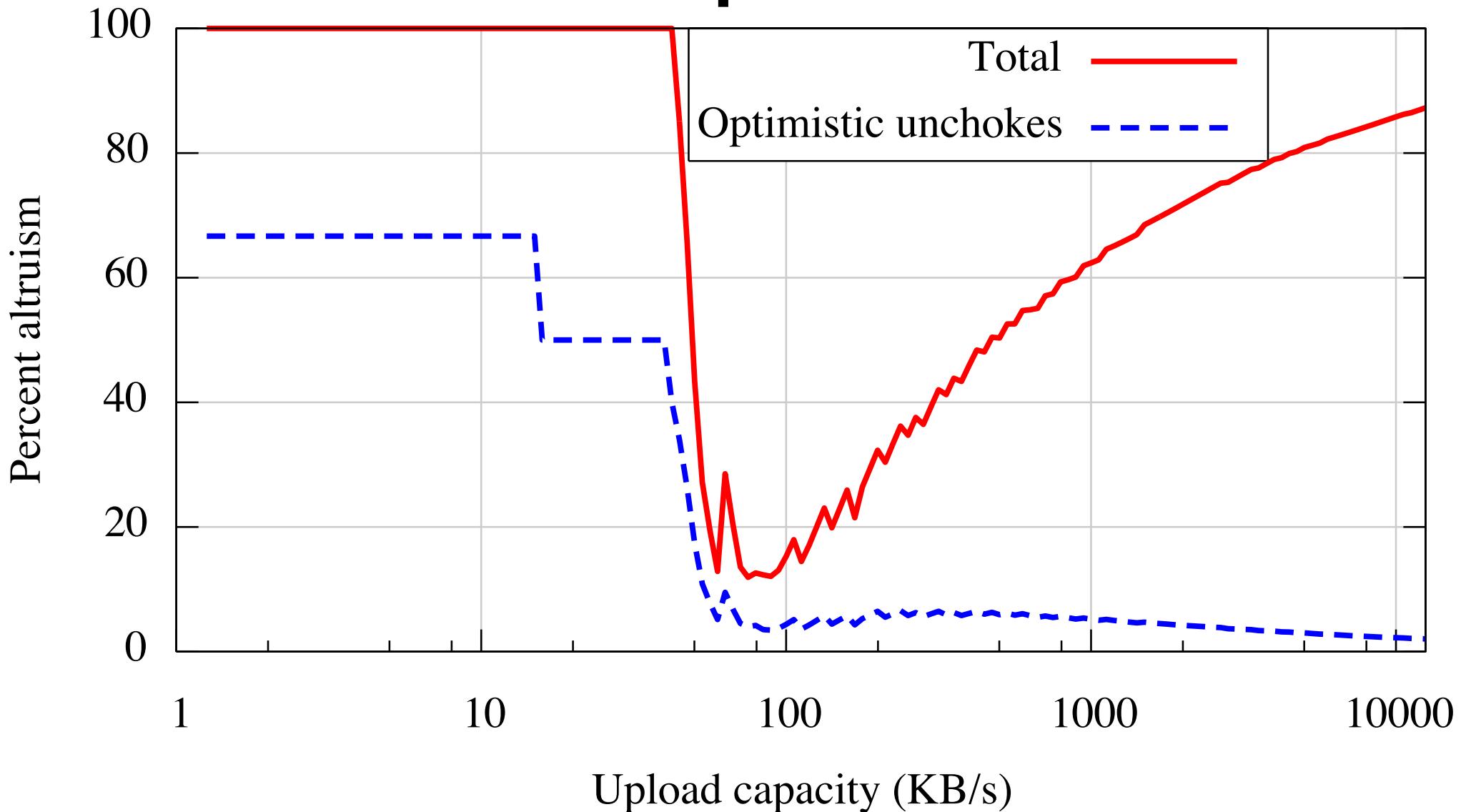
# Some observations about upload/download bandwidth



Altruistic upload as a function of rate

The powerful peers donate a large part of their bandwidth

# Some observations about upload/download bandwidth



Altruism when defined as upload capacity not resulting in direct reciprocation. The strong peers contribute more than they get.

