Spotify — Behind the Scenes

Gunnar Kreitz

Spotify
gkreitz@spotify.com

KTH, September 29 2011

What is Spotify?

- Lightweight on-demand streaming
- Large catalogue, over 15 million tracks
- Available in US and 7 European countries
- Over 10 million users across Europe, over 2 million subscribers
- Fast (median playback latency of 265 ms)
- Legal

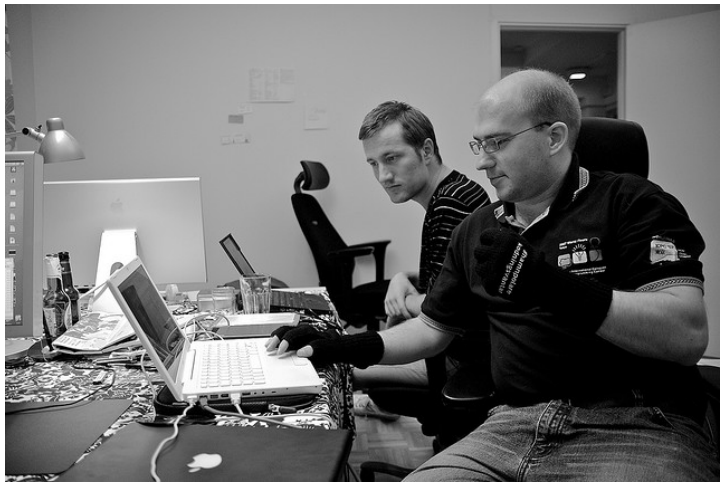
Business Idea

- More convenient than piracy

Business Idea

- More convenient than piracy
- Spotify Free (ads, 10h/month after 6 months)
- Spotify Unlimited (no ads, on computer)
- Spotify Premium (no ads, mobile, offline, API)

Spotify Tech Team



- Most developers in Stockholm
- Very talented people
- Proud of the product
- Team size: > 100
- We're growing fast and hiring!

Development Environment

- Scrum methodology with three week sprints
- Some cross-functional teams, some specialized project teams
- Kanban for some teams
- Scrum teams consist of programmers, testers and designers
- Hack days

Technical Design Goals

- Available
- Fast
- Scalable
- Secure

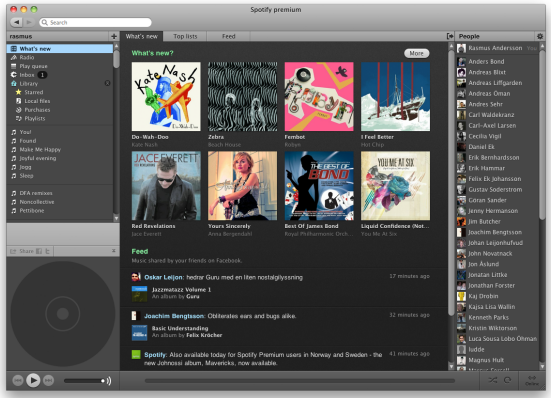
The Importance of Being Fast

- How important is speed?
- Increasing latency of Google searches by 100 to 400ms decreased usage by 0.2% to 0.6% [Brutlag09]
- The decreased usage persists
- Median playback latency in Spotify is 265 ms (feels immediate)

The forbidden word

(By http://www.flickr.com/photos/marxalot/, CC BY-SA 2.0)

Client Software



- Desktop clients on Linux (preview), OS X and Windows
  - Windows version works well under Wine
- Smartphone clients on Android, iOS, Palm, Symbian, Windows Phone
- libspotify on Linux, OS X and Windows
- Sonos, Logitech, Onkyo, and Telia hardware players
- Mostly in C++, some Objective-C++ and Java

Client Software vs. Web-based

- Web-based applications are easier to update and maintain
- Web-based don't need to be installed
- Client software still gives better user experience
- Volume control, separate application, faster
- Auto-upgrades eases parts of installation pain

