Device-to-Device Mobile Data Offloading for Music Streaming

Sylvia T. Kouyoumdjieva and Gunnar Karlsson
ACCESS Linnaeus Center, School of Electrical Engineering
KTH Royal Institute of Technology, Stockholm, Sweden
Email: {stkou, gk}@ee.kth.se

***Abstract*—Device-to-device communication (also referred to as opportunistic networking) is considered a feasible means for offloading mobile data traffic. Due to the sporadic nature of contact opportunities, applications in the domain of device-to-device communication are assumed to be delay-tolerant, with content delivery deadlines being in the order of hours. However, predictions suggest that by 2020 more than 75% of the traffic volumes at mobile operators will be generated by multimedia contents which is often seen as data served in real-time. In this paper we explore how the concept of opportunistic networking can be used for dissemination of real-time streaming contents for users in urban environments without degrading quality of experience. We first present a general framework for offloading multimedia data that is organized in terms of playlists, and we then investigate the performance of the framework in realistic urban environments using the music streaming service Spotify as a use-case. Our results show that it is feasible to use opportunistic device-to-device communication in the context of music streaming. We demonstrate that the system performance is insensitive to a number of parameters such as playlist length distribution, and initial content availability distribution, however it exhibits sensitivity towards the amount of requested data and the node density.**

***Index Terms*—mobile data offloading, device-to-device communication, opportunistic networking, music streaming, Spotify.**

I. Introduction

The proliferation of mobile devices has changed tremendously the way in which people consume information. The huge amounts of data delivered to mobile users on an everyday basis requires mobile operators to face the challenge of increased traffic volumes in their networks. In fact, predictions are that by 2020 mobile operators will experience around 24 EB of monthly mobile data traffic. Approximately 75% of this traffic is expected to be generated by multimedia contents [1]. Mobile operators are thus exploring different possibilities for offloading traffic volumes to alternative networks, and one of the promising solutions that has been proposed is device-to-device (also called opportunistic) communication. Opportunistic communication allows devices in proximity to exchange data directly with each other without relying on infrastructure.

Opportunistic communication has been perceived as a means of disseminating and offloading delay-tolerant contents characterized with content delivery deadlines in the order of hours such as news or software updates. However with the increasing amounts of multimedia contents such as music and video delivered to mobile devices, offloading solely delay-tolerant contents may not be enough to reduce the traffic volumes at mobile operators. Thus, we here question whether the concept of device-to-device opportunistic communication could also be applicable for disseminating data with much shorter deadline requirements, such as streaming data. Only few studies [2], [3], [4] evaluate direct device-to-device communication as an alternative for delivering on-demand streaming to mobile devices, however the solutions rely on the presence of infrastructure for supporting the dissemination process. Instead, we evaluate the feasibility of an infrastructureless solution for offloading streaming data while maintaining the quality of experience for the end user.

The contributions of this work are two-fold. We first introduce a framework for disseminating real-time streaming contents in an opportunistic manner. We then evaluate the performance of the framework via extensive trace-driven simulations based on realistic mobility traces that recreate mobility patterns of urban users. We base our evaluation on the popular music streaming service Spotify as a use-case. The rationale behind choosing a music streaming service as a use-case is the length of the flows. Music streams are long-lived (in the order of hours) and although they do not require high data rates, due to their duration they result into high traffic volumes. Our results show that opportunistic device-to-device communication is a viable means for offloading streaming data. The system performance exhibits high sensitivity towards the node density and the amount of requested data, but it is insensitive towards the distribution of the playlist length and the initial content availability.

The rest of this paper is structured as follows. In Section II we present our proposed general framework for offloading multimedia contents via device-to-device communication. Section III presents a popular music streaming service, Spotify, which we further use as a case study for evaluating the framework. Sections IV and V summarize the evaluation scenario and results, while Section VI discusses the related work and positions our proposal with respect to it. Finally, we conclude in Section VII.

II. Opportunistic Dissemination and Streaming

Opportunistic device-to-device communication allows nodes equipped with mobile devices to exchange data with one

ISBN 978-3-901882-83-8 © 2016 IFIP

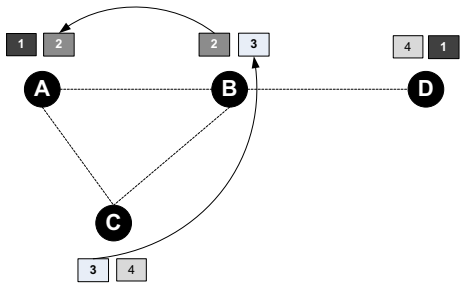


Fig. 1. An example with four nodes {A, B, C, D} interested in four content items {1, 2, 3, 4}. Each node has one content item in its cache (left rectangle) and one content item in its requested playlist to obtain in the nearest future (right rectangle). Dashed lines show nodes in direct communication range; solid lines show possibilities for content exchange.

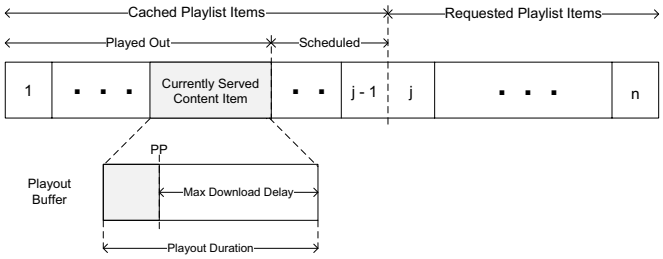


Fig. 2. The cached playlist holds items that are already available on the device; these can be either items that have been played by the device, or items that have been downloaded for future usage. The requested playlist holds items that still need to be downloaded. The playout point is denoted with PP.

another when in direct communication range. Data exchange is only possible if nodes share one or more common interests. An interest is expressed in the form of a subscription to a specific content category through a publish/subscribe interface exposed by an opportunistic content dissemination system [5]. Nodes encounter other peers, and are consecutively able to discover and download contents stored on those peers, as they move through an area. However, without keeping track of the mobility pattern of each and every node in the system, it is not possible to predict beforehand neither when any two devices with shared interest would be in direct communication range, nor how long the duration of their contact would be. Due to this unpredictability, opportunistic communication is mostly proposed as an offloading solution for delay-tolerant content, and is often considered inappropriate for sharing real-time contents such as streaming data.

Streaming data *is* provided in real-time when considering the data that is delivered to a device. However, shifting the viewpoint towards contents changes the definition from real-time data into *data with tight delay constraints*. We define contents to have tight delay constraints if its playout is in real-time but its delivery to the requesting application happens within some predefined delay boundaries. Contrary to traditional on-demand data delivery, the delay boundaries in this case are more generous. Let us illustrate this concept with an example, Fig. 1. Four mobile nodes {A, B, C, D} are interested in obtaining four content items {1, 2, 3, 4} constituting a *playlist*. The dashed lines show direct communication links between nodes. At the beginning, all playout buffers are empty. Thus, each of the nodes begins by streaming one of the items of interest from a server via the cellular network and playing it out; node A streams item 1, node B streams item 2, etc. Since all nodes are interested in all four of the content items, the application at each node is aware both of the currently played content item, as well as of other content items that are not available on the device but are part of the playlist; we call this the *request playlist*. In this example, the request playlist of node A consists of item 2, the request playlist of node B consists of item 3, etc. Normally node A would request item 2 from the server and stream it in real-time once it is done streaming item 1. However, if node A's application is aware of the items in its request playlist, it may as well attempt to obtain these items in advance in alternative ways, for example via opportunistic device-to-device communication, *before* they are requested by the application. Then, once the streaming of item 1 is over, item 2 can be played seamlessly from a local cache. In our example, node A has item 2 in its request playlist, while node B which is a neighbor of node A already has item 2 which it has obtained via streaming; node B could consecutively share item 2 upon request with node A.

In the context of delivering multimedia to mobile devices, a node can either *stream* the data in real-time directly from a server via the cellular network, or it can *download* the data from a peer via an opportunistic contact. Contents available in the cache of a node is then *played out* whenever the application requests it. Observe that play out is oblivious whether the data source is local (the device's cache) or remote (the server).

Fig. 2 illustrates the general concept of cached and requested playlist items. A *cached* playlist item is already available on the device. Observe that we do not specify how the cached item has been made available to the device. For instance, it could have been streamed via a cellular network, or downloaded via an opportunistic contact. A *requested* playlist item is a content item that the device is expected to provide to the application in the future. A special case of a cached item is the currently served content item, i.e. the item that is being played out to the requesting application. The currently served content item defines the maximum download delay as the time until the next play out. In case that the currently consumed content item is the last item in the cached playlist, the maximum download delay measures how long the node could search new content items in neighboring devices *before* its buffer under-flows. Algorithm 1 presents the framework for offloading streaming data with tight delay constraints. As long as the request playlist of a node is not empty (line 3), the node actively searches for peers in its vicinity that could provide it with useful contents. Upon downloading a content item from a neighbor, the node stores the item in its scheduled playlist, and removes it from the request playlist (lines 7 and 8). Whenever the playout buffer reaches the end of the currently consumed content item (line 10), the requesting application checks whether there are any scheduled content items; if this

Algorithm 1 Framework for Offloading of Streaming Data

1: $\mathcal{S} \leftarrow$ set of scheduled content items
2: $\mathcal{R} \leftarrow$ set of requested content items
3: **while** $\mathcal{R}$ is not $\varnothing$ **do**
4: search for peers in vicinity
5: **if** available peer with shared interest **then**
6: download content item
7: add content item to $\mathcal{S}$
8: remove content item from $\mathcal{R}$
9: **end if**
10: **if** playout buffer is $\varnothing$ **then**
11: **if** $\mathcal{S}$ is $\varnothing$ **then**
12: **if** replay **then**
13: replay cached content item
14: **else**
15: stream content item from server
16: **end if**
17: **else**
18: play out next content item
19: **end if**
20: **end if**
21: **end while**

is the case, the first of the scheduled content items is played out from the local cache (line 18). However, if the scheduled content items playlist is empty, the requesting application has two options: either to replay a locally cached content item that has already been played out before (line 13) or to stream a new content item from the server via the cellular network (line 15).

The playout system in a device can be described as a G/G/1 queueing system, Fig. 3. Nodes arrive in the communication range of an observer node j with mean arrival rate λ. However, only a subset of these nodes could be useful to the observer node. A contact is useful if it can deliver contents of interest to the observer node while in its communication range. Useful contacts occur at a content solicitation rate $\lambda' < \lambda$, and can be described with a distribution of useful inter-contact times $f(t)$. The distribution of the useful inter-contact times is defined by the underlying node mobility, as well as the popularity of the content items in the requested playlist of the observer node. The duration of the currently served content item τ in node j defines the service time. The service time follows a distribution $g(\tau)$. Opportunistic mobile data offloading for streaming services is then defined as feasible *if* the mean inter-arrival time of useful nodes to node j is larger than the mean service time for node j, $\mathbb{E}[f(t)] > \mathbb{E}[g(\tau)]$. Contrary to traditional queueing theory, the main objective for the system is to operate in a saturated state.

III. Music Streaming with Spotify

Spotify is a popular online peer-assisted music streaming service [6]. Spotify offers an extensive music catalog with more than 20 million music tracks to desktop, mobile and web users. In the wired domain, users can stream music directly from the Spotify servers or via peers who have already

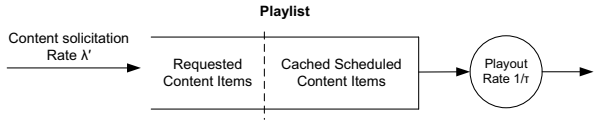


Fig. 3. The playout system in a device represented as a G/G/1 queueing system.

```
<track href="spotify:track:5eCgNATwXgRc4mZx9NymGJ">
   <name>Waiting For Love</name>
   <artist href="spotify:artist:1vCWHaC5f2uS3yhpwWbIA6">
      <name>Avicii</name>
   </artist>
   <album href="spotify:album:0LUr5Q06EQu7QIid7cACFU">
      <name>Waiting For Love</name>
      <released>2015</released>
      <availability>
         "AD", "AR", ... "TW", "UY"
      </availability>
   </album>
   <track-number>1</track-number>
   <length>228.750000</length>
   <popularity>96</popularity>
</track>
```

Listing 1. Sample data structure of a Spotify track. Some parameters are omitted in order to improve readability.

downloaded the track and cached it on their devices. It has been shown that approximately 40% of all data delivered to users is provided by peers instead of by the server. In the wireless domain, however, users are currently only allowed to download tracks directly from the Spotify servers. Recently Spotify announced that the usage of their service on mobile devices has surpassed the usage of desktop and web clients, with more than 50% of users streaming music on their smartphones or tablets [7]. This shift in usage significantly increases the traffic volumes on the servers as well as on the mobile operator, and would ultimately require a shift in the way data is provisioned to devices in the wireless domain.

A. Spotify Data Analytics

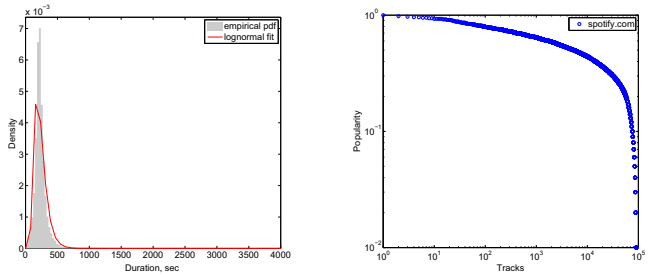
Spotify provides an application programming interface (API)[1] which allows developers to access and extract meta data about tracks, artists as well as users and their playlists. Searching through Spotify's database is performed by a simple GET request:

`GET http://api.spotify.com/v1/search?q=text&key`$_i$`=val`$_i$

Here, `text` denotes a free text, for instance a name of a track, while `key`$_i$ and `val`$_i$ are additional key-value filtering options for improving the search results. For example, in this section we make use of the `market` key which expects an ISO 3166-1 alpha-2 country code as a value to narrow down search results to a specific country. Another popular filtering key option is `type` which can take values of `track, artist` and `album`.

The data provided by the API in response to a GET search request is presented in the form of a JSON object, and in essence it consists of a collection of key-value pairs associated with a particular request. A sample data structure

[1]Spotify's API is available at http://developer.spotify.com/web-api/



(a) (b)

Fig. 4. Statistics collected from 100 000 tracks on Spotify. (a) Distribution of track durations. (b) Distribution of track popularity.

corresponding to a search of a popular track is presented in Listing 1; the format is converted into XML for the sake of readability. The track element is the highest level element in the structure, and it contains various meta data associated with the particular track, such as the length of a track (in milliseconds), the associated artist and album, as well as the track availability across the world. Each track has a unique identifier in the form of a URI which is provided by the `href` key.

To obtain a better understanding of the parameters of data distributed by the Spotify service, we implemented a simple crawler in Python and collected statistics of 100 000 tracks out of the Spotify catalog. Fig. 4(a) presents the distribution of track durations. Most tracks have durations of few minutes, with the mean track duration being approximately 225 s. However we see that few tracks exhibit much longer durations; these are for example classical music pieces. None of the general probability distributions is able to fully describe the distribution of track durations however the lognormal distribution gives the closest fit.

Spotify also provides their own estimate of the popularity of tracks as a value between 0 and 100, with 100 denoting the most popular tracks. Popularity is estimated with respect to the total number of times a track has been played, as well as how recently it has been played. Fig. 4(b) illustrates the popularity distribution of 100 000 tracks plotted on a loglog scale. (We here represent popularity with values between 0 and 1 on the y-axis to improve readability.) Only few tracks have a high popularity score. Those are the tracks that could potentially be solicited by neighboring peers instead of streamed directly from the Spotify server in the wireless domain. We therefore further focus on evaluating only high popularity tracks. However we should note that when considering tracks for disseminating via opportunistic device-to-device contacts, we should not look into track popularity on global scale. Instead, since device-to-device communication allows data exchange among devices in proximity, we should attempt to describe data popularity in a smaller and more local scope, i.e. the scope of a country or even an area in a country. Spotify currently provides data at the scope of a country in its availability tag.

Fig. 5 shows the popularity distribution of tracks currently present in the Top 50 chart for three European countries:

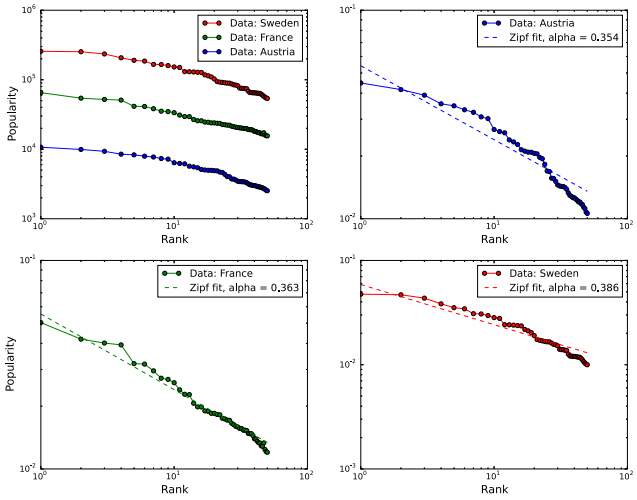


Fig. 5. The popularity distribution of the top 50 tracks per country can be described with a Zipf distribution. Popularity is calculated based on the number of times a track has been played in the course of a week. Although tracks tend to have different popularity across countries, the parameter describing the Zipf distribution does not vary significantly.

Austria, France and Sweden. (We take into account only data from European countries instead of the USA or China due to the fact that the area is smaller and the statistics represent a more localized view of the track popularity.) The graph plots the popularity of tracks in terms of the times a track has been played with respect to the position of the track in the chart. Users in different countries exhibit different usage patterns of the Spotify service, with the service being most commonly used in Sweden, and least used in Austria (upper left in Fig. 5). We then fit the popularity curve into a Zipf distribution, and we estimate the parameter α for each of the three datasets. It is interesting to see that although the usage statistics per country vary significantly (the most popular song in Sweden is played more than 250 000 times, while in Austria – less than 10 000 times), the Zipf distribution that describes the popularity exhibits similar behavior (with $\alpha \in [0.354, 0.386]$).

Finally, we provide a rough quantitative measure of the maximum offload that could be achieved if popular tracks are to be disseminated via opportunistic contacts instead of streaming the contents directly from the Spotify servers. Table I shows the amount of data downloaded from the server when the duration of each song has a mean of 225 s, the audio stream is encoded with Ogg Vorbis q9 with 320 kbps (the typical encoding scheme for premium users of Spotify). Currently, the top 50 most popular tracks amount to hundreds of terabits of data downloaded from a server, and traversing a mobile network which could potentially be offloaded to alternative networks.

B. Spotify and Opportunistic Communication

In order to be able to examine the performance of a music streaming service such as Spotify in the opportunistic domain, we first need to define how the current protocols and data formats would fit into the context of device-to-device

TABLE I
TOP 50 CHART: TOTAL WEEKLY DOWNLOADS AND MAXIMUM OFFLOAD CAPACITY WHEN AUDIO STREAMS ARE ENCODED WITH OGG VORBIS Q9 AT 320 KBPS.

Country	Total downloads (millions)	Offload capacity (terabits)
Austria	0.2 M	16 Tb
France	1.3 M	80 Tb
Sweden	5.4 M	380 Tb

communication. We thus present a sample publish/subscribe framework for opportunistic networks. Similar to the publish/subscribe interface exposed currently by Spotify to its users [8], in the opportunistic domain users should be able to express interest in the tracks they wish to listen to. However, while in the wired domain a user is able to subscribe to a playlist or to an artist or to another user, in the wireless domain such a fragmentation in subscriptions would lead to an underutilization of the opportunistic network. For instance, a node is less probable to encounter another peer which is subscribed to the same playlist, however a user has higher chances of encountering another node which is subscribed to the same track. The reason is that tracks may exist in multiple playlists, and they are always defined by a single unique identifier.

IV. EVALUATION SCENARIO

A. Mobility scenario

In order to realistically recreate pedestrian mobility, we use the Walkers traces [9] captured in Legion Studio [10], a commercial simulator initially developed for designing and dimensioning large-scale spaces via simulation of pedestrian behaviors. Its multi-agent pedestrian model is based on advanced analytical and empirical models which have been calibrated by measurement studies. Each simulation run conducted in Legion Studio results in a mobility trace file, containing a snapshot of the positions of all nodes in the system every 0.6 s.

Fig. 6 presents an urban outdoor scenario considered in our evaluation; the outdoor scenario recreates an actual part of downtown Stockholm. The scenario consists of a grid of interconnected streets with lengths varying between 20 m and 200 m. Each street has a width of 2 m which is representative of a sidewalk. In terms of cellular coverage, we assume that the area of the outdoor scenario corresponds to the area of a single cell of a mobile operator. Nodes enter into the urban area according to an arrival rate λ via one of the fourteen passages that connect the area to the outside world, and roam around the area until they reach an exit passage. The mean sojourn time of nodes is approximately 295 s. The active area of the scenario is 5872 m^2; observe that the active area defines the area through which nodes can actually move, i.e., the streets. Throughout their lifetime nodes are constantly moving in the observed area, therefore the scenario can be characterized as a high mobility scenario. More details on the scenario can be found in [11].

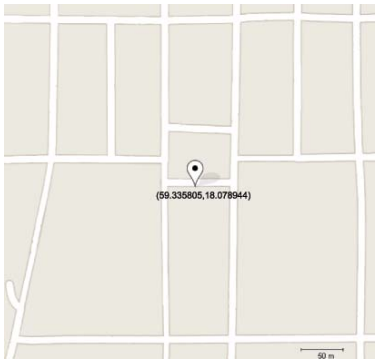


Fig. 6. Urban scenario: a grid of streets representing a part of an actual downtown area in Stockholm.

B. Simulation Setup

In our evaluation scenario we assume that all nodes carry mobile devices on which the Spotify application is installed. Communication between nodes may occur when they are in direct communication range; we set the range to be 10 m. Since most technologies dedicated to device-to-device communication, such as Wi-Fi Aware [12] and LTE-Direct [13], are not yet mature, we here do not focus on evaluating the performance of opportunistic mobile data offloading with respect to a specific underlying technology. Instead we only assume that the data rate between nodes is regulated to 10 Mbps.

Nodes have some contents preloaded into the caches of their devices, and while this content is played out, they attempt to obtain one or more tracks from their request playlist. Observe that from the viewpoint of the Spotify service, nodes may be subscribed to different playlists. However, as long as these playlists contain the same track, opportunistic device-to-device data exchange is possible. In this work we only evaluate the performance of the system with respect to these overlapping popular tracks as part of the request playlist. We believe that such an evaluation is realistic based on the track popularity distribution exhibited in the previous section. We assume that a total of $\mathcal{N}$ tracks, belonging to one or more Spotify playlists, are available in the area, and nodes may be subscribed to a set of them, or to all of them. Based on the data collected from Spotify in Section III we assume that the mean duration of a track is 225 s, with a standard deviation of 30 s. The popularity of tracks follows a Zipf distribution with parameter $\alpha = 0.368$; observe that the value of α is chosen in correspondence with the results obtained in Section III. For each simulation run we first remove the transient phase, and we then collect statistics of 1000 nodes in steady state. For all results the mean values are plotted with a 95% confidence interval.

C. Performance metrics and configurations

We focus on the following three metrics in our evaluation:

- **User satisfaction**: The user satisfaction is a measure of the percentage of users who are able to obtain *at least* one track from their request playlist via an opportunistic contact with a neighboring node before the maximum

download delay associated with the currently consumed content item is reached.

- **Offload ratio**: The offload ratio is a measure of the amount of data obtained via opportunistic contacts with respect to the total amount of data in the request playlist of a node. We only account for downloads from peers that lead to transfer of an entire track.
- **Inter-download time**: The inter-download time defines the interval between two consecutive encounters with nodes that are able to provide contents of interest to a given node. The ability of a node to provide contents of interest is in turn defined by the local subscriptions and the contact duration. Contacts with duration shorter than the minimum transfer time for a track are discarded.

Although energy consumption is an important metric in the context of device-to-device communication, we do not address it in this work; instead we refer the reader to [14] where we present power-saving algorithms for opportunistic mobile data offloading.

V. Results

A. Effect of Request Window Size

Let us assume that all nodes are interested in obtaining a playlist, constituting a total of $\mathcal{N} = 50$ tracks. Nodes that enter the area have one track pre-loaded in their cache, and are initially interested in obtaining k tracks from the request playlist. A request window of size $k = 1$ denotes that a node is interested in obtaining the next item on its playlist. We refer to this type of playlist as a *preset* playlist. A preset playlist may for instance contain a collection of podcast episodes which need to be reproduced in the correct order at the end device. A node that has a request window of size $k > 1$ is interested to obtain any of the next k tracks on its playlist; there is no need to preserve the order of tracks on the playlist, hence the playlist is *random*. An example of a random playlist is a playlist of independent music tracks which can be played in any order. Note that in this study we do not consider a scenario in which nodes can interact with the playlist, i.e., we assume that nodes do not fast-forward or skip a track during their lifetime in the system.

Fig. 7 shows the effect of the request window size on the offload ratio for the urban scenario with two different arrival rates. The offload ratio increases both with the size of the request window and with the density of nodes in the area. In fact, when a critical mass of nodes is present in the area, even at relatively small values of the request window $k = 5$, the offload ratio is approximately 90%. However, the offload ratio should not be considered in isolation; instead it should be coupled with the user satisfaction. A user is *satisfied* when its buffer does not underflow, i.e. when it is able to obtain at least one track out of its request playlist before the maximum download delay is reached. Observe that even if the offload ratio is low, the user satisfaction may be much higher (Table II) even at low densities. The reason is that the offload ratio provides a measure with respect to the overall size of

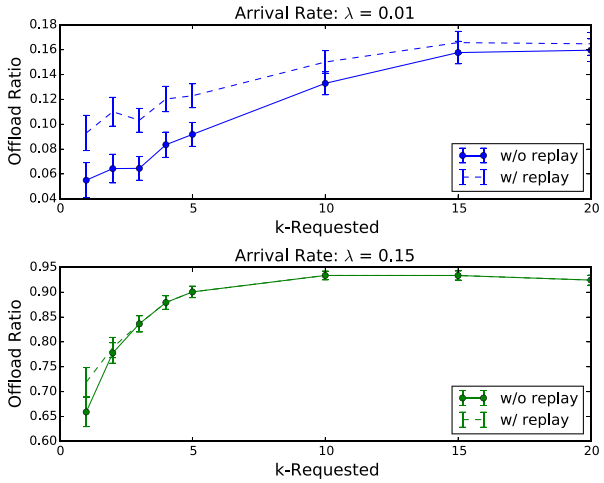


Fig. 7. Effect of the request window size for $\lambda = \{0.01, 0.15\}$ nodes/s on the offload ratio with and without replay. Observe the difference in the maximum value on the the y-axis.

TABLE II
User satisfaction with respect to the request window size.

Arrival rate	Request Window Size					
	1	2	3	4	5	10
λ = 0.01 n/s	6%	11.6%	16.9%	25%	31.9%	60.5%
λ = 0.15 n/s	66.9%	91.1%	97.3%	99%	100%	100%

the request window (thus the whole playlist), while the user satisfaction provides a measure for the momentary state of the node (what is to be played next).

Nodes that do not download a track before the maximum download delay is reached have two options: either to stream the track directly via the cellular network, or to replay one of the previous tracks that are already available in the device cache. The first strategy gives us the lower bound of offload ratio - no node is willing to replay tracks; the second strategy - the upper bound of the offload ratio - all nodes prefer to replay tracks rather than to download contents via the cellular network. Fig. 7 shows that at low densities, replaying tracks allows for a marginal improvement of the offload ratio at low values of the request window size, however as the request window size increases the offload ratio achieved with and without application of a replay strategy becomes comparable. At higher densities the replay strategy is almost never used since nodes have enough contact opportunities to download fresh contents.

B. Effect of Node Density

Fig. 8 illustrates the effect of the node density on the system performance in terms of offload ratio. We vary the arrival rate of nodes in the area from $\lambda = 0.01$ n/s to $\lambda = 0.15$ n/s for each entry point into the observed area, and we evaluate the offload ratio for two values of the requested playlist size, $k = 1$ (preset playlist) and $k = 10$ (random playlist). The results in Fig. 8 confirm that the system performance improves

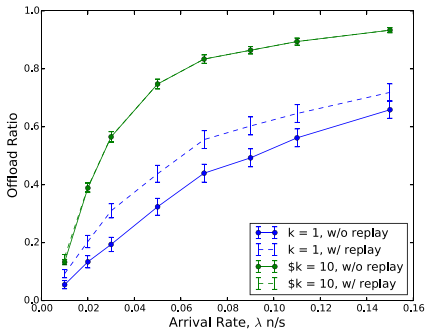


Fig. 8. Effect of node density on the offload ratio for two different values of the request window size, $k = 1$ (preset playlist) and $k = 10$ (random playlist).

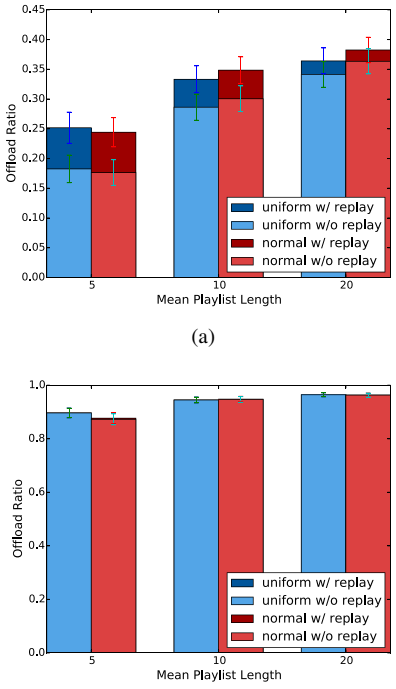
with the increase of node density. It is interesting that even at low node densities, track dissemination via device-to-device communication achieves relatively high offload ratio (more than 50% for $\lambda = 0.03$ n/s) when the size of the request playlist is large ($k = 10$) and the tracks from the request playlist can be downloaded in random order. Furthermore, a larger request window size results in faster increase of the offload ratio, especially for small values of the arrival rate. On the contrary, when the playlist is preset with $k = 1$, the change in the offload ratio between two consecutive values of the arrival rate is almost linear.

The results in Fig. 8 also illustrate that replay strategies are useful for preset playlists regardless of the node density. On the other hand, when the playlist is set to random with a large request window size, replaying tracks does not improve performance significantly even in sparsely populated scenarios ($\lambda = 0.01$ n/s).

C. Effect of Playlist Length

Users have different music preferences; thus nodes often would not be interested in obtaining all $\mathcal{N}$ tracks but only a subset $\mathcal{S} \subset \mathcal{N}$ of these tracks. In this section we evaluate whether the distribution of the mean playlist length of the subset $\mathcal{S}$ affects the system performance. Upon entering in the area, each node subscribes to a subset $\mathcal{S}$ of tracks distributed either uniformly or normally over all tracks in the original set $\mathcal{N}$. A subset of the playlist $\mathcal{S}$ is already available on each device upon entering the area. According to [6], approximately 50% of tracks that are requested by the Spotify application can be found locally in the cache of the requesting device. We thus assume that initially the cache of each device is populated with j tracks chosen uniformly at random from the $\mathcal{S}$ tracks in the playlist, with the mean value of j being $\mathbb{E}[j] = |\mathcal{S}|/2$ where $|\mathcal{S}|$ is the cardinality of the playlist set. The size of the request window of a node is then calculated as $|\mathcal{S}| \quad j$.

Fig. 9 shows how the distribution of the playlist length $|\mathcal{S}|$ affects the offload ratio for scenarios with different density. The offload ratio exhibits sensitivity towards the mean value of the playlist length, however it is not sensitive towards the distribution of the playlist length. Again, adopting replay



(a)

(b)

Fig. 9. Effect of playlist length distribution on the offload ratio with and without replay for (a) λ = 0.01 nodes/s, and (b) λ = 0.15 nodes/s. Observe the difference in the maximum value on the the y-axis.

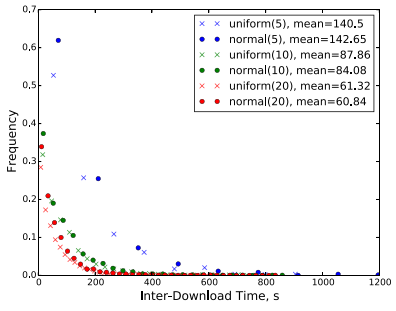
TABLE III
PERCENTAGE OF NODES THAT DO NOT OBTAIN A SINGLE TRACK DURING THEIR LIFETIME IN THE SYSTEM UNDER DIFFERENT DISTRIBUTIONS OF THE PLAYLIST LENGTH.

Playlist Length	Arrival Rate			
	λ = 0.01 n/s		λ = 0.15 n/s	
	Uniform	Normal	Uniform	Normal
$\mathbb{E}[j]$ = 5	76.5%	75.2%	32.9%	34.2%
$\mathbb{E}[j]$ = 10	50.2%	45.2%	18.5%	15.2%
$\mathbb{E}[j]$ = 20	31.7%	25.3%	10.4%	8.8%

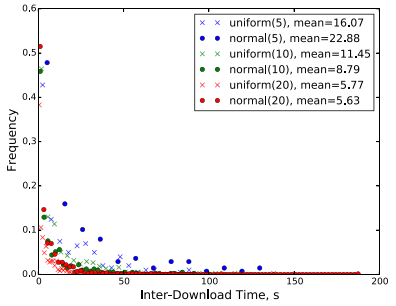
strategy is only beneficial for scenarios with low node densities, especially for short playlists. As the mean length of the playlist increases, and with the increase of participating nodes in the system, replaying tracks does not enhance the system performance.

We further take a look into the distribution of inter-download times, Fig. 10. Regardless of the node density, the shorter the length of the playlist $|\mathcal{S}|$, the longer the inter-download times. Furthermore, the distribution of playlist length does not significantly affect the distribution of inter-download times.

Finally, we evaluate the effect of playlist length distribution on content delivery. Table III shows the percentage of nodes that are not able to obtain a single track throughout their lifetime in the system. Similar to the offload ratio, the percentage of nodes that are not able to obtain contents via opportunistic contacts is strongly dependent on the mean length of the playlist as well as the node density in the area, however it is not affected by the actual distribution of the playlist length.



(a)



(b)

Fig. 10. Effect of playlist length distribution on the inter-download times for (a) λ = 0.01 nodes/s, and (b) λ = 0.15 nodes/s. Observe the difference in the maximum value on the the y-axis.

TABLE IV
AVERAGE INTER-DOWNLOAD TIMES FOR DIFFERENT DISTRIBUTION OF THE INITIAL CONTENT AVAILABILITY WITH MEAN $m = 25$ TRACKS.

Distribution	Arrival Rate	
	λ = 0.01 n/s	λ = 0.15 n/s
Deterministic	42.2 s	3.8 s
Uniform	45.2 s	3.8 s
Normal	45.2 s	3.8 s

D. Effect of Initial Content Availability

Lastly we examine the effect of initial content availability. We assume that all nodes are interested in obtaining all $\mathcal{N}$ tracks from a playlist, and that on average they have already stored half of the playlist in their local caches. We vary the distribution of cached tracks, and examine three different distributions: deterministic (exactly 50% of the items from the playlist are cached), uniform and normal. Table IV shows the average inter-download times for each of the distributions; we see that the inter-download times are not sensitive to the distribution of the initial contents available in the cache.

VI. Related work

Recent solutions for alleviating traffic load on cellular networks can be divided in two main categories: offloading to femtocells or existing Wi-Fi networks [15], [16], [4], and offloading through opportunistic device-to-device communication. Although the large body of work produced in the area of offloading cellular traffic pertain to femtocells and Wi-Fi networks, such approach is limited to possible deployments and the availability of Internet access. Using opportunistic communication for mobile data offloading does neither depend on available deployment, nor on Internet access, and has thus become a popular candidate for traffic offloading in recent years. A number of studies attempt to optimize the traffic volumes delivered to end users through opportunistic communication. In [17], Han et al. study a target-set selection problem for choosing initial data carriers in order to minimize the amounts of mobile data traffic. Lu et al. propose an opportunistic forwarding protocol for increasing the probability of data delivery [18]. Since mobile data offloading makes use of battery resources on battery-powered mobile devices, a body of work in the mobile offloading domain has been concentrated towards reducing energy consumption [14], [19]. However, all of the studies assume that opportunistic communication is used for offloading delay-tolerant contents. Instead, we evaluate whether opportunistic communication could be used for offloading data with tight delay constraints such as streaming multimedia organized in playlists.

Few works evaluate the performance of on-demand video streaming services in the presence of opportunistic contacts. In [2] Yoon et al. present a mobile peer-to-peer video-on-demand application which uses peers for delivering parts of a video stream to the video player. The application however relies heavily on the presence of a centralized scheduler which coordinates the data exchange among the peers, and it utilizes a complementary download link when contents cannot be fetched in real-time from nodes in the vicinity. In [3], Siris et al. present testbed experiments of multi-source mobile video streaming which exploits mobility and throughput prediction for prefetching video data in caches located at hotspots that the mobile will encounter and device-to-device communication to opportunistically obtain parts of a video from neighboring mobile devices. A delay-tolerant networking approach relying on multiple links and routing is applied to live-streaming by Morgenroth et al. in [20]. In contrast to these works, we do not aim to offload the currently served content item but instead we make use of opportunistic contacts to deliver contents that would be needed by the node in the near future. Furthermore, we do not rely on infrastructure for supporting the content delivery, neither do we incorporate routing for finding appropriate peers to deliver the stream. In [21], Keller et al. propose MicroCast, a system for cooperative video streaming among mobile devices in close proximity. MicroCast is however designed to operate for small number of participants in a static setting, and is thus not applicable for mobile scenarios with users on-the-go. Our approach is closest to the work of Ding et al. [4] in which they examine prefetching of non-live streaming contents. However, they rely on Wi-Fi infrastructure, on profiling user mobility patterns and on location reporting for determining the access points on which to offload future multimedia streams. Instead, our solution is purely based on opportunistic communication. Recently Jimenez et al. evaluated the effect on energy consumption when mobile devices are introduced as participants in Spotify's peer-to-peer overlay in the wired domain [22]. As opposed to them, we here evaluate a solution which creates

a separate overlay in the wireless domain and alleviates the communication with Spotify's servers which allows offloading mobile data from the operators' networks. Lastly, in [23] Danihelka et al. propose a hybrid architecture for video sharing based on cloud logic and device-to-device communication. Similar to us, the authors aim to offload the next to-be-played-out piece of video contents within a predefined deadline. The proposed architecture however requires a signalling feedback to assist the device-to-device content spreading. Instead, in our work we do not require nodes to provide feedback to the network, and the content dissemination is purely opportunistic.

VII. Conclusion

It is expected that by 2020 more than 75% of the traffic volumes at mobile operators will be generated by real-time multimedia services such as audio and video streaming. Mobile operators are thus seeking solutions for offloading mobile data with an alternative being the use of opportunistic device-to-device communication. However, opportunistic communication has been perceived mainly as a means for disseminating delay-tolerant information. In this work we studied whether the mechanisms of opportunistic device-to-device communication could be applied to a particular class of real-time services, namely music streaming; music streaming is characterized by long-lived flows resulting into high traffic volumes. We first introduced a framework for opportunistic content sharing in the context of multimedia streaming services. We then evaluated the feasibility of opportunistic device-to-device communication for streaming data using the popular music streaming service Spotify as a use-case, and performed extensive trace-driven simulations of realistic pedestrian mobility in urban environments. The main findings of our work can be summarized as follows:

- Opportunistic device-to-device communication is viable for offloading streaming data. The performance is enhanced when the inter-download times are shorter than the mean track duration.
- The system performance depends on the node density and the request window size. In sparse areas, the system performance could be improved by adopting replay strategies to avoid buffer under-flows. In dense scenarios opportunistic device-to-device communication achieves up to 90% offload ratio even at low values of the request window size.
- The system performance exhibits low sensitivity towards the distribution of parameters such as playlist length and initial content availability. However, it is sensitive towards the mean values of these parameters. This indicates that it is sufficient to take into account mean values when designing systems that use opportunistic device-to-device communication for streaming services.

As part of our future work, we plan to implement the proposed model on actual devices and to perform field tests to verify the feasibility of our approach.

References

[1] "Cisco visual networking index: Global mobile data traffic forecast update 2014 - 2019 white paper," 2015.

[2] H. Yoon, J. Kim, F. Tan, and R. Hsieh, "On-demand video streaming in mobile opportunistic networks," in *Proc. IEEE PerCom 2008*, March 2008, pp. 80–89.

[3] V. A. Siris and D. Dimopoulos, "Multi-source mobile video streaming with proactive caching and d2d communication," in *Proc. IEEE WoWMoM*, June 2015, pp. 1–6.

[4] A. Ding, B. Han, Y. Xiao, P. Hui, A. Srinivasan, M. Kojo, and S. Tarkoma, "Enabling energy-aware collaborative mobile data offloading for smartphones," in *Proc. IEEE SECON*, June 2013, pp. 487–495.

[5] Ó. Helgason, E. A. Yavuz, S. Kouyoumdjieva, L. Pajevic, and G. Karlsson, "A mobile peer-to-peer system for opportunistic content-centric networking," in *Proc. ACM SIGCOMM MobiHeld workshop*, 2010.

[6] 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*, Aug 2010, pp. 1–10.

[7] "Spotify makes the shift to mobile with 52 percent of listening now on phones and tablets," http://techcrunch.com/2015/01/10/music-is-a-mobile-linchpin/, accessed: 2015-07-30.

[8] V. Setty, G. Kreitz, R. Vitenberg, M. van Steen, G. Urdaneta, and S. Gimåker, "The hidden pub/sub of spotify: (industry article)," in *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*, ser. DEBS '13. New York, NY, USA: ACM, 2013, pp. 231–240.

[9] S. T. Kouyoumdjieva, Ó. R. Helgason, and G. Karlsson, "CRAWDAD data set kth/walkers (v. 2014-05-05)," Downloaded from http://crawdad.org/kth/walkers/, May 2014.

[10] "Legion Studio," http://www.legion.com/.

[11] Ó. Helgason, S. T. Kouyoumdjieva, and G. Karlsson, "Opportunistic communication and human mobility," *Mobile Computing, IEEE Transactions on*, vol. 13, no. 7, pp. 1597–1610, July 2014.

[12] Wi-Fi Alliance, "Wi-fi aware: Better proximity technology for personalized experiences," July 2015.

[13] Qualcomm Technologies Inc., "Lte-direct trial," Tech. Rep., Feb. 2015.

[14] S. T. Kouyoumdjieva and G. Karlsson, "Energy-aware opportunistic mobile data offloading for users in urban environments," in *IFIP Networking*, May 2015, pp. 1–9.

[15] K. Lee, J. Lee, Y. Yi, I. Rhee, and S. Chong, "Mobile data offloading: How much can wifi deliver?" *Networking, IEEE/ACM Transactions on*, vol. 21, no. 2, pp. 536–550, April 2013.

[16] L. Gao, G. Iosifidis, J. Huang, and L. Tassiulas, "Economics of mobile data offloading," in *Proc. IEEE INFOCOM*, April 2013, pp. 3303–3308.

[17] B. Han, P. Hui, V. Kumar, M. Marathe, J. Shao, and A. Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 5, pp. 821–834, May 2012.

[18] L. Xiaofeng, H. Pan, and P. Lio, "Offloading mobile data from cellular networks through peer-to-peer wifi communication: A subscribe-and-send architecture," *Communications, China*, vol. 10, no. 6, pp. 35–46, June 2013.

[19] V. F. Mota, D. F. Macedo, Y. Ghamri-Doudanez, and J. M. S. Nogueira, "Managing the decision-making process for opportunistic mobile data offloading," in *Proc. IEEE NOMS*, May 2014, pp. 1–8.

[20] J. Morgenroth, T. Pögel, and L. Wolf, "Live-streaming in delay tolerant networks," in *Proc. ACM CHANTS*. New York, NY, USA: ACM, 2011, pp. 67–68.

[21] L. Keller, A. Le, B. Cici, H. Seferoglu, C. Fragouli, and A. Markopoulou, "Microcast: Cooperative video streaming on smartphones," in *Proc. ACM MobiSys*. New York, NY, USA: ACM, 2012, pp. 57–70.

[22] R. Jimenez, G. Kreitz, B. Knutsson, M. Isaksson, and S. Haridi, "Integrating smartphones in spotify's peer-assisted music streaming service," KTH, Royal Institute of Technology, Stockholm, Sweden, Tech. Rep. Diva-134609, 2013.

[23] J. Danihelka, D. Giustiniano, and B. Plattner, "On a cloud-controlled architecture for device-to-device content distribution," in *Proc. ACM CHANTS*, New York, NY, USA, 2015, pp. 19–24.