P2P Applications

Reti di Elaboratori

Corso di Laurea in Informatica

Sapienza – Università di Roma

Versione originale delle slides fornita da Dora Spenza e Marco Barbera

P2P Paradigm

- Late 80's
- Became popular in 1999-2001 thanks to
- Napster was shut down by court order in 2001 due to copyright violation
- New P2P clients were developed: Gnutella, Kazaa, BitTorrent
- As of today, **43-70%** of Internet traffic is generated by P2P applications (Feb 2009)

Peer-to-peer (P2P) networks

Server-based Network

- A type of network in which each workstation (peer) has equivalent capabilities and responsibilities

P2P Network

- Differs from client/server architectures, in which some computers are dedicated to serving the others

Peer-to-peer (P2P) networks

Server-based Network

- Each peer can act as a server or as client
- Each peer does not necessarily have to be always active, see for example BitTorrent
- Peers join and leave the network continuously
- P2P networks are very dynamic networks!!

P2P Network

P2P networks: Goal

- Increase <u>scalability</u>
- Increase resources <u>availability</u>
- Increase <u>fault-tolerance</u>
- <u>Cost</u> reduction
- Increase peer <u>privacy</u>
- Provide a framework for <u>dynamic</u> scenarios

Server-based Network

P2P Network

P2P networks: **Challenges**

- Peers are not <u>reliable</u> (*e.g.,* disconnections, low bandwidth)
- Peers are heterogeneous, with different computational power and storage capacity
- Resource discovery
- Security and resource integrity

Server-based Network

P2P Network

P2P Flavours

- P2P networks available today come in different flavours
- They can be classified depending on how the **overlay network** of peers is organized
- Unstructured networks (*e.g.,* GNUTella, Kazaa): do not impose any topology on the overlay network (*i.e.,* peers connect randomly to each other). Searching is performed by flooding the network with queries
- Structured networks (*e.g.,* Kad, certain flavours of BitTorrent): typically a **D**istributed **H**ash **T**able (DHT) allowing distributed and efficient search of peers with specific content

P2P Flavours

- Hybrid P2P networks, that mix P2P and client/server architectures:
    - to simplify the join to the network of a new peer (*bootstrap* problem)
    - to improve resource discovery
- Example: A (set of) server(s) provide a **centralized** resource index (e.g., Napster):
    - Resource discovery is straightforward, but…
    - Single point of failure
    - Performance bottleneck and infrastructure cost

P2P Flavours

- Hierarchical P2P networks (e.g., Skype)
- There are special peers called *super-peer* with additional functionalities
  - Usually selected among the more "powerful" peer nodes
  - Useful to simplify resource discovery
  - Each peer is connected to a super-peer that manage a **local** resource index
  - If a peer requests a resource and the resource is not on the local index, the super-peer forwards the request in flooding to other super-peers
  - Flooding limited between super-peers → greater efficiency…
  - …and local index improves searching performances!!

An alternative to content delivery: CDN

- A Content Delivery Networks (CDN) is a system of servers distributed across the Internet
- Goal: provide third-party contents to end-users with high availability and cost
- Key idea: content replication close to end-users
- Example: Akamai
  - 170000 servers in 102 countries
  - Akamai delivers between 15-30% of all Web traffic
  - Akamai delivers over 2 trillion daily Internet interactions

Example: P2P vs CDN

Server-based Network

- Lower costs
- More <u>scalable</u>
- P2P networks can take advantage of the **upload** bandwidth capacity of the clients (peers) that are downloading a given content
- Peers in a P2P network become part of a big, decentralized (and potentially very efficient) CDN
- However, a P2P network is not always easy to manage and **QoS** can be a problem

P2P Network
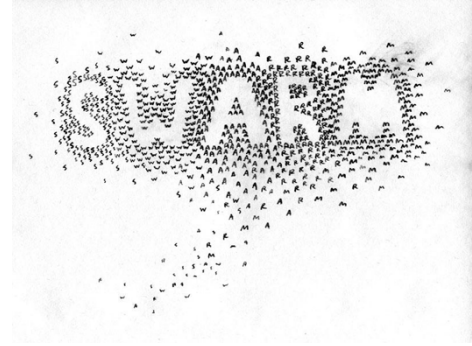
Understanding P2P protocols

BitTorrent

- P2P file distribution system
- Designed and implemented (Python) by Bram Cohen in 2001
- Dozen of free clients
- January 2012: 150 million active users
- Used to distribute large amounts of data over the Internet: not only media content, but also Linux distributions, scientific data sets, …

BitTorrent overview

- A separate *torrent* for each file
- Peers simultaneously upload and download pieces of file within the torrent
- The set of all active peers in a torrent is called the *swarm*

Two types of peers

For each *torrent* the set of active peers is divided into:

Seeds: clients that have a complete copy of the file and that continue to serve other peers
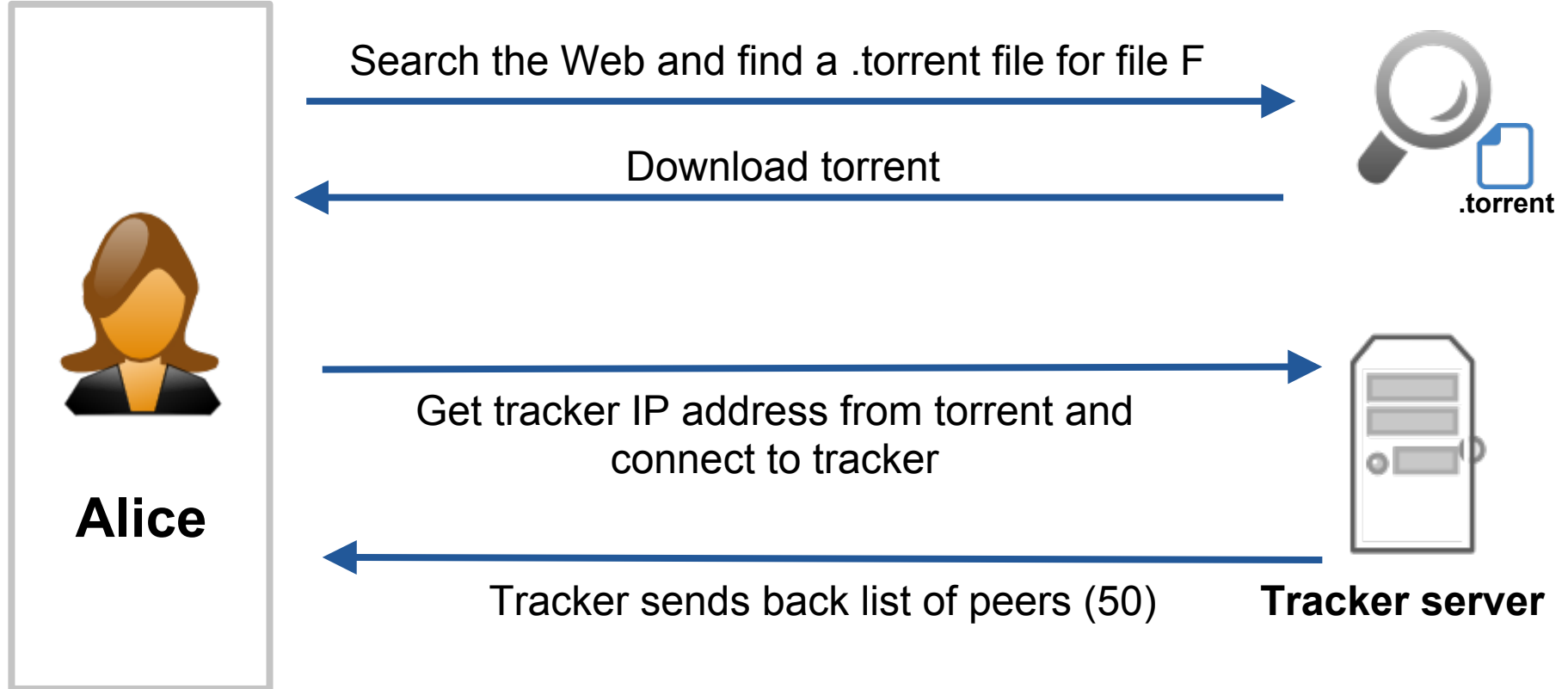
Leechers: clients that are still downloading the file (Alice)

How to download a file?

- Users need to discover which peers hold a copy of the file (at least a seeder!)
- Search for a **.torrent** file on the Web
- Torrent file include the address of a centralized server (the *tracker*) that helps peers finding each other
- Connect to the tracker and receive the list of peers having a copy of the file

Discovering peers for a file F

Alice

Search the Web and find a .torrent file for file F

Download torrent

.torrent

Get tracker IP address from torrent and connect to tracker

Tracker sends back list of peers (50)

**Tracker server**

The Tracker

- Not involved in the actual distribution of files!
- Keeps information about peers currently active
- Peers report their state to the tracker every 30 minutes, and when joining or leaving the network
- New clients receive from the tracker the IP address of 50 randomly chosen active peers