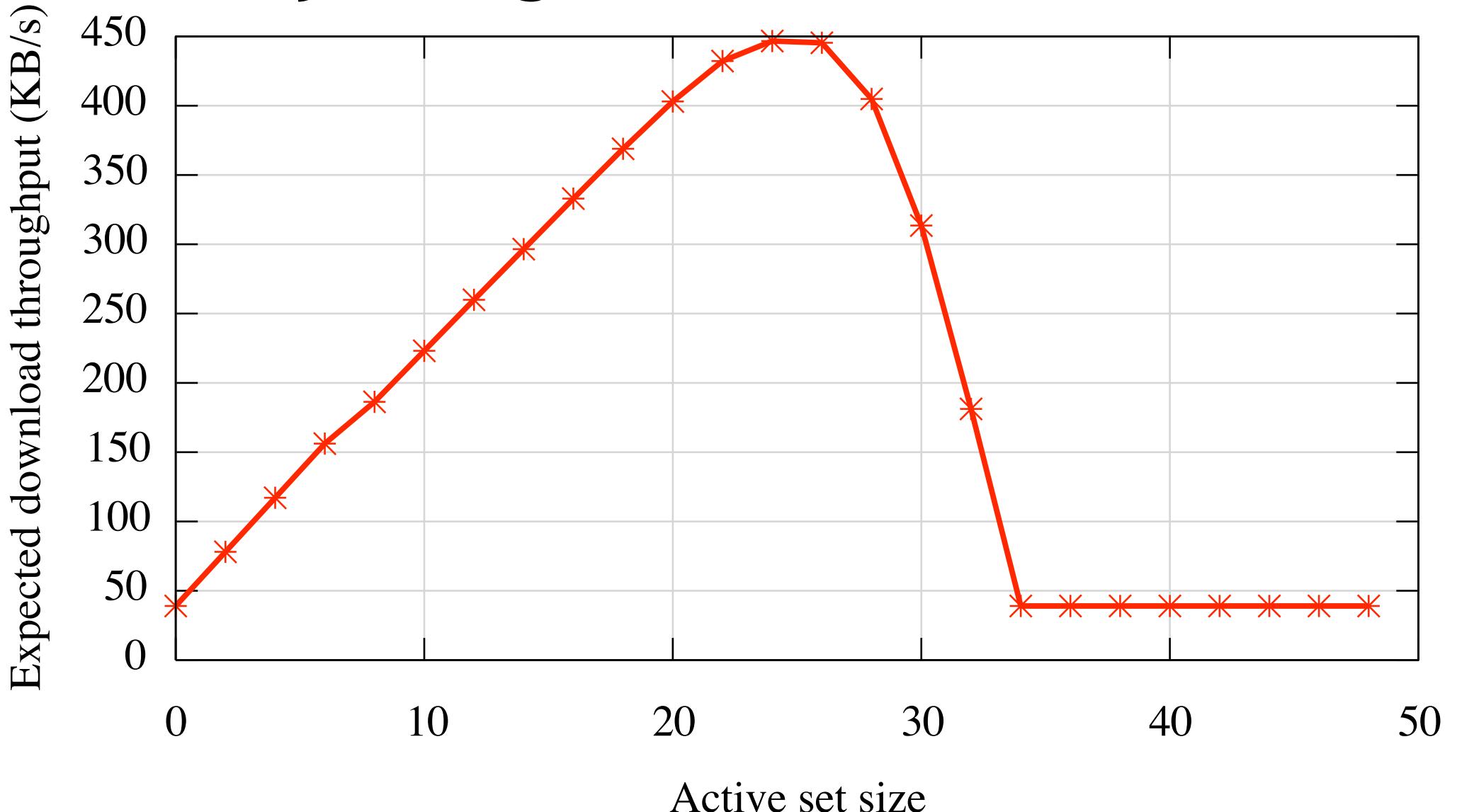
# A Sybil attack

- Looking at the data it seems that low capacity peers have disproportionately high performance.
- An obvious attack is then disguising a high capacity peer as multiple low capacity peers.
  - flooding the local neighbourhood of high capacity peers these Sybils increase the likelihood of tit-for-tat reciprocation
  - and of receiving optimistic unchoke
- Such attacks may be mitigated by disallowing multiple connections from one IP address.

