Apple tv – business canvas

Business

Music-on-Demand Streaming Gunner Kermit KITH - Royal Institute ofTechnology, and Spottily Stockholm, Sweden Email:SE Abstract? Spottily is a music streaming service offering low- latency access to a library of over 8 million music tracks. Streaming is performed by a combination of client- server access and a peer-to-peer protocol. In this paper, we give an overview of the protocol and peer-to-peer architecture used and provide measurements of service performance and user behavior.

The service currently has a user base of over 7 lion and has been available in six European countries since October 2008. Data collected indicates that the combination of the client-server and peer-to-peer paradigms can be applied to music streaming with good results. In particular, 8. 8% of music data played comes from Spottily servers while the median playback latency is only 265 ms (including cached tracks). We also discuss the user access patterns observed and how the peer-to-peer network affects the access patterns as they reach the server. . Spottily is a streaming music service using peer-to-peer techniques. The service has a library of over 8 million tracks, allowing users to freely choose tracks they wish to listen to and to seek within tracks. Data is streamed from both servers and a peer-to- peer network. The service launched in October 2008 and now has over 7 million users in six European countries. The service is offered in two versions: a free version withadvertisement, and a premium, pay-per-month, version.

The premium version includes some extra features such as the option to stream music at a higher vitiate, and to synchronize playbills for offline usage. Both versions of the service allow unlimited streaming, and a large majority of users are on the free version. The music catalog is the same for

both free and premium users with the exception of some pre- releases exclusive to premium users. However, due to licensing restrictions, the tracks accessible to a user depends on the user's home country. One of the distinguishing features of the Spottily client is its low playback latency.

The median latency to begin playback of a track is 265 ms. The service is not web-based, but instead uses a proprietary client and protocol. A. Related Services There are many different on-demand music streaming seer- vices offered today. To our knowledge, all such services but Spottily are web-based, using either Adobe Flash or a web browser plug-in for streaming. Furthermore, they are pure client-server applications without a peer-to-peer component. Freddie Enamel¤SE Among the more well-known such services are Anapest, Reap- Soddy, and Wee.

The application of peer-to-peer techniques to on-demand streaming is more prevalent when it comes to video-on- demand services. Such services include Jots, Applied, and Upstream. These vary between supporting live streaming (us- ally of a large number of channels), video-on-demand access, or both. While there are many similarities between video-on- demand and music-on-demand streaming, there are also many Territories; Including user Demeanor, ten size AT streaming dejects, Ana ten mummer of objects offered for streaming.

A service offering on-demand streaming has many things in common with file-sharing applications. For instance, the mach- animism for locating peers in Spottily are similar to techniques from Bitterroot and Neutral. B. Related Work Leveraging the scalability of peer-to-peer networks to per- form media

streaming is a well-studied area in theacademicliterature. Most such systems are concerned with eve stream- inning, where viewers watch the same stream simultaneously. This setting is different in nature from the Spottily application, where a user has on-demand access to a large library of tracks.

The peer-to-peer live streaming literature can be roughly divided into two general approaches [1]: tree-based (e. G. [2]), and mesh-based (e. G. [3], [4]), depending on whether they maintain a tree structure in the overlay. While both techniques have their advantages, intuitively it seems that a mesh structure is a better fit for on-demand streaming applications. There have been several studies assuring the perform- mange and behavior of large peer-to-peer systems, describing and measuring both on-demand streaming [5], [6], and file- sharing [7], [8] protocols.

We believe that there is high value in understanding how peer-to-peer techniques perform in today's networks. Human et al. [5] describe the Applied video-on-demand streaming system, and also present measurements on its per- performance. To the best of our knowledge, their work is the only other detailed study of an on-demand streaming system of this size, with a peer-to-peer component. While there are naturally many similarities between Applied and Spottily, there are also many preferences, including the overlay structure and Spottily focus on low latency techniques. C.