Contacting peers

- Once received the list of IP addresses from the tracker, Alice tries to establish a TCP connection with each of them
- *Peer set*: peers to which Alice is connected
- It changes over time!
- If nodes in the peers set become less than 20, Alice contacts the tracker again to obtain a new list
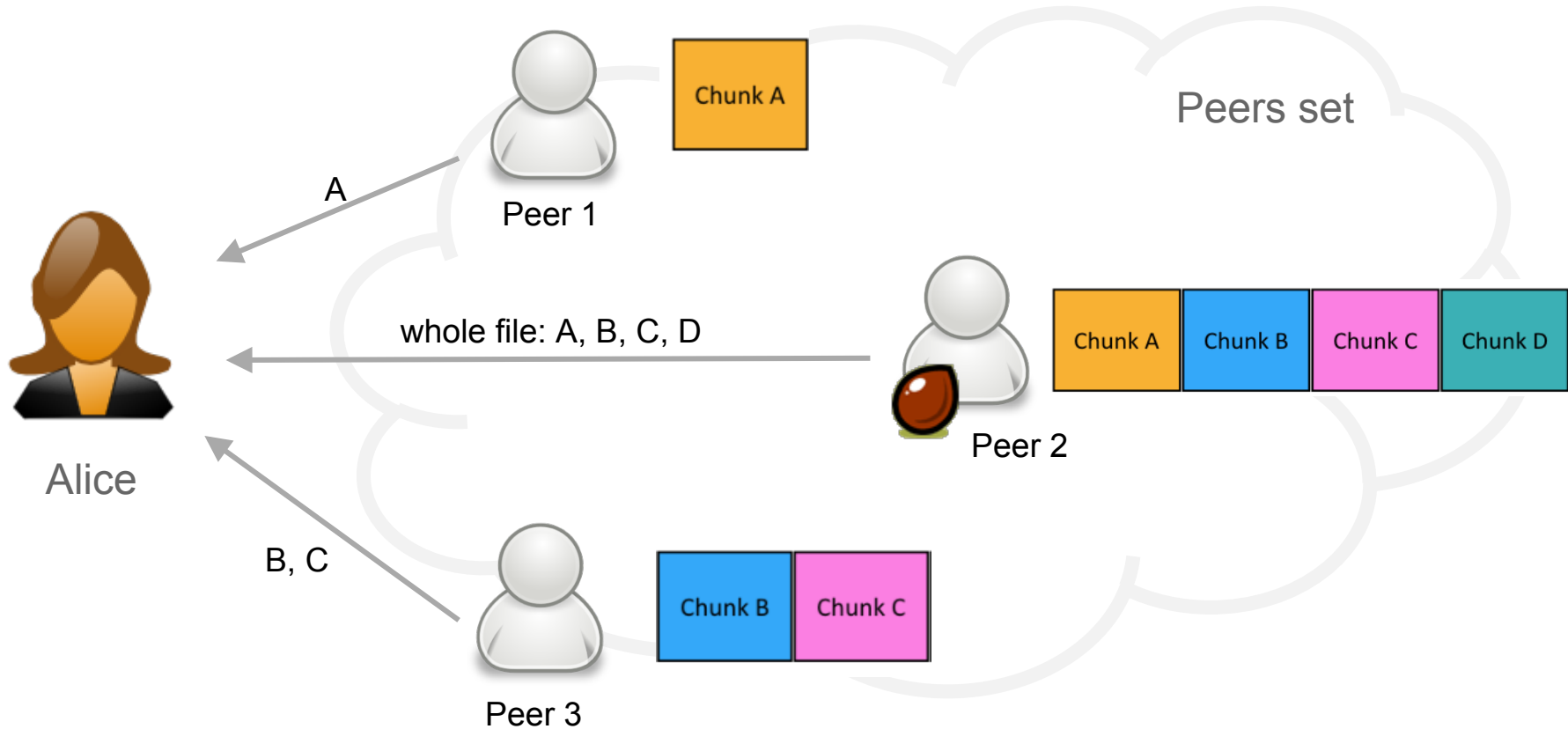
File chunks

