Performance Factors in Peer-to-peer file distribution networks

Martijn van Eenennaam – Selected Topics in Telematics 2007

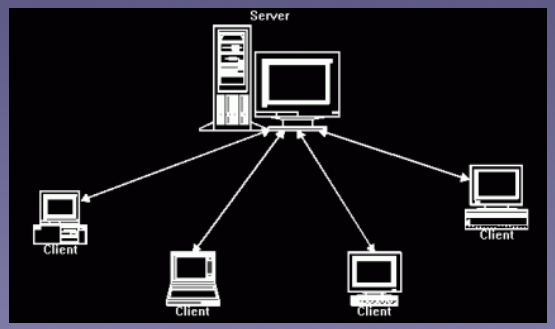
Problem

* Large file – say a Linux distribution

* Thousands of users can't wait to get their hands on it

* It is release-day, what happens?

Traditional Client-Server architecture:



All communication with Server -> bottlenecked by server's uplink!

Server-Client solutions

* Large serverfarms with lots of bandwidth

* Multiple servers (mirroring)

* Load-balancing infrastructure

These solutions are **expensive**, don't **scale** too well and are not **self-organising**.

These solutions still cannot deal easily with *flashcrowds*, with every user initiating a download performance **degrades**.

Let users redistribute the file!

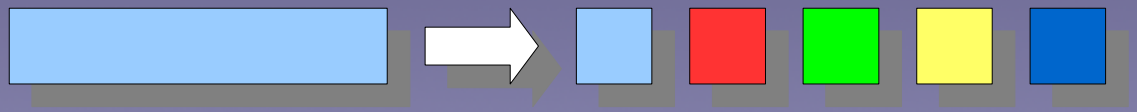
P2P Solution

Let users redistribute the file!

Benefits:

* Huge aggregate bandwidth
* Automatic mirroring of data
* No expensive serverfarms needed

* Some solutions transfer *entire* files – no option for large files due to high **churn** rates
* Better to break file into smaller **chunks**:



Users redistribute these chunks individually.

Chunking

The use of chunks comes with several nice opportunities:

* Granularity of transmission units is small –> low loss in case of corruption or incomplete transmission due to churn

* Parts of the same file can be downloaded from different sources -> speed boost

* Redistribution of (part of) a file can commence quickly

* Chunking allows intelligent chunk selection – rarest first, random

This technique is also called *swarming*.

P2P Solution – distribution methods

Biersack et. al. propose three distribution models:

* Linear Chain

* Tree

* Forest of Trees

These models can be used to gain insight in the factors that play a role in file distribution.

BitTorrent, a system deployed on a large scale, uses a **Mesh** which is in essence a dynamic forest of trees **–** more on that later...

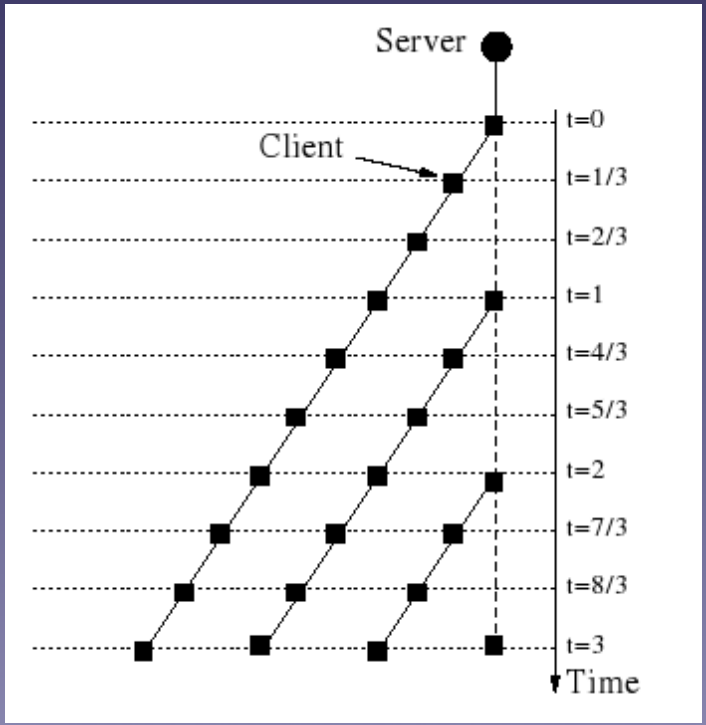
Linear Chain Model

* Server transfers entire file in chunks to one client at a time
* Client shares received chunks and transfers a chunk to one other peer at a time

Flaws:

* Server does not use full bandwidth (uplink >> user's downlink)

* Chunks allow selective upload of some chunks – not used in this scheme



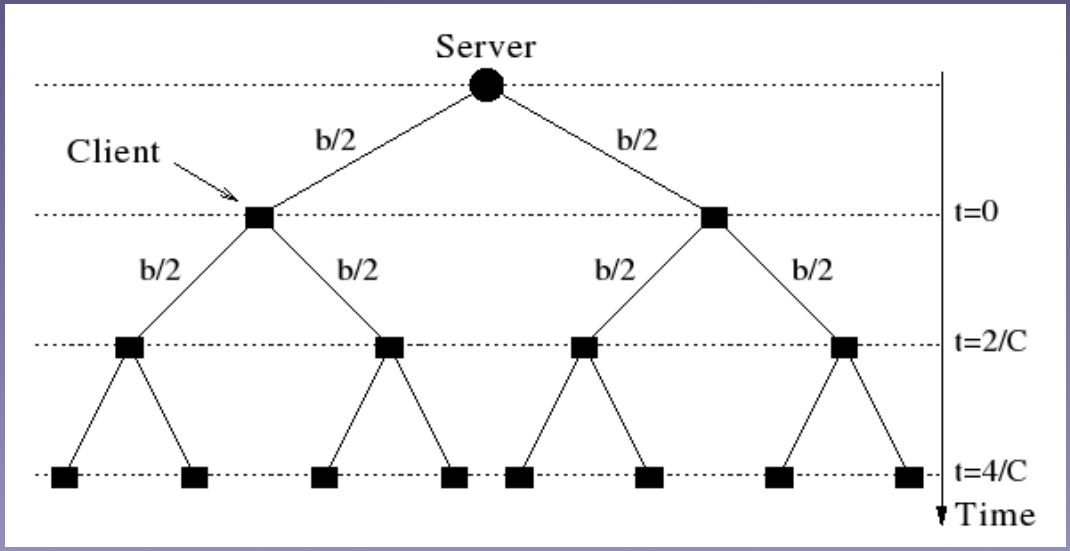
Trace for file split in three chunks

Tree (Tree^k) Model

* Server transfers in parallel to *k* clients
* Client shares received chunks with *k* other peers simultaneously

Flaws:
* Available bandwidth shared with *k* peers, total download time will increase factor *k*

* Unfair: peers download a file of *b* bytes but must upload *k***b* bytes -> conflicts with connection types such as ADSL where downstream > upstream