# Adaptively resizing the active set

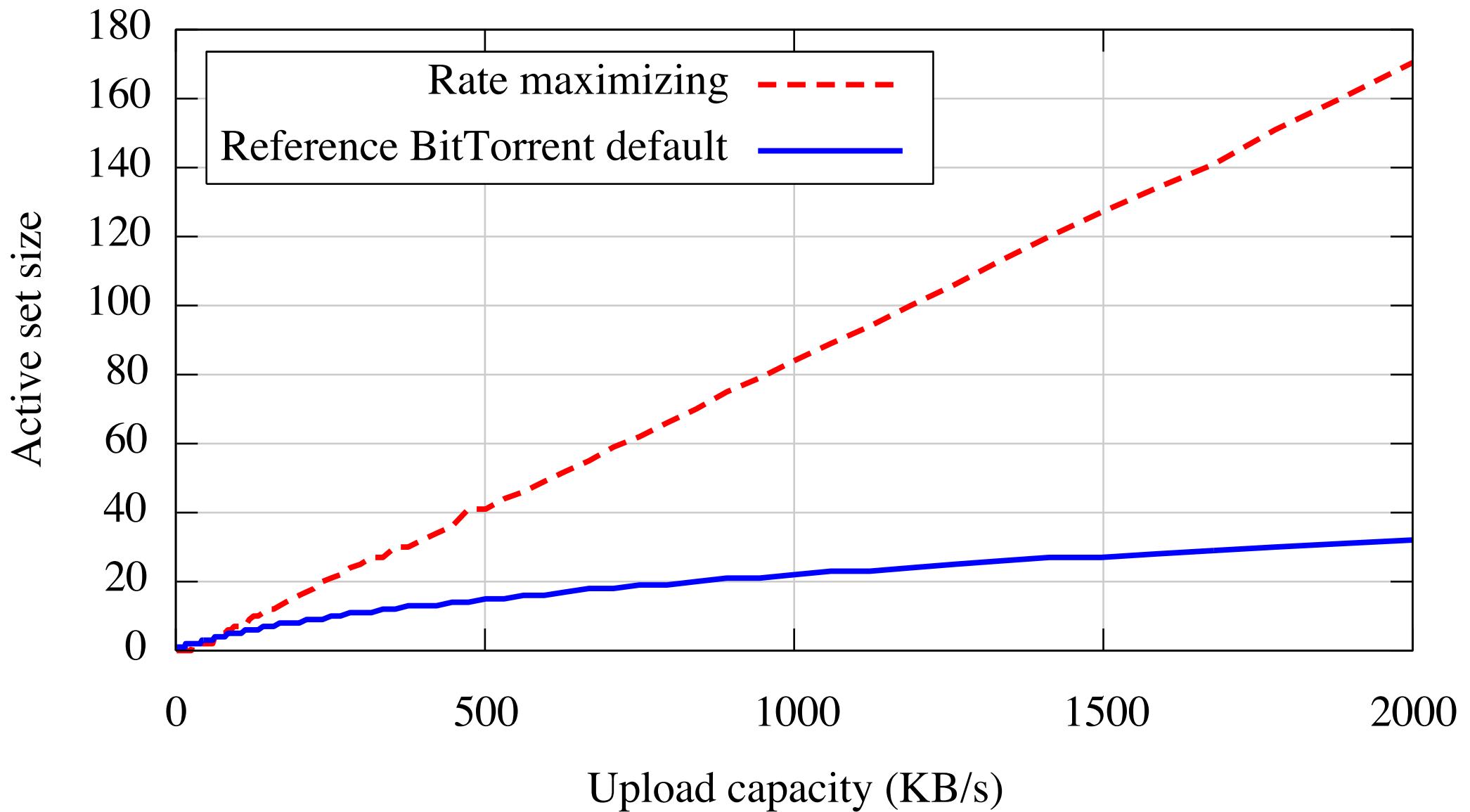
- From the data it seems that high capacity peers upload “too much” to their neighbours compared with what they get back.
  - that would imply that having more neighbours in the active set would be beneficial.
- If the equal split capacity distribution of the swarm is known, we can derive the active set size that maximises the expected download rate.

# Adjusting the active set size



Expected download throughput for a peer with 300 KB/s upload

# Adjusting the active set size



Optimal active set size as a function of upload capacity

# BitTyrant's unchoke algorithm

- For each neighbouring peer  $p$  BitTyrant maintains estimates of the upload rate required for reciprocation  $u_p$ ,
  - as well as measured download throughput  $d_p$ .
- Peers are then ordered by  $d_p/u_p$  and unchoked in order until the sum of  $u_p$  terms exceeds the upload capacity.

# BitTyrant's unchoke algorithm

For each peer  $p$ , maintain estimates of expected download performance  $d_p$  and upload required for reciprocation  $u_p$ .

Initialize  $u_p$  and  $d_p$  assuming the bandwidth distribution in Figure 2.

$d_p$  is initially the expected equal split capacity of  $p$ .

$u_p$  is initially the rate just above the step in the reciprocation probability.

Each round, rank order peers by the ratio  $d_p/u_p$  and unchoke those of top rank until the upload capacity is reached.

$$\underbrace{\frac{d_0}{u_0}, \frac{d_1}{u_1}, \frac{d_2}{u_2}, \frac{d_3}{u_3}, \frac{d_4}{u_4}, \dots}_{\text{choose } k \mid \sum_{i=0}^k u_i \leq cap}$$

At the end of each round for each unchoked peer:

If peer  $p$  does not unchoke us:  $u_p \leftarrow (1 + \delta)u_p$

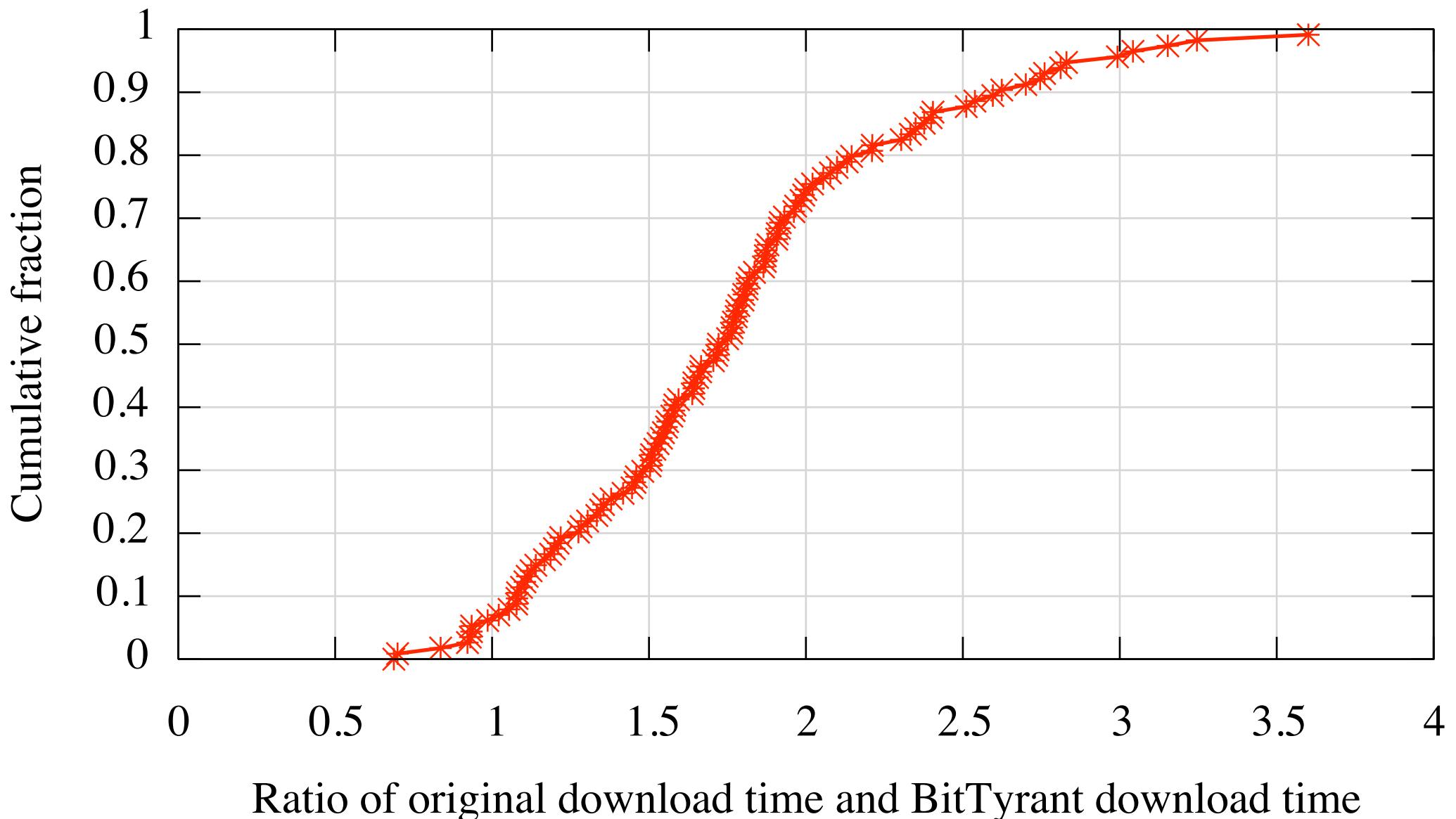
If peer  $p$  unchoke us:  $d_p \leftarrow \text{observed rate}$ .

If peer  $p$  has unchoke us for the last  $r$  rounds:

$$u_p \leftarrow (1 - \gamma)u_p$$

- $\gamma = 10\%$
- $\delta = 20\%$
- $r = 3$

# BitTyrant in a regular swarm



# Summary

- BitTyrant performs well in a regular swarm—where it lives off the altruism of the other peers.
- High bandwidth peers really benefit from BitTyrant.
- It also lives well in a swarm of only BitTyrant peers—as long as these are altruistic, i.e., they still contribute excess capacity.
- But when the entire BitTyrant swarm is acting selfishly the performance takes a serious hit. Selfish meaning that the peer will never use excess capacity.

# Summary

- **We have seen a number of ways to attack BitTorrent:**
  - Sybil attacks, piece lying, peer eclipsing
- **The BitTyrant system, a strategic BitTorrent client, was presented. BitTyrant increases download speed by:**
  - varying the active set size based on the reciprocation and,
  - making sure that you only give what is necessary to other peers.

# Summary

- **Scalability**
  - Highly scalable and widely used
- **Fairness**
  - You are only involved if you are interested in a particular file, give and ye shall receive...
- **Integrity and security**
  - Files are integrity checked – peers may be malicious
- **Anonymity, deniability, censorship resistance**
  - Not a part of the protocol – transactions can be (and are) followed, and trackers can certainly be shutdown