- In BitTorrent files are divided into pieces (**chunks**) of size between 64 KB and 1 MB (typically 256 Kb)
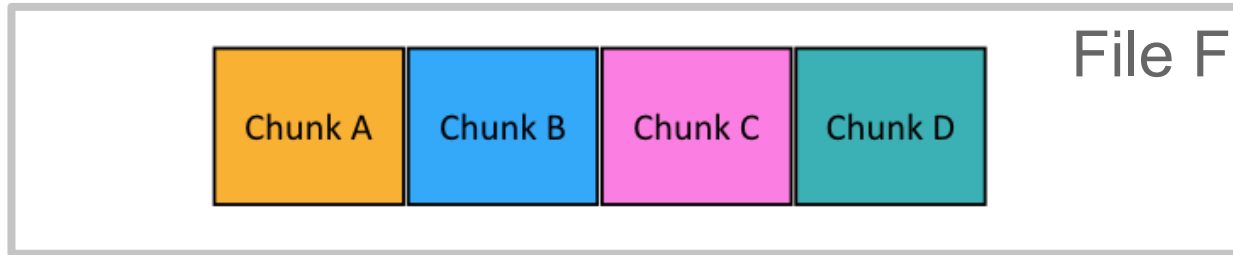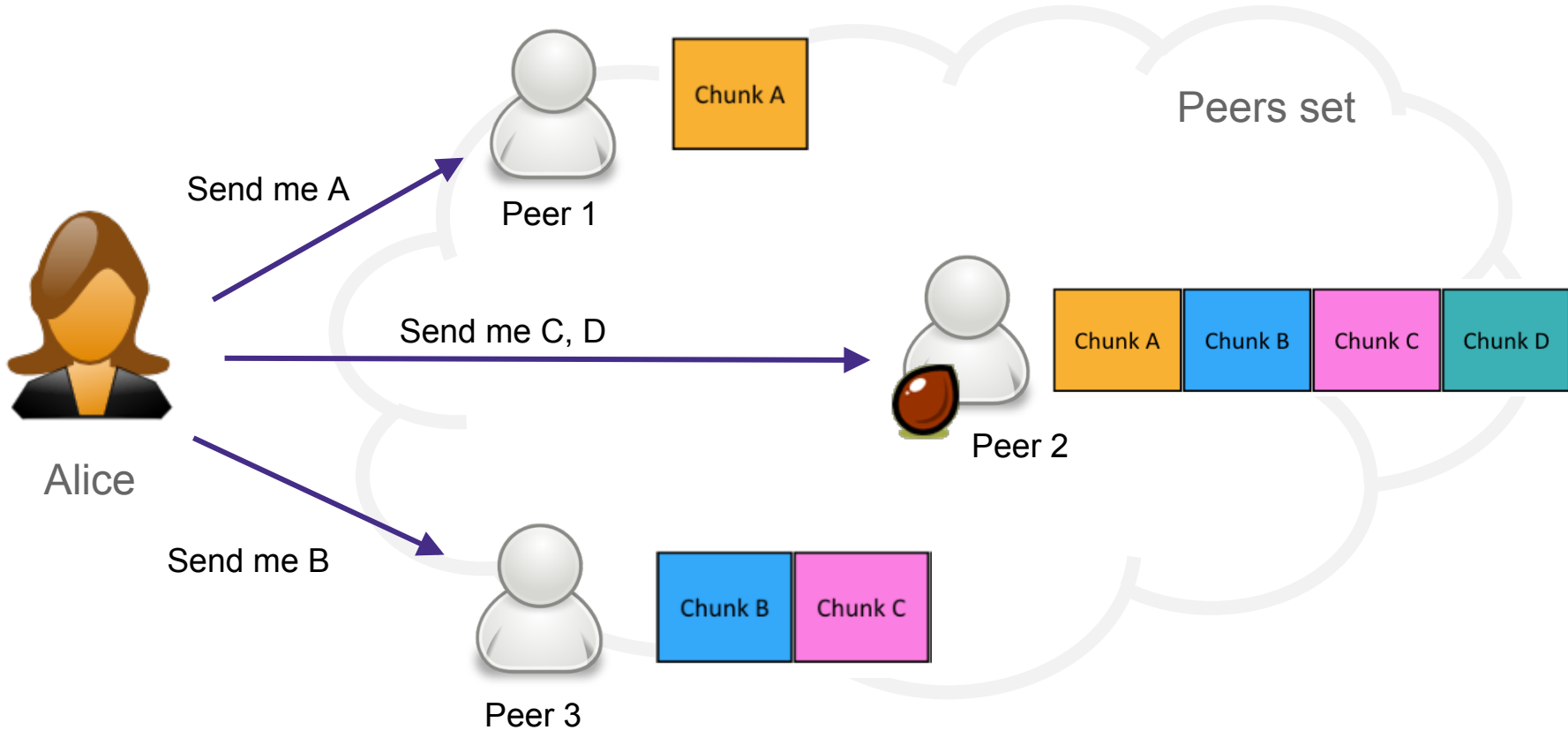

- When Alice enters the torrent for file F, she has no chunks
- Each peer in the peers set have a subset of chunks from F
- Alice periodically asks each node in the peers set for the list of chunks they have

Peers send their chunks list to Alice

Multiple simultaneous downloads

Downloading chunks

- Alice downloads chunks from multiple peers and keeps track of the download rate from each of them
- In which order file chunks are downloaded?
- **(Local) rarest first**: based on the chunks list received by her peers set, Alice determines which chunk (among those she does not have) is the rarest one in her peers set

Downloading rarest first

- Chunks that are more common are left for later
- By replicating the rarest chunks as quickly as possible, the risk of getting them completely lost as current peers leave the torrent is minimized

Exception

- **Random first**: when a new user joins the torrent, the first chunks to download are randomly selected, as rare chunks, being usually present on only one peer, would be downloaded more slowly

Uploading chunks

- As soon as Alice downloads her first chunk, she can start uploading to other peers
- Alice has a limited number of upload slots to allocate to other peers
- How to choose which peers to serve?
- **Tit for tat**: exchanging upload bandwidth for download bandwidth

Trading chunks

- Alice continuously measures her download rate from the other peers
- She uploads chunks to the 4 peers from which she is downloading at the <u>highest rate</u>
- Every 10 seconds she recalculates the *four top peers*
- In addition, every 30 seconds she picks a peer at random and uploads chunks to her

Choking & Unchoking

- The five peers to which Alice uploads are said to be *unchoked*
- All the other peers in the swarm are *choked*, i.e., they do not receive any chunk from Alice
- Unchoking a random peer every 30 seconds (*optimistic unchoking*):
  - ensures that newcomers get a chance to join the swarm
  - allows to potentially discover better partners