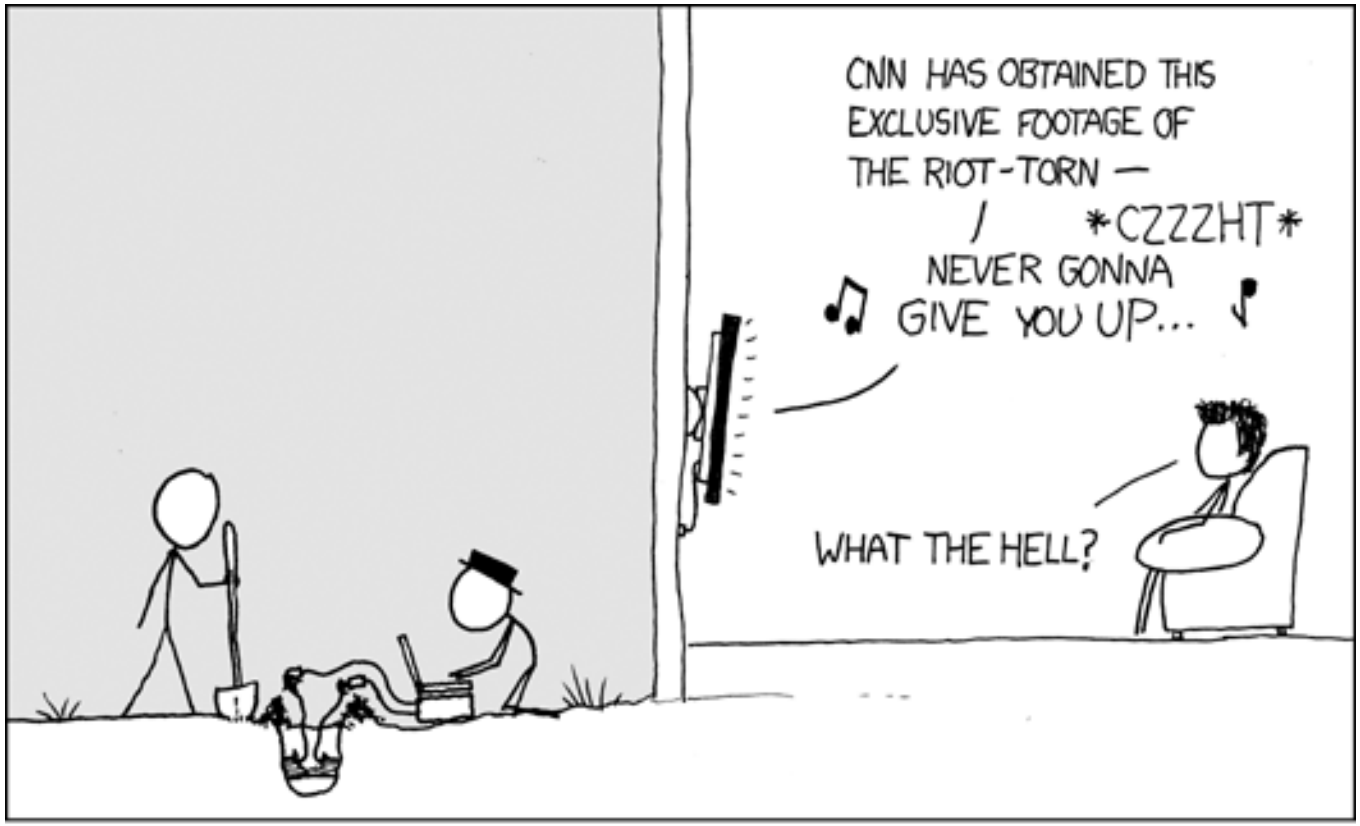
Everything is a link

- spotify: URI scheme
- `spotify:track:6JEK0CvvjDjjMUBFoXShNZ#0:44`
- `spotify:user:gkreitz:playlist:4W5L19AvhsGC3U9xm6lQ9Q`
- `spotify:search:never+gonna+give+you+up`

Everything is a link

- spotify: URI scheme
- `spotify:track:6JEK0CvvjDjjMUBFoXShNZ#0:44`
- `spotify:user:gkreitz:playlist:4W5L19AvhsGC3U9xm6lQ9Q`
- `spotify:search:never+gonna+give+you+up`
- New URI schemes not universally supported
- `http://open.spotify.com/track/6JEK0CvvjDjjMUBFoXShNZ#0:44`