A tree with degree *k* = 2

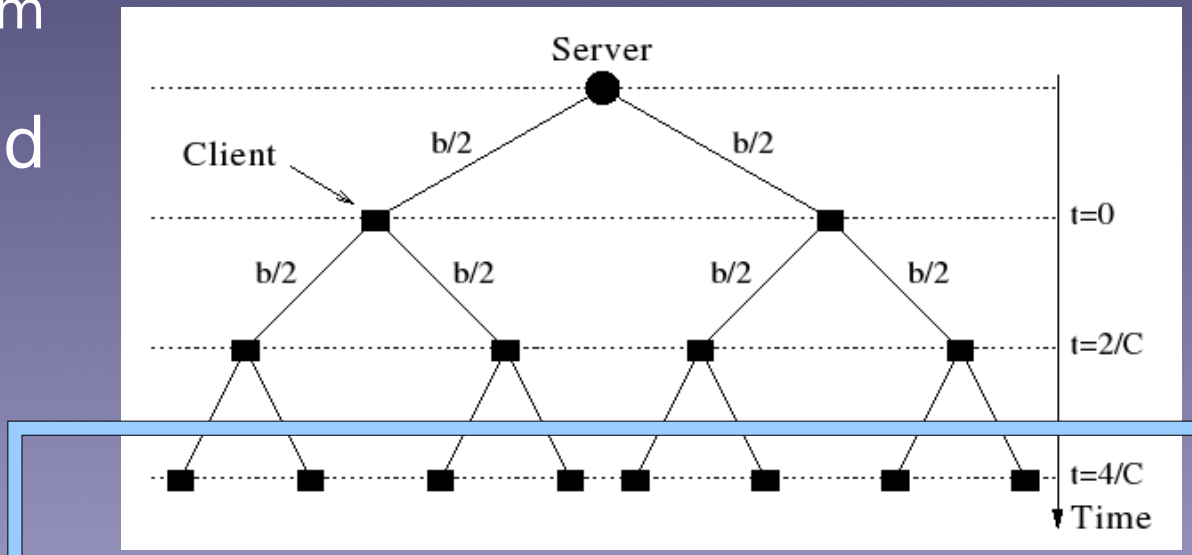
Tree (Tree^k) Model

* Server transfers in parallel to *k* clients
* Client shares received chunks with *k* other peers simultaneously

Flaws:
* Available bandwidth shared with *k* peers, total download time will increase factor *k*

* Unfair: peers download a file of *b* bytes but must upload *k***b* bytes -> conflicts with connection types such as ADSL where downstream > upstream

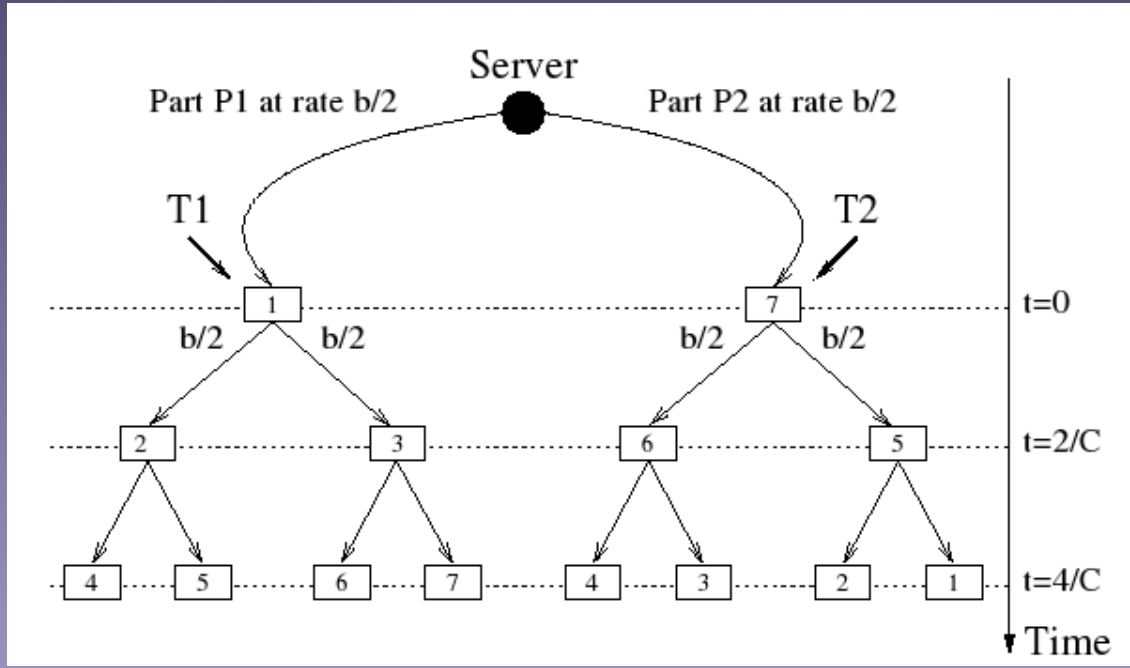
* Many users upload NOTHING!