Optimistic unchoking

1. Alice optimistically unchokes Bob
2. Alice becomes one of Bob's top 4
3. Bob sends data to Alice
4. Alice becomes one of Bob's top 4
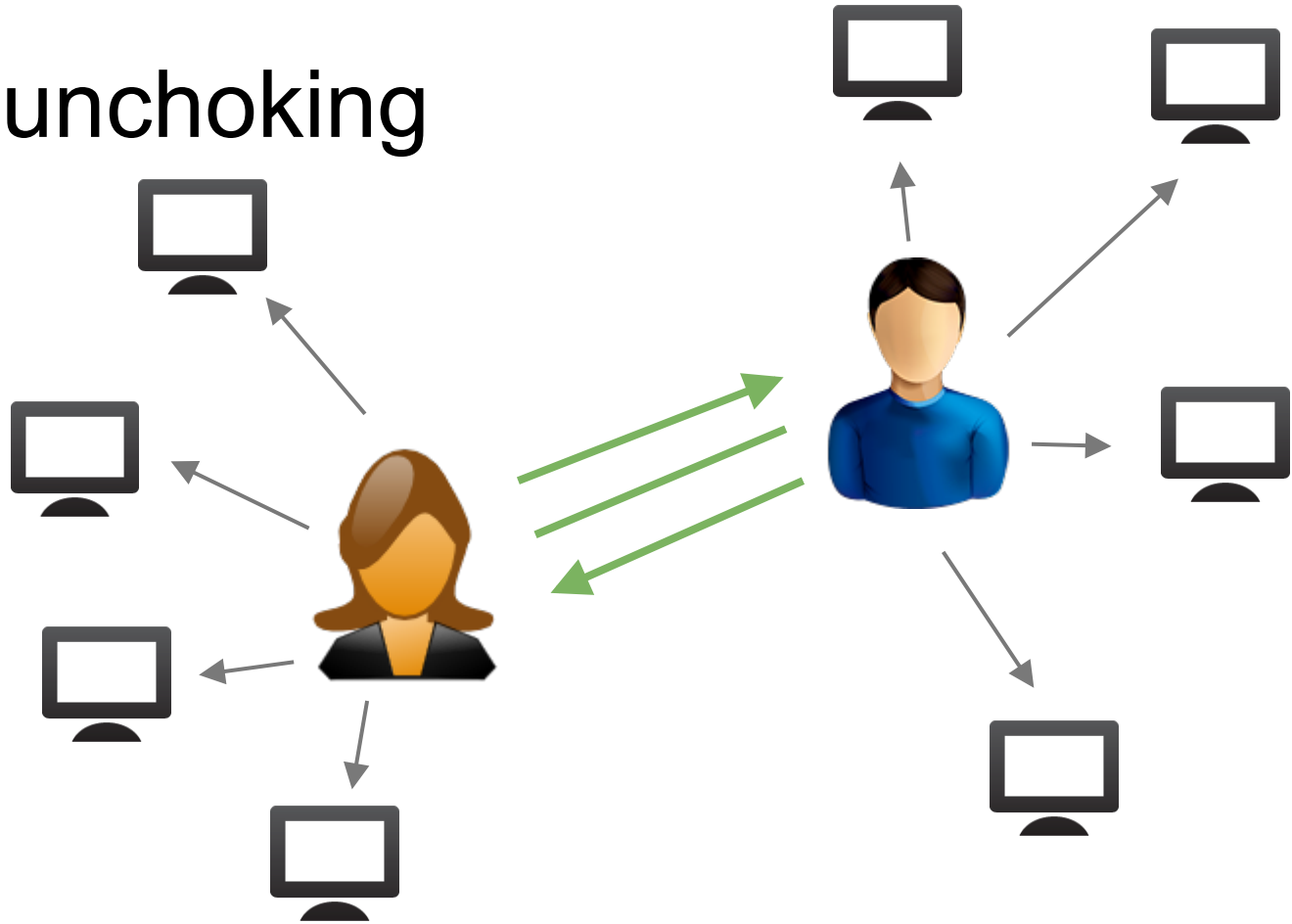
BitTorrent Pros and Cons

Pros:
- Proficiently uses partially downloaded files
- Discourages *free-loading* by rewarding fast uploaders
- Works well for hot content

Cons:
- High latency and overhead for small files
- Less useful for unpopular content
- Does not support streaming
- Leech problem
- Not a pure P2P protocol: single point of failure (the tracker)

Understanding P2P protocols

Spotify: Overview

- Spotify is a <u>peer-assisted</u> on-demand music streaming service
- Active users: Over 60 million
- Number of songs: Over 30 million
- Number of songs added per day: Over 20,000
- Available in more than 58 countries
- Efficient: <u>Only ~ 250ms playback latency on average!</u>
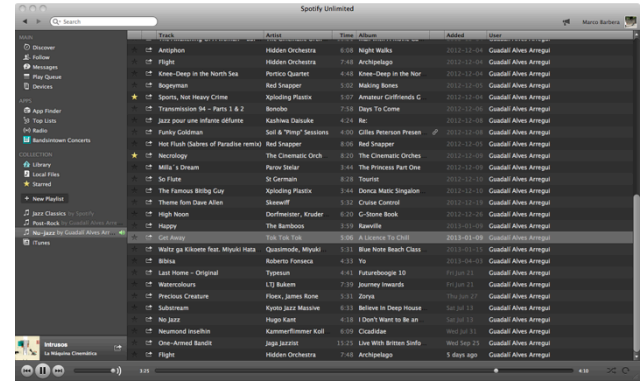
# Spotify: Overview

