IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 2, 2020, accepted September 24, 2020, date of publication October 7, 2020, date of current version October 21, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3029319

MobRec – Mobile Platform for Decentralized Recommender Systems

FELIX BEIERLE, (Member, IEEE), AND SIMONE EGGER
Service-Centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin, 10587 Berlin, Germany
Corresponding author: Felix Beierle (beierle@tu-berlin.de)

We acknowledge support by the German Research Foundation and the Open Access Publication Fund of TU Berlin.

ABSTRACT Recommender systems recommend new movies, music, restaurants, etc. Typically, service providers organize such systems in a centralized way, holding all the data. Biases in the recommender systems are not transparent to the user and lock-in effects might make it inconvenient for the user to switch providers. In this paper, we present the concept, design, and implementation of MobRec, a mobile platform that decentralizes the data collection, data storage, and recommendation process. MobRec's architecture does not need any backend and solely consists of the users' smartphones, which already contain the users' preferences and ratings. Being in proximity in public places or public transportation, data is exchanged in a device-to-device manner, building local databases that can recommend new items. One of biggest challenges of such a system is the implementation of unobtrusive device-to-device data exchange on off-the-shelf Android devices and iPhones. MobRec facilitates such data exchange, building on Google Nearby Messages with Bluetooth Low Energy. We achieve the successful exchange of data within 3 to 4 minutes, making it suitable for the described scenario. We demonstrate the feasibility of decentralized recommender systems and provide blueprints for the development of seamless multi-platform device-to-device communication.

INDEX TERMS Device-to-device communication, mobile ad hoc networks, mobile applications, pervasive computing, recommender systems, social networking services, ubiquitous computing.

I. INTRODUCTION

Recommender systems are ubiquitously available. They recommend items from different domains, for example, media to consume (e.g., Spotify, Netflix) or points of interests (POIs) to visit (e.g., Yelp, Google Maps). However, existing recommender systems have several drawbacks. Existing providers typically operate in a centralized manner: the service provider holds all the data and recommends items based on algorithms that are not visible to the user. This can introduce certain limitations and biases. Limitations often are that only items will be recommended that are available with the service provider, e.g., Netflix will only recommend items available in their catalog. Possible biases could be that the recommender algorithms favor items that create more profit for the service provider. Typically, the mentioned service providers are interested in retaining their user base and create lock-in effects. For example, movies bought on iTunes cannot be transferred to another service provider, effectively locking the user and his/her collection in.

The infrastructure that could be a solution for the limitations of centralized recommender systems is already in the palms of its users. The smartphone can store lots of information about its user and his/her interests, e.g., regarding preferred restaurants, music, or movies. Equipped with capabilities for device-to-device communication, users can exchange data with each other. When considering recommender systems based on content-based filtering or collaborative filtering, data about similar items and similar users are needed. Data about the properties of items can be retrieved through public APIs (e.g., Google Places, Spotify, The Movie DB (TMDB)). Finding similar users might be simple with smartphones: spending time at the same location might imply similarity – at least to a certain degree. Additionally, from our previous research, other methods of determining similarity between users based on smartphone data are available [1], [2]. Thus, exchanging data between smartphones in proximity in a device-to-device fashion allows to create local databases

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Maaz Rehan.

that allow to filter for similar users. This data can be used for on-device recommender systems that are not limited to a single external service provider.

Combining and expanding approaches from device-to-device computing (e.g., [3], [4]) and decentralized recommender systems (e.g., [5]–[7]), in this paper, we propose a modular architecture for recommender systems for virtually any domain, building on the existing infrastructure of smartphones. Our architecture consists of collaborative data collection paired with data exchange via device-to-device communication and local recommender systems running on each device, supported by third-party service providers where appropriate. There are some challenges to overcome when developing such a platform. Some data is already readily available on smartphones, for example the most frequently visited locations. Other user preferences/ratings that cannot be assessed automatically might have to be entered manually or retrieved from external service providers, e.g., music listened to, or favorite movies. Some short-distance wireless technologies, like NFC, Bluetooth, or WiFi Direct, are available on most modern smartphones, and software libraries exist for device-to-device communication. However, from an application layer perspective, utilizing such libraries for seamless data exchange between smartphones remains a challenge for multiplatform apps.

In this paper, we propose an mobile platform for decentralized recommender systems. We refer to the platform as well as our prototypical implementation as MobRec.

The main contributions of this paper are:

- An extensive overview of existing device-to-device connection approaches, including the benefits and drawbacks associated with them.
- Developing an approach for collecting data and exchanging data in a device-to-device fashion for multiplatform apps (Android and iOS).
- Proposing a general modular architecture for a service-provider-independent, backend-less mobile platform for recommender systems.
- A prototypical implementation of a minimum viable product (MVP) showcasing the feasibility of our complete MobRec architecture.[1]
- An extensive evaluation of the device-to-device connection module.

We sketched the general idea of MobRec in our work-in-progress paper [8]. This paper builds on the results of [8] and extends it with the details regarding device-to-device communication, and the implementation and evaluation of MobRec.

The remainder of this paper is structured as follows. In Section II, we describe the requirements of MobRec. In Section III, we give an overview of related research areas. Section IV gives a detailed overview about the state-of-the-art in device-to-device communications research and technology. A prototypical implementation of a minimum viable product showcases the feasibility of our complete architecture. Details about the design and implementation of MobRec are given in Section V and Section VI. In Section VII, we evaluate MobRec, focusing on device-to-device capabilities, before concluding in Section VIII.

[1]The code is publicly available at https://github.com/TU-Berlin-SNET/MobRec.

II. REQUIREMENTS

The general concept is that smartphone users exchange data relevant for recommender systems when they pass each other. Such a system especially works in urban areas. Today, already more than half of the world's population lives in urban areas and the trend is that this percentage is increasing.[2] There are several things to consider when designing MobRec. In the following, we describe the technical (functional) requirements of our systems.

In order for MobRec to work, it should be deployable on off-the-shelf smartphones. Android (74.3%) and iOS (24.76%) have a combined market share of almost 100% of the global mobile operating system market.[3] The first requirement (R1) is to build a multiplatform app for both Android and iOS.

At the core of MobRec is the idea of an app running in the background, exchanging data with other users. We derive the next two requirements based on this. First, the exchange of data when two smartphones are in proximity must not require explicit user interaction in order to establish a connection (R2). If that were the case, background data exchange would no longer be possible and would require too much user effort. The next aspect is related but can be defined distinctly: the transfer of data has to be done in the background (R3). That is, the system has to be capable of transferring data from and to nearby devices while our app is running in the background. R2 and R3 can be described as the *broadcasting* of data, i.e., having a $1 : n$ relationship between sender and receiver without explicit connection establishment. R2 describes the broadcasting requirement from a user-interaction perspective, whereas R3 describes it from a technical perspective.

Considering that MobRec is a mobile platform for recommender systems, the domain of items that should be possible to be recommended is not restricted. As Section IV will show, most device-to-device approaches that support broadcasting of messages only allow for very small payloads. Supporting a variety of different domains, in order for it to be feasible to recommend items from each supported domain, each device potentially needs to broadcast larger amounts of data (R4). As we will design a workaround for R4 in Section V-B, we do not quantify the exact size requirements here.

To summarize, the four requirements for our system are:

multi-platform
R1 (Android and iOS)

[2]https://www.un.org/development/desa/en/news/population/2018-revision-of-world-urbanization-prospects.html, accessed 2020-07-20

[3]https://gs.statcounter.com/os-market-share/mobile/worldwide, accessed 2020-02-28

R2 no explicit connection establishment (from a user perspective)
R3 transferring data while running in the background
R4 transferring larger amounts of data

Especially the aspect of utilizing a background service that uses a wireless interface to broadcast and scan for messages, makes battery drain a concern that comes to mind. We focus on R1–R4 in this work. Once we designed and implemented an app with technologies that support these requirements, we propose optimizations for battery consumption in Section VII-D.

III. RELATED WORK

In this section, we review related work from multiple related fields: decentralized recommender systems, ubiquitous social networking, and mobile sensing.

The terms *distributed* and *decentralized* are often used interchangeably. We want to make a distinction though. *Centralization* refers to the governance of a system. In a centralized system, one service provider holds control over the whole system, and stores and processes all the data. *Distributed*, on the other hand, refers to computation and data storage. For example, Facebook is a centralized Social Networking Service provider that can utilize computing and data storage in a distributed fashion. Email is an example for a decentralized service: server providers interact via standardized interfaces and the user can choose his/her service provider or host his/her own email server.

Among the most frequent concerns with centralized services is the potential misuse of data and loss of user privacy. A single service provider holds all the data and can use it beyond the level to which users intended to share it with the service provider [9]–[11]. A second concern are lock-in effects. Centralized service providers like iTunes or Spotify typically strive to retain their users and make it difficult to switch to other providers. Transferal of bought movies or created playlists is not easily possible – or not at all.

Decentralized recommender systems typically use peer-to-peer network overlays [5], [6], [12]. Gossip protocols are often used to find similar peers to connect to. Communities of peers with similar interests exchange item ratings among each other. In contrast to such peer-to-peer scenarios, the ad hoc broadcasting approach in MobRec only establishes short-term connections between devices to exchange data. Especially [5] and [7] follow similar approaches compared to our proposed system. In [5], decentrally stored data from the web is used for a recommender system running on the user's personal computer. Since that paper's publication (2005), the development of mobile devices enable mobile and ubiquitous scenarios depicted in this paper. In [7], the authors propose that smartphones exchange data in a device-to-device fashion and calculate their own recommendations via collaborative filtering. The focus of that paper is on the recommender algorithm that is evaluated with a music data set. For device-to-device communication, WiFi Direct is proposed.

Other related fields are those of ubiquitous recommender systems and context-aware recommender systems (CARS). They consider items in proximity or consider the user's current context while recommending items, respectively [13]. In contrast to these approaches, our proposed system is universal in the sense that any type of item can be recommended, independent of the item's physical proximity or the user's context.

In a broader sense, our proposed system is related to the field of *ubiquitous social networking*, sometimes referred to as *proximity-based (mobile) social networking*. Overall, a variety of scenarios is addressed in this field. The most common one is the incentivization of social interaction, i.e., the recommendation of physically proximate users. On a technical level, this is often achieved by device-to-device transfer of data, meaning two devices, typically smartphones, connect wirelessly in an ad hoc manner without going through infrastructure like routers or hotspots.

In SMILE, the authors utilize real-life encounters to exchange randomly created symmetric keys in a device-to-device manner [14]. Only users who actually met can then read encrypted messages stored on a server. E-Smalltalker aims at incentivizing real-life smalltalk by exchanging interests profiles between users in proximity [15]. In our own previous work, we proposed using all available smartphone data and automatically comparing it in order to determine similarity in terms of interest and personality [1], [16]. In [17], Yang and Hwang proposed a mobile recommender system for point-of-interest (POI) recommendations that utilizes data exchanged in a device-to-device manner. Talk2Me, presented in [18], is a prototype that combines augmented reality with social networking. Basic profile information is sent to nearby devices via device-to-device communication. Along with the profile, a so-called *face-signature* is sent, allowing the receiving device to use this signature to recognize the sending user when scanning faces with the camera in smartphones or smart glasses.

One of the core ideas of our proposed system is that a lot of the data indicating the user's taste is readily available on the smartphone. The concept behind this is called *mobile sensing*, measuring data with the smartphone's sensors and storing the measurement. In previous work, we presented an Android app for smarphone data tracking [19] and analyzed the different data categories that are available [20]. Additionally, there are frameworks avaialble for mobile sensing for both Android and iOS, for example, Sensus [21], LiveLabs [22], or AWARE [23].

IV. STATE-OF-THE-ART IN DEVICE-TO-DEVICE COMMUNICATION

In this section, we give an extensive overview of related work in the field of device-to-device communications, structured by technology used. This includes Bluetooth, WiFi, and different frameworks. The technical details given in this section might help researchers and developers in choosing appropriate device-to-device technologies for other use cases with

different requirements as well. In the summary, we give details which technologies fit our requirements given in Section II.

Note that we focus on the technologies from an application layer perspective. There is a lot more related work regarding the lower layers of the OSI model. Additionally, there is lot more related work focusing on IoT (Internet of Things) scenarios with sensors instead of smartphones.

A. BLUETOOTH

In several related works published until 2011, data is exchanged using Classic Bluetooth and a direct device-to-device connection [24]–[27]. In SpiderWeb, each mobile device functions as server or client [26]. While being in the server role, the device publishes a service that clients can search for and connect to. On current devices and mobile OSs, Classic Bluetooth is still available. However, Apple restricts connections to other iOS devices, which makes device-to-device communication via Classic Bluetooth impossible for multi-platform applications (R1). The second requirement that Classic Bluetooth cannot fulfill, is R2. For Classic Bluetooth data transfer, an explicit connection has to be established, which typically requires user interaction.

There are several papers that propose solutions that avoid having to establish explicit connections between devices [28]–[32]. E-Shadow uses WiFi and Bluetooth without establishing explicit connections [28]. This is done by utilizing the WiFi SSID, the Bluetooth device name, and the Bluetooth Service Discovery Protocol (SDP) to broadcast data. The three different technologies are used to cover different physical ranges. The numbers given in [28] are as follows. For WiFi SSID, they indicate a range of 50 m and a size of 32 bytes. For Bluetooth device names, they state a range of 20 m and a size of 2,000 bytes. For Bluetooth SDP, they state a range of 10 m and a size of 1,000 bytes. Other related papers also suggest exploiting the Bluetooth device name [30] or Bluetooth SDP [14], [29], [33]. While the proposed Bluetooth-related solutions – device name and SDP – circumvent having to establish a connection, they have the drawback of being related to Classic Bluetooth. Thus, they do not benefit from the energy efficiency introduced with Bluetooth Low Energy (BLE). Furthermore, Apple's restrictions regarding the connection of Bluetooth devices in iOS within the MFi (Made for iPhone/iPod/iPad) program[4] could make it impossible to utilize these approaches.

In 2010, BLE was introduced in Version 4.0 of the Bluetooth Core Specification.[5] Many of the features of Classic Bluetooth are inherited while a low latency and very low energy consumption is achieved [34]. Most current smartphones support this standard.

Two different modes are available in BLE: undirected advertising/scanning (AD/SC) mode and central/peripheral connection (C/P) mode [34]. In AD/SC mode, small payloads can be broadcast without the need of an established connection. C/P mode allows larger payloads but there has to be an explicit connection between devices. To be more specific: devices in peripheral mode can advertise their presence while central nodes can connect to those nodes [34]. BLE is often used in IoT scenarios with sensors. Having multiple sensors in peripheral mode allows a central node to periodically connect to the sensors. Peripheral nodes cannot communicate with other peripheral nodes.

The authors of [34] present a framework called BlueNet for IoT scenarios that allows the switching of roles in BLE. Sikora *et al.* use AD/SC mode for data exchange between two smartphones [35]. They report about a range of about 40 m and a size of 37 bytes. The authors report that, at least for Android, the device switches automatically between advertising and scanning mode, and the software developer can influence the delay between switches but not define it in an exact manner.

B. WiFi

WiFi, IEEE 802.11 standard, has two modes: *infrastructure* and *ad hoc* mode. Infrastructure mode is the common mode of devices connecting to access points, whereas ad hoc mode allows for communication between devices directly.

The infrastructure mode is specifically *not* device-to-device communication. However, by creating a WiFi access point and letting other devices join the created network, effectively, device-to-device communication can be implemented in this mode. With ShAir, a middleware with this approach is presented in [36]. Furthermore, popular file sharing applications use this approach, for example the app Xender.[6] According to the FAQs on the app's website, file sharing is done by being in the same WiFi network or by creating access points on the smartphone. SHAREit[7] works the same way. The disadvantage of this approach is that running in the background is not possible without disrupting the user's experience: while a software based on this approach is active, the user probably is not able to be connected to his/her usual WiFi network. This violates R3 (transferring data in the background).

Some papers suggest using the WiFi SSID for broadcasting small amounts of data [28], [32]. However, iOS does not allow changing the WiFi SSID programmatically. Furthermore, if the SSID is bound to an opened access point on the device, this typically means that the device cannot be connected to another WiFi network, which again violates R3.

The main issue with WiFi ad hoc stems from the fact that it never was widely deployed in the market [37]. In [38], the authors use the WiFi ad hoc mode on an Android device. This mode is not available by default, an extension had to be compiled into the Linux kernel. There is still a lack of support of WiFi ad hoc since the publication of that paper (2016), which shows that only a very limited set of Android devices

[4]https://developer.apple.com/programs/mfi/, accessed 2020-07-20
[5]https://www.bluetooth.com/specifications/archived-specifications/, accessed 2020-07-20
[6]https://www.xender.com/, accessed 2020-07-20
[7]https://www.ushareit.com/, accessed 2020-07-20

would be able to run such an application. Furthermore, WiFi ad hoc mode is not supported on iOS [32].

The WiFi Alliance developed another WiFi mode for device-to-device communication, WiFi Direct, which is supported on Android devices with version 4.0 and higher [39].[8] There are several related papers dealing with WiFi Direct for device-to-device communication [18], [37], [40]–[43]. While the Talk2Me prototype was developed utilizing WiFi Direct, Shu *et al.* describe how it is not mature enough and wasn't used for the evaluation [18]. Instead they used UDP over WiFi with devices connected to the same access point. In WiFi Direct, every connection needs to be confirmed manually, workarounds might be possible though. By default, however, this violates R2 (no explicit connection establishment). Furthermore, iOS does not support WiFi direct, violating R1 (multi-platform app). Apple instead offers its own device-to-device framework that is only available for iOS devices.

WiFi Aware, sometimes called NAN (Neighbor Awareness Networking) is another approach by the WiFi Alliance for device-to-device communication. Android implements its functionality with version 8. The developer websites indicate that its functionality is dependent on the actual WiFi hardware and firmware.[9] Although some websites report that WiFi Aware is based on Apple's AWDL (Apple Wireless Direct Link) technology, Apple does not seem to support WiFi Aware. According to the website of the WiFi Alliance, there are currently less than 50 smartphones specifically certified for WiFi Aware.[10] Thus, we regard this technology as not widespread enough to be considered for our purposes.

WLAN-Opp was developed based on IEEE 802.11 and tethering between smartphones [44]. It is supposed to serve as an alternative for WiFi ad hoc and WiFi Direct given their shortcomings and limited availability. The implementation of WLAN-Opp is for Android only and not maintained.[11]

C. FRAMEWORKS

The open source community, as well as some companies have developed frameworks aiming at providing abstractions for device-to-device communication. Google and Apple as the major mobile OS providers also provide their own solutions. In this section, we will give an overview and highlight key characteristics.

AllJoyn is a software framework that allows devices to communicate with other devices in proximity.[12] There are a few project building on top of AllJoyn [45]–[47]. As the latest release is from 2017, we assume the project is not actively maintained anymore.

Thali is an open source project with the goal of enabling device-to-device computing. The code is not actively maintained[13] and only exists as a Cordova[14] plugin. The developers specifically highlight the issue of connecting Android and iOS in a device-to-device manner, stating they only found a workaround.[15] It consists of using BLE for finding other devices and then manually joining a WiFi access point opened on another device. This procedure violates R2 as manual user interaction is required.

Some companies offer frameworks for device-to-device communication. Ueoaa AG's p2pkit[16] is a multi-platform framework for seamless device-to-device computing. However, the code does not seem actively maintained.[17] OpenGarden's FireChat app[18] for offline messaging via message exchange in a device-to-device manner gained some attention during times of government censorship and unavailability of Internet connections.[19] OpenGarden's Meshkit SDK though, mentioned, for example, in [48], cannot be found online and is not part of the company's GitHub repository.[20] Bridgefy[21] follows the same goal of offline device-to-device communication. Their free plan allows for 30 monthly offline users.[22] Broadcasting, i.e., the connectionless sending of messages to devices in proximity, only works in the mesh mode of the framework and the maximum message size then is 2048 bytes.[23]

There were and are some products and apps available with device-to-device functionalities. Hand-held gaming devices from Nintendo and Sony were offering data exchange with nearby players in proximity.[24] This is a feature specific to each gaming system and does not work across devices from Nintendo and Sony.

Both Apple and Google provide frameworks that enable developers to build apps that are able to communicate with nearby devices. Apple's framework is called MultipeerConnectivity[25] and uses different technologies like WiFi and Bluetooth for communication and is only supported by iOS devices. Google's Nearby framework[26] uses technologies such as Bluetooth, WiFi, and audio and consists of

[8] WiFi Direct has initially been called WiFi Peer-to-Peer.

[9] https://developer.android.com/guide/topics/connectivity/wifi-aware, accessed 2020-07-20

[10] https://www.wi-fi.org/product-finder-results?sort_by=certified&sort_order=desc&certifications=56, accessed 2020-07-20

[11] https://github.com/saschat/WLAN-Opp, accessed 2020-07-20

[12] https://github.com/alljoyn/core-alljoyn/releases, accessed 2020-07-20

[13] https://github.com/thaliproject/Thali_CordovaPlugin, accessed 2020-07-20

[14] A framework for multi-platform mobile application development, see https://cordova.apache.org/ (accessed 2020-07-20).

[15] http://thaliproject.org/Android-and-iOS-interop/, accessed 2020-07-20

[16] http://p2pkit.io/, accessed 2020-07-20

[17] https://github.com/Uepaa-AG, accessed 2020-07-20

[18] https://www.opengarden.com/firechat/, accessed 2020-07-20

[19] https://en.wikipedia.org/wiki/FireChat, accessed 2020-07-20

[20] https://github.com/opengarden, accessed 2020-07-20

[21] https://www.bridgefy.me/, accessed 2020-07-20

[22] https://www.bridgefy.me/pricing.html, accessed 2020-07-20

[23] https://github.com/bridgefy/bridgefy-ios-developer/blob/master/README.md, accessed 2020-07-20

[24] https://www.nintendo.com/3ds/built-in-software/streetpass/how-it-works and http://us.playstation.com/psvita/apps/psvita-app-near.html, both accessed 2020-04-10

[25] https://developer.apple.com/documentation/multipeerconnectivity, accessed 2020-07-20

[26] https://developers.google.com/nearby/, accessed 2020-07-20

two different APIs: Nearby Connections and Nearby Messages. Nearby Connections allows direct communication with other devices in proximity without the need for an Internet connection. It is only available for Android. Nearby Messages requires an Internet connection and only allows the exchange of small payloads but it is available for both Android and iOS. Google describes that they utilize ''a combination of Bluetooth, Bluetooth Low Energy, Wi-Fi and near-ultrasonic audio''.[27] Using these technologies, tokens are exchanged between devices. After receiving a common token, Google's servers distribute the payload to the receiving device. Although the messages are relayed through Google's servers, the documentation emphasizes that ''Nearby Messages is unauthenticated and does not require a Google Account''.[28] The maximum payload size is 100 kibibyte, i.e., 102400 bytes.

D. SUMMARY

Table 1 gives an overview of the device-to-device communication approaches that we disregard for our proposed system.[29]

Because of Apple's mentioned restrictions regarding certifications for Bluetooth devices, the three approaches related to Classic Bluetooth are not readily available on iOS: Classic Bluetooth, Bluetooth Device Name, Bluetooth SDP. WiFi ad hoc, WiFi Direct, WiFi Opp, WiFi Aware, and Google Nearby Connections are not available on/for iOS devices. Apple Peer Connectivity is not available on Android devices.

Regarding R2, exchanging data without manual user interaction, we observed that iOS only allows the connection to new WiFi access points after manual user interaction. This leads us to disregard WiFi infrastructure mode and Thali which uses this workaround for device-to-device communication. Changing the WiFi SSID is not programmatically possible on iOS, which violates R2 as well. BLE C/P needs the explicit connection between devices, so it violates R2 as well.

We note that p2pkit violates R3, the transfer of data in the background. It does not enable the exchange of data between two iOS devices that are not actively used.[30] The OpenGarden Meshkit is not to be found and thus we exclude it. Both AllJoyn's and p2pkit's code does not seem to be maintained, the latest releases of both are three years old at the time of writing.

Implementing mutliple device-to-device approaches in one app would likely result in interferences at the wireless interfaces or excessive battery drain. Both Apple Multipeer Connectivity and Google Nearby utilize BLE, for example.

[27] https://developers.google.com/nearby/messages/overview, accessed 2020-07-20

[28] https://developers.google.com/nearby/messages/overview, accessed 2020-07-20

[29] Additionally, we disregard the solutions by Nintendo and Sony because they are proprietary solutions for their respective gaming devices and are not available for smartphones.

[30] http://p2pkit.io/developer/support/faq/, accessed 2020-07-20

TABLE 1. Excluded approaches for device-to-device communication.

Reason	Technology	Comment
R1	Classic Bluetooth	no iOS support
R1	Bluetooth Device Name	no iOS support
R1	Bluetooth Service Discovery Protocol (SDP)	no iOS support
R1	WiFi ad hoc	no iOS support
R1	WiFi Direct	no iOS support
R1	WiFi Aware	no iOS support
R1	WiFi Opp	no iOS support
R1	Google Nearby Connections	no iOS support
R1	Apple Multipeer Connectivity	no Android support
R2	WiFi infrastructure	no automatic connection on iOS devices
R2	WiFi SSID	not programmatically changeable on iOS
R2	Thali	no automatic connection on iOS devices
R2	BLE C/P	requires explicit connection establishment
R3	p2pkit	no background data exchange support for iOS to iOS; and code not maintained
other	OpenGarden Meshkit	SDK not available
other	AllJoyn	code not maintained

Using both technologies and trying to combine their capabilities this way would likely not work well because the BLE interface could most likely just be used by one of the frameworks at each time.

This leaves three options that fulfill R1, R2, and R3: BLE SC/AD, Bridgefy, and Google Nearby Messages. In the next section, we will design our application based on the results of this section.

V. DESIGN OF MobRec

In Fig. 1, we illustrate the proposed general modular architecture of MobRec. The three main components of the system are *Data Collection*, *Data Exchange*, and *Recommender System*. Data Collection is responsible for getting data about the user. Data Exchange is responsible for getting data from other users. The Recommender System utilizes all available data for recommending items to the user. The mobile OS provides components for sensors (for example for tracking the user's location for inferring his/her favorite POIs) and wireless interfaces (for exchanging data).

External service providers might be needed (or be useful) in order to retrieve metadata about items, utilize existing systems, or offload data or computational tasks. Fig. 1 shows dashed lines for optional connections to third party service providers. Data Collection might use this to retrieve data about the user or to enrich already available data, e.g., find out the genre of the songs the user listened to.

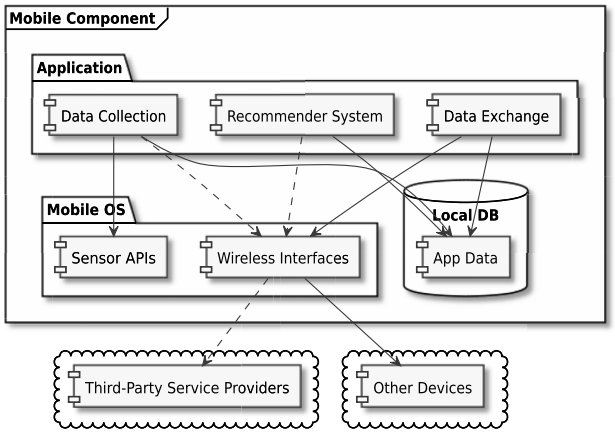

FIGURE 1. Architecture components of MobRec.

The Recommender System can optionally be relayed to an external service provider.

The system should be developed in a modular way in order to be able to exchange components easily. Consider the multitude of device-to-device approaches. Technological advances or the development of new frameworks could offer shorter connection times, and higher bandwidths, or larger transmission ranges. We then might want to exchange the Data Exchange module. Similarly, advances in recommender systems and machine learning might offer better recommendations, creating the need to replace the module or offload certain tasks to components available from external service providers.

Privacy and Security. The example domains used throughout this paper are music, movies, and POIs. Some people advertise their music taste publicly through t-shirts, posters, or stickers. Movie taste and preferences for POIs are public in the sense that people see each other at the cinema or at the POI. In contrast to religion or politics, for example, music, movies, and places to visit are rather topics for small talk conversations and are associated with much less sensitive information. Thus, overall, we expect most of the potential user base of a system like MobRec to be ok with sharing their preferences of music, movies, and POIs. Deeper discussions about privacy, and about mechanisms to give more control to the user about which data is shared, is left for future work.

Regarding security, the most critical part will be the device-to-device interface. By focusing on the application layer and building on existing solution modules for the device-to-device interfaces, we should not be introducing new security risks on top of those present in the used solutions. In this paper, we focus on the architecture and prototypical implementation of MobRec, a deeper security analysis is left for future work.

A. DATA COLLECTION

We identify three different possibilities to retrieve user data:

1) AUTOMATIC DATA TRACKING

Via mobile sensing, already, information about the user's interests and preferences is available. Most music player apps allow for tracking the played back songs (cf., e.g., [49]). Additionally, papers like [50] show further links between behavior and implicit ratings. In [50], links are shown between geolocation histories and implicit place ratings. Thus, we assume to be able to use automatically tracked data for either finding similar users or for finding implicit ratings.

The data that can be tracked automatically on Android and iOS might differ. In order to create a multiplatform system and ensure that the same data points are available on all systems, additional ways of retrieving the user's ratings are necessary. Fig. 2 shows the sequence diagram of automatic context data tracking (mobile sensing).

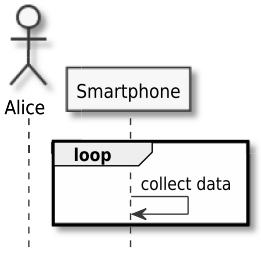

FIGURE 2. Data Collection: Automatic data tracking (mobile sensing).

2) QUERYING THIRD PARTY SERVICE PROVIDERS

In order to minimize necessary user effort, the second method we suggest is retrieving data from existing service providers. For example, Spotify's API enables application developers to fetch recently played tracks.[31] Similarly, both Apple Music[32] and Deezer[33] also allow developers to get most recently played tracks. According to Statista, these three music streaming service providers make up 57% of the worldwide music streaming market, with Spotify and Apple Music being the two biggest service providers.[34] Regularly retrieving recently played back music yields a complete music listening history indicating implicit user ratings. Fig. 3 shows the sequence diagram for the collection of data from a third-party service provider.

3) MANUAL USER INPUT

For data that is neither automatically trackable nor available via third parties, the user should be able to enter it manually. By defining an ontology for categories and terms that can be exchanged between users, compatibility between the data from different collection methods can be ensured.

[31] https://developer.spotify.com/documentation/web-api/reference/player/get-recently-played/, accessed 2020-07-20

[32] https://developer.apple.com/documentation/applemusicapi, accessed 2020-07-20

[33] https://developers.deezer.com/api, accessed 2020-07-20

[34] https://www.statista.com/statistics/653926/music-streaming-service-subscriber-share/, accessed 2020-07-20

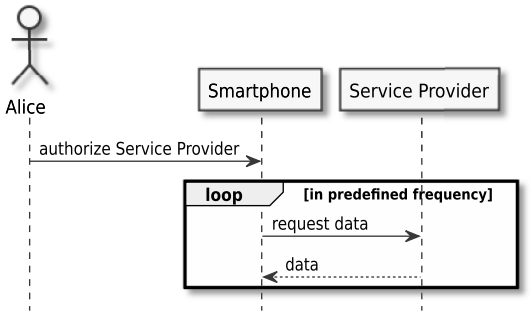

FIGURE 3. Data Collection: Querying third-party service providers.

Pre-defined categories can be movies, music, or restaurants, where recommender system are often used, but any other category would be possible as well. Service providers like The Movie DB (TMDB),[35] for example, can be used to help employ globally valid identifiers for each item, in this case, for each movie. Fig. 4 shows the sequence diagram for manual data collection. *Catalog Service Provider* denotes a service provider that offers structured information about a specific category, like the mentioned The Movie DB.

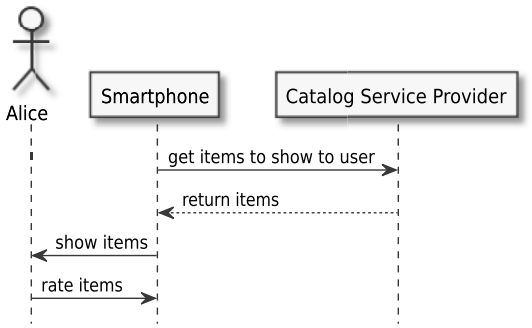

FIGURE 4. Data Collection: Manual user input.

B. DATA EXCHANGE (DEVICE-TO-DEVICE Communication)

The three remaining approaches for device-to-device communication from our overview in Section IV are BLE SC/AD, Bridgefy, and Google Nearby Messages. All of them seem to fulfill R1, R2, and R3. None of them, however, fulfills R4, transferring larger payloads. In this section, we present our workaround utilizing cloud storage providers. We then investigate the three remaining technologies to decide which one to choose for the implementation.

1) SIZE-LIMITATION WORKAROUND WITH CLOUD STORAGE PROVIDERS

Building on the existing device-to-device approaches, we present a workaround to facilitate the broadcasting of large payloads while fulfilling R1 (multiplatform app), R2 (no explicit connection establishment), and R3 (data transfer in the background). Fig. 5 visualizes our workaround. First, Alice authorizes the system to access her account at some Cloud Storage Provider (CSP) like Dropbox, Google Drive, etc. Alternatively, she could use her own cloud storage. In some predefined frequency, Alice's data is then uploaded to the CSP and shared via a public URL. This URL is then broadcast via one of the above-mentioned approaches. As only the URL is shared, which can be further shortened via a URL shortener service, the available small payloads should suffice. Another user, Bob in Fig. 5, receives the broadcast with the URL and can download Alice's publicly shared data. Optimizations like waiting for a WiFi connection can easily be implemented. Note that the only required user interaction by Alice or Bob is the authorization of the CSP, which only has to be done once.

2) BLUETOOTH LOW ENERGY (BLE)

When using BLE AD/SC mode, custom data can be sent in different fields that are part of the advertisement data. In that advertisement data, we could broadcast the URL pointing to Alice's data at her CSP. In order to broadcast, the device needs to be in peripheral mode, which both Android and iOS support, fulfilling R1. Both scanning and advertising do not require manual user interaction, fulfilling R2. Android allows for both scanning and advertising while the app is in the foreground or background, fulfilling R3. Apple also allows Bluetooth-related tasks to be done while running in the background. However, the scanning intervals are longer which might lead to two passing users missing each other if they do not stay in proximity for long enough. We conducted tests on real devices that showed that iPhones advertising while our app was running in the background could not be discovered by any other device (tested with Android smartphones, iPhones, and MacOS laptop). Because of this limitation, upon closer inspection, we do not consider R3 fulfilled.

3) GOOGLE NEARBY MESSAGES

R1 and R2 are fulfilled for Google Nearby Messages. Looking deeper into R3, exchanging data while the app is in the background, for Android, the documentation describes that scanning should only be done while in the foreground. However, in the background it is still possible to scan for beacon messages.[36] In order to do that, the Nearby Messages Client needs to specify a strategy that only uses BLE. For iOS, both background advertising and scanning are supported. Again, a strategy only using BLE has to be defined for this. It seems like Google worked around the issues regarding iOS and background advertising we reported about in the previous paragraph. We did not find an exact description how Google implemented this. Google's documentation states that background subscriptions are more energy-efficient but provide lower reliability and higher latencies. The remainder of this

[35]https://www.themoviedb.org/, accessed 2020-07-20

[36]https://developers.google.com/nearby/messages/android/get-beacon-messages, accessed 2020-07-20

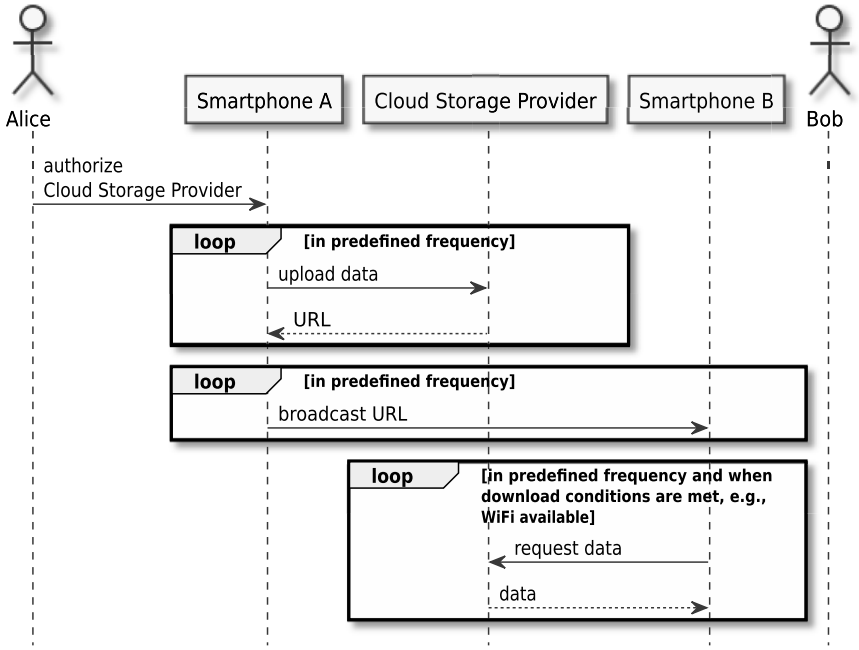

FIGURE 5. Design of the data exchange via a third-party cloud storage provider.

paper will show that Google Nearby Messages is still a viable solution despite its limitations.

4) BRIDGEFY

Comparing Bridgefy to Google Nearby Messages, we see two major drawbacks for Bridgefy. First, the maximum payload size is 2048 bytes whereas the payloads in Nearby Messages can be 50 times that size. The bigger payloads in Nearby Messages will allow us to send more data before downloading data from the Cloud Storage Provider. Second, the framework is a commercial product and the free version restricts the number of offline users. Thus, while Bridgefy in principal might be a viable solution, we opt to go with Google Nearby Messages for our prototype.

5) SUMMARY

The data transfer with Google Nearby Messages is, because of the described relay over Google's servers, strictly speaking not direct device-to-device transfer between two devices. We still chose it for our prototypical implementation because of the described benefits of being free of cost and supporting a larger payload. In our view, the ubiquity of Internet connections allows for using a service that requires Internet connection. Because of the modular design, we could replace the Data Exchange module with one utilizing Bridgefy, then having direct device-to-device communication.

C. RECOMMENDING NEW ITEMS

This paper focuses on an architecture that facilitates decentralized recommender systems. In the prototype, we will relay the recommendation task to external service providers. In this section, we sketch the challenges recommender algorithms in MobRec will face, and pose potential solutions.

When employing a local recommender system on the smartphone, additional data is needed. For content-based filtering, the properties of items have to be known. Third-party service providers can help with retrieving such needed metadata about items. For user-based collaborative filtering, information about the similarity of users is utilized. Whereas services like Spotify or Netflix have very large databases with millions of users, the local databases in MobRec will be much smaller and thus there is a lower likelihood of finding similar users.

We see two possible solutions for this problem. First, we could let each user disseminate more than just his/her own item preferences/ratings and let him/her also send data from previous encounters [51] – this would also address the cold start problem new users will face. Another approach is to calculate the similarity of users in a different way, independent of the users' ratings. In psychology, the *propinquity effect* is the well-studied effect that physical proximity is a good predictor of forming interpersonal bonds [52], [53]. Having a unique identifier for each user and counting the number of times and/or the duration of being in proximity would then likely predict a higher bond. Additional methods are available for determining similarity in proximity-based applications. In [1], we developed and evaluated a method for estimating similarity based on users' context data using probabilistic data structures. In [2], we developed a privacy-preserving method for determining the similarity of two users based on their text messaging data. Both of those

methods can be implemented in our proposed architecture to find similar users, without having the need to have users that rated the same items. Future work will have to show to what extent the propinquity effect or the mentioned similarity metrics yield valuable similarity indications for user-based collaborative filtering. Future work could also investigate the feasibility of approaches like federated learning, effectively exchanging trained models or updates to models for recommendations [54].

Following this idea of having separate similarity data and ratings, in the following sections, we distinguish between two data types:

- *simdata*
- *ratingsdata*

The basis is the assumption that an estimated similarity, for example based on smartphone data, will yield an indication of similar ratings of items. Thus, not only when we find similar users via *ratingsdata*, but also when finding similar users via *simdata*, can we recommend new items to the user. Note that we do not evaluate this assumption because that would likely require the deployment of the whole system and data collection with lots of users including feedback on the given recommendations. Instead, our implementation uses existing third party service providers for recommendations based on similar people (determined by *simdata*) that the user has met.

In order to keep the information fresh, MobRec can simply (re-)download data from users met in the past. The process is then that (at least some) data from each user is automatically tracked, either by mobile sensing or from external service providers, and updated on the user's cloud storage provider. After Bob has met Alice, he knows her URL and can just download her latest data. In our prototypical implementation, where recommendations are relayed to external service providers, up-to-date data is available, for example, TMDB is updated constantly, and the latest movies can be recommended.

Another field which has gained less attention in industry and academia, is that of group recommender systems (e.g., [55]). With its ad hoc nature and immediate preference data exchange, MobRec is ideally suited to be used for pervasive group recommendation scenarios. Exchanging data between several users in a group setting, a local recommender system can calculate recommendations based on the given data, considering the preferences of each user. When utilizing an external service provider for a recommendation, most likely, before contacting it, the preferences of each group member have to be combined into one group profile as most providers will only recommend items for a single user.

VI. IMPLEMENTATION OF MobRec

In this section, we describe the implementation of MobRec. The idea is to have a minimum viable product (MVP) that shows all core functionalities. In the following, we present the core modules of our architecture and describe what frameworks and third party service providers we utilized.

A. MULTIPLATFORM DEVELOPMENT

In order to be able to reach almost 100% of all smartphone users, an app for Android and iOS has to be developed (R1). For the implementation of our MVP, we opted for Ionic,[37] which is an SDK (Software Development Kit) built on top of Cordova, a framework for multiplatform development. Using such multiplatform frameworks is an alternative to developing two distinct apps, allowing to have one code base for both apps. Multiplatform frameworks take a few different approaches in how they work. Often, the differences between the approaches lie in the programming language used and in how the UI is rendered. The latter often either is part of a web component that is displayed within a browser inside the app, or is rendered with native components. This typically results in a trade-off between performance (native is better) and and ease/speed of development (webapp is faster). Looking at statistics about the most used frameworks among developers from 2019,[38] for multiplatform development, we observe that of the surveyed developers, 10.5% reported using React Native, for Cordova it is 7.1%, for Xamarin 6.5%, and for Flutter 3.4%.

B. DATA COLLECTION

In this section, we give details about the implementation of the data collection in MobRec, structured by the three methods given in Section V-A.

1) AUTOMATIC DATA TRACKING

Cordova plugins for accessing the user's location, also while the app is running in the background, are readily available.[39] When tracking the user's location, frequent visits at points of interest or restaurants can indicate preference, and ratings can be inferred. When implementing location tracking, the trade-off is typically between accuracy, frequency, and battery drain. For the users, no interaction is required besides the system confirmation that our app can access his/her location.

In our implemented MVP, we use the location traces of a user as *simdata*. Each location point is first transformed into a Geohash,[40] a short string representation of a latitude/longitude pair. Then, each of the user's locations are entered into a 1-hash Counting Bloom Filter and compared via *CBF-Dice*, a metric we developed in [1].

2) QUERYING THIRD-PARTY SERVICE PROVIDERS

Some third party service providers allow the user – or an application on behalf of the user – to export the items consumed with that provider. This is an easy way to track the user's taste. Listening to music is one of the most com-

[37] https://ionicframework.com/, accessed 2020-07-20

[38] https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/, accessed 2020-07-20

[39] https://www.npmjs.com/package/@mauron85/cordova-plugin-background-geolocation, accessed 2020-07-20

[40] https://web.archive.org/web/20080305223755/http://blog.labix.org/#post-85 and http://geohash.org/, both accessed 2020-07-20

mon activities with smartphones.[41] Spotify has by far the most subscribers in the market of music streaming services (36% market share[42]). Given the user's permission, we access the user's 50 most recently played tracks using OAuth 2.0.[43] Retrieving these recently played tracks regularly yields implicit ratings by the user – based on the assumption that the more a user listened to a track, the more he/she likes it. From the user's side, authorizing our app to access Spotify is the only action he/she must take.

3) MANUAL USER INPUT

Globally, 37% of internet users use Netflix.[44] Watching movies and TV shows is a common pastime and recommending new items in these fields is a common task for recommender systems. At the time of writing, Netflix does not offer a publicly available API, though their website offers the functionality of downloading a *viewing activity* list. However, if our system wants to recommend movies, only tracking those movies available on Netflix will limit the available range of movies: as of 2018, Netflix only offered 4010 movies (in the US).[45] With around 650 films released each year in the US alone,[46] this is not a high number. We use the publicly available *The Movie DB*[47] (TMDB) API to create a visual interface for the user to rate movies. The Movie DB contains 562,522 movies.[48] When designing the visual interface and functionality, we followed the approach of the MovieLens project as described in [56]. This includes searching, rating movies, adding them to a watchlist, and feedback on recommendations (rate, add to watchlist, not interested). Additionally to movies, TV shows are also available via the TMDB API, and thus available for the users of our prototype to rate.

4) SUMMARY

Fig. 6 shows all three approaches for collecting and processing data. Note that only the manual user input needs user interaction. Getting data from third party service provider Spotify only requires a one-time authorization (not shown in the figure). Automatic data tracking is done in the background and uses native libraries of Android and iOS. Once the collected data changed, we can automatically update Alice's *ratingsdata* and *simdata*, denoted in the figure as belonging to Alice with the suffix *_a*.

[41] https://www.pewresearch.org/internet/2015/04/01/us-smartphone-use-in-2015/, accessed 2020-07-20

[42] https://www.statista.com/statistics/653926/music-streaming-service-subscriber-share/, accessed 2020-07-20

[43] https://developer.spotify.com/documentation/web-api/reference-beta/#endpoint-get-recently-played, accessed 2020-07-20

[44] https://www.statista.com/statistics/758369/netflix-video-usage-region/, accessed 2020-07-20

[45] https://www.businessinsider.de/netflix-movie-catalog-size-has-gone-down-since-2010-2018-2, accessed 2020-07-20

[46] According to http://data.uis.unesco.org/

[47] https://www.themoviedb.org/, accessed 2020-07-20

[48] https://www.themoviedb.org/faq/general, accessed 2020-07-07.

C. DATA EXCHANGE (DEVICE-TO-DEVICE Communication)

As described in Section V, the data exchange with nearby users is designed to utilize the Google Nearby Messages API. As the library only supports small payloads of 100 kibibyte, the workaround with uploading the user's *ratingsdata* and *simdata* to a CSP and sharing the public URL of that file, was used. In the following, we present details about the utilization of the Google Nearby Messages API and the sequence of the data exchange.

The Google Nearby Messages API for Android is available in Java and Kotlin and iOS developers can use Swift or Objective-C. In order to integrate the library into an Ionic application, a plugin is required to invoke calls to the native libraries from the JavaScript code. We only found one Cordova plugin that supports the Google Nearby Messages API.[49] However, the implementation is only available for Android. Furthermore, the Android implementation is not configured to work in the background. We developed a custom plugin that solves those issues. Both on Android as well as on iOS, the publishing strategies are set to work in the background, utilizing BLE.

Based on using the described workaround for device-to-device size limitations (cf. Section V-B), additionally to *ratingsdata* and *simdata*, we define the following data types:

- *dataset.* This contains the *ratingsdata* and optionally additional information like a nickname or profile picture, etc. Note that it does not contain *simdata*.
- *cspurl.* This is the URL pointing to the publicly available *dataset* available at a Cloud Storage Provider (CSP).

Fig. 7 shows the initialization of the device-to-device communication utilizing the workaround with a CSP. Alice authorizes access to her account with the CSP. In the MobRec MVP, we use Google Drive. The platform-independent JavaScript code then handles the authorization for Google Drive via OAuth 2.0 and uploads Alice's dataset *dataset_a*. Note that if some data types are not present, for example because Alice did not rate any items yet, parts or the whole set might be empty. The CSP returns *cspurl_a*.

Fig. 8 shows the sequences for updating data at the CSP, broadcasting and scanning via the BLE interface, and receiving broadcast messages. Whenever *ratingsdata_a*, the nickname, profile picture, etc., changes, *dataset_a* is updated. *cspurl_a* stays the same and does not need to be updated. In our implementation, *simdata_a* is small enough to be sent with the payload broadcast via Google Nearby Messages. We trigger the broadcasting of messages from the JavaScript code. The publishing itself, i.e., broadcasting messages via Google Nearby Messages via BLE, is done with our native code plugin. The broadcast message consists of the *cspurl_a* and *simdata_a*. Similarly, subscribing, i.e., listening for broadcast messages from other app instances in BLE range, is also triggered via JavaSciprt code and executed via our native code module.

[49] https://github.com/hahahannes/cordova-plugin-google-nearby, accessed 2020-07-20

