An Experimental Study of Client-Side Spotify Peering Behaviour

Martin Ellis
School of Computing Science
University of Glasgow
Email: ellis@dcs.gla.ac.uk

Stephen D. Strowes
School of Computing Science
University of Glasgow
Email: sds@dcs.gla.ac.uk

Colin Perkins
School of Computing Science
University of Glasgow
Email: csp@csperkins.org

***Abstract*—Spotify is a popular music-streaming service which has seen widespread use across Europe. While Spotify's server-side behaviour has previously been studied, little is known about the client-side behaviour. In this paper, we describe an experimental study where we collect packet headers for Spotify traffic over multiple 24-hour time frames at a client host. Two distinct types of behaviour are observed, when tracks are being downloaded, and when the client is only serving requests from other peers. We also note wide variation in connection lifetimes, as seen in other studies of peer-to-peer systems. These findings are relevant for improving Spotify itself, and for the designers of other hybrid peer-to-peer and server-based distribution architectures.**

I. Introduction

Spotify (http://www.spotify.com/) is a commercial music streaming service that is widely used across Europe, and has recently been rolled out in the US. It employs a hybrid data distribution model with a combination of peer-to-peer sharing of data and a server-side infrastructure. Although some data on server-side Spotify traffic patterns has been published [6], the behaviour of clients in the wild remains unstudied. This data might be useful in finding aspects of the protocol or application which can be improved, such as measuring the geographic distribution of peers to see whether latencies can be reduced, or assessing how successful the peer network is at serving content.

In this paper, we present a preliminary study of the client-side behaviour of the Spotify protocol; in particular, we are interested in the peer-to-peer interactions, rather than those between client and server. We capture packet headers, and use the resulting traces to analyse the size of the peer set over time, and connection lifetimes of the peers.

Our contributions are client-side measurements of Spotify's behaviour and initial analysis of these measurements to examine the operation of Spotify's peer-to-peer protocol in the wild. We investigate how often our client connects with new peers, how long they stay connected, and how much data they transfer. Our results show that the client's connection-forming behaviour is quite different when downloading new tracks, compared to when playing from the local cache. In particular, connection lifetimes are typically shorter when the client is in the "fetching" state. This result is interesting since it suggests that new connections need to be formed to obtain new tracks in the playlist, meaning that the tracks of interest are not available from existing peers.

Similar work has focused on other peer-to-peer systems, such as Skype [4], [2] and PPLive [5]. However, since Spotify uses a hybrid architecture, rather than a purely peer-to-peer network, and since its content consists of short music tracks rather than interactive telephony or large video files, we expect the behaviour of the peers to be quite different.

The remainder of this paper is organised as follows. We briefly describe the Spotify protocol and application in §II, outline our experimental setup in §III, and present our results in §IV. We discuss related work in §V, and conclude in §VI.

II. Spotify Overview

The Spotify service is available with several levels of membership, with a free version supported by advertisements, as well as subscription services, and is available for Windows, OS X, Linux, and several mobile platforms. It recently reached ten million users, with one million of those subscribing to the Unlimited or Premium services[1]. The protocol operates entirely using TCP transport, since TCP provides reliable transport and congestion control. Clients maintain connections to the server and to other peers, multiplexing both data and control traffic over these connections. The clients also use UPnP to ask home gateways (if present) to allow incoming TCP connections to the client. This avoids some of the problems caused by NAT on home networks, since the Spotify clients do not attempt explicit NAT traversal [6].

Tracks are transmitted (and cached by the clients) in an encrypted format, and cannot be played outside Spotify. The size of the client cache is determined by the user, defaulting to 10% of available disk space, but being at most 10GB in size. Each time clients try to play a track, they first search their on-disk cache to see if the track has already been downloaded before searching for the track online.

To locate tracks, the Spotify client uses two methods. First, the client uses a tracker at the Spotify servers, which maintains "a list of the 20 most recent peers for each track" [6]. Second, the client can also search using peers it is already connected to. These peers also forward the query to *their* neighbours, so that all peers within a distance of two hops of the searcher receive the query. Peers which have cached the requested track will send a response to the client which initiated the search.

[1] http://www.spotify.com/uk/about/press/background-info/

When playing from a playlist, the client takes advantage of "predictable track selection" to start downloading the next track before the end of the track currently playing. When the current track has around 30 seconds left, the client will search for the next track on the peer network. If these searches have not succeeded with 10 seconds remaining of the current track, the beginning of the next track is downloaded from the server. To allow this, all tracks are available on the Spotify servers, which lets the service maintain low playback latency.

To reduce the state required at the client and stateful firewalls, each client limits the size of its peer set. According to [6], the clients are configured with "soft" and "hard" limits of 50 and 60 connections, respectively. The client will not make new connections above the soft limit, and the number of connections should never go above the hard limit. To maintain these limits, the client will periodically prune connections, using a number of criteria to rank the current peers (including number of bytes transferred to and from the peer, number of successful searches passed through the peer, and number of tracks of interest the peer has). Using these criteria, the current peers are ranked, and the "least useful" are disconnected.

For a more in-depth discussion of the Spotify protocol and architecture, including a description of the streaming protocol, peer-to-peer overlay, and client design, as well as measurements of server-side performance, refer to [6].

III. Methodology

We use a dedicated machine running Linux to play Spotify with an Unlimited subscription. Using a separate machine running FreeBSD 8.1 as a transparent bridge, we capture all traffic to and from the Linux machine using `tcpdump`, for offline analysis. However, since the payloads of the Spotify traffic are encrypted, we limit our analysis to using information in the packet headers, including source and destination IP address and port, TCP flags, and payload size. Both these machines are connected to the Internet using a commodity NAT/ADSL modem, attached to a residential ADSL line. We acknowledge that this preliminary study uses a single observation point, which may not be representative of all Spotify clients. However, the characteristics of the traffic we observe between our client and the servers and to other peers are interesting, especially since different modes of behaviour appear to be present when the client is actively fetching data compared to when it is playing from its local cache.

We measure the traffic exchanged when music is being played by our client. To understand the effect of content/popularity on our measurements, we use three playlists for our measurements of data traffic:

- *100-random*, which includes 100 tracks chosen using Spotify's radio feature, using the widest possible set of music preferences to select tracks randomly.
- *500-rollingstone*, based on Rolling Stone magazine's "500 greatest tracks of all time" [1]. This includes around 450 tracks, since some of the tracks were unavailable in the UK when we conducted our experiments, probably due to licensing issues.
- *100-rollingstone*, the first 100 tracks of *500-rollingstone*.

These playlists were chosen to investigate how content popularity affects the peering behaviour of our client. For each of the playlists, we collect a 24-hour trace, starting the client with an empty on-disk cache, so that all the tracks in the playlist are downloaded each time. The *100-rollingstone* and *100-random* playlists have around six hours of content, so will stop fetching new content roughly one quarter of the way through the trace, while the *500-rollingstone* playlist has around 24 hours of content, fetching new content throughout the measurement period. Each experiment was repeated three times to mitigate the effect of random variations, and to validate the characteristics experienced for each playlist. The playlists are available at http://sdstrowes.co.uk/research/spotify/.

IV. Results

Since our focus is on Spotify's peer-to-peer behaviour, and the majority of the data we observe is transferred over the peer network ($\sim$99%), for the remainder of the paper we analyse only the peer-to-peer traffic. To do this, we filter out traffic to and from known Spotify IP ranges, and on ports 80, 443, and 4070 (which are used for communication with the server, and with other sites as part of the client).

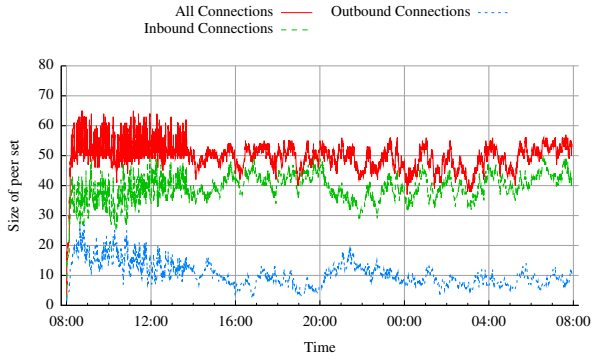
In our analysis of the traces, we look at how the peer set size varies over time, and at the effect of playlist content (between *100-rollingstone* and *100-random*), and playlist length (between *100-rollingstone* and *500-rollingstone*). We also investigate Spotify's connection-forming behaviour, examining the connection lifetimes, and activity over short timescales.

A. Peer set size over time

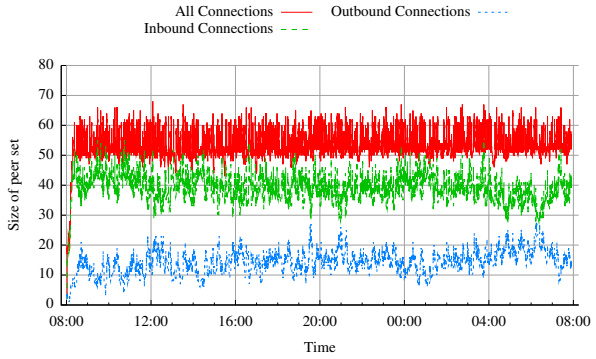
Figure 1 shows the size of the peer set over time, in both the *100-rollingstone* and *500-rollingstone* playlists ("outbound" connections are initiated by our client, and "inbound" are initiated by peers). Note that the peer set sizes roughly match the limits described in Section II. Two states are visible within the *100-rollingstone* playlist. From the start until around 14:00, while our client is actively fetching tracks, we note a more variable trend in the peer set size, followed by a less variable period. Since the *500-rollingstone* playlist lasts longer (24 hours rather than six), the measurements of this playlist show our client fetching tracks for the entire measurement period. We note that for the *100-rollingstone* playlist (and in the *100-random* playlist, not shown), some outbound connections are still being opened by our client, even in the less variable period after the first six hours of playing (when no new tracks are being fetched). The reason for this behaviour is explained in [3]; when a client attempts to connect to a peer, it also forwards a request via the server for the peer to open a connection back to the original client. When one of these succeeds, the other is terminated, before any data is transferred.

B. Connection lifetimes

Since there is variation in the peer set, we next examine connection lifetimes to gain insight into the efficiency of the peer network. Figure 2 shows histograms of the connection

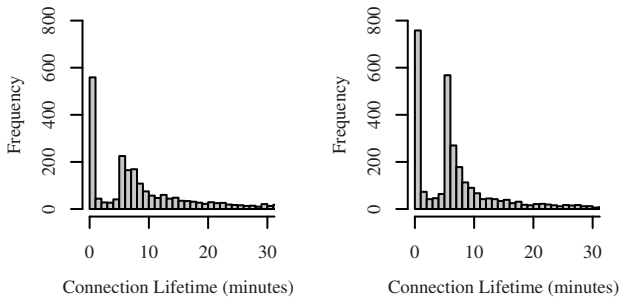


(a) *100-rollingstone*2



(b) *500-rollingstone*2

Fig. 1. Peer set size over time (*100-rollingstone* and *500-rollingstone*)



(a) *100-random*3 (b) *100-rollingstone*3

Fig. 2. Connection lifetimes histograms (*100-random* and *100-rollingstone*)

lifetimes for the *100-random* and *100-rollingstone* playlists (all the replicas show similar shapes, so we show representative examples). In these plots we use one-minute bins, and observe that two peaks are present in both plots. The first peak shows that many connections are opened and closed within a minute (as we discuss later, many of these last for less than one second). The second peak is around five to six minutes; we note that this is around the length of a song, and suggest that connection duration is correlated to track length.

Figure 3 shows the complementary CDF of the connection lifetimes, drawn on log-log scale. As in other studies of peer-to-peer behaviour [4], [5], [7], we find the shapes to be consistent with a long-tailed distribution of connection lifetimes,

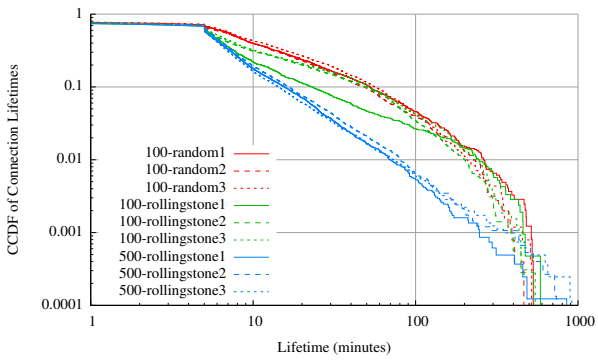


Fig. 3. Log-log complimentary CDF of connection lifetimes

with the lifetime distributions showing a roughly straight line on log-log axes. Figure 3 shows a difference between the *500-rollingstone* playlist (which shows a clear straight line along the body of the distribution, a feature commonly attributed to power-laws) and the other playlists. These results show that for the *500-rollingstone* playlist (during which the client spends more time in the "fetching-content" state, as discussed in Section IV-A), connection lifetimes are typically shorter, indicating higher churn in the client's peer set.

C. TCP activity

We examine the TCP activity over short periods of time, to better understand when connections are created and how long they last, and in particular, to understand why so many connections last around five minutes. Figure 4 shows a 60-minute snapshot of the activity in one of our replicas (again, we observe similar behaviour in the other traces, and show a representative example). Each point represents TCP activity, and each line on the y-axis represents a different connection. We observe bursts of activity every few minutes, with new connections being formed, some lasting for quite short periods, and others lasting longer. These bursts of new connections are reasonably evenly spaced, around every five minutes, roughly coinciding with the length of songs. Therefore, we believe that each of these bursts of activity is associated with our client approaching the end of the current track, and making new connections to gather the next track, as described in [6]. We speculate that once the client obtains a list of peers which have the next track in its playlist, it opens multiple connections to these peers. When the track is obtained, the other connections are closed, resulting in the short connections visible in Figure 4. These short connections are also visible in the box highlighted in Figure 5, which shows the length of connections and the volume of data transferred. Connections with very short lifetimes (∼1 second or less) tend to transfer very little data (less than 1kB), implying they are not used to transfer tracks. Note that Figure 5 does not include connections which transfer no data before being closed, such as the "two-way" connections described in Section IV-A.

We note that this bursty connection-forming behaviour is present on all the playlists. This is slightly unexpected since

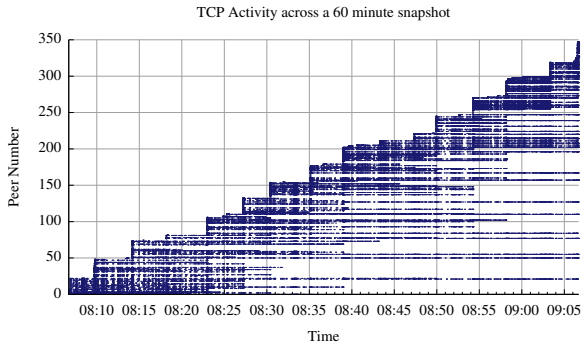


Fig. 4. TCP activity over 60-minute snapshot (*100-random*3)

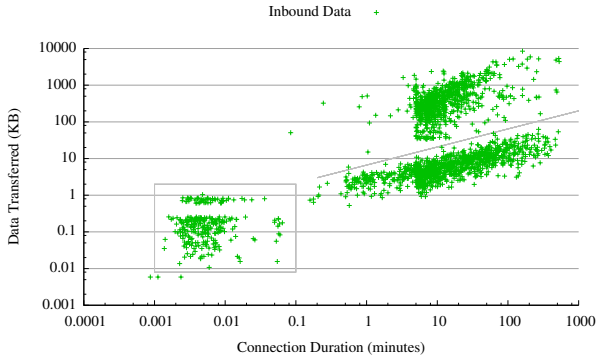


Fig. 5. Connection duration / data transferred (*100-random*1, inbound)

we expected that playing the *rollingstone* playlists would introduce us to peers with similar musical preferences (improving the chance our existing peers would have the next track in the playlist). If these results are reflective of all Spotify clients, this suggests that the existing set of peers consistently fails to provide requested tracks. Since this connection behaviour is predictable, it also provides a mechanism to classify network connections as being Spotify peer-to-peer traffic.

Further work will focus on trying to discover whether some playlists (e.g., with tracks classified as Spotify's most popular) change this behaviour, and gather most data from the existing peer set rather than forming new connections.

V. Related Work

The operation of the Spotify protocol, and a server-side perspective of the peer-to-peer network behaviour is described in [6]. In contrast, we study Spotify from the perspective of an individual peer participating in the network, connected to the Internet using a typical home setup (using a commodity NAT and ADSL line). Similar studies have looked at other peer-to-peer systems and protocols, including Skype [2], [8], and peer-to-peer video streaming systems [5], [7]. Like these studies, we observe long-tailed behaviour in lifetimes of connections, as illustrated in Figure 3. However, some important differences are present. Since Spotify deals with non-interactive content (unlike Skype), it has a less stringent requirement for low latency connections between peers, and can cope with situations when tracks cannot be located from the peer network. Similarly, since Spotify transfers relatively small music tracks, rather than streaming video (as in P2PTV systems), it can transfer data in bursts, as shown in Figure 4.

VI. Conclusions And Future Work

In this paper, we have presented the first client-side measurements of Spotify's peering behaviour. We show that connection-forming behaviour is quite different between when the client is fetching content, and when it is playing only from its local cache, with differences in the rate of new connections, and the connection lifetimes. We note in particular that when new tracks are being downloaded, the rate of new connections is quite bursty. We believe that these bursts of activity represent the client downloading the next track in the playlist, and that connections to many peers are opened in parallel, to improve the likelihood obtaining the track. The predictable bursts of new connections we observe when the client is fetching tracks could provide a technique to identify Spotify traffic.

These results highlight a number of avenues for future work. Firstly, repeating these experiments with a larger number of clients will allow us to validate the preliminary findings outlined here. Additionally, further experiments using clients located on different types of network (such as academic or mobile networks) would produce interesting comparisons, since the different network capacity and stability is likely to affect the peering behaviour. Since our playlists may not have introduced the musical locality of preference we expected, future work should choose these more carefully, possibly by playing from entire albums, or by using Spotify's list of most popular tracks. This should allow us to determine whether playing popular content reduces connection churn.

Overall, we show a client-side perspective of how the Spotify protocol operates, and discuss interesting features of our data. This work, and the future work described above, can be used to improve the performance of the Spotify protocol, and other protocols for hybrid peer-to-peer distribution.

Acknowledgements

This work was supported by Cisco Research and the UK EPSRC. Thanks to Tristan Henderson and Saleem Bhatti for suggestions and comments on earlier drafts of this paper.

References

[1] "The 500 Greatest Songs of All Time," *Rolling Stone*, no. 963, 2004.
[2] S. A. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol," in *Proc. IEEE INFOCOM*, 2006.
[3] M. Goldmann and G. Kreitz, "Measurements on the Spotify Peer-Assisted Music-on-Demand Streaming System," in *Proc. IEEE P2P*, 2011.
[4] S. Guha, N. Daswani, and R. Jain, "An Experimental Study of the Skype Peer-to-Peer VoIP System," in *Proc. IPTPS*, 2006.
[5] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A Measurement Study of a Large-Scale P2P IPTV System," *IEEE Transactions on Multimedia*, vol. 9, no. 8, 2007.
[6] G. Kreitz and F. Niemelä, "Spotify - Large Scale, Low Latency, P2P Music-on-Demand Streaming," in *Proc. IEEE P2P*, 2010.
[7] T. Silverston and O. Fourmaux, "Measuring P2P IPTV Systems," in *Proc. ACM NOSSDAV*, 2007.
[8] K. Suh, D. R. Figueiredo, J. Kurose, and D. Towsley, "Characterizing and detecting relayed traffic: A case study using Skype," in *Proc. IEEE INFOCOM*, 2006.