- Spotify uses a proprietary protocol, but:
  - some of its internals have been described by researchers working at Spotify (http://www.csc.kth.se/~gkreitz/spotify-p2p10/)
  - a third-party OSS alternative client has been released (http://despotify.sourceforge.net/)
  - *… but* since September 2013, Despotify is not compatible anymore with Spotify :-(
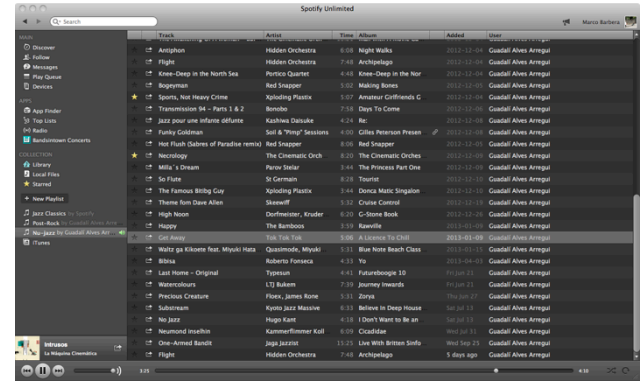
Spotify: Architecture

- Spotify uses a *hybrid* content distribution method, combining:
    - a client-server access model
    - a P2P network of clients
- Main **advantage**: <u>only ~ 8.8% of music data comes from the spotify servers!</u> The rest is shared among the peers (although mobile devices **do not** participate to the P2P network)
- Possible **drawbacks**:
    - playback latency (*i.e.,* the time the user has to wait before the track starts playing)
    - (potentially) complex design
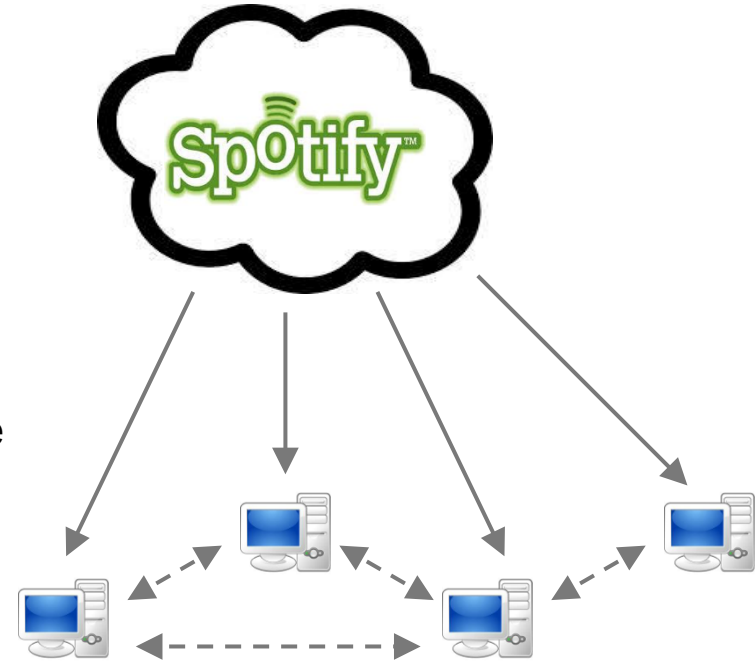
Spotify: P2P Network

- Spotify uses an **unstructured** P2P overlay topology.
  - the network is built and maintained by means of **trackers** (similar to BitTorrent)
  - **no** super peers with special maintenance functions (as opposite to Skype)
  - **no** Distributed Hash Table to find peers/content (as opposite to Kad)
  - Discovery messages get forwarded to other peers for two hops at most
- **Advantages**:
  - keeps the protocol simple
  - keeps the bandwidth overhead on clients low
  - reduces latency
- This is possible because Spotify can leverage on a *centralized* and *fast* CDN in the backend (as opposite to the completely distributed P2P networks)

Spotify: Caching

- Spotify clients store the already played tracks in a **cache**. By default, the cache uses at most 10% of disk space (capped to 10GB, but never less than 50MB).
- Around 56% of clients have a maximum cache size of 5GB.
  - **advantage**: increases the chances that a client can get a track from the P2P network (lower load on the Spotify servers).
  - **advantage**: reduces the chances that a client has to re-download already played tracks.
  - **drawback**: impacts on the users' disk
    - an LRU *cache-eviction* policy is used that removes the **L**east **R**ecently **U**sed (*i.e.,* played) track.
    - caches are large (as compared to the typical track size), so this is not a big deal.

Spotify: Sharing Tracks

- A client cannot upload a track to its peers unless it has **the whole** track
  - **advantage**: this choice greatly simplifies the protocol and keeps the overhead low, as clients do not have to communicate (to their peers or to the server) what parts of a track they have.
  - **drawback**: reduces the number of peers a client can download a track from (*i.e.,* slower downloads).
    - tracks are small though (few MB each), so this has a limited effect

Spotify: Locating Peers

- There are two ways a client can locate the peers:
  - ask the tracker servers
  - ask the other peers

Spotify: Locating Peers (tracker)

- To balance the load among its tracker servers a peer randomly selects which server to connect to.
- Each server is responsible for a **separate** and **independent** P2P network of clients.
  - **advantage**: does not require to manage inconsistencies between the servers' view of the P2P network
  - **advantage**: the architecture scales up nicely (at least in principle). If more users join Spotify and the servers get clogged, just add a new server (and a new P2P network)
- To keep the discussion simple, we assume there is only one server.