Our Contribution In this paper we give an in-depth description and evaluation of Spottily. We discuss the general streaming protocol in Section II, and go into more details on the peer-to- peer parts in Section Ill. Furthermore, in

Section 'V, we present and comment on data gathered by Spottily while operating its service. We give detailed measurements on many aspects of the service such as latency, stutter, and how much data is offloaded from the server by the peer-to-peer protocol. Some measurement data is also resented in Sections II and Ill; that data was collected as described in Section IV-A. II.

SPOTTILY OVERVIEW The Spottily protocol is a proprietary network protocol designed for streaming music. There are clients for SO X and Windows as well as for several smartened platforms. The Windows version can also be run using Wine. The smartened clients do not participate at all in the peer-to-peer protocol, but only stream from servers. Since the focus of this paper is the evaluation of peer-to-peer techniques we will ignore the smartened clients in the remainder of this paper. The clients are closed-source footwear available for free download, but to use a client, a Spottily user account is needed.

Clients automatically update themselves, and only the most recent version is allowed to access the service. The user interface is similar to those found in desktop MPH players. Users can organize tracks into playbills which can be shared with others as links. Finding music is organized around two concepts: searching and browsing. A user can search for tracks, albums, or artists, and she can also browse? for instance, when clicking on an artist name, the user is presented with a page displaying all alludes Teetering Tanat artist.

Loll streams are encomia using cog Overdo Walt a default quality of sq, which has variable vitiate averaging roughly 160 Kbps. Users with a premium

subscription can choose (through a client setting) to instead receive Cog Boris in sq quality, averaging roughly 320 Kbps. Both types of files are served from both servers and the peer-to-peer network. No re- encoding is done by peers, so a peer with the sq version of a track cannot serve it to one wanting the sq version. When playing music, the Spottily client monitors the sound card buffers.

If the buffers under, the client considers a stutter to have occurred. Stutters can be either due to network effects, or due to the client not receiving sufficient local resources to decode and decrypt data in a timely manner. Such local starvation is generally due to the client host doing other (predominantly 1/0-intensive) tasks. The protocol is designed to provide on-demand access to a large library of tracks and would be unsuitable for live broadcasts. For instance, a client cannot upload a track unless it has the whole track.

The reason for this is that it simplifies the protocol, and removes the overhead involved with communicating what parts of a track a client has. The drawbacks are limited, as tracks are small. While JODI is the most common transport protocol in streaming applications, Spottily instead uses TCP. Firstly, have- inning a reliable transport protocol simplifies protocol design and implementation. Secondly, TCP is nice to the network in that Tic's congestion control is friendly to itself (and thus other applications using TCP), and the explicit connection signaling helps statuses firewalls.

Thirdly, as streamed material is shared in the peer-to-peer network, the re-sending of lost packets is useful to the application. Between a pair of hosts a

ingle TCP connection is used, and the application protocol multiplexes messages over the connection. While a client is running, it keeps a TCP con- mention to a Spottily server. Application layer messages are buffered, and sorted by priority before being sent to the proper- eating system's TCP buffers. For instance, messages needed to support interactive browsing are prioritize over bulk traffic. A. Caching Caching is important for two reasons.

Firstly, it is common that users listen to the same track several times, and caching the track obviates the need for it to be re- downloaded. Sec- monody, cached music data can be served by the client in the peer- to-peer overlay. The cache can store partial tracks, so if a client only downloads a part of a track, that part will generally be cached. Cached content is encrypted and cannot be used by other players. The default setting in the clients is for the maximum cache size to be at most 10% of free disk space (excluding the size of the cache itself), but at least 50 MBA and at most 10 B.

The size can also be configured by the user to be between 1 and 100 B, in 1 KGB increments. This policy leads to most client installations having large caches (56% have a maximum size of 5 KGB or more, and thus fit approximately 1000 tracks). Cache eviction is done with a Least Recently Used (LOUR) policy. Simulations using data on cache sizes and playback logs indicate that, as caches are large, the choice of cache eviction policy does not have a large effect on cache efficiency. This can be compared to the finding of Human et al. 5] that the Applied system gained much efficiency by changing from LOUR to a more complex, weight-based evaluation process. However, in their setting, the objects in the cache re movies, and their caches are of a size such that a client can only cache one, or a few movies.

B. Random Access to a Track I en asleep case Tort a streaming music player Is when tracks are played In a predictable order. Assuming sufficient band-width available, this allows the player to begin fetching data needed to play upcoming tracks ahead of time (prenticing).

A more difficult, and perhaps interesting, case is when the user chooses a new track to be played, which we refer to as a random access. We begin by describing the random access case and then proceed to discuss prenticing in Section II-C. Approximately 39% of playbacks in Spottily are by random access (the rest start because the current track finished, or because the user clicked the forward button to skip to the next track). Unless the client had the data cached, it makes an initial request to the server asking for approximately 1 5 seconds of music, using the already open TCP connection.

Simultaneously, it searches the peer-to-peer network for peers who can serve the track, as described in Section Ill-C. In most cases the initial request can be quickly satisfied. As the client already has a TCP connection to the server no 3- way handshake is needed. Common TCP congestion avoidance algorithms, such as TCP New Reno [9] and CUBIC [10], maintain a congestion window limiting how large bursts of data can be sent. The congestion window starts out small for a new connection and then grows as data is successfully sent.

Normally, the congestion window (on the server) for the long- lived connection between client and server will be large as data will have been sent over it. This allows the server to quickly send much, or all, of the response to the latency-critical initial request without waiting for Sacks from

the client. The connection to the server is long-lived, but it is also bursts in tauter. For instance, if a user is streaming from a popular album, a very large fraction of the traffic will be peer-to-peer traffic, and the connection to the server will then be almost unused.

If the user then makes a random access playback there is a sudden burst of traffic. RFC 5681 [1 1] states that an implementation should reduce its congestion window if it has not sent data in an interval exceeding the retransmission timeout. Linux kernels can be configured to disable that reduction, and when Spottily did so the average playback latency decreased by approximately 50 ms. We remark that this was purely a server-side configuration change. If a user jumps into the middle of a track (seeks), the client treats the request similarly to a random access, immediately requesting data from the server as described above.

The Cog Boris format offers limited support for seeking in a streamingenvironment, so Spottily adds a custom header to all their files to better support seeking. C. Predictable Track Selection Most playbacks (61 occur in a predictable sequence, I. E. Because the previous track played to its end, or because the user pressed the forward button. Clients begin prenticing the next track before the currently playing track has been played to implosion. With prenticing there is a trade-off between cost and benefit. If clients begin prenticing too late the prenticing may not be sufficient to allow the next track to immediately start playing.

If they prefects too early the bandwidth may be wasted if the user then makes a random request. The clients begin searching the peer-to-peer

network and start downloading the next track when 30 seconds or less remain of the current track. When 10 seconds or less remain of the current track, the client preferences the beginning of the next track from the server if needed. We did tot have data to directly measure how good the choice of these parameters are. But, we can measure now oaten a user won listens to a track Tort ten rotation AT ten track minus t seconds continues to the next track.

If a user seeks within a track, the measured event does not correspond to playback coming within t seconds of the end of the track. For t = 10, 30, the next scheduled track was played in 94%, and 92% of cases, respectively. This indicates that the choice of parameters is reasonable, possibly a bit conservative. During a period of a few weeks, all clients had a bug where prenticing of the next track was accidentally disabled. This allows us to directly measure the effects of prenticing on the performance of the system.

During a week when prenticing was disabled the median playback latency was 390 ms, compared with the median latency over the current measurement period of 265 ms. Furthermore, the fraction of track playbacks in which stutter occurred was 1. 8%, compared to the normal rate of 1. 0%. D. Regular Streaming While streaming, clients avoid downloading data from the server unless it is necessary to maintain playback quality or keep down latency. As discussed in Section 11-8, when the user makes a random access, an initial request for data is sent to the revere.

Clients make local decisions about where to stream from depending on the amount of data in their play-out buffers. The connection to the server is

assumed to be more reliable than peer-connections, so if a client's buffer levels are low, it requests data from the server. As long as the client's buffers are sufficiently full and there are peers to stream from, the client only streams from the peer-to-peer network. There is also an " emergency mode" where, if buffers become critically low (less than 3 seconds of audio buffered during playback), the client pauses uploading data to its peers.

The reason for this is that many home users have asymmetric connection capacity, a situation where JACK compression can occur and cause degradation of TCP throughput [12]. The " emergency mode" has been in the protocol since the first deployment, so we have not been able to evaluate its effects. A given track can be simultaneously downloaded from the server and several different peers. If a peer is too slow in satisfying a request, the request is resent to another peer or, if getting the data has become too urgent, to the server.

While streaming from a server, clients throttle their requests such that they do not get more than approximately 1 5 seconds ahead of the current playback point, if there are peers avail- able for the track. When downloading from the peer-to-peer network, no such throttling occurs and the client attempts to download the entire currently playing track. If a user changes tracks, requests relating to the current track are aborted. Files served within the peer-to-peer network are split into chunks of 16 KGB.

When determining which peers to request chunks from, the client sorts peers by their expected down- load times (computed as the number of bytes of outstanding requests from the peer, divided by the average download speed

received from that peer) and greedily quests the most urgent chunk from the peer with the lowest estimated download time (and then updates the expected download times). This means that chunks of a track are requested in sequential order. As all peers serving a file have the entire file, requesting blocks in-order does not affect availability or download speeds.

A client can at most have outstanding requests from a given peer for data it believes the peer can deliver within 2 seconds. An exception to this is that it is always allowed to have requests for 32 KGB outstanding from a peer. If the estimated download time for a Deadlock exceeds ten pilot In tale at wanly en Deadlock Is name, Tanat Deadlock Is not requested. E. Play-out Delay Streaming applications need to employ some mechanism to combat the effects of packet loss and packet delay variation. Several different options have been suggested in the literature; for a survey see [13].

Spottily clients do not drop any frames or slow down the pullout-rate, and are thus non delay-preserving in the nomenclature of [13]. As TCP is used as transport protocol, all data requested will be delivered to the application in-order, but the rate at which data is delivered is significantly affected by network conditions such as packet loss. If a buffer under occurs in a track, the Spottily client pauses playback at that point, re-performing latency adjustment. As discussed by Liana et al. [14], there is a trade-off between initial playback latency, receiver buffer size, and the stutter free probability.

Spottily clients do not limit the buffer size, and thus the crux of the problem is the appropriate modeling of the channel and using that information to adjust the initial playback latency. As a simplification, the client only

considers the channel to the server for latency adjustment. Spottily clients use a Moravian model for throughput as observed by the client (I. E. Affected by packed delay variation, packet loss, and TCP congestion control). Clients make obese- ovations of throughput achieved while it is downloading from the server to estimate a Markova chain.

Only data collected during the last 15 minutes of downloading is kept and used. The states of the Markova chain is the throughput during 1 second, discredited to 33 distinct levels between O and 153 Kbps (more granular at lower throughput). The model is not used to compute an explicit playback latency. Instead, before playback has commenced, the client periodically uses the Markova chain to simulate the playback of the track, beginning with the runner amount of buffered data, and the current data throughput. Each such simulation is considered as failing or passing, depending on if an under occurred or not.

The client performs 100 simulations and if more than one of them fails, it waits longer before begin- inning playback. During these simulations the client makes the simplifying assumption that data is consumed at a constant rate despite the fact that the code used has a variable vitiate encoding. Ill. SPECIFY'S PEER-TO-PEER NETWORK Spottily protocol has been designed to combine server- and peer-to-peer streaming. The primary reason for developing a peer-to-peer based protocol was to improve the scalability of the service by decreasing the load on Spottily servers and bandwidth resources.

An explicit design goal was that the usage of a peer-to-peer network should not decrease the performance in terms of playback latency for music or the amount of stutter. While that design goal is addressed by the reliance on a server for latency-critical parts, it puts demands on the efficiency of the peer-to-peer network in order to achieve good offloading properties. We discussed in Sections II-B and II-D owe the clients combine streaming from the peers and servers. In this section, we give an overview of the peer-to-peer network. A.

