Tribler Mobile: P2P Video Streaming from and to Mobile Devices

Raul Jimenez*, Arno Bakker†, Björn Knutsson*, Johan Pouwelse†, Seif Haridi*
* KTH - Royal Institute of Technology, Stockholm, Sweden
† Delft University of Technology, Delft, The Netherlands

***Abstract*—Peer-to-peer (P2P) mechanisms allow users with limited resources to distribute content to a large audience, without the need of intermediaries.**

These P2P mechanisms, however, appear to be ill-suited for mobile devices, given their limited resources: battery, bandwidth, and connectivity. Even Spotify, a commercial straming service where desktop clients stream about 80% of the data via P2P, does not use P2P on mobile devices.

This paper describes Tribler Mobile, a mobile app that allows users to broadcast their own videos to potentially large audiences directly from their devices. Our system delegates most of the distribution tasks to *boosters* running on desktop computers. Our mechanisms are designed to be fully-decentralized and consider mobile devices' limitations.

Tribler Mobile is available as open-source software and have been installed by almost 500 users on their Android devices.

I. Introduction

This paper presents a P2P video streaming platform where mobile devices are well integrated. As a prove of concept, we have modified an existing P2P system and deployed the infrastructure to allow mobile devices not only stream from the P2P network, but also publish new content to it.

We make the very conservative assumption that uploading data from mobile devices is always expensive due to cost related to increased energy consumption on battery-powered devices and traffic (monthly data caps are common on mobile networks). In practice, we expect a more favorable environment where uploading costs are much lower for some mobile devices; for instance, when a device is connected to a power supply and on a wired or Wifi connection. Furthermore, more generous data caps and more energy-efficient devices may lower uploading costs on mobile devices in the future.

In this environment, a mobile device publishing new content should ideally upload a single copy of the content to the P2P network, and then immediately stop all P2P traffic. This is basically what cloud-based applications do, and they can do it because cloud servers are reliable and they are always ready to accept data. In P2P networks, however, each peer is assumed to be unreliable and publishers have no easy-to-use mechanisms to place peers on stand-by, waiting for new content.

We are the first to propose, implement, and deploy a complete system where mobile devices can publish content directly into the P2P network and let other peers distribute the content to a potentially large audience (including other mobile devices). In the process of building this system, we develop a reverse peer discovery mechanism (described in Section IV-A) that (1) traverses NAT gateways —these gateways are widely-deployed on Wifi and mobile networks— and (2) limits energy consumption by preventing other peers from establishing connections to the mobile device (those connections would switch the device's radio interface to high-power state even after the P2P app has terminated).

To distribute content to other mobile devices, our P2P system organizes *desktop peers* (peers running on desktop computers) to form a sort of content distribution network (CDN). Some of these desktop peers would be run by users who consume the content, and as a side-effect, contribute to its distribution to both desktop and mobile peers. This paper, however, is mainly focused on what we call *boosters*.

A *booster* is a peer whose main task is to augment (or boost) the distribution capacity of a publisher or source. There could be different reasons why a user chooses to cede resources to a publisher by boosting its content: personal relationship, political/activist support, in exchange for services or money, et cetera.

Boosters are meant to provide the capacity demanded by *leechers*. A leecher is a peer that downloads data from other peers, without uploading data to others[1]. While the term leecher is often used pejoratively to indicate that these leechers are free-riders, we use it without any negative connotation. On the contrary, our system is designed to address the extra capacity demanded by leechers by creating extra capacity provided by boosters.

This paper presents a system design that is generic and flexible enough to be deployed in different contexts. From an open-community best-effort system to a commercial P2P streaming service where boosters are mainly (or exclusively) run by the publisher to guarantee quality of experience. While open-community services tend to demand fully-decentralized solutions, commercial services may keep some subsystems centralized for monitoring and control purposes and to ease deployment.

As a proof of concept, we have implemented and deployed a complete prototype which consists of these elements:

- A mobile app, called *Tribler Mobile*, offers P2P video streaming playback, as well as publishing new content to the P2P network. For instance, a video recorded with the smartphone's camera.

[1]Throughout the paper, content is always streamed, though we use *download* and *upload* to indicate data flow's direction.

- A desktop application, called *Tribler*, which can be configured to automatically boost certain content (e.g., published by a friend) as soon as it is published.[2]
- A subscription and publisher authentication mechanism (Section IV-B). We use Twitter in our prototype in a way when a booster subscribes to `@user`, it will boost all content posted by that user. That is, when `@user` records a video and publishes it, a tweet is automatically posted on `@user`'s timeline and that video will then be boosted by all boosters subscribed to `@user`.
- A DHT-based reverse peer discovery (Section IV-A that addresses two issues at once. When the mobile device (MD) is behind a NAT gateway, it allows the MD to upload the content by establishing direct connections to boosters. When the MD is not behind a NAT gateway, it prevents connection attempts from other peers, which would increase energy consumption even after the mobile app had been terminated.

The rest of this paper is organized as follows. Background is presented in Section II. Section III introduces the roles in the system and how they interact with each other. In Section IV, we describe each of the infrastructure services supporting our system, both in abstract terms and our prototype's concrete implementation details along with a short discussion about trade-offs and alternative implementations. Finally, Section V concludes.

II. Background

Internet content streaming services are very popular and growing fast. Not only on large-screen wire-connected desktops but also battery-powered wireless-connected mobile devices. In fact, all trends indicate that mobile devices already account for a large part of streaming, and it is growing far faster than on desktop platforms. For instance, mobile devices consumed 41% of YouTube's total traffic in the third quarter of 2013, up from 25% in 2012 and just 6% in 2011.[3]

Streaming content to a large audience is far from trivial. In general, it is impractical for an individual or a small company to deploy its own content distribution system. While there are platforms that will distribute your content for a fee (e.g., content distribution networks) or placing ads (e.g., YouTube), there is an alternative: *P2P content distribution*.

A. Peer-to-Peer Content Distribution

On the desktop environment, where desktop computers and laptops are commonly connected to a power source and connected to the Internet though Wifi or a wired network, P2P offers the possibility of augment the publisher's distribution capacity with resources from its viewers. That is, the viewers not only consume resources but also contribute their own, increasing scalability and reducing costs to the publisher.

File-sharing communities use P2P to distribute content by each peer contributing its own resources (mainly storage and bandwidth). P2P systems have evolved to be fully-distributed and self-organized, gradually removing centralized services (e.g., tracking and torrent files acquisition in BitTorrent, which initially relied on centralized services, have now been fully decentralized).

Yet, mobile devices have not been well integrated on P2P networks. Conventional wisdom suggests that these devices are ill-suited for P2P for several reasons: (1) uploading would drain batteries and deplete 3G monthly traffic allowances too fast for most users, (2) connectivity issues caused by widely-deployed network address translator (NAT) gateways, and (3) for some publishers, client-server (e.g., content distribution networks (CDNs)) distribution costs are low enough to disregard investments in P2P-based optimizations.

Spotify, a commercial music streaming service, is a good example to illustrate this point. On desktop platforms, Spotify uses P2P mechanisms to increase scalability and reduce costs. According to their own measurements, 80% of the music is delivered by its P2P network [2]. On mobile platforms, however, all music is currently delivered by Spotify's servers.

B. Mobile P2P

The introduction of Internet-enabled mobile devices led some researchers to investigate the feasibility of mobile devices participating in P2P systems.

Bakos et al. [3], [4] simulated Gnutella on a variety of wireless topologies. Then, Nurminen and Noyranen [5] studied BitTorrent's impact on energy consumption on Symbian-based smartphones on both 3G and Wifi networks, considering it feasible to run full peers on mobile devices (download and upload) since BitTorrent's tit-for-tat mechanism [6] rewards uploaders with faster downloads, thus decreasing energy consumption through a shorter total downloading time. Notice that tit-for-tat is less relevant in seeding-rich environments, like the one we propose in this paper.

Kelényi and Nurminen studied the energy aspects of a mobile devices on BitTorrent's Mainline DHT [7]. They concluded that running a full peer on a mobile device is unfeasible due to continuous incoming queries that keep the radio interface on a high-energy state. On the other hand, their client-only implementation allowed mobile devices to use DHT-based services with minimum energy cost.

In a paper currently under review [8], we explored the participation of mobile devices in Spotify's P2P network, in a way that these devices can stream data not only from Spotify's servers (as it currently happens) but also from other users running desktop Spotify clients. When we enabled P2P on the Spotify's Android app, we found that Spotify's gossip-like peer discovery mechanism increases energy consumption (moderately on Wifi and dramatically on 3G). After applying backwards-compatible modifications, energy consumption decreased to a level just slightly higher than the P2P-disabled official app on both Wifi and 3G.

A survey by Hoque et al. [9] showed numerous approaches at optimizing energy consumption of multimedia streaming on Wifi, 3G and LTE networks found in the literature, categorized

[2]Tribler [1] is an existing P2P application. We just modified it to support boosting.

[3]http://www.computerworld.com/s/article/9243331/ (Nov. 2013)

by layer: physical, link, application, and cross layer. Hoque and his colleagues went on to propose their own approach using crowd-sourced viewing statistics and evaluated it on 3G and LTE networks [10]. On these networks, there is a trade-off between (1) pre-fetching large chunks at the risk of downloading useless data if the user stops watching, (2) and fetching small chunks, at the risk of increasing total energy consumption because wireless interfaces stay powered for several seconds after each data transfer. Although these studies focused on client-server streaming, they provide insights that are also applicable to P2P streaming on mobile devices.

C. NAT Gateways

Network address translator (NAT) gateways are used to allow multiple devices to access the Internet using a single IP address. Practically, every Wifi-enabled router (including ADSL, and cable modems provided by Internet service providers) features NAT functionality with few exceptions (e.g., KTH's Wifi network provides public IP addresses). That is, most Wifi connections go through a NAT gateway.

Hätönen et al [11] studied 34 different home gateway models and observed noticeable differences among NAT implementations, reporting that no gateway used the parameter values recommended by the IETF standard for NAT gateways [12].

Mobile networks also use NAT mechanisms, mainly to multiplex an ever-scarcer number of IP addresses. Mäkinene and Nurminen [13] characterized the NAT policies of six major mobile operators, observing that existing TCP NAT traversal techniques work in the majority of these networks. Wang et al. [14] developed a mobile app which allowed them to collect data regarding NAT and firewall policies deployed in over 100 mobile ISPs. Less than a quarter of the these mobile ISPs provided public IP addresses.

NAT gateways supports relatively well client-server applications where clients initiate connections and the server has a public IP address. When a connection is established by a host behind a NAT gateway (client), the gateway will forward packets between client and server, for the most part.

On P2P networks, peers should ideally be able to establish connections to each other because each peer is able to both consume and offer services. On the Internet, however, NAT gateways severely restrict connectivity, making it hard to establish a connection to a peer behind a NAT gateway.

In general, NAT gateways hinder P2P performance [15]. When the only peer with a copy of the content is behind a NAT gateway, content distribution might never start. This paper addresses the particular problem of a piece of content's original source being behind a NAT gateway.

There are many NAT traversal mechanisms to allow hosts behind NAT gateways to establish direct connections to each other. For instance, NatCraker [16], Usurp [17]. Halkes and Pouwelse's UDP NAT puncturing [18] provide an overview of UDP NAT traversal techniques used on the Internet. While our current prototype does not use any of these mechanisms, we are considering integrating some of these mechanisms in future versions.

D. PPSP and Libswift

The Peer-to-Peer Streaming Protocol (PPSP) suite [19] is currently in the process of being standardized at the Internet Engineering Task Force (IETF)[4]. *Libswift*[5] is the reference implementation of PPSP's transport protocol (previously known as *Swift* [20], [21]) and it is available under an open-source license.

PPSP inherits many of its properties from BitTorrent [6], while introducing several improvements. Unlike BitTorrent, PPSP was explicitly designed for multimedia streaming, both on-demand and live; although it can also be used as generic multi-party transfer protocol.

Since PPSP is based on UDP, it is easier to traverse NAT gateways and achieve lower delays. PPSP uses a hashing scheme (Merkle hashes) that greatly improves the dissemination of integrity metadata, reducing user-perceived start-up delay compared to BitTorrent, specially when considering large amounts of data such as a high-definition feature film. While the few hashes needed by a PPSP peer to start checking integrity are likely to fit in a single UDP, a BitTorrent peer must obtain every single hash (several kilobytes packed in a *.torrent file*) before being able to perform any integrity check.

PPSP has been deployed on desktop computers, mainly through its integration on Tribler [1] and a browser plug-in project in collaboration with Wikipedia [22]. Preliminary experiments of *libswift* on Android showed that it was feasible for a modern smartphone to run *libswift* [23].

PPSP has also been proposed as a P2P-based implementation of information-centric networks (ICN). PPSP's Merkle hashes provide naming properties very similar to those proposed in other ICN systems, making PPSP a good candidate for piece-meal adoption on the current IP-based network infrastructure, unlike other ICN proposals that require a clean-slate redesign of the networking infrastructure [20], [21].

III. Roles and Interactions

Before we describe the different parts of the system, we define the different roles in the system and an example to illustrate how the system works and how the different roles interact with each other.

A. Roles

We define roles broadly because our goal is to design a flexible system that is able to support different applications: from best-effort community services to commercial services with strict quality of experience requirements.

- **source**
  User has a piece of content that does not exist in the P2P network yet. Sources can upload content to peers, leechers, and boosters. A source running on a mobile device is referred as **mobile source**.

[4]PPSP Working Group: https://datatracker.ietf.org/wg/ppsp/ (Nov. 2013)
[5]http://libswift.org/ (Nov 2013)

- **peer**
  User downloads and watches video, uploading it to others as a side-effect (as it happens in P2P-based file-sharing systems). Although some mobile devices would be able to function as peers, we conservatively assume that all mobile devices are leechers. Thus, we will use the term **desktop peer** in this paper.
- **leecher**
  Typical operation of mobile devices (**mobile leecher**). Leechers can download and playback the content, but unlike peers, leechers do not upload. We do not consider leechers as free-riders and the system should provide them whenever possible.
- **booster**
  Peer that is configured to automatically redistribute (*seed*) content according to some user-controlled policy (e.g., subscription).
- **meta-booster**
  User who posts content metadata to a subscription service. Boosters subscribed to this meta-booster will automatically start boosting the content, according to their policies.
- **sharer**
  User who helps spreading the content in the same way it is done in social media platforms. That is, sharing pointers (e.g., links) to the data instead of distributing the data themselves.

B. Community-Supported Video Streaming: an Example

The NGO "Bits International" encourages its members to record videos with their smartphones to document their actions and raise awareness. BI members have published their videos on web-based video sharing platforms before but a recent video upload was too controversial and was taken down alleging it was "inappropriate", despite the fact it was a clear exercise of freedom of speech and nobody questioned its legality.

Our example is fictional but takedowns do happen in actual platforms, mainly related to controversial topics[6] and dubious copyright takedowns[7]. In the absence of a court order, the platform owner must decide whether to allow a particular video to be distributed or not.

BI has now decided to use Tribler Mobile to distribute their videos and asks its members to collaborate by sharing and boosting these videos. Now, as long as users donate enough resources in the form of bandwidth and storage (via **boosters**), content can be distributed to a large audience.

Adam, one BI activist, records a new video on his smartphone and uses Tribler Mobile to upload it. The smartphone (**mobile source**) is ready to send the data to others. The app also generates metadata that Adam posts (**meta-booster**) on a subscription service (red dashed arrow from Adam's mobile source in Figure 1). As soon as the metadata is posted, all **boosters** subscribed to Adam's content will download the data from the mobile source and each other (solid black arrows among Adam's mobile source and Adam's boosters). Once a complete copy of the data has been uploaded, Adam can stop uploading and exit the app, thus saving battery. Notice that we have not specified how subscription works nor how mobile source and boosters find each other; we will cover those mechanisms in detail in Sections IV-A and IV-B, respectively.

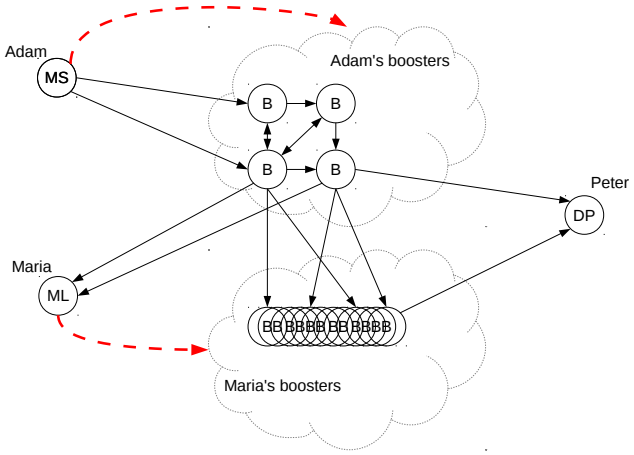

Fig. 1. Interaction among a mobile source (MS), boosters (B), a mobile leecher (ML), and a desktop peer (DP). Dashed red arrows represent the flow of meta-data from publisher to subscribers. Solid black arrows represent data flows.

Adam also shares a link (**sharer**) on social networks to notify *followers* and *friends* that there is a new video available for streaming. Maria, one of Adam's followers, opens the link on her smartphone (**mobile leecher**), streaming the video from boosters (notice that, in this case, the mobile source is already off-line). Maria likes the video and posts the link on the subscription service (**meta-booster**). This action adds much distribution capacity since hundreds of boosters (including her own computer at home) are subscribed to Maria. Maria is happy to contribute in ways that do not involve draining her battery and mobile data traffic.

Peter opens the link Adam shared on his computer and streams the video from boosters, his computer not only downloads data but it also uploads it to others (**desktop peer**).

In this example, as in our prototype, boosters boost all content they are subscribed to. This is not a limitation of the system in itself but just a simplification to accelerate prototype implementation and deployment. More advanced boosters would decide whether to boost a piece of content depending on different factors.

We define the term booster broadly to allow for a broad spectrum of boosters that may be combined with incentive mechanisms, both centralized (e.g., private BitTorrent trackers [24], [25], [26]) and distributed (e.g., BarterCast [27], [28]

[6]http://news.cnet.com/8301-1023_3-57513354-93/no-easy-outs-for-youtube-in-islam-video-controversy/ (Nov. 2013)

[7]https://www.eff.org/press/releases/lawrence-lessig-strikes-back-against-bogus-copyright-takedown (Nov. 2013)

and Dispersy [29]). In this paper, however, we will mainly focus on *altruistic boosters*.

IV. Infrastructure Services

We use PPSP (introduced in Section II-D) as transport protocol for peers to transfer data to each other but other infrastructure services are needed to coordinate peers.

This section describes each of the main infrastructure services supporting the system. For each one, we first expose general requirements and challenges, proposing generic mechanisms which are open to different implementations, according to the specific needs and resources. Then, we provide a description of our prototype's implementation details and discuss their properties and limitations. In some cases, we discuss alternative implementations that may be used in different contexts.

A. Peer Discovery

In the example in Section III-B (Figure 1), peers connect and transfer data to each other. How does a peer find one or more peers where the requested data is stored and ready to be transferred? A *peer discovery mechanism* keeps track of peers sharing a piece of content and peers can request a list of peers, given a content identifier (identifiers are a hash string derived from the data itself and can be used for integrity protection).

Popular P2P networks use two complementary kinds of peer discovery mechanisms: tracker and gossip-like. Some systems (e.g., Spotify) use both of them [2]. In this paper, we focus on tracker-like peer discovery.

A tracker is a centralized registry where all peers sharing a piece of content are registered. For each request, the tracker returns a short list of peers (if there is any). Trackers started as centralized services running on a single machine and have evolved to become decentralized services. For instance, nowadays BitTorrent can use both centralized trackers, where the service is run on a single (or few) server(s), and decentralized DHT-based trackers, where the service is run by millions of DHT-enabled BitTorrent peers.

1) Challenges: One of the side-effects of IPv4's address exhaustion is the massive deployment of NAT gateways. NAT gateways are commonly used where several devices need Internet connection but only one public IP address is available. Nowadays, most Wifi access points are also NAT gateways and much of the mobile operators do not provide a public IP address, but a private one behind a large NAT gateway [13], [14]. Therefore, mobile devices connected through Wifi or mobile networks are very likely to be *NATed*.

NAT gateways are a great challenge for P2P networks because NATs restrict connectivity. In general, it is difficult to establish a connection to a *NATed peer* and even harder if both are NATed. Several techniques have been proposed and deployed but NATs remain a main challenge for P2P-based systems.

This paper focuses on the particular challenge of uploading new content from a mobile source. Since it is the only source available, we must guarantee that a connection can be established between the mobile source and, at least, a peer (a booster in our case).

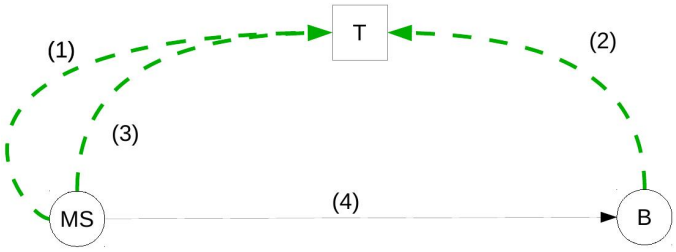

Fig. 2. Reverse peer discovery. **(1)** Mobile source (MS) periodically requests peers from the tracker (T) but it does not announce itself, T returns an empty list. **(2)** a booster (B) requests peers from T and it announces itself, T returns an empty list. **(3)** MS requests peers, T returns a list containing B. **(4)** MS establishes a connection to B, data transfer begins.

Popular P2P systems like BitTorrent are pull-based systems. In BitTorrent, a peer offering new content (a smartphone with the original copy, in this case) would register themselves in the peer discovery service (tracker) and wait for other peers to establish connections. Unfortunately, that smartphone is likely to be NATed, making it very hard to establish connections from other peers to the smartphone.

Mobile devices with an open connection (i.e., not NATed) are challenging in a different way. Normally, once a peer is registered in a tracker, this peer cannot unregister itself. Instead, the tracking mechanism only *unregisters* peers if they have not renewed their registration within a period of time of (commonly several minutes). Thus, if a mobile peer registers itself and it happens to be reachable, it runs the risk of receiving connections attempts over a long time period, which brings the radio to a high-power state, thus consuming extra energy. These connection attempts will reach the device even after the P2P app no longer runs on the device.

Thus, we need to make sure that he mobile peer does not register itself in the tracker to limit energy consumption.

2) Booster Discovery: We have reversed peer discovery so that the mobile source establishes connections to other peers, to increase the success rate of mobile sources uploading new content to the P2P network and reduce energy consumption.

We present now a general description of the mechanism. Then, we provide implementation details regarding our prototype implementation.

A mobile source uploading new content to the P2P network posts the content's identifier on the subscription service (Section IV-B) and requests a list of peers from the tracker in few-seconds intervals, as shown in Figure 2.

As soon as a booster is notified of a new piece of content, the booster requests peers and registers itself in the tracker. In the next periodic tracker request, the mobile source will receive the booster's address (IP address and UDP port) and establish a connection to the booster. If the booster were not reachable, the mobile source keeps querying the tracker for more peers until it establishes connections and transfers a full copy of the content. At that point, the mobile app can be

stopped. Boosters will continue distribution to other boosters, peers, and mobile leechers.

Ensuring successful data upload from a mobile source is relatively simple and cheap. The user himself (or a trusted party) can set up a non-NATed booster at home or on a rented server and configure it to boost his own content.

It is worth emphasizing that this simple reverse mechanism allows us to build a fully decentralized streaming service where mobile devices can upload content even in the presence of NAT gateways. We plan to include other NAT traversal mechanisms in the future.

3) Prototype Implementation: In our prototype, we aim for a fully-decentralized system and thus do not rely on a centralized tracker. Instead, we use a DHT-based mechanism: BitTorrent's Mainline DHT (MDHT).

In particular, we use Pymdht, a MDHT implementation in Python designed to collect experimental data and to be easy to modify. We have used this implementation before [30], [31], and it is publicly available as open-source software. In this case, we modify Pymdht to support the reverse lookup mechanism described above.

On our modified Tribler desktop application, boosters that lookup a content identifier and find no peers, announce themselves to the DHT-based tracker even though they have no content to offer. This is done by sending an `announce_peer` request to the appropriate nodes in the DHT overlay.

On the Tribler Mobile app, there are two different configurations: mobile source and mobile leecher.

The mobile source sends `get_peers` requests to the DHT periodically (20 seconds in the current implementation) but it does not announce itself. As soon as the DHT returns a lists of boosters, the mobile source establishes connections to a subset of them and uploads data. The app user is shown a status screen and he should wait until a full copy of the content has been transferred before closing the app. Once the app is closed, the app will not send/receive traffic, limiting energy consumption.

The mobile leecher performs a single DHT lookup, sending `get_peers` requests to DHT nodes, and as the mobile source, it does not announce itself. As soon as the DHT returns peers' addresses, the mobile leecher establishes connections and downloads data from them, starting playing back as soon as the playback buffer is sufficiently populated.

There are two other major differences between the mobile and the desktop implementations. First, while the desktop implementation is basically vanilla Pymdht, the mobile implementation does not perform any DHT routing and maintenance tasks, ignoring all requests from other nodes. There are two reasons for this: (1) NATed DHT nodes cannot properly route DHT lookups, harming lookup performance for other nodes if they try [30], and (2) to avoid future requests from other peers that would power up the radio interface even after the app has been closed.

The second difference is that we translated Pymdht from Python to Java due to the current poor support for Python libraries on Android. Nevertheless, the translation was done with great care to make both Java and Python implementations equivalent.

B. Subscription Service

Boosters require a subscription service to receive notifications of newly published content metadata. Once a booster receives a notification, it will decide whether to boost (i.e., fetch and redistribute the data), according to its configuration. The functionality needed from the subscription service will depend on the boosting incentive mechanisms deployed in the system.

Our goal is to design a flexible system that supports a wide range of boosting incentives. On the other hand, our prototype provides a concrete implementation of a simple mechanism as a proof of concept.

Existing P2P-based file-sharing systems have different incentive mechanisms. BitTorrent, for instance, only has incentives for peers currently downloading data from other peers (tit-for-tat [32] and super-seeding [33]). There are only indirect incentives for *seeding* (uploading data to others after the peer has completely downloaded a piece of content). If there are enough users seeding, it is likely that the peer that seeded one piece of content at some point in time will download a different piece of content from another altruistic peer in the future. This may be viewed as a kind of indirect reciprocity.

Private BitTorrent trackers introduce a direct incentive mechanism for seeding. A private tracker keeps track of each peer's share ratio (bytes uploaded divided by bytes downloaded). Typically, the tracker enforces a minimum ratio by refusing to return a list of peers to those peers whose ratio is lower than the minimum [25], [26], [24].

BarterCast [27] allows peers to trade bandwidth among each other in a decentralized manner, converting bandwidth into currency. Our long-term goal is to integrate a similar mechanism to support fully-decentralized boosting incentives.

1) Prototype Implementation: Our prototype uses Twitter as subscription service. We first describe how it is used with an example, and then discuss its advantages and limitations.

When Bob wishes to contribute to the distribution of content published by Alice, Bob simply adds Alice's Twitter handle (`@alice`) to his booster's configuration. Then, Bob's booster will periodically monitor `@alice` for new *tweets* containing content metadata in the form of a PPSP URI schema (i.e., ppsp://f37...f1). Shortly after Alice *tweets* the PPSP URI, Bob's booster is notified of the new content and it starts downloading and redistributing the content whose identifier is f37...f1. It is worth mentioning here that the identifier is derived through cryptographic hash derived from the data, providing data integrity protection.

As a side-effect, Alice is also *sharing* a PPSP URI. That means that Alice's followers can click on the URI and start streaming video to their devices (either desktop or mobile) from boosters. If any of these followers re-posts (*retweet*) the URI, the boosters subscribed to that follower will also start boosting, adding distribution capacity.

Fig. 3. Posting of a PPSP.me link on Twitter.

We could have used RSS subscription, which most BitTorrent clients already support. Twitter, however, has millions of interconnected users [34] and offers additional user-friendly features (retweet in particular) that produce powerful network effects [35]. In our case, they enable fast and wide dissemination of content metadata.

Other advantages of using Twitter as boosting subscription mechanism are: (1) it couples content to a person or organization, making it relatively simple to decide where to donate resources (via boosting), and (2) a single system handles meta-boosting and sharing.

The obvious limitation of using Twitter is having a single point of failure and the risk that Twitter would not accept PPSP URIs for commercial, legal, or any other reason. That is why our long-term goal is to switch to a fully-distributed pub-sub service.

C. PPSP.me link translator

This part is technically not part of the system, but just a convenient utility to bootstrap deployment. We include it in this paper because we consider incremental deployment a critical part of a successful system.

Tribler Mobile uses its own URI schema: ppsp://hash. Unlike HTTP URLs (e.g., http://kth.se/en), PPSP URIs do not contain any information regarding the content's location. In fact, PPSP could be considered information-centric, as discussed in Section II-D. Instead, data's location is discovered via a peer discovery service, as we discussed in Section IV-A.

Unfortunately, browsers and mobile apps do not implement any handle for PPSP URIs yet, showing an error to the user who tries to open the *PPSP link*.

To bootstrap deployment, we have implemented a simple link translator which leads users to useful information about Tribler Mobile and how to playback content referred by a PPSP link.

Thus, when Alice records a video on her smartphone and chooses to publish it via Tribler Mobile, our software will generate an HTTP URL in the form of http://ppsp.me/hash for Alice to *tweet* it (see Figure 3 for an actual example).

Then, Alice's followers who click on the ppsp.me link, they will land on a simple web page with instructions to install Tribler Mobile. Once Tribler Mobile is installed, the user is asked whether he wants to always open ppsp.me links using Tribler Mobile instead of a browser.

While the ppsp.me web service is a single point of failure, it is not a critical service, just a convenient way of introducing new users to Tribler Mobile. Users who already have Tribler Mobile installed do not contact ppsp.me servers at all, and directly proceed to find peers and stream content from them. Even users who have not installed Tribler Mobile yet can find information on the web about how to install the app, should ppsp.me be temporally (or even permanently) unavailable.

Notice that ppsp.me links have no effect on boosters, since boosters can parse both PPSP URIs and ppsp.me links, and treat them equally.

V. Conclusion

We have presented a P2P video streaming platform where mobile devices are well integrated, considering their limitations: battery, mobile data caps, and limited connectivity due to NAT gateways.

In particular, we are the first to propose, implement, and deploy a complete system where mobile devices can publish content directly into the P2P network and let other peers (boosters) distribute the content to a potentially large audience (including other mobile devices).

Our contributions include a reverse peer discovery mechanism that addresses two important issues. First, it allows mobile devices to traverse NAT gateways —these gateways are widely-deployed on Wifi and mobile networks. Second, it saves battery because the mobile device is never registered on the peer discovery service, thus other peers will never contact the mobile device —the device's radio must switch to high-power state when receiving packets.

Our prototype has been built incrementally. Whenever possible we have used existing proven-to-work components, implementing backwards-compatible modifications to adapt each component to our requirements. For instance, our reverse peer discovery mechanism is backwards-compatible with BitTorrent's Mainline DHT, allowing us to leverage this multi-million DHT overlay.

Our long-term goal is to evolve this prototype into a fully-decentralized system without any single point of failure (our current prototype relies on Twitter). Furthermore, we plan to integrate more sophisticated NAT traversal mechanisms, and incentive mechanisms to create a marketplace for publishers and boosters.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Seventh Framework Programme (QLectives), SSF (E2E-Clouds), and EIT ICTLabs.

The authors would like to thank Elric Milton, Niels Zeilemaker, Mihai Capotă for their help setting up the infrastructure.

REFERENCES

[1] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. J. Epema, M. Reinders, M. R. van Steen, and H. J. Sips, "Tribler: a social-based peer-to-peer system: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 2, pp. 127–138, 2008.

[2] G. Kreitz and F. Niemela, "Spotify - Large Scale, Low Latency, P2P Music-on-Demand Streaming," in *P2P'10*, 2010.

[3] B. Bakos, G. Csúcs, L. Farkas, and J. K. Nurminen, "Peer-to-peer protocol evaluation in topologies resembling wireless networks. an experiment with gnutella query engine."

[4] B. Bakos, L. Farkas, and J. K. Nurminen, "P2p applications on smart phones using cellular communications," in *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*, vol. 4. IEEE, 2006, pp. 2222–2228.

[5] 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.

[6] B. Cohen, "Incentives Build Robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer Systems*, vol. 6. Berkeley, CA, USA, 2003.

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

[8] R. Jimenez, G. Kreitz, B. Knutsson, M. Isaksson, and S. Haridi, "Integrating Smartphones in Spotify's Peer-Assisted Music Streaming Service."

[9] M. Hoque, M. Siekkinen, and J. Nurminen, "Energy efficient multimedia streaming to mobile devices — a survey," vol. PP, no. 99, 2012, pp. 1–19.

[10] M. A. Hoque, M. Siekkinen, and J. K. Nurminen, "Using crowd-sourced viewing statistics to save energy in wireless video streaming," in *Proceedings of the 19th annual international conference on Mobile computing and networking*, ser. MobiCom '13. New York, NY, USA: ACM, 2013, pp. 377–388.

[11] S. Hätönen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, and M. Kojo, "An experimental study of home gateway characteristics," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 260–266. [Online]. Available: http://doi.acm.org/10.1145/1879141.1879174

[12] R. Braden, *RFC 1122 Requirements for Internet Hosts - Communication Layers*, Internet Engineering Task Force, Oct. 1989. [Online]. Available: http://tools.ietf.org/html/rfc1122

[13] L. Makinen and J. K. Nurminen, "Measurements on the feasibility of tcp nat traversal in cellular networks," in *Next Generation Internet Networks, 2008. NGI 2008*. IEEE, 2008, pp. 261–267.

[14] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 374–385, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2043164.2018479

[15] B. Ford, "Peer-to-peer communication across network address translators," in *In USENIX Annual Technical Conference*, 2005, pp. 179–192.

[16] R. Roverso, S. El-Ansary, and S. Haridi, "Natcracker: Nat combinations matter," in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, 2009, pp. 1–7.

[17] S. Niazi and J. Dowling, "Usurp: Distributed nat traversal for overlay networks," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science, P. Felber and R. Rouvoy, Eds. Springer Berlin Heidelberg, 2011, vol. 6723, pp. 29–42.

[18] G. Halkes and J. Pouwelse, "Udp nat and firewall puncturing in the wild," in *NETWORKING 2011*, ser. Lecture Notes in Computer Science, J. Domingo-Pascual, P. Manzoni, S. Palazzo, A. Pont, and C. Scoglio, Eds. Springer Berlin Heidelberg, 2011, vol. 6641, pp. 1–12.

[19] A. Bakker, P. R., and V. Grishchenko, "Peer-to-Peer Streaming Peer Protocol (PPSPP)," 2013.

[20] F. Osmani, V. Grishchenko, R. Jimenez, and B. Knutsson, "Swift: the missing link between peer-to-peer and information-centric networks," in *Proceedings of the First Workshop on P2P and Dependability*, ser. P2P-Dep '12. New York, NY, USA: ACM, 2012, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/2212346.2212350

[21] V. Grishchenko, F. Osmani, R. Jimenez, J. Pouwelse, and H. Sips, "On the design of a practical information-centric transport," PDS Technical Report PDS-2011-006, Tech. Rep., 2011.

[22] A. Bakker, R. Petrocco, M. Dale, J. Gerber, V. Grishchenko, D. Rabaioli, and J. Pouwelse, "Online video using bittorrent and html5 applied to wikipedia," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, 8 2010, pp. 1 –2.

[23] R. Petrocco, J. Pouwelse, and D. Epema, "Performance analysis of the libswift p2p streaming protocol," in *12-th IEEE International Conference on Peer-to-Peer Computing (P2P12)*. IEEE, 2012. [Online]. Available: http://dx.doi.org/10.1109/P2P.2012.6335790

[24] J. J.-D. Mol, J. A. Pouwelse, D. H. Epema, and H. J. Sips, "Free-riding, fairness, and firewalls in p2p file-sharing," in *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*. IEEE, 2008, pp. 301–310.

[25] M. Meulpolder, L. D'Acunto, M. Capotă, M. Wojciechowski, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "Public and private bittorrent communities: a measurement study," in *Proceedings of the 9th international conference on Peer-to-peer systems*, ser. IPTPS'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863145.1863155

[26] Z. Liu, P. Dhungel, D. Wu, C. Zhang, and K. Ross, "Understanding and improving ratio incentives in private communities," in *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, 2010, pp. 610–621.

[27] M. Meulpolder, J. Pouwelse, D. H. J. Epema, and H. Sips, "Bartercast: A practical approach to prevent lazy freeriding in p2p networks," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1–8.

[28] R. Delaviz, N. Andrade, and J. A. Pouwelse, "Improving accuracy and coverage in an internet-deployed reputation mechanism," in *Peer-to-Peer Computing'10*, 2010, pp. 1–9.

[29] N. Zeilemaker and J. Pouwelse, "Open source column: Tribler: P2p search, share and stream," *SIGMultimedia Rec.*, vol. 4, no. 1, pp. 20–24, Mar. 2012. [Online]. Available: http://doi.acm.org/10.1145/2206765.2206767

[30] R. Jimenez, F. Osmani, and B. Knutsson, "Connectivity properties of Mainline BitTorrent DHT nodes," in *9th International Conference on Peer-to-Peer Computing 2009*, Seattle, Washington, USA, 9 2009.

[31] ——, "Sub-second lookups on a large-scale kademlia-based overlay," in *11th International Conference on Peer-to-Peer Computing 2011*, Kyoto, Japan, 8 2011.

[32] B. Cohen, "Incentives to Build Robustness in BitTorrent," in *Proceedings 1st Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, Jun. 2003, http://bittorrent.com/bittorrentecon.pdf.

[33] Z. Chen, Y. Chen, C. Lin, V. Nivargi, and P. Cao, "Experimental analysis of super-seeding in bittorrent," in *Communications, 2008. ICC '08. IEEE International Conference on*, 2008, pp. 65–69.

[34] M. Gabielkov and A. Legout, "The complete picture of the twitter social graph," in *Proceedings of the 2012 ACM conference on CoNEXT student workshop*. ACM, 2012, pp. 19–20.

[35] D. Boyd, S. Golder, and G. Lotan, "Tweet, tweet, retweet: Conversational aspects of retweeting on twitter," in *System Sciences (HICSS), 2010 43rd Hawaii International Conference on*, 2010, pp. 1–10.