Spotify: Locating Peers (tracker)

- The server maintains a **tracker**, similarly to BitTorrent.
  - as opposite to other systems, however, the server **does not** keep track of all the peers who can serve each track
  - rather, it keeps a list of the ~ 20 most recent clients that **played** each track
  - clients **do not** report to the server the content of their caches!
- **Advantages**:
  - less resources on the server side
  - simplifies the implementation of the tracker
- **Drawback:** only a **fraction** of the peers can be located through the tracker
  - this is not a big issue, since clients can ask the other peers (next slide)

Spotify: Locating Peers (P2P)

- Each client is connected to a set of **neighbors** (other clients) in the P2P network.
  - these are the peers the client has previously uploaded a track to, or has previously downloaded a track from
- When a new track has to be downloaded, a client can search its neighborhood for peers that have stored the track in their cache
- The peers can, in turn, forward the search request to their own peers in the network
  - the process **stops at hop distance 2** in the overlay network
- each query has a unique ID, to allow ignoring duplicate queries

Sp🟢tify: Locating Peers (P2P)
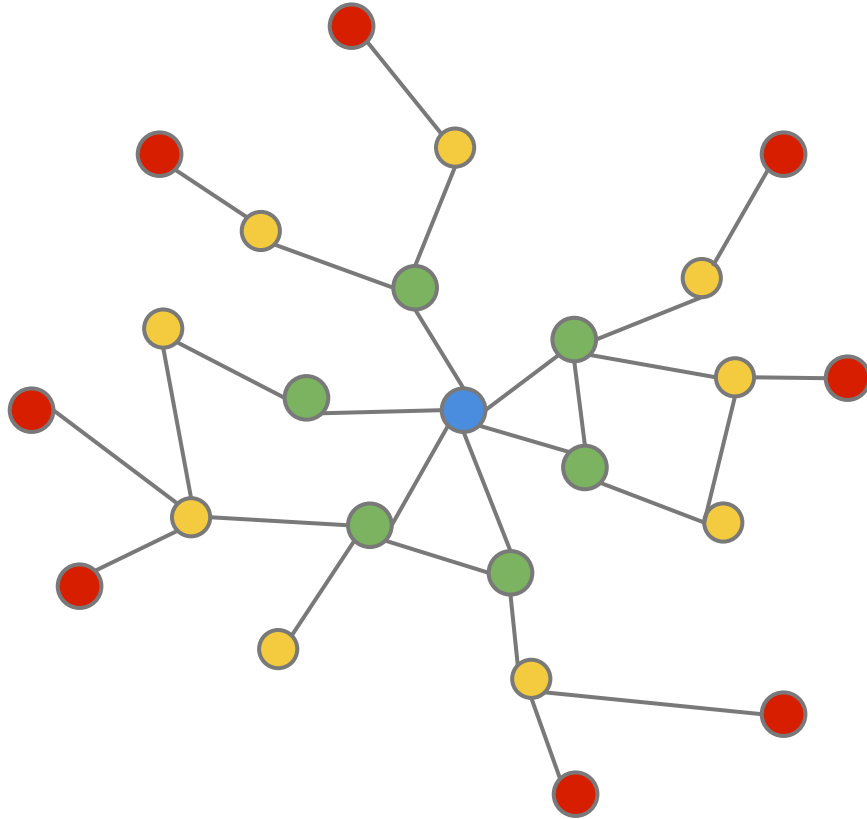
client

neighborhood

two-hops-distant peers

other peers

Spotify: Neighborhood Maintenance

- A client uploads to **at most 4 peers at any given time**
  - helps Spotify behaving nicely with concurrent application streams (*e.g.,* browsing)
- Connections to peers do not get closed after a download/upload
  - **advantage**: reduces time to discover new peers when a new track has to be played
  - **drawback**: keeping the state required to maintain a large number of TCP connections to peers is expensive (in particular for home routers acting as stateful firewall and Network Address Translation (NAT) devices)
- To keep the overhead low, clients impose both a **soft** and a **hard** limit to the number of concurrent connections to peers (set to 50 and 60 respectively)
  - when the soft limit is reached, a client stops establishing new connections to other peers (though it still accepts new connections from other peers)
  - when the hard limit is reached, no new connections are either established or accepted

Spotify: Neighbor Maintenance

- When the soft limit is reached, the client starts pruning its connections, leaving some space for new ones.
- To do so, the client computes an **utility** of each connected peer by considering, among the other factors:
    - the number of bytes sent (received) from the peer in the last 60 (respectively 10) minutes
    - the number of other peers the peer has helped discovering in the last 10 minutes
- Peers are sorted by their utility, and the peers with the least total scores are disconnected.