General Structure The peer-to-peer overlay used is an unstructured network, the construction and maintenance of which is assisted by trackers. This allows all peers to participate in the network as equals so there are no " supersedes" performing any special network- maintenance functions. A client will connect to a new peer only when it wishes to download a track I t tanks ten peer NAS. It locates peers Kelly to nave a track It Is looking for through the mechanisms described in Section Ill-C. As discussed in Section II-A, clients store (relatively large) local caches of the tracks they have downloaded.

The content of these caches are also what the clients offer to serve to their peers. As tracks are typically small and as live streaming is not supported, a simplification made in the protocol is that a client only offers to serve tracks which it has completely cached. This allows for a slightly simpler protocol, and keeps the protocol overhead down. There is no general routing performed in the overlay net- work, so two peers wishing to exchange data must be directly connected. There is a ingle message forwarded on the

behalf of other peers, which is a message searching for peers with a specific track.

The rationale for the lack of routing in the overlay is to keep the protocol simple and keep download latencies and overhead down. B. A Split Overlay Network The service is currently run from two data centers, one in London and one in Stockholm. A peer uniformly randomly selects which data center to connect to, and load is evenly spread over both data centers. Each data center has an indeed- pendent peer-to-peer overlay. Thus, the peer-to-peer overlay is in fact split into two overlays, one per site. The split is not complete since if a client loses its connection to the server, it reconnects to a new server.

If it reconnected to the other site it keeps its old peers but is unable to make any new connections to peers connected to servers at its old site. For simplicity of presentation, we will describe the protocol as having a single overlay network. C. Locating Peers Two mechanisms are used to locate peers having content the client is interested in. The first uses a tracker deployed in the Spottily back-end, and the second a query in the overlay network. The problem of locating peers is somewhat different in music- n-demand streaming compared to many other settings.

As tracks are small, a client generally only needs to find one, or a few peers to stream a track from. However, as tracks are also short in duration, downloading new tracks is a very frequent operation, and it is important to minimize the overhead. Furthermore, the lookup time becomes a big issue, which is one of the reasons for Spottily not using a Distributed Hash Table

(DOT) to find peers. Other reasons for not implementing a DOT include keeping the protocol simple and keeping overhead down. The functionality of the tracker is similar, but not identical, to that of a tracker in the Bitterroot protocol [15].

It maintains a mapping from tracks to peers who have recently reported that they have the track. As a peer only offers to serve a track if it has the whole track cached, peers listed in the tracker have the whole track. As two complementary mechanisms are used, the tracker can be simplified compared to many other system. In particular- lar, the tracker only keeps a list of the 20 most recent peers for each track. Furthermore, clients are only added to the tracker when they play a track and do not periodically report the contents of their caches, or explicitly notify the tracker when content is removed.

This helps in keeping overhead down, and simplifies the implementation of the tracker. As clients keep a TCP connection open to a Spottily server, the tracker knows which clients are currently online. When a client asks the trackers for peers who have a track, the tracker replies with up to 10 peers who are currently online. The response is limited in size to minimize overhead. I Alton to ten tracker-Dates peer searches, clients also seen search requests In the overlay network, similar to the method used in Neutral [16].

A client in search of a track sends a search request to all its neighbors in the overlay, who forward the quest to all their neighbors. Thus, all peers within distance two of the searcher in the overlay see the request, and send a

response back if they have the track cached. Search queries sent by clients have a query id associated with them, and peers remember the 50 most recent searches seen, allowing them to ignore duplicate messages. This limited message forwarding is the only overlay routing in the Spottily peer-to-peer protocol.

When a client is started, how does it get connected to the peer-to-peer network? If it was still listed in the tracker for some tracks then it is Seibel that other clients will connect to it asking for those tracks. If the user starts streaming a track, it will search the peer-to-peer network and connect to peers who have the track, thus becoming a part of the overlay. D. Neighbor Selection Keeping the state required to maintain a large number of TCP connections to peers is expensive, in particular for home routers acting as statuses firewall and Network Address Translation (NAT) devices.