Links contain opaque id:s



(Image from XKCD, `http://www.xkcd.com/351`)

Metadata API

- Simple, http-based API
- Search and lookup
- `http://ws.spotify.com/lookup/1/?uri=spotify:track:6JEK0CvvjDjjMUBFoXShNZ`
- `http://ws.spotify.com/search/1/artist?q=foo`
- Developer resources: `http://developer.spotify.com/`

Overview of Spotify Protocol

- Proprietary protocol
- Designed for on-demand streaming
- Only Spotify can add tracks
- 96–320 kbps audio streams (most are Ogg Vorbis q5, 160 kbps)
- Peer-assisted streaming

Spotify Protocol

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

Caches

- Player caches tracks it has played
- Default policy is to use 10% of free space (capped at 10 GB)
- Caches are large (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 P2P overlay

Streaming a Track

- Request first piece from Spotify servers
- Meanwhile, search Peer-to-peer (P2P) for remainder
- Switch back and forth between Spotify servers and peers as needed
- Towards end of a track, start prefetching next one

TCP Congestion Window

- TCP maintains several windows, among them `cwnd`
- `cwnd` is used to avoid network congestion
- A TCP sender can never have more than `cwnd` un-ack:ed bytes outstanding
- Additive increase, multiplicative decrease

TCP Congestion Window

- TCP maintains several windows, among them `cwnd`
- `cwnd` is used to avoid network congestion
- A TCP sender can never have more than `cwnd` un-ack:ed bytes outstanding
- Additive increase, multiplicative decrease
- What to do with `cwnd` when a connection sits idle?
- RFC 5681 (TCP Congestion Control) says:

> *Therefore, a TCP SHOULD set cwnd to no more than RW before beginning transmission if the TCP has not sent data in an interval exceeding the retransmission timeout.*

TCP Congestion Window and Spotify

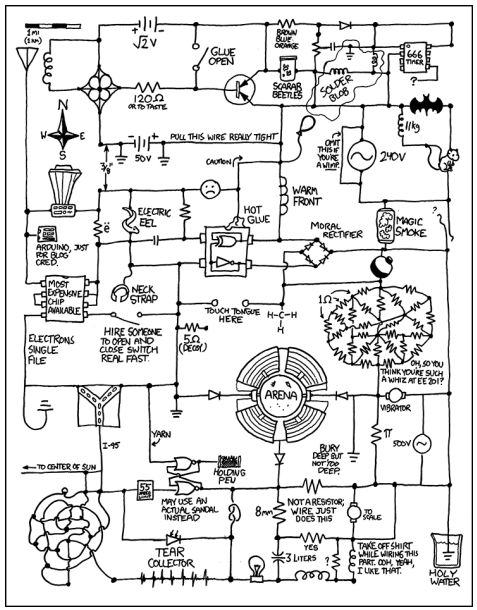
- Spotify traffic is bursty
- Initial burst is very latency-critical
- Want to avoid needless reduction of congestion window
- Configure kernels to not follow the RFC 5681 SHOULD.

When to Start Playing?

- Minimize latency while avoiding stutter
- TCP throughput varies
  - Sensitive to packet loss
  - Bandwidth over wireless mediums vary
- Model throughput as a Markov chain and simulate
- Heuristics

Image by ronin691 http://www.flickr.com/photos/ronin691/3482770627, CC BY SA 2.0

Security Through Obscurity



- Client must be able to access music data
- Reverse engineers should not
- So, we can't tell you exactly how our client works
- Plus, we need to apply software obfuscation

Image by XKCD http://xkcd.com/730/, CC BY NC 2.5

Security Through Obscurity

(Image from XKCD, `http://www.xkcd.com/257`)

P2P Goals

- Easier to scale
- Less servers
- Lass bandwidth
- Better uptime
- Fun!

Music vs. Movies

Music

- Small (5 minutes, 5 MB)
- Many plays/session
- Large catalog
- Active users

Movies

- Large (2 hours, 1.5 GB)
- High bit rate

Music vs. Movies

Music

- Small (5 minutes, 5 MB)
- Many plays/session
- Large catalog
- Active users

Main problem: peer discovery

Movies

- Large (2 hours, 1.5 GB)
- High bit rate

Main problem: download strategy

P2P Structure

- Unstructured network (not a Distributed Hash Table)
- Edges are formed as needed
- Nodes have fixed maximum degree (60)
- No overlay routing
- Neighbor eviction by heuristic evaluation of utility

P2P Structure

- All peers are equals (no supernodes)
- A user only downloads data she needs
- P2P network becomes (weakly) clustered by interest
- Oblivious to network architecture

Brief Comparison to BitTorrent

- One (well, three) P2P overlay for all tracks (not per-torrent)
- Does not inform peers about downloaded blocks
- Downloads blocks in order
- Does not enforce fairness (such as tit-for-tat)
- Informs peers about urgency of request

Finding Peers

- Sever-side tracker (BitTorrent style)
  - Only remembers 20 peers per track
  - Returns 10 (online) peers to client on query
- Broadcast query in small (2 hops) neighborhood in overlay (Gnutella style)
- LAN peer discovery (cherry on top)
- Client uses all mechanisms for every track

Downloading in P2P

- Ask for most urgent pieces first
- If a peer is slow, re-request from new peers
- When buffers are low, download from central server as well
  - When doing so, estimate what point P2P will catch up from
- If buffers are very low, stop uploading

Limit resource usage

- Cap number of neighbors
- Cap number of simultaneous uploads
  - TCP Congestion Control gives “fairness” between connections
- Cap cache size
- Mobile clients don’t participate in P2P

P2P NAT Traversal

- Asks to open ports via UPnP
- Attempt connections in both directions
- High connection failure rate (65%)
- Room for improvement

Security in our P2P Network

- Control access to participate
- Verify integrity of downloaded files
- Data transfered in P2P network is encrypted
- Usernames are not exposed in P2P network, all peers assigned pseudonym

Avoiding hijacking

- A peer cannot ask peers to connect to arbitrary IP address/port
  - Avoiding DDoS issues
- Misbehaving peers are reported

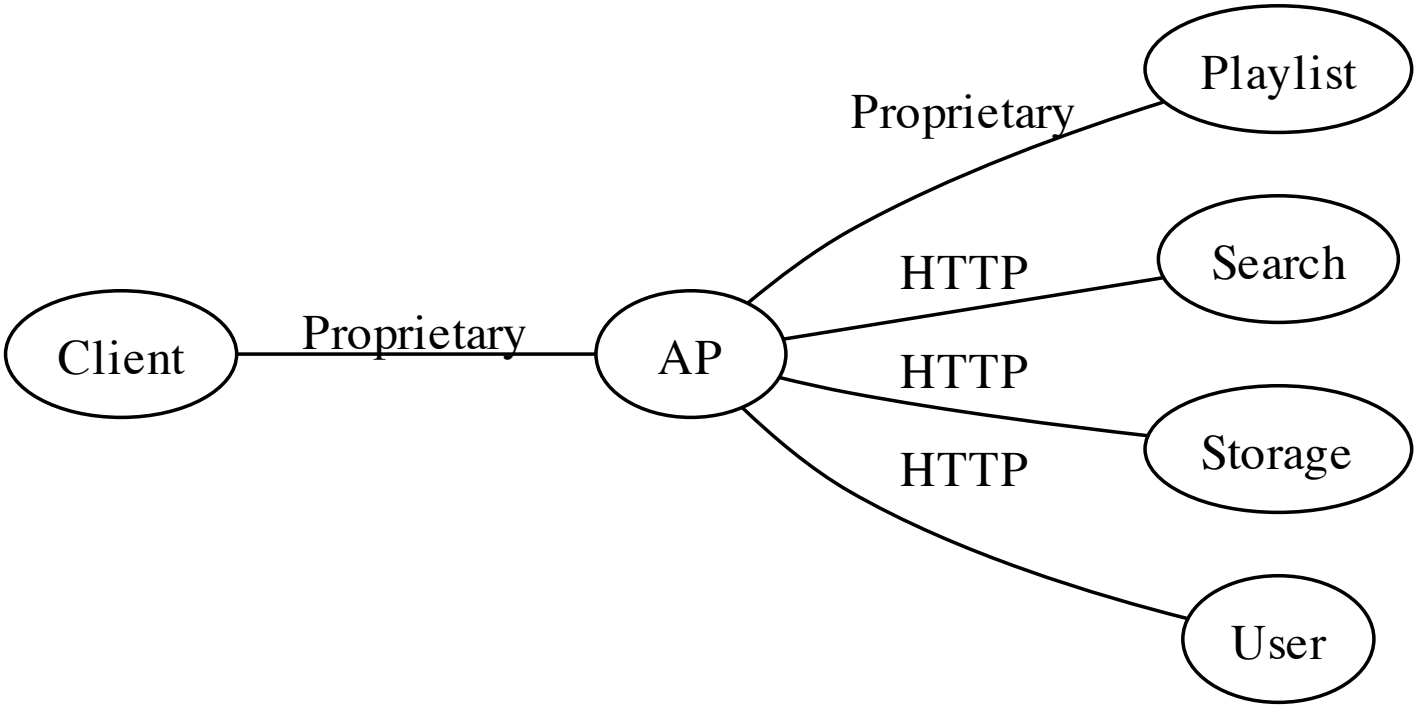
Back End Software

- Comprised of many small services
  - Do one task and do it well
- Python, C++, Java, Scala
- No common framework (yet)

High-level overview

- Client connects to an *Access Point* (AP)
- AP handles authentication and encryption
- AP demultiplexes requests, forwards to backend servers
- Gives redundancy and fault-tolerance

High-level overview (cont'd)



Locating an Access Point

- DNS SRV lookup of `_spotify-client._tcp.spotify.com`
- GeoDNS to return access point close to you
- Fallback to A record for `ap.spotify.com`
- Seeing problems with large responses (TCP DNS in home routers)

Communcating with backend servers

- Most common backend protocol is HTTP
- Some services need to push information, e.g. playlist
  - Currently, each such service has its own protocol
  - Moving towards a more unified backend protocol