Spotify: Playing a Track

- The main objective is to keep the **playback latency low**
  - playback latency: time to wait before the track can be played smoothly (like buffering time on Youtube)
- Around 61% of tracks are played in a predictable order (*i.e.,* the previous track has finished, or the user has skipped to the next track)
  - playback latency can be reduced by predicting what is going to be played next.
- The remaining 39% are played in random order (*e.g.,* the user suddenly changes album, or playlist)
  - predicting what the user is going to play next is too hard. Playback latency may be higher

Spotify: Random Access

- When tracks are played in an unpredictable (random) order, fetching them just using the P2P network would negatively impact the playback delay.
- Why?
  - searching for peers who can serve the track takes time (mostly because of multiple messages need to be exchanged with each peer)
  - some peers may have poor upload bandwidth capacity (or may be busy uploading the track to some other client)
  - a new connection to a peer requires some time before start working at full rate (check out the lectures about TCP congestion control)
  - P2P connections are unreliable (*e.g.,* may fail at any time)

Spotify: Random Access

- How to solve the problem?
- Possible solution: use the *fast* Spotify Content Delivery Network (CDN)
  - **drawback**: more weight on the Spotify CDN (higher monetary cost for Spotify.. and possibly to its users too)
- <u>Better</u> solution: use the Spotify CDN <u>asking for the first 15 seconds</u> of the track only.
  - **advantage**: this buys a lot of time the client can use to search the peer-to-peer network for peers who can serve the track.
  - **advantage**: the Spotify CDN is used just to recover from a critical situation (in this case, when the user has started playing a random track)

Spotify: Sequential Access

- When users listen to tracks in a predictable order (*i.e.,* a playlist, or an album), the client has plenty of time to **prefetch** the next track before the current one finishes.
- **Problem**: you don't really know whether the user is actually going to listen to the next track or not. If the user plays a random track instead of the predicted one, you end up having wasted bandwidth resources.
- **Solution**: start prefetching the next track only when the previous track is about to finish, as Spotify has experimentally observed that:
  - when the current track has only 30 seconds left, the user is going to listen to the following one in 92% of the cases.
  - when 10 seconds are left, the percentage rises to 94%
- The final strategy is:
  - 30 seconds left: start searching for peers who can serve the next track
  - 10 seconds left: if no peers are found (critical scenario!), use the Spotify CDN

Sp🟢tify: Regular Streaming

- The client continuously monitors the **playout buffer** (*i.e.,* the portion of the song that has been downloaded so far but not already played)
- If the buffer becomes too low (< 3 seconds) the client enters an **emergency mode**, where:
  - it stops uploading to the other peers
    - this is especially useful in asymmetric connections (*e.g.,* aDSL), whose download capacity is negatively affected by concurrent uploads (check out the lectures on TCP)
  - it uses the Spotify CDN
    - this helps in the case the client fails to find a reliable and fast set of peers to download the chunks from

Spotify: Regular Streaming

- Tracks are split in 16KB **chunks**.
- A track can be simultaneously downloaded from the CDN and the P2P network.
- If both CDN and P2P are used, the client never downloads from the Spotify CDN more than 15 seconds ahead of the current playback point.
- To select the peers to request the chunks from, the client sorts them by their expected download times and **greedily** requests the most urgent chunk from the top peer.
  - expected download times are computed using the average download speed received from the peers
  - if a peer happens to be too slow, another peer is used

Spotify: Conclusions

- Spotify is a nice example of modern system for content distribution
    - it uses a CDN for centralized content delivery (recently switched to Amazon Cloudfront, a relatively new competitor of Akamai)
    - its data centers are backed up by Amazon S3 (a popular choice of many other systems too, such as Dropbox)
- Mixing a centralized and a P2P network helps keeping the monetary cost **low** (bandwidth does not come for free!)
- The P2P network is very simple, thanks to the extremely efficient CDN that backs it up
- A few good key design choices help getting the most of the P2P network, limiting the typical problems that may affect it (*e.g.,* latency, reliability)

Spotify: Conclusions

- Spotify probably keeps evolving as the number of users increase. *e.g.,* it recently switched to Amazon CloudFront:
    - https://aws.amazon.com/solutions/case-studies/spotify/,
    - https://d36cz9buwru1tt.cloudfront.net/aws-media-summit-2011/aws_spotify_summit_pavley_SW_2.pdf
- As any closed system, it is hard to get a clear and up-to-date view of its internals. As a consequence, some of the internals we presented in this overview may have been changed. Still, is a very interesting case study, from which a number of lessons can be learned
- A more complete technical overview of Spotify, and other interesting studies can be found on G. Kreitz's homepage: http://www.csc.kth.se/~gkreitz/

Want to know more?

Incentives Build Robustness in BitTorrent

Bram Cohen
bram@bitconjurer.org

May 22, 2003

Abstract

The BitTorrent file distribution system uses tit-for-tat as a method of seeking pareto efficiency. It achieves a higher level of robustness and resource utilization than any currently known cooperative technique. We explain what BitTorrent does, and how economic methods are used to achieve that goal.

each peer's download rate be proportional to their upload rate. In practice it's very difficult to keep peer download rates from sometimes dropping to zero by chance, much less make upload and download rates be correlated. We will explain how BitTorrent solves all of these problems well.

1.1 BitTorrent Interface

Incentives Build Robustness in BitTorrent, **Bram Cohen**
Workshop on Economics of Peer-to-Peer Systems, June 2003

Want to know more?

- A journey inside BitTorrent:
  https://www.youtube.com/watch?v=GTDRgzuW_No

- Technical seminar by Bram Cohen on **BitTorrent Live**:
  https://www.youtube.com/watch?v=VfbRhSrJ4qA
- A Measurement Study of the **Wuala** On-line Storage Service:
  http://www.eurecom.fr/en/publication/3772/download/rs-publi-3772.pdf
- Kademilia a Peer-to-peer Information System based on the XOR metric:
  http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf