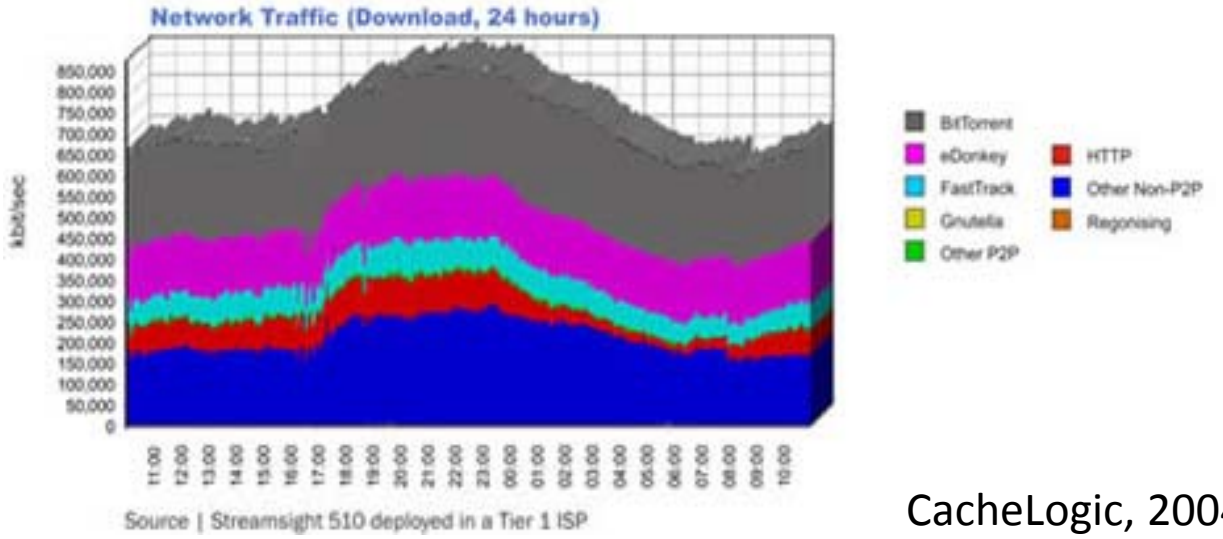
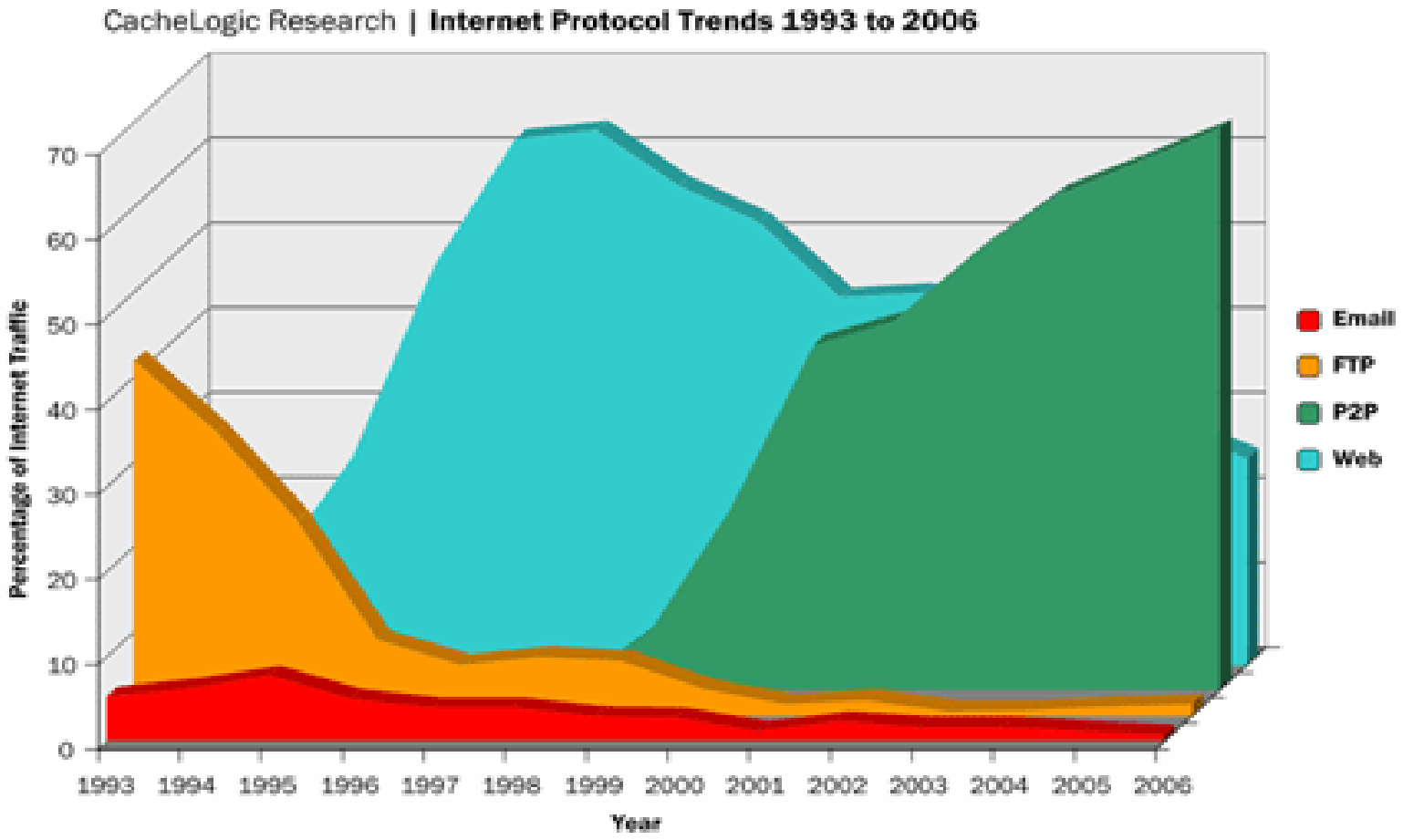


CacheLogic, 2004

PEER-TO-PEER APPLICATIONS

More data
(Source: CacheLogic)



Spotify - Large Scale, Low Latency, P2P Music-on-Demand Streaming

Gunnar Kreitz, Fredrik Niemelä

IEEE P2P'10

Following slides are adapted from authors' slides at P2P in 2010 & 2011.

SPOTIFY

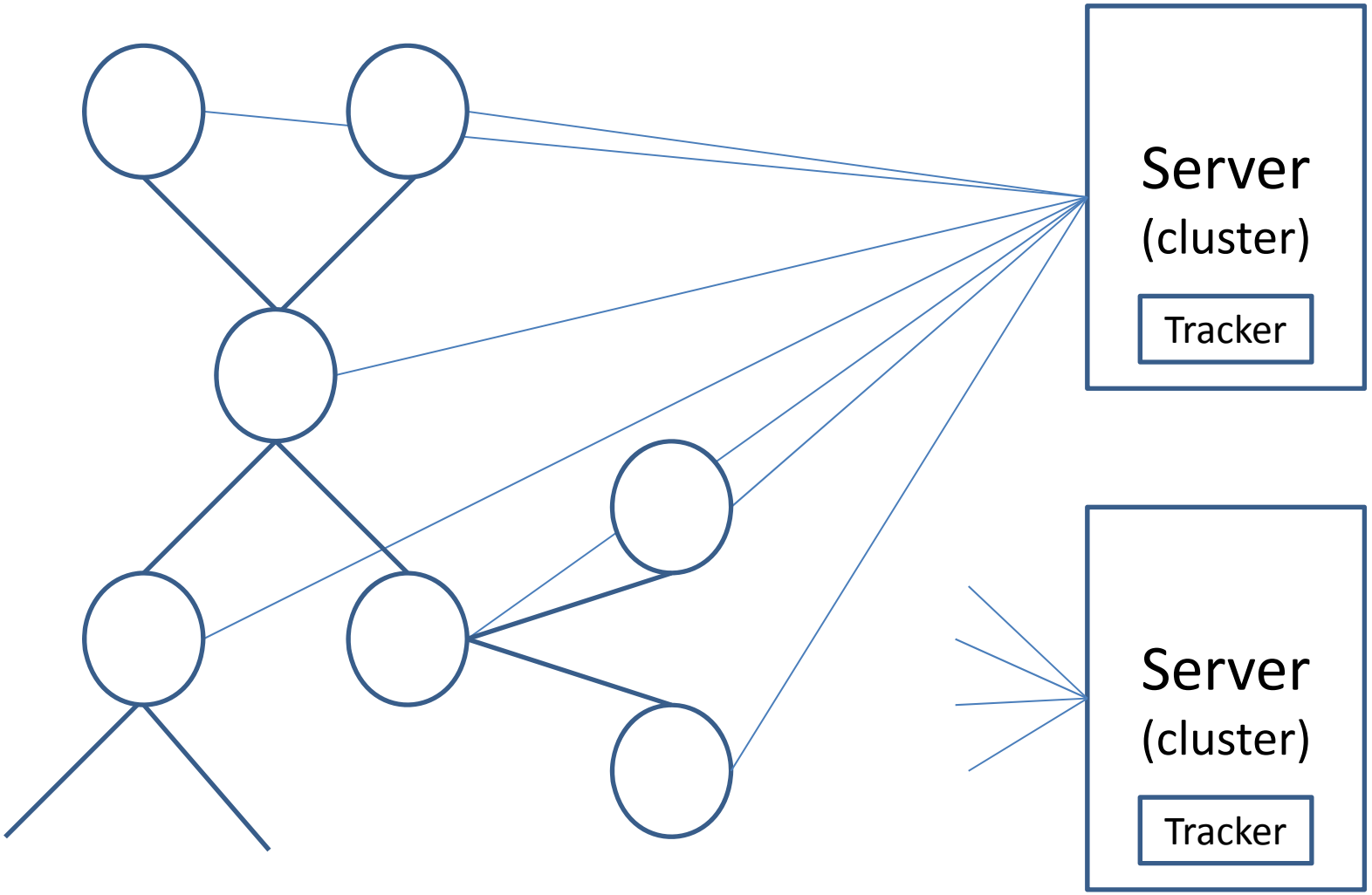
Spotify.com, 2004

- Commercially deployed system, KTH start-up (Sweden)
- Peer-assisted on-demand music streaming
- Legal and licensed content, only
- Large catalogue of music (over 15 million tracks)
- Available in U.S. & 7 European countries, over 10 million users, 1.6 million subscribers (in 2004)
- Fast (median playback latency of 265 ms)
- Proprietary client software for desktop & phone (not p2p)
- Business model: Ad-funded and free & monthly subscription, no ads, premium content, higher quality streaming

Overview of Spotify Protocol

- Proprietary protocol
- Designed for on-demand music streaming
- Only Spotify can add tracks
- 96–320 kbps audio streams (most are Ogg Vorbis q5, 160 kbps)
- Relatively simple and straightforward design
- Phased out in 2014: "*We're now at a stage where we can power music delivery through **our** growing number of **servers** and ensure our users continue to receive **a best-in-class service***."
- Conclusion: *Commercially*, P2P technology is good for **startups** who demand more resources than their servers offer. Avoid "***death by success***".

Spotify architecture: Peer-assisted



Why a Peer-to-peer Protocol?

- **Improve scalability** of service
- **Decrease load** on servers and network resources

- Explicit design goal
  - Use of peer-to-peer should not decrease overall performance (i.e., playback latency & stutter)

Peer-to-peer Overlay Structure

- Unstructured overlay
- Does not use a DHT
  - **Fast lookup required (Hybrid p2p)**
  - Let us do some rough estimates (ping times)
    - Latency UK – Netherlands ~ 10 ms and up
    - Latency across EU more like ~ 80 ms and up
    - Latency US – Europe ~ 100 ms and up
    - Playback latency ~265 ms
    - <1% Playbacks have stuttering
  - **Simplicity of protocol design & implementation**

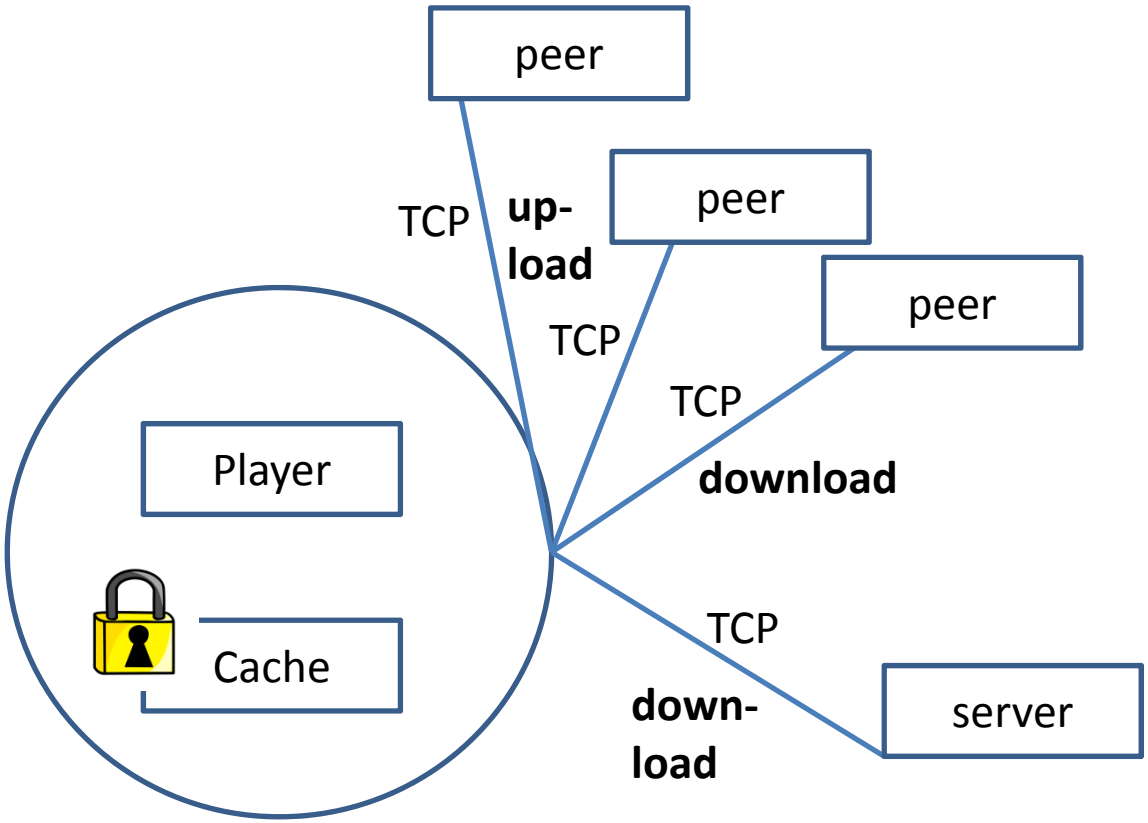
Peer-to-peer Overlay Structure

- Nodes have fixed maximum degree (60)
- Neighbour eviction by heuristic evaluation of utility
- Looks for and connects to new peers when streaming new track
- Overlay becomes (weakly) clustered by interest
- Client only downloads data user needs

Finding Peers

- Server-side tracker (cf. BitTorrent)
  - Only remembers 20 peers per track
  - Returns 10 (online) peers to client on query
- Clients broadcast query in small (2 hops) neighbourhood in overlay (cf. Gnutella)
- Client uses both mechanisms for every track

Peers


peer
peer
peer
TCP
up-
load
TCP
TCP
download
Player
Cache
TCP
down-
load
server

Protocol

- (Almost) everything encrypted
- (Almost) everything over TCP
- Persistent connection to server while logged in
- Multiplex messages over a single TCP connection

Caches

- Client (player) caches tracks it has played
- Default policy is to use 10% of free space (capped at 10 GB)
- Caches are often larger (56% are over 5 GB)
- Least Recently Used policy for cache eviction
- Over 50% of data comes from local cache
- Cached files are served in peer-to-peer overlay (if track completely downloaded)

Streaming a Track

- Tracks are decomposed into 16 kB chunks
- Request first chunk of track from Spotify servers
- Meanwhile, search for peers that cache track
- Download data in-order (chunk by chunk via TCP)
- Towards end of a track, start prefetching next track

Streaming a Track

- If a remote peer is slow, re-request data from new peers
- If local buffer is sufficiently **filled**, only download from peer-to-peer overlay
- If **buffer** is getting **low**, download from central server as well
  - Estimate at what point p2p download could resume
- If **buffer** is **very low**, stop uploading

Security Through Obscurity, ☹

- Music data lies encrypted in caches
- Client must be able to access music data
- Reverse engineers should not be able to access music data
- Details are secret and client code is obfuscated

- **Do not do this "at home"**
  - ***Security through obscurity*** is a bad idea
  - It is a matter of time until someone hacks the Spotify client (cf. the various Skype reverse engineering efforts)

Data sources: 8.8% from servers, 35.8% from p2p network, 55.4 % from caches

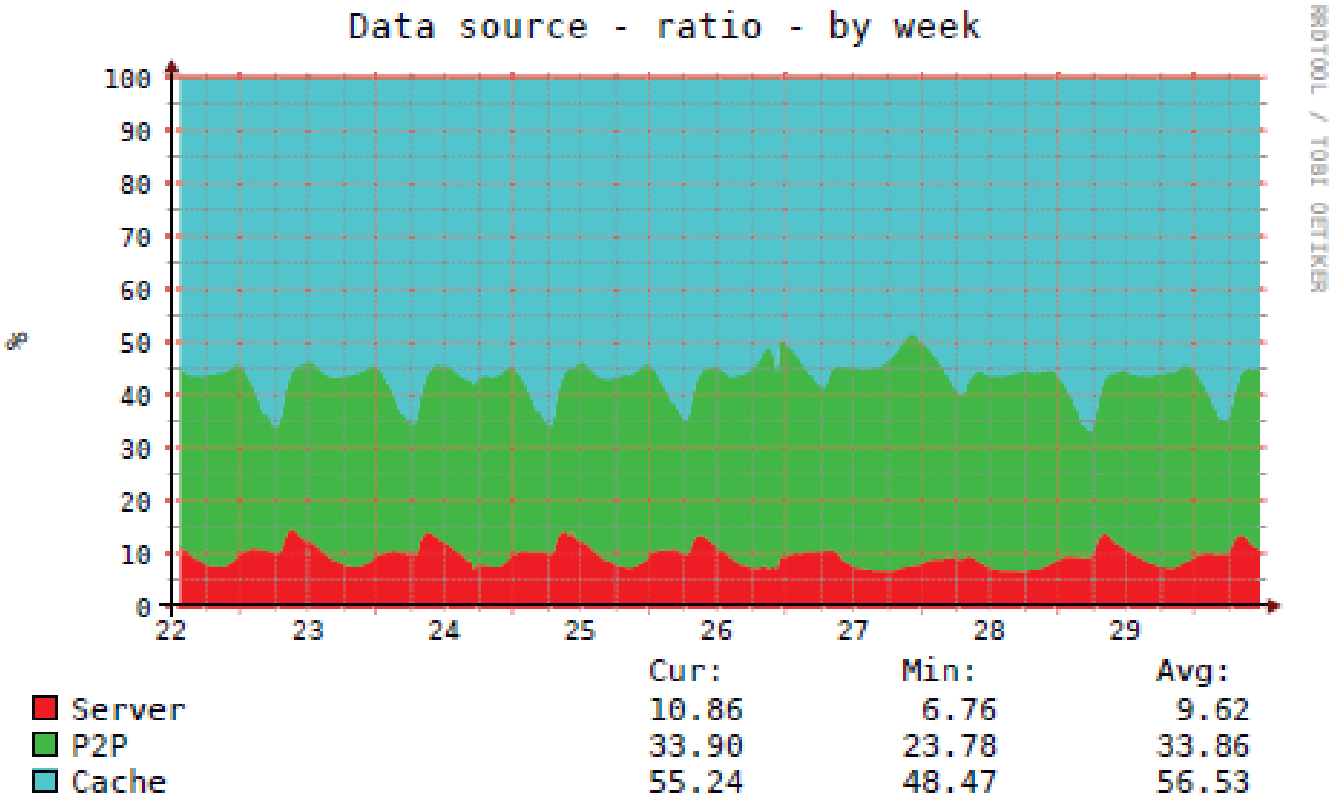
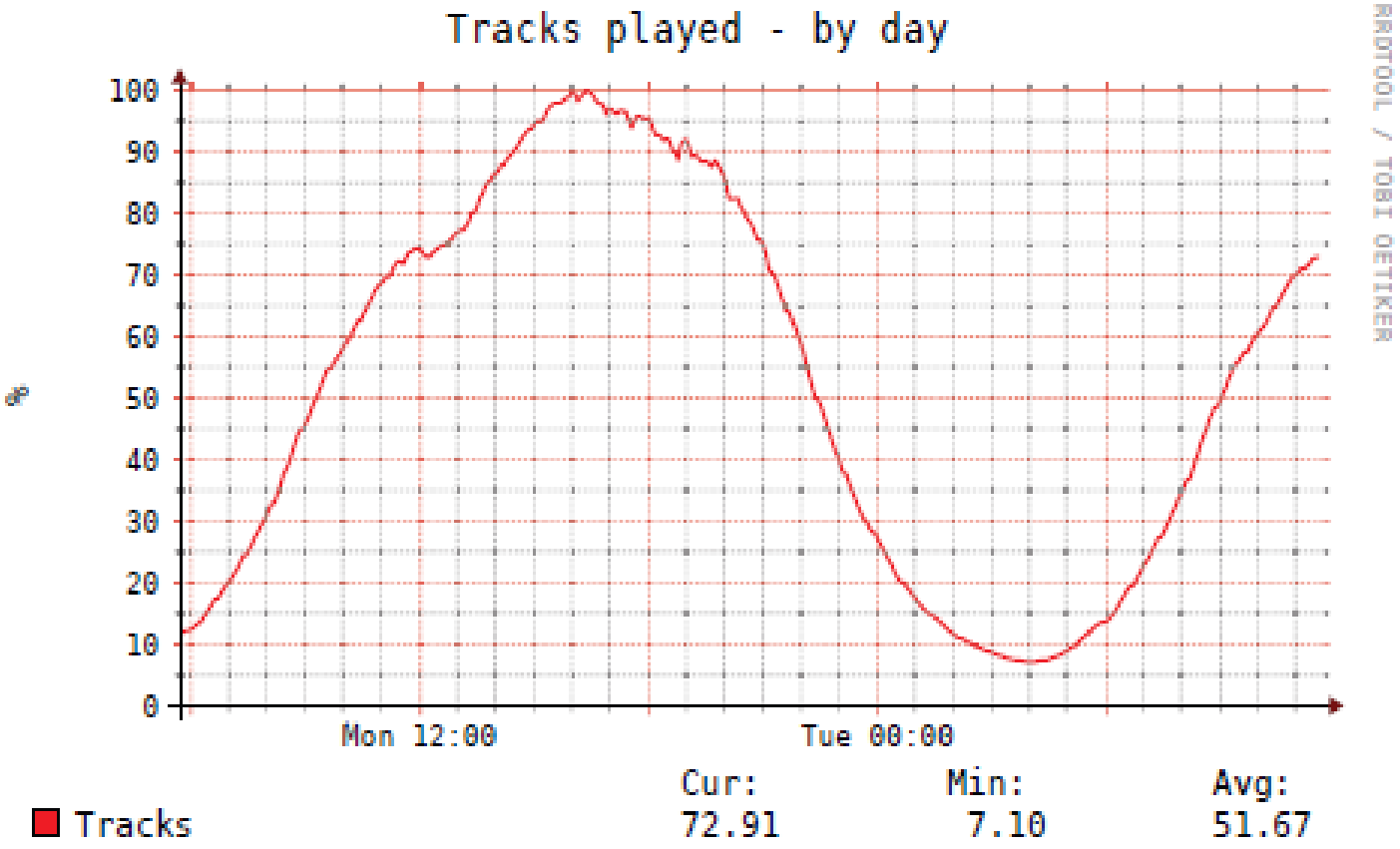


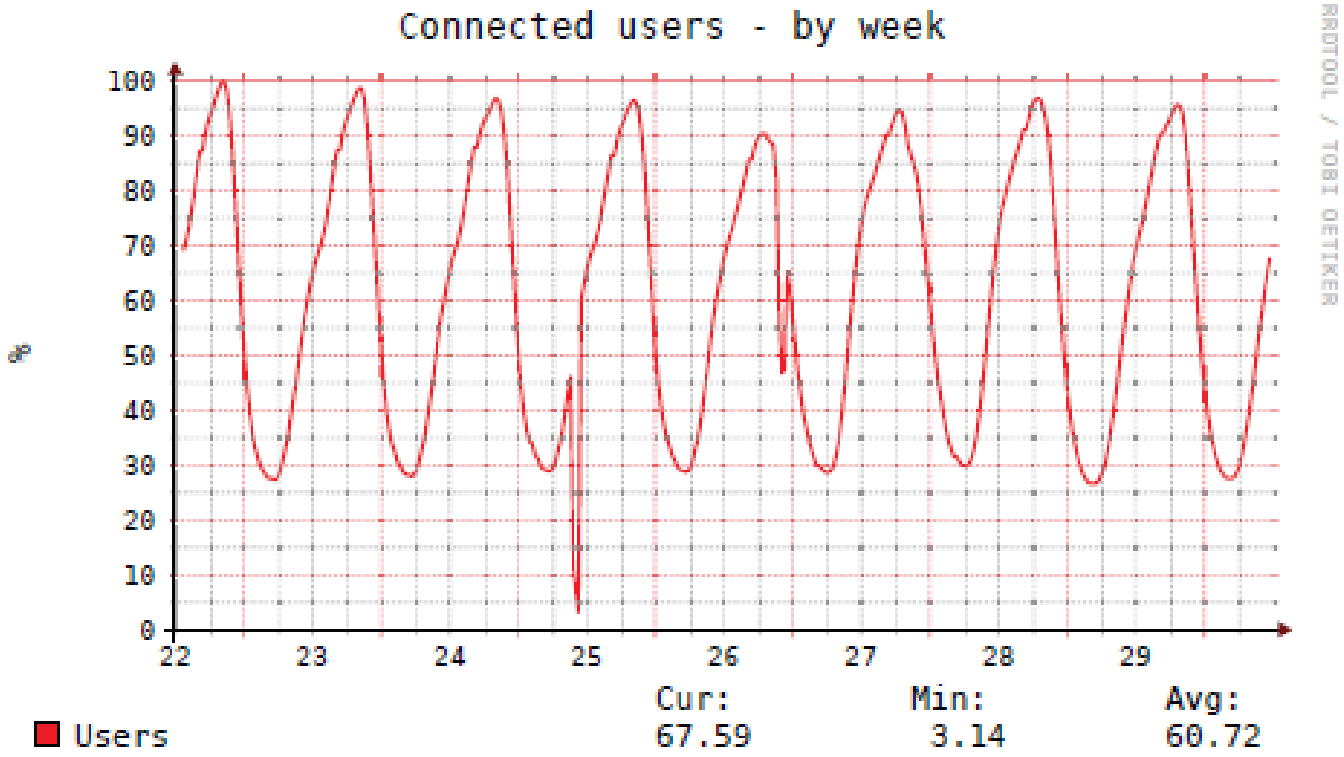
Figure 2. Sources of data used by clients

Tracks played



(a) Tracks played

Users connected



(b) Users connected

Key Points

- Simplicity of architecture, protocol, design
- Peer-assisted, i.e., rely on centralized server
- Use of peer-to-peer techniques for scalability and avoid heavy, over-provisioned infrastructure
- Use of centralized tracker