Parallel Trees (Ptree^k) Model

* Server transfers chunks to *k* peers simultaneously
* A peer is interior in one tree, leaf in other *k-1* trees
* Server transfers a different chunk to each tree
* A peer receives *k* chunks in parallel
* A peer redistributes a chunk *k*-times

Benefits: fairness guaranteed, server bandwidth used optimal



Choosing Chunk Size and Number

Biersack et. al. have shown that the number of chunks a file is broken into has effect on performance.

$$T_{Linear}(C,N) \approx \frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2 \cdot N}{C}}$$

$$T_{Tree}(C,k,N) = k + \lfloor \log_k(\frac{N}{k}) \rfloor \cdot \frac{k}{C}$$

$$T_{PTree}(C,k,N) = 1 + \lfloor log_k N \rfloor \cdot \frac{k}{C}$$

Choosing larger *C* generally decreases the time to completely transfer the file to *N* peers.

In practice, choosing *C* too large will result in increased overhead (both in transmission and hashing / tracking).

Using a Mesh Architecture: BitTorrent

The BitTorrent system uses a dynamic Mesh architecture, peers use bidirectional links that are continually reconfigured.

Except for a lookup-service this system is completely distributed.

To publishing a file:

* A .torrent file is placed on a webserver

* .torrent contains *metadata*:
  - filename
  - filesize
  - hashing information
  - url of a tracker

* A tracker enables downloaders to rendez-vous

* File is made available by starting a client with a 'seed'

File Distribution on a BitTorrent Network

* A user downloads a .torrent file

* BT client contacts tracker

* Tracker returns random list of peers
  - a random graph has beter resilience against churn than a tree

* BT client contacts peers to obtain chunks
  - first chunk obtained randomly
  - remaining chunks downloaded 'rarest first'
  - client called a 'leecher'

* BT client occasionally initiates transfer with other peers to find faster download rates

* Transfer is **bidirectional**, cooperate by uploading to peers
  - tit-for-tat policy

* When a download completes a client can remain uploading
  - client called a 'seeder'

BitTorrent in Practice

Izal et. al. conducted an analysis of a large torrent. The tracker log was analysed statistically and an instrumented client was used.

