Integrating Smartphones in Spotify's Peer-Assisted Music Streaming Service

Raul Jimenez*, Gunnar Kreitz*†, Björn Knutsson*, Marcus Isaksson† and Seif Haridi*
*KTH - Royal Institute of Technology, Stockholm, Sweden
†Spotify, Stockholm, Sweden

***Abstract*—Spotify is a large-scale peer-assisted music streaming service. Spotify's P2P network serves 80% of music data to desktop clients. On the other hand, the rapidly growing number of mobile clients do not use P2P but instead stream all data from Spotify's servers.**

We enable P2P on a Spotify mobile client and empirically evaluate the impact of P2P protocols (in particular low-bandwidth traffic between peers) on energy consumption, both on 3G and Wifi. On 3G, current P2P protocols are highly energy inefficient, but simple modifications bring consumption close to the client-server configuration. On Wifi, the extra energy cost of enabling P2P is much lower.

Finally, we propose a protocol modification to further integrate mobile devices in Spotify's P2P network according to their capabilities (power source, access network). This allows us to break the artificial division between desktop and mobile platforms and dynamically adapt as resources become (un)available to the device.

I. Introduction

While Peer-to-peer (P2P) technology has enjoyed great success for content delivery over the "regular" Internet (wired and Wifi), this is currently not the case for the mobile Internet and the rapidly increasing population of smartphones. In this paper, we challenge the conventional wisdom regarding P2P on such devices, and using the P2P network and clients of the popular Spotify music service, we demonstrate that doing so can lead to significant benefits, while incurring few, if any, of the dire consequences predicted.

Spotify is a *peer-assisted* music streaming service, with over 20 million tracks and over 24 million users in 28 countries.

Currently, Spotify uses its P2P network as a major source of the music played by its desktop client (80% of the total volume), while its own servers primarily handle cases where the P2P network is unable to provide data in a timely fashion to a client.

On its *mobile platforms* (Android, iOS, Windows Phone, etc.) this it not the case, however. These clients follow conventional wisdom, and get 100% of their data directly from Spotify's servers. This situation is not unique to Spotify —we are not aware of any large-scale P2P system where mobile platforms are well integrated— despite the dramatic increase in use of such devices.

At first glance, conventional wisdom appear to be well-founded. P2P systems are predicated on peers contributing "spare" resources for the greater benefit of all, so trying to include a peer which is severely resource constrained (limited battery, storage, CPU and bandwidth, the latter often even capped and metered) may seem quite unwise, and we agree.

Our contention is with the application of it, which we believe to often be too broad. Sure, contributing resources from a mobile phone with those limitations is probably a bad idea, but just letting it *download* from other peers should be no different than having it download from a server.

And what about when that mobile is plugged into the charger and using the Wifi network at home? Then its limitations are no different than those of a laptop, so why can't we let it upgrade itself to a "full" peer then?

In this paper, we will explore these and other related questions and issues by way of trying to let Spotify's client for mobile devices use and participate in the Spotify P2P network. Along the way, we will also examine the P2P protocols and the power consumption they are responsible for on both 3G and Wifi wireless networks, in the hope that by better understanding the causes, it will be possible reduce power consumption.

We observe that power consumption is determined not only by traffic volume, but also by its shape. In wireless networks there are high fixed costs for starting transmission and low marginal costs for additional bytes once the radio interface is powered up. This means that protocols continuously exchanging small amount of data are very power inefficient. Unfortunately, that is exactly the traffic pattern many P2P protocols (including Spotify's) exhibit.

We start from Spotify's desktop client, compiled to run on a mobile system, and from examining the power used on both Wifi and 3G, we explore ways to reduce, sometimes drastically, the power consumed by it. Our modifications include preventing the device from uploading to reducing the "trickling" traffic, and we end up with energy consumption is comparable to the non-P2P configuration.

We also discovered that on Wifi, the increase in energy consumption due to "trickling" traffic is much smaller than we had expected, based on related work [7], [9].

We conclude by proposing guidelines and methods for how mobile devices can be integrated into P2P-systems with a minimum of added power consumption when the systems are battery powered and using 3G, and with most, if not all, the benefits of "full" peers when they are AC powered and on Wifi networks.

We believe, based on our experience with other P2P protocols and networks, that the majority of our conclusions and

proposals are applicable not only to Spotify's P2P network, but to varying degrees, to all the P2P protocols and networks currently in wide use.

We also believe that getting mobile systems integrated into these P2P networks is of paramount importance if P2P is to stay relevant, given the rapid growth of mobile devices population and the ways they are replacing traditional (desktop) systems.

A. Our Contributions

In this work, we make a case study of the feasibility of integrating mobile devices into a deployed, large-scale, peer-assisted system. In this study, we make several finds on energy consumption patterns, showing the surprising efficiency of the unmodified Spotify P2P protocol on Wifi. We also demonstrate how simple backwards-compatible adaptions can drastically reduce energy consumption on 3G.

Our case study leads us to a discussion which could serve as a foundation for future research: how can overlay designs best be updated to accomodate the new class of mobile devices? These devices have new characteristics as participants in peer-to-peer overlays as they can up-front be flagged as resource-constrained, but may also swiftly change status when on Wifi and connected to a power supply.

Lastly, the tools we developed for this study have been released as open source, available at http://people.kth.se/~rauljc/spotify/. We believe these may be of independent use to analyze and understand power consumption of networked applications. In particular, some potential energy optimizations for the current (client-server) Spotify protocol were discovered using these tools.

II. Spotify and P2P

This section provides a brief overview of Spotify and its peer-assisted mechanisms. See [1], [2] for more technical details.

The core compontent in Spotify's service is music delivery. Clients can request music tracks from both Spotify's servers and its P2P network. At present only desktop clients use the P2P network, and the mobile clients only request data from Spotify's servers.

Desktop clients prefer downloading tracks from the P2P network. They only stream from the server to guarantee low-latency and gap-less playback. For instance, when a user selects and plays a track, the server provides the beginning of the track while the client requests the remaining data from the P2P network. When playing from a playlist, the client requests the next track from the P2P network a few seconds before the current track ends, downloading the whole track from the P2P network in the best case. In any case, the client can always fall back to requesting data from the server to prevent music interruptions and stutter.

To minimize the amount of data delivered by servers, a client requests data in streaming mode, avoiding requests more than 15 seconds ahead of the current playback point. On the P2P network, whole tracks are downloaded as fast as possible.

On mobile platforms, where P2P is not available, clients request tracks in large chunks: an initial request of around 1.5 MB and then requests of up to 5 MB. The reason behind this streaming strategy is a trade-off attempting to minimize both energy consumption and delivery of unnecessary data. We will discuss energy consumption in Section III.

A. Peer Discovery

Spotify clients use two different peer discovery mechanisms: 1) a centralized tracker and 2) a distributed overlay. A design rationale for having two mechanisms is that it allows both to be imperfect and at times miss peers, making them more efficient and cheaper to implement. These two mechanisms have similarities to those of BitTorrent and Gnutella, respectively.

The tracker service is implemented in Spotify's backend. Compared to BitTorrent's tracker, an important difference is that a Spotify client will only announce itself to the tracker after it has downloaded a complete copy of the music track. That is, Spotify's tracker only tracks seeders. Spotify's tracker also limits the amount of peers it tracks and the number of responses to queries to keep overhead low.

The second mechanism is an unstructured overlay similar to Gnutella's and is very relevant to this paper, since its networking patterns greatly affect energy consumption on mobile devices.

To form the overlay, every peer establishes and maintains a number of TCP connections to other peers (called neighbors), the details of how neighbors are selected can be found in [1], [2].

To find peers offering a given track, a client sends a query to all its neighbors and these neighbors in turn forward the query to their neighbors. That is, all peers at most two hops away from the client will receive the query and respond if they happen to have the requested track.

On the other end, each peer constantly receives queries and keep-alive pings from neighbors. In terms of peak bandwidth and total number of bytes, the overhead is relatively small compared to data transfers. On wireless networks, however, continuous low-bandwidth traffic is considered expensive in terms of energy consumption. We will discuss energy consumption in wireless networks in Section III.

III. Energy Consumption in Wireless Networks

One of the main concerns for users and smartphone app developers is battery life. The proliferation of battery-hungry apps make smartphones' batteries last shorter and the availability of energy monitors give users the tools to identify and uninstall apps that provide low value per joule.

Several studies have investigated energy consumption on mobile devices. Zhang et al. [3] and Yoon et al. [4] studied in detail power consumption caused by the different hardware components of a smartphone, including Wifi and 3G.

In this paper, we focus on power consumption caused by network activity in Wifi (IEEE 802.11g) and 3G. While a brief description of their power properties is given in this section, we refer to the studies above for further details.

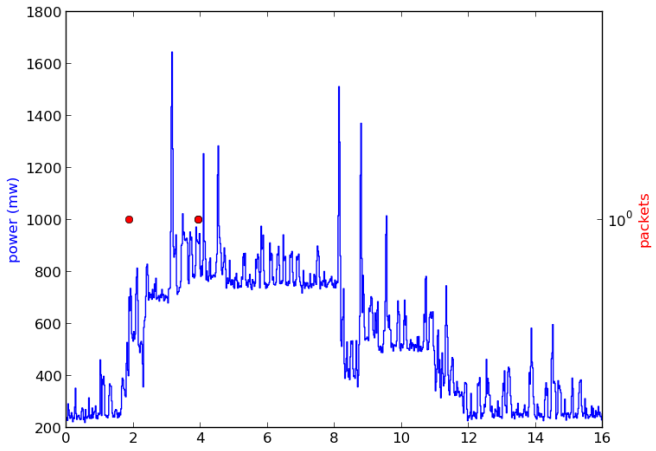

Fig. 1. A ping-response exchange shows power transitions on 3G: DCH (2-8) and FACH (8-12). Notice the latency of around 2 seconds.

A. 3G

There are three power states in the 3G radio resource controller (RRC): IDLE, FACH, and DCH. To transmit data, the radio needs to be powered up to DCH where the device acquires a dedicated channel for communication. After a few seconds of inactivity, the radio is demoted to FACH, where it is assigned a shared channel. In FACH, power consumption is about half of DCH and the state can be promoted to DCH quickly if more data needs to be transmitted. After a few further seconds of inactivity, the radio is powered down to IDLE where it consumes very little power.

Figure 1 illustrates 3G power states on an Android smartphone. Approximately two seconds into the graph, the phone sends a TCP packet to a server (red dots represent packets captured by tcpdump). To transmit the packet, the 3G radio transitions into DCH, increasing power consumption. Around two seconds, the packet reaches the server, which sends a TCP ACK back. Finally, about four seconds into the graph, the smartphone receives the TCP ACK.

This example illustrates the long initial delay (actual RTT was below 100 ms) applications are subjected to. Precisely, to counter such long delay for subsequent packets, the interface stays in DCH for around four seconds and an extra four seconds in FACH before it powers down back to IDLE. Inactivity timeouts are controlled by network operators and can vary widely.

B. Wifi

Wifi data transfer is much more efficient than 3G, specially when power saving mechanism (PSM) is enabled. PSM lets the radio interface sleep and just wake up at regular intervals to listen for beacons from the access point. These beacons indicate whether the mobile device has received data. This data is buffered by the Wifi access point and can be retrieved by the client when it wakes up.

While PSM adds latency, it saves energy. Modern smartphones support adaptive PSM (PSM-A) which keeps the wireless interface powered for an extra period of time to lower latency and increase throughput. In modern smartphones this extra period (i.e. energy tail) has been estimated to be approximately 200 ms [5].

A detailed description of Wifi power consumption can be found in [6]. A recent survey on energy-efficient streaming further elaborates on power saving methods in Wifi [5].

IV. Experimental setup

To measure and understand the sources of power consumption, we have run experiments in different networks and locations. While details are slightly different in each experiment, all experiments show the same patterns. Given that we are interested in these patterns rather than details, we consider the experiments we document in this paper to be representative.

We run each experiment sequentially on a Galaxy Nexus phone running Android 4.2.1. For each Spotify client configuration (see next section for details), we instruct the app to play a playlist for 31 minutes. The playlist played is *Top 100 tracks currently on Spotify*[1] as of June 20th 2013. We chose this playlist because its tracks are well seeded, making sure that the mobile client would find enough peers to download all data from the P2P network when P2P was enabled.

Spotify uses the Ogg Vorbis format for streaming in three different quality alternatives: q3 (96 kbps), q5 (160 kbps), and q9 (320 kbps). The defaults are q3 and q5 for mobile devices and desktop clients, respectively.

To be able to download data from existing desktop clients, we select *High quality* (q5) on the mobile device. We also make sure that Facebook reporting and last.fm scrobbling were disabled so that communication with these services would not interfere with our experiments. For the same reason, we use a phone exclusively for these experiments with no extra apps installed and background synchronization disabled.

In addition, we check all connections the client makes in each experiment. Despite our efforts to eliminate non-Spotify traffic from the experiments, we do observe long-lived connections to Google's servers (*.1e100.net) in which small amounts of data are exchanged regularly.

Each experiment starts with an empty cache and the first minute is discarded, to exclude user interaction when the display is powered. During the 30 minutes we analyze, the display is off and headphones are plugged in. Volume is set to the lowest setting (volume setting does not affect power consumption [3]).

Each configuration is evaluated both on 3G and Wifi. For the entire duration of the experiments the smartphone is stationary. We use an enterprise 3G package from Swedish operator Telia with no monthly data caps. According to Android, the mobile network types used are: HSDPA:8, HSPA+:15, and UMTS:3. On Wifi, we report the experiments run on the Wifi

[1] http://play.spotify.com/user/spotify/playlist/4hOKQuZbraPDIfaGbM3lKI

infrastructure provided by KTH in its Kista campus. We have also repeated the experiments at Spotify's Stockholm offices with no major differences in the results, despite the fact that the former provides public and the latter private IP addresses.

To measure power consumption, we use a Monsoon Power Monitor[2], which feeds power at constant voltage to the phone while collecting up to 5000 power samples per second. During recalibration periods, the device fails to report samples. We compensate these missing samples by reducing the sampling frequency. For instance, 50 samples per second where used in Figure 1. That is, every 20 ms our tools output a single aggregate sample by averaging all samples reported by the device in that period of time. In the next section, we show figures using one aggregated sample per second (1 Hz) to better illustrate power patterns on 3G by removing noisy power spikes.

To capture network traffic on the mobile device, we run *tcpdump* provided by Shark for Android[3].Since tcpdump requires root privileges, we rooted the phone using Wug's Nexus Root Toolkit v1.6.2[4].

Our tools analyze the traffic capture files, generating useful information such as per host traffic analysis as well as the graphs included in this paper.

While these tools have been created specifically for this project, they are designed to be generic. In fact, they are used by Spotify to identify and remove spurious data traffic from its production client-server mobile client. These tools are written in Python and are available under an open-source license at http://people.kth.se/~rauljc/spotify.

V. Experimental Results

A. Baseline: P2P Disabled

The first configuration we evaluate is the standard Spotify app[5], which we use as *baseline* for comparison. We compile the unmodified source code for version 0.6.0 and manually install the Android package on our test mobile device. We repeat the same process for the other two configurations evaluated in this section.

This *baseline* configuration does not use P2P, instead downloading all data directly from Spotify's servers. The smartphone client makes fewer and larger requests compared to its desktop counterpart, following recommendations for Android developers[6]. A primary reason for this recommendation is that it is much more energy efficient to download the whole track as fast as possible than performing long low-bandwidth transfers on wireless networks. We refer the reader to a recent survey [5] for an in-depth study of energy-efficient multimedia streaming on wireless networks.

[2]http://www.msoon.com/LabEquipment/PowerMonitor/

[3]https://play.google.com/store/apps/details?id=lv.n3o.shark

[4]http://forums.androidcentral.com/verizon-galaxy-nexus-rooting-roms-hacks/147177-automated-wugs-nexus-root-toolkit-v1-6-2-updated-12-29-12-a.html

[5]https://play.google.com/store/apps/details?id=com.spotify.mobile.android.ui

[6]http://developer.android.com/training/efficient-downloads/efficient-network-access.html

Baseline downloads each music truck in chunks. An initial download of 1.5 MB, followed of up to 5 MB chunks. This simple heuristic attempts to save power while avoiding premature download of data that will not be used if the user skips the track a few seconds into playback, which is common user behavior.

Figure 2 shows a typical track playback. The figure shows power consumption (blue line) and the number of IP packets transmitted within one second (red dots, log scale). Power is shown in one-second sample aggregation (as explained in Section IV) to remove noise and emphasize power patterns. This visualization helps us to clearly see the impact of network traffic on power consumption.

The track is downloaded in two chunks. At second 39, 1.5 MB are downloaded, the rest of the track (3.7 MB) is downloaded at second 100. In the figure, these downloads are clearly identifiable by a cloud of dots above 300 packets/s.

The 3G energy tail is clearly seen and it lasts several seconds. It is also worth noting that it takes around two seconds between the client's request for content and the first data packet from the server, this time is spent negotiating and setting up the channel. The phone's 3G radio is active during these two seconds. In the figure, this is visible as a red dot indicating one packet just before the download.

The figure also shows periods of low network activity where a few packets are exchanged triggering a long period of high power consumption. The worst offender is found at second 212. A 3-packet exchange keep-alive ping (sent 11 bytes, received 11 bytes, ACK) between the client and the server keeps the 3G radio interface up for around 10 seconds.

Similar patterns are triggered by notifications and reporting messages. For instance, to report that a particular track has been played.

Despite our efforts to eliminate non-Spotify traffic from the experiments, we also observe a DNS query plus a 17-packet exchange (4202 bytes) with Google's servers (arn06s02-in-f19.1e100.net) at 170 s.

In this particular example, downloading periods (including startup delay and tail) consume 32 joules in 39 seconds. Low-bandwidth traffic consumes 43 J in 68 s and while low-power periods consume 41 J in 143 s. This means that bundling the low-bandwidth traffic together with a download would significantly reduce energy consumption.

This optimization is out of the scope of this paper. Although, as a result of this research, Spotify will modify its protocol to reduce power consumption on mobile devices.

B. Peer: Unmodified P2P

Next, we enable P2P on the mobile device with the same configuration as the Spotify desktop client. Although this configuration is capable of uploading data to other peers, there is no data upload in our experiments, allowing us to study solely download traffic and P2P overhead.

Compared to baseline, there are two major differences with respect to traffic patterns: single download and continuous low-bandwidth traffic.

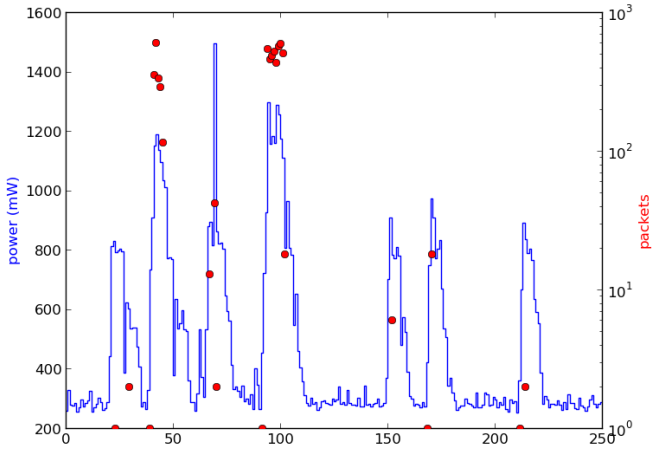

Fig. 2. Baseline 3G. Track download is done in two bursts. Notice energy end tail. Power sampling: 1 Hz.

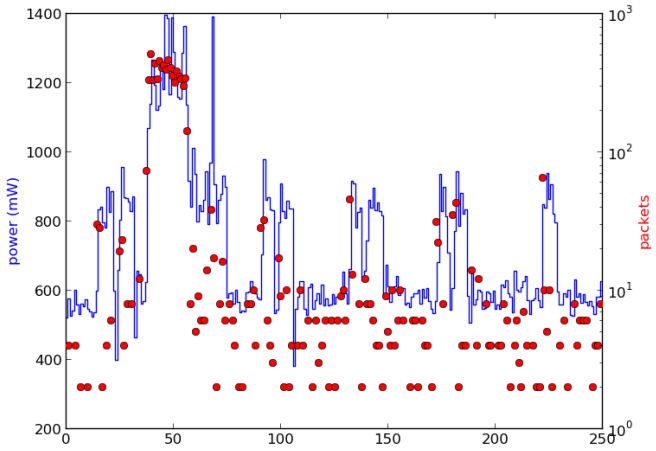

Fig. 3. Peer 3G. Track download is done in one burst. Energy end tail keeps 3G constantly awake, consuming extra power. Power sampling: 1 Hz.

Figure 3 shows a single large dot cloud (around second 50), which was expected, since downloads from the P2P network are always done in a single chunk as described in Sectionbackground-spotify. While this is more energy efficient, it risks downloading useless data if the user skips the track.

As expected, Spotify's Gnutella-like query flooding mechanism (used to find peers) generates a continuous flow of low-bandwidth traffic, keeping the 3G radio interface powered almost continuously, dramatically increasing energy consumption. Over a 30 minute period, this configuration almost doubles energy consumption compared to baseline, as shown in Section V-D.

Figure 4 shows the same configuration on Wifi. Notice that we increased aggregation sampling to 50 Hz in the power graph to show power spikes, given the much shorter tail energy

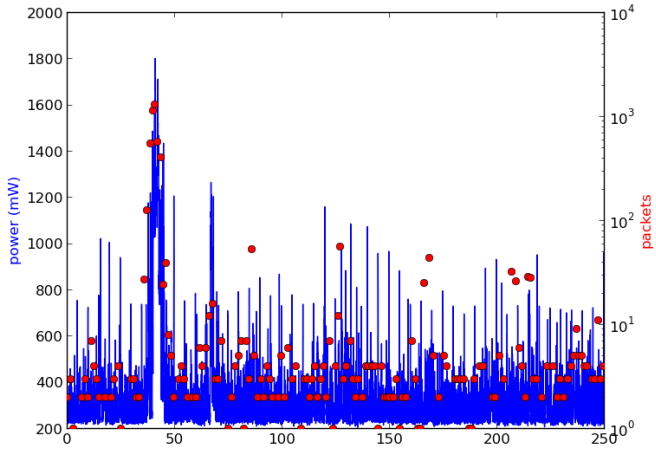

Fig. 4. Peer Wifi. Music download is done in one burst. Energy end tail is not that bad in Wifi. Power sampling: 50 Hz.

period compared to 3G.

We observe that Wifi is much more power-efficient handling low-bandwidth traffic. It is also much better compared to a 2008 study evaluating constant low-bandwidth traffic generated by another P2P protocol [7]. We speculate this improvement comes from Wifi power saving methods implemented in modern devices.

C. Leecher: Reducing Energy Consumption

In some contexts, P2P upload from mobile devices is very expensive. For instance a battery-powered device on 3G data plan with a scarce monthly allowance.

We can completely disable upload on mobile devices. This makes mobile peers free riders or *leechers*. Leechers decrease scalability because a leecher demands resources from the system while providing none. On the other hand, in peer-assisted systems, they can download data from other peers, offloading servers.

Leeching can be done by simply refusing uploading to peers requesting data. A better approach is, however, to never register on the peer discovery services, so other peers do not connect the leecher to request data at all.

As described in Section II-A, Spotify has two complementary peer discovery mechanisms: centralized tracker and decentralized flooding search.

Standard peers announce files to the centralized tracker as soon as they are completely downloaded and stored in cache. We modified leechers to use the tracker to find peers but never announce files.

The distributed search is a more complex mechanism. Each peer keeps connections to a number of neighbors. Whenever it needs to find peers, a peer queries all its neighbors, which will 1) reply if they have the file and 2) forward the query to all its neighbors.

We modify the configuration to never reply to queries (no reply means "I don't have the requested data"), and drop

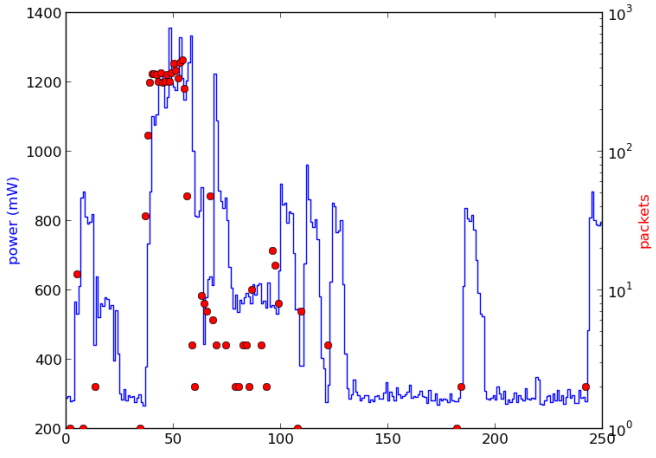

Fig. 5. Leecher 3G. Music download is done in one burst. Energy end tail keeps the 3G radio in a non-idle state for 60 seconds, which is what it takes to drop a neighbor.

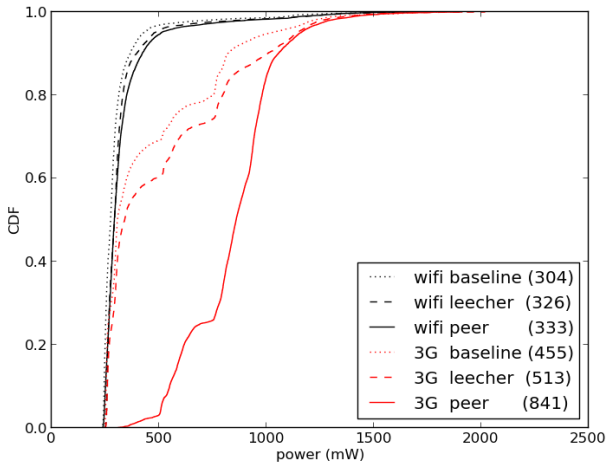

Fig. 6. Power consumption CDF. Legend shows average power consumption (mW).

connections after 60 seconds. While this approach has its drawbacks, it clearly shows the impact of low-bandwidth traffic patterns produced by some P2P protocols on energy consumption. We will later propose improvements to this configuration.

Figure 5 shows that the bulk of low-bandwidth traffic disappears by dropping P2P neighbor connections, dramatically reducing energy consumption while the whole track is downloaded from the P2P network.

A more aggressive dropping policy would further reduce energy consumption at a risk of affecting search performance and ability of downloading data from peers.

While this approach has the benefits of using the distributed search to find peers and being backwards compatible, it has some drawbacks. For instance, it introduces peers that are not using the protocol as originally intended and artificially increase churn[7] by dropping connections after 60 seconds.

Ideally, leechers should be able to find peers using distributed search while not being constantly queried by other peers. One simple solution, which we propose for future work, is to label connections to leechers in a way that peers never send queries through these connections. This would stop the energy-inefficient trickling traffic, while providing distributed search for leechers. While we elaborate on this proposal in Section VI-B, we could not evaluate it empirically because it is backwards incompatible.

D. Total Energy

Figure 6 shows a comparison of the amount of power consumed by each configuration over the 30-minutes experiments, allowing comparison between different client and network configurations. Average power consumption is included in the graph's legend.

[7]Rate at which peers join/leave a P2P overlay.

Starting at the bottom left, we observe that all lines but one are bundled together between 250 and 350 mW. This power consumption corresponds to playback and output audio to headphones while the wireless radio is idle.

All three configurations tested on Wifi (black lines) consume less than 350 mW more than 90% of the time. There is a relatively small power gap between baseline and the other two configurations, showing that modern mobile devices can efficiently handle low-bandwidth traffic on Wifi, minimizing the impact of inefficient Wifi on P2P overlays found in the literature (e.g. [7]).

On 3G, we can clearly see a staircase-shaped line which corresponds to 3G's power states and their long inactivity timeouts. In the extreme case of the *peer* configuration, the 3G radio is constantly powered in a non-idle state and spends most of the time in DCH (highest power state). As discussed before, this situation is caused by a constant flow of low-bandwidth traffic generated by the distributed search protocol.

We also observe that the simple modification of dropping P2P connections after 60 seconds is enough to greatly decrease energy consumption. A more aggressive dropping policy would further reduce the gap between baseline and leecher.

These results suggest that it is possible to integrate mobile devices into P2P, at least as leechers. Even with a naive implementation of a leecher, the energy cost in Wifi is moderate. On 3G, the gap between baseline is much larger but we were able to greatly reduce it with simple backwards-compatible protocol modifications. In the next section, we propose further improvements, introducing backwards-incompatible modifications.

VI. Moving Forward

A. Peers and Leechers

Until now, we have hinted at how to categorize mobile devices into peers and leechers. Here, we propose a heuristic

that we consider appropriate for Spotify.

Ideally, participants in the P2P network should be categorized according to their available resources. That would partially eliminate an artificial distinction between devices and platforms (operating systems). And, more importantly, it would allow switching from one category to another as resources become available.

Our experiments have focused on energy consumption because battery life span is a major issue on mobile devices. But not all mobile devices are battery-powered all the time. Many users charge their devices at night, while at work or driving. Many devices are docked into sound systems and there are already docking stations in the market that convert a smartphone or tablet into a full-fledged desktop computer with all necessary peripherals.

Another factor is data caps. Currently, most mobile data plans offered by operators have data caps. A common deal is a monthly allowance of 0.5 to 2 GB, including both up- and down-stream traffic. In fact, many applications consider this limitation, limiting large downloads (e.g. Spotify's off-line synchronization) to be done only on Wifi, by default.

Given these two factors, we consider as *peers* those mobile devices connected to a power supply and using Wifi. Given the widespread use of data caps on mobile networks and the lack of a standard way to automatically detect these caps, we consider all mobile devices connected to a mobile network as *leechers*, regardless of their power status.

B. Protocol Modification

In our experiments, we tried to create two peer categories: peer and leecher. Unfortunately, we could not evaluate a true leecher configuration due to backward incompatibility issues. That is, the leecher received queries from existing peers and the only way to stop them was dropping peer connections which, in turn, artificially increased churn.

In this section, we propose a backwards incompatible modification to the P2P protocol modification that not only addresses this problem but it also tackles seamlessly switching between peer and leecher according to the resources available to the mobile device at any moment.

While deploying backwards incompatible modifications into a large-scale P2P network is hard, we believe that the benefits of integrating mobile devices will compensate the deployment costs.

The modification affects the way peers establish connections to neighbors and how these connections are used. Currently, all peers are considered equal and they frequently query each other to find peers and request data. They also send keep-alive pings to check that the neighbor is still alive (it is common for P2P protocols to assume that peers are unreliable).

Our modification introduces a flag where a participant in the P2P network indicates to its neighbors whether it is a peer or a leecher. Connections to neighbors flagged as leechers are never used to send queries or keep-alive pings. In addition, leechers never announce themselves to the central tracker. That is, leechers cannot be discovered by any of the two peer discovery mechanisms, thus no peer will try to establish a connection to a leecher.

As a result, each leecher will be connected to a number of neighbors (all of them full peers). Whenever a leecher needs to download a track, it will query its neighbors until it finds a peer offering the requested data.

Peers will have neighbor connections to both peers and leechers. The only difference with the current protocol is that peers will not send nor forward queries to leechers.

Notice that these modifications not only stop energy-hungry low-bandwidth traffic but also shields the P2P network from artificial churn. In the modified protocol, a leecher going off-line does not affect the overlay because no message is ever routed through leechers.

This design lets switching categories without the need of a disconnection plus a full bootstrap into the P2P overlay. If a peer becomes a leecher (e.g. the device is disconnected from a power supply), it will drop its neighbor connections to leechers and notify its peer neighbors of its new status, quickly stopping all incoming queries. In the opposite case where a leecher becomes a peer, the status notification will allow both incoming queries from existing neighbors and incoming connections from both peers and leechers.

Switching networks (IP address) is a special case and may, but not necessarily, involve switching categories as well. Provided each device has an identifier independent of its network address (as it happens in most P2P networks), it can reconnect to all its neighbors. The neighbors will update the new IP address and category.

VII. Related Work

There are a number of studies investigating the integration of mobile devices in P2P systems. Nurminen and Nöyränen [8] measured power consumption of SymTorrent—a BitTorrent client for Symbian—both in 3G and Wifi. They report a higher energy consumption on 3G, which is mainly caused by lower download speed on 3G, thus keeping the radio powered for a longer period of time. They also measure "full peer" versus "client only" but only on Wifi.

Kelényi and Nurminen [7], [9] measure energy consumption of Mainline DHT, BitTorrent's decentralized peer discovery mechanism. They highlight the high energy cost of continuous low-bandwidth traffic on Wifi. They reduce energy consumption by selectively dropping DHT queries.

A recent study by Nurminen [10] evaluates energy consumption of BitTorrent and BitTorrent's Mainline DHT. While it studies energy consumption using a Nokia device on a 3G network, it focuses on how BitTorrent's incentives affect download time, and thus energy consumption.

Peer-to-Peer Streaming Protocol (PPSP) is at late stage in the IETF standarization process[8]. Petrocco et al. [11] evaluated the performance of Libswift (reference implementation of PPSP) on Android. The study includes power comparison, using hardware measurements, of Libswift and YouTube playing

[8]http://datatracker.ietf.org/wg/ppsp/charter/

a video stream, using hardware measurements. They did not consider peer discovery, focusing on streaming on Wifi.

RapidStream [12] is P2P video streaming prototype on Android. The paper focuses on design and practicalities with the Android platform. Superficial energy measurements (phone's battery indicator) were given in the paper.

There is a considerable number of design papers and network performance and mobility issues on mobile P2P but do not address energy consumption. For instance, Eittenberger et al. [13] evaluate BitTorrent on WiMax and Berl and Meer [14] consider incentives when integrating mobile devices as leechers in the eDonkey network.

VIII. Conclusion

In our efforts to integrate mobile devices into Spotify's P2P system, we have studied the issues that difficultate their integration.

We make the case against treating mobile devices as resource-constrained devices, but instead categorize them according to the resources they have available. This includes seamlessly switching categories as resources become (un)available to the device.

In particular, we consider two categories: peer and leecher. The peer category is reserved for devices that are capable of performing all tasks expected of a full peer (download and upload data, provide peer discovery service) at low cost to the client in terms of energy and bandwidth. The rest of the devices are considered leechers and are able to download data from the P2P network (offloading servers) with no, or minimal, increase of energy and bandwidth consumption.

Our experiments show that low-bandwidth traffic used by many P2P systems is particularly energy-inefficient on 3G. On Wifi, on the other hand, this traffic does not significantly increase energy consumption. Presumably, this is due to energy saving improvements implemented on modern mobile devices.

We also show in our experiments that a simple backwards-compatible modification goes a long way reducing energy consumption on 3G.

Finally, we propose a protocol modification that would further reduce energy consumption for leechers while allowing them to be integrated in the P2P network. This modification is backwards-incompatible, which is the reason why we have not empirically evaluated it.

Additionally, the tools we developed for this study have been released as open source, available at http://people.kth.se/~rauljc/spotify/. We believe these may be of independent use to analyze and understand power consumption of networked applications. In particular, some potential energy optimizations for the current (client-server) Spotify protocol were discovered using these tools.

Disclaimer

We would like to emphasize that this is an academic research paper and should not be taken as indicative of Spotify's product plans.

Acknowledgment

We thank Spotify employees, in particular Javier Ubillos and his team, who not only provided technical support and were involved in the research discussions, but also provided a great working environment.

We also thank engineers from Telia and Deutsche Telekom who helped us understanding the problem from an operator point of view.

References

[1] G. Kreitz and F. Niemela, "Spotify – large scale, low latency, p2p music-on-demand streaming," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, 2010, pp. 1–10.

[2] M. Goldmann and G. Kreitz, "Measurements on the spotify peer-assisted music-on-demand streaming system," in *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on*, 2011, pp. 206–211.

[3] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES/ISSS '10. New York, NY, USA: ACM, 2010, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1878961.1878982

[4] C. Yoon, D. Kim, W. Jung, C. Kang, and H. Cha, "Appscope: Application energy metering framework for android smartphone using kernel activity monitoring," in *USENIX ATC*, 2012.

[5] M. Hoque, M. Siekkinen, and J. Nurminen, "Energy efficient multimedia streaming to mobile devices — a survey," in *IEEE Communications Surveys & Tutorials*. IEEE, 2013.

[6] Y. Xiao, P. Savolainen, A. Karppanen, M. Siekkinen, and A. Ylä-Jääski, "Practical power modeling of data transmission over 802.11g for wireless applications," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, ser. e-Energy '10. New York, NY, USA: ACM, 2010, pp. 75–84. [Online]. Available: http://doi.acm.org/10.1145/1791314.1791326

[7] I. Kelenyi and J. Nurminen, "Optimizing energy consumption of mobile nodes in heterogeneous kademlia-based distributed hash tables," in *Next Generation Mobile Applications, Services and Technologies, 2008. NGMAST '08. The Second International Conference on*, 2008, pp. 70–75.

[8] J. Nurminen and J. Noyranen, "Energy-consumption in mobile peer-to-peer - quantitative results from file sharing," in *Consumer Communications and Networking Conference, 2008. CCNC 2008. 5th IEEE*, 2008, pp. 729–733.

[9] I. Kelenyi and J. Nurminen, "Energy aspects of peer cooperation measurements with a mobile dht system," in *Communications Workshops, 2008. ICC Workshops '08. IEEE International Conference on*, 2008, pp. 164–168.

[10] J. Nurminen, "Energy efficient distributed computing on mobile devices," in *Distributed Computing and Internet Technology*, ser. Lecture Notes in Computer Science, C. Hota and P. Srimani, Eds. Springer Berlin Heidelberg, 2013, vol. 7753, pp. 27–46. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36071-8_3

[11] R. Petrocco, J. Pouwelse, and D. H. J. Epema, "Performance analysis of the libswift p2p streaming protocol," in *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, 2012, pp. 103–114.

[12] P. Eittenberger, M. Herbst, and U. Krieger, "Rapidstream: P2p streaming on android," in *Packet Video Workshop (PV), 2012 19th International*, 2012, pp. 125–130.

[13] P. M. Eittenberger, S. Kim, and U. R. Krieger, "Damming the torrent: Adjusting bittorrent-like peer-to-peer networks to mobile and wireless environments," *Advances in Electronics and Telecommunications*, vol. 2, no. 3, pp. 14–22, 2011.

[14] A. Berl and H. de Meer, "Integrating mobile cellular devices into popular peer-to-peer systems," *Telecommunication Systems*, vol. 48, no. 1-2, pp. 173–184, 2011.