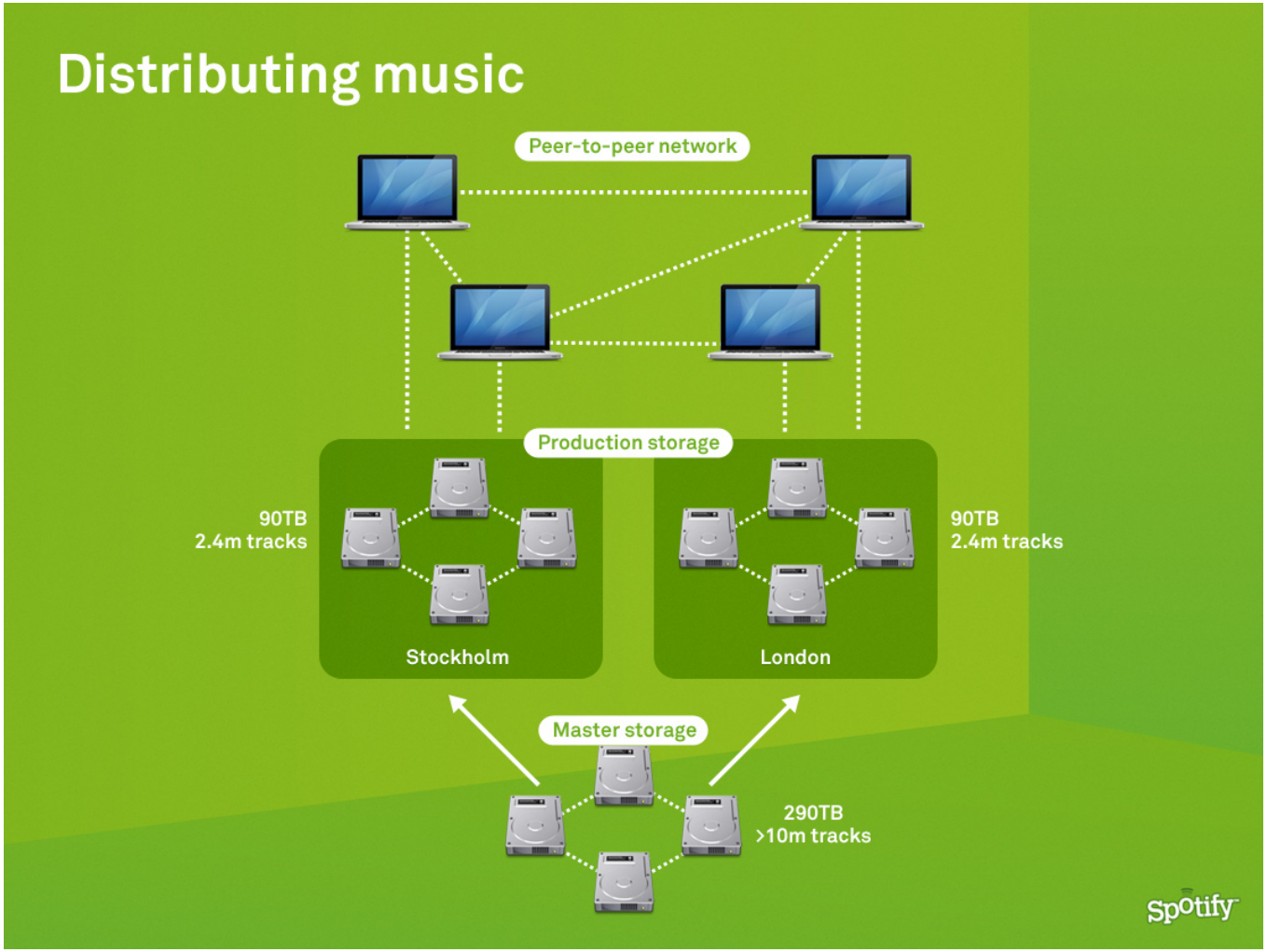
Lookup, version 1

- Put content on random servers
- Multicast UDP to find server
- Each server has a small daemon with an index, responding to lookup queries
- Scaling issues

Lookup, version 2

- DNS-based (using TXT records) Consistent Hashing
- Each client knows entire keyspace
- Each server handles parts of keyspace
- Hash key to find master server
- Repeated hashing to find slaves

Storage


Distributing music
Peer-to-peer network
Production storage
90TB
2.4m tracks
Stockholm
London
90TB
2.4m tracks
Master storage
290TB
>10m tracks
Spotify

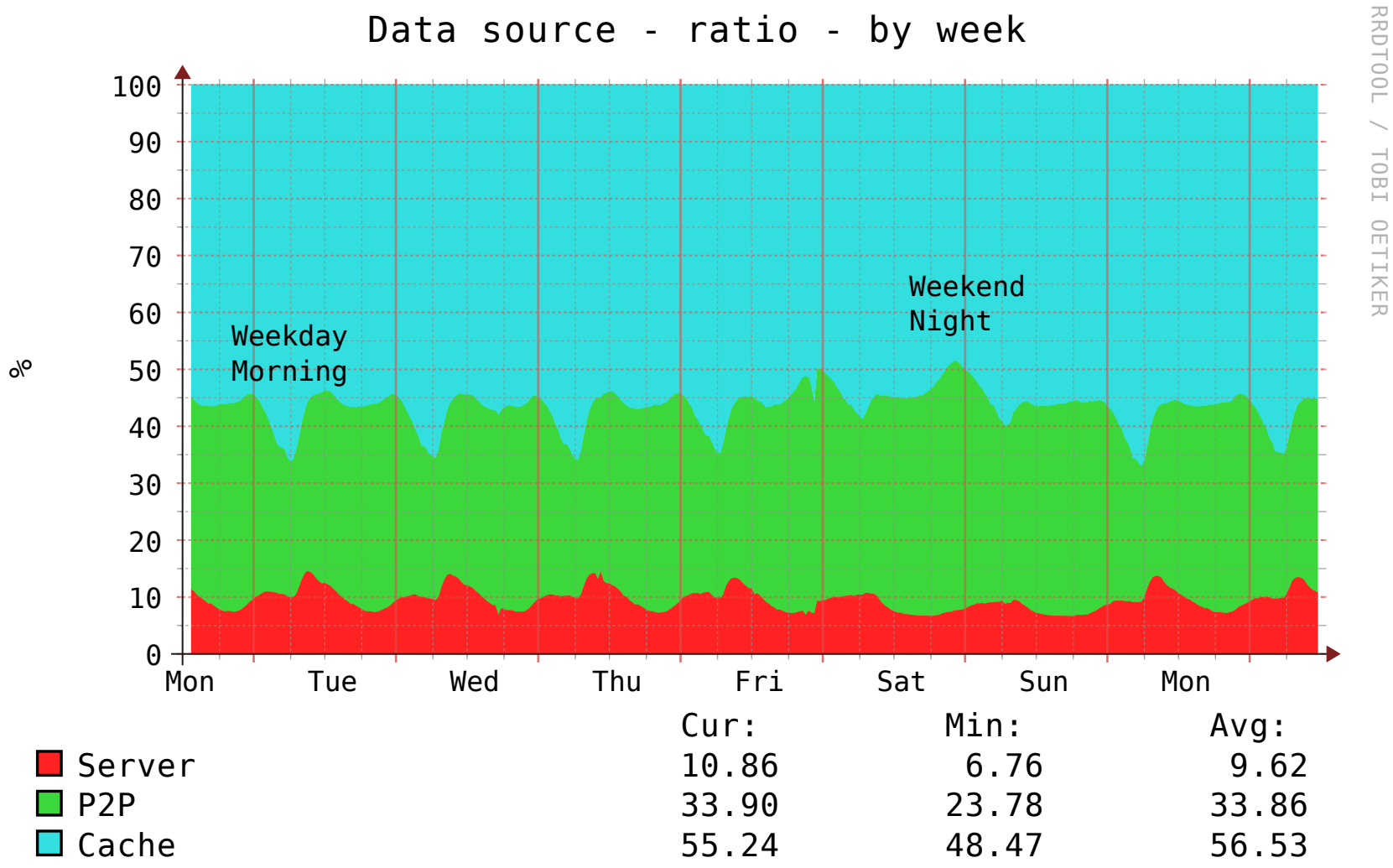
Playlist

- Our most complex service (!)
- Simultaneous writes with automatic conflict resolution
- Publish-subscribe system to clients
- Changes automatically versioned, transmits deltas
- Terabyte sizes of data

Evaluation

- So, how well does it work?
- Collected measurements 23–29 March 2010
- (Before Facebook integration, local files, . . .)

Data Sources



Data Sources

- Mostly minor variations over time
  - Better P2P performance on weekends
  - P2P most effective at peak hours
- 8.8% from servers
- 35.8% from P2P
- 55.4% from caches

Latency and Stutter

- Median latency: 265 ms
- 75th percentile: 515 ms
- 90th percentile: 1047 ms
- Below 1% of playbacks had stutter occurrences

Finding Peers

Table: Sources of peers

Sources for peers	Fraction of searches
Tracker and P2P	75.1%
Only Tracker	9.0%
Only P2P	7.0%
No Peers Found	8.9%

- Each mechanism by itself is fairly effective

Protocol Overhead

Table: Distribution of application layer traffic in overlay network

Type	Fraction
Music Data, Used	94.80%
Music Data, Unused	2.38%
Search Overhead	2.33%
Other Overhead	0.48%

- Measured at socket layer
- Unused data means it was cancelled/duplicate

More measurements

- Recently, we investigated more general network properties
- How many behind NATs? How many with UPnP support?
- How many IPs does each user connect form over a week?
- Does this vary between weekdays and weekends?
- Does this vary between countries?
- See our P2P'11 paper for data and details

Thank you! Questions?

gkreitz@spotify.com