* Redhat 9, 1.77GB, followed over a period of five months

* 180,000 clients downloaded of which 51,000 in first five days

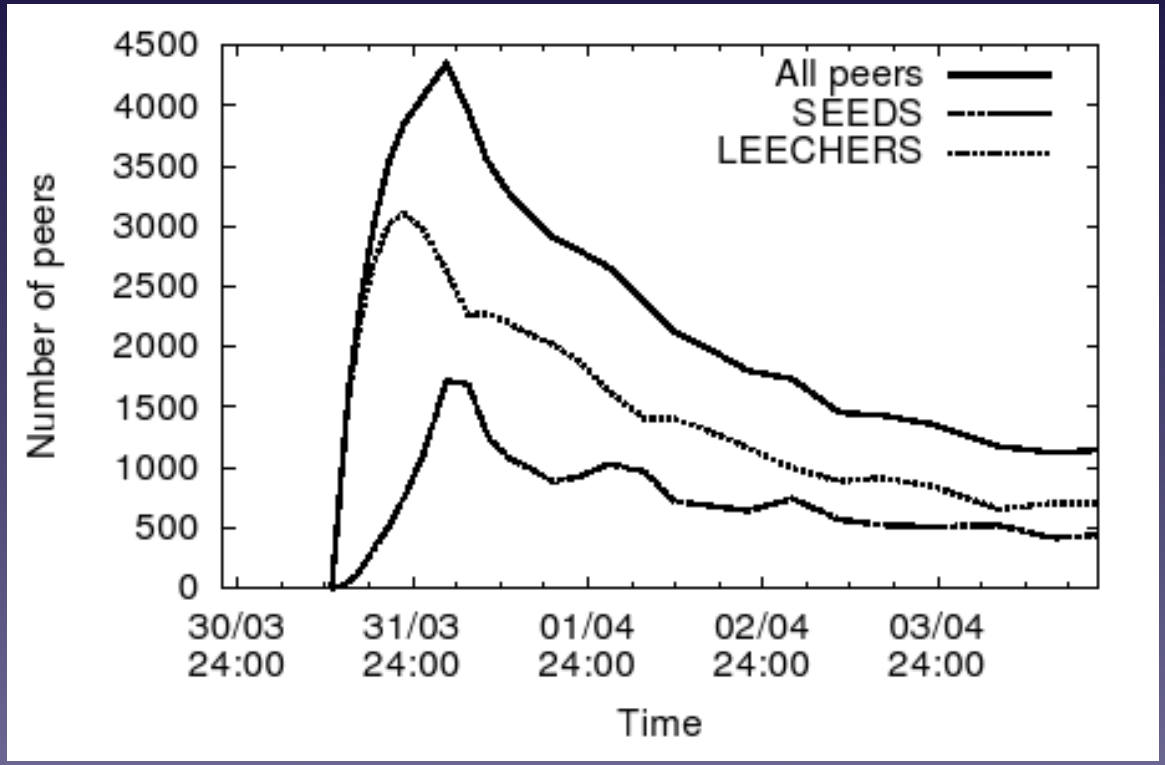
Let us assume 100% efficiency and no idletime, consider what this would imply when using a Client-Server architecture:

(51,000 x 1,77GB) / (5 * 24 * 3600) = 213.97MB/sec

-> this would take 18 100mbit connections!

Aggregate download rate peaked at over 800MB/sec

BitTorrent in Practice - continued



* #leechers quickly rises after publication

* #seeds is low at start, BT redistributes chunks as soon as they are complete

* BitTorrent is tuned to duplicate a file as quickly as possible, rarest-first chunk-selection guarantees availability of all chunks of a file

* BitTorrent not so much a 'share' network but a 'rapid replication' system

On a more Social note...

A research by Krishnan et. al. focusses on the *Impact of Free-Riding on Peer-to-Peer Networks*.

* Ability to distribute seen as a socio-economical *common*, a commodity everybody can use (such as air, grassy meadows etc.)

* Users take their share unselfishly (*prisoner's dilemma*)

* Free-riding: using resources without providing resources in return

* Several reasons:
  - high upload rate chokes download rate in some connections
  - paranoid of prosecution for redistribution of (potential) illegal material

* Studies reveal P2P networks are quite resistant to free-riding, could be more optimal with less free-riding

* BitTorrent solves this with tit-for-tat policy: free-riders get lower priority

Performance Factors – a Conclusion

Several performance factors in p2p file distirbution networks have been identified:

* Bandwidth (of initial server and peers)

* Number of peers

* Number of chunks a file is broken into

* Chunk selection strategy

* Peer selection strategy

* Loss of performance due to churn, robustness

* User's willingness to share (altruism). Some clients allow to limit upload speed, number of connections or disable upload completely

References

Ernst W. Biersack , Pablo Rodriguez, Pascal Felber *Performance Analysis of Peer-to-Peer Networks for File Distribution*

Bram Cohen *Incentives Build Robustness in BitTorrent*

M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garc´s-Erice *Dissecting BitTorrent: Five Months in a Torrent's Lifetime*

R. Krishnan, M.D. Smith, Z. Tang, R. Telang, *The Impact of Free-riding on Peer-to-Peer Networks*

E. Adar, B.A. Huberman, *Free Riding on Gnutella*

Front image: *Gears* by Peter Neumann, www.peterneumann.com