Thus, each client has a maximum number of peers it may be connected to at any given time. Clients are configured with both a soft and a hard limit, and never go above the hard limit. The client does not make new connections above the soft limit and periodically prunes its connections to keep itself below the soft limit (with some headroom for new connections). These limits are set to 50 and 60, respectively. When a client needs to disconnect one or more peers, it performs a heuristic evaluation of the utility of each connected peer.

The intention is for the heuristic to take into account both how useful the connection is to the evaluating peer, as well as how useful the link is to the overlay as a whole. The client sorts all its connected peers according to 6

criteria: bytes sent in the last 10 minutes, Yates sent in the last 60 minutes, bytes received in the last 10 minutes, bytes received in the last 60 minutes, the number of peers found through searches sent over the connection in the last 60 minutes, and the number of tracks the peer has that the client has been interested in in the last 10 minutes.

For each criterion, the top scoring peer in that criterion gets a number of points, the second peer a slightly lower number, and so on (with slightly different weights for the different criteria). Peers with a raw score of O for a criterion do not get any points for that criterion. The errs points are then summed over all the criteria, and the peers with the least total scores are disconnected. The client simultaneously uploads to at most 4 peers.

This stems from the fact that TCP congestion control gives fairness between TCP connections, so many simultaneous uploads can have adverse effects on other internet usage, in particular for a home user with small uplink bandwidth. E. State Exchanged Between Peers A client wanting to download a track will inform its neighbors of its interest in that track. The interest notification also contains a priority, where the client informs its peer of the urgency of the request. Currently, three discrete levels (in falling order of priority) are used: currently streaming track, prenticing next track, and offline synchronization.

A serving client selects which peers to service requests from by sorting them by the priority of the request, and previously measured upload speed to that peer, and then offer to service the requests of the top 4 peers. Peers are Interment whenever toner status changes (IT toner requests Decode

quiescently prioritize to be serviced, or if their requests no longer are). F. NAT Traversal All traffic in the peer-to-peer network uses TCP as transport protocol so the most moon protocols for NAT traversal, e. G. STUN [1 7], are not immediately applicable as they are designed for UDP.

While there are techniques for performing TCP NAT traversal as well [1 8], Spottily clients currently do not perform any NAT traversal. This lack of NAT traversal is mitigated by two factors. Firstly, when a client wishes to connect to a peer a request is also forwarded through the Spottily server asking the connected to attempt a TCP connection back to the connecter. This allows the connection to be established provided one of the parties can accept incoming connections. Secondly, clients use the Universal Plug n' Play (Upon) protocol to ask home routers for a port to use for accepting incoming connections. B) Stutter during playback Playback latency and music stutter over a week. A. Measurement Methodology (a) Tracks played Figure 1 . The weekly usage pattern of the Spottily service. Data has been normalized too 0-1 scale. (a) Playback latency Hogue s. Figure 2. Sources of data used by clients Both Spottily clients and servers perform continuous in-sedimentation and monitoring of the system. Most client measurements are aggregated locally before being sent to the server. For instance, reports on connection statistics are sent every 30 minutes to the server.

The raw log messages are collected and stored on log servers and in a Hoodoo cluster (an open-source map-reduce and distributed storage implementation), where they are avail- able for processing. There is also a real-time monitoring system, based on the open-source Minimum monitoring system, storing aggregated data and generating graphs based on the log

messages and instrumentation of Spottily servers. Most of our graphs are based on the aggregated Minimum databases, while most aggregate statistics (e. G. , median playback latency) ere computed from the raw log files.

In the graphs presented in this paper, min and bag gives the minimum and average (per time unit) values taken over the measurement period, and cur denotes the current value when the measurement was made. Values below 1 will be denoted with units m or u, denoting mill (10-3) and micro (10-6 respectively. 'V. PEER-TO-PEER EVALUATION In this section, we will present and discuss measurements indicating the performance of the Spottily system with focus on the peer-to-peer network performance. For business reasons, some data is presented as ratios rather than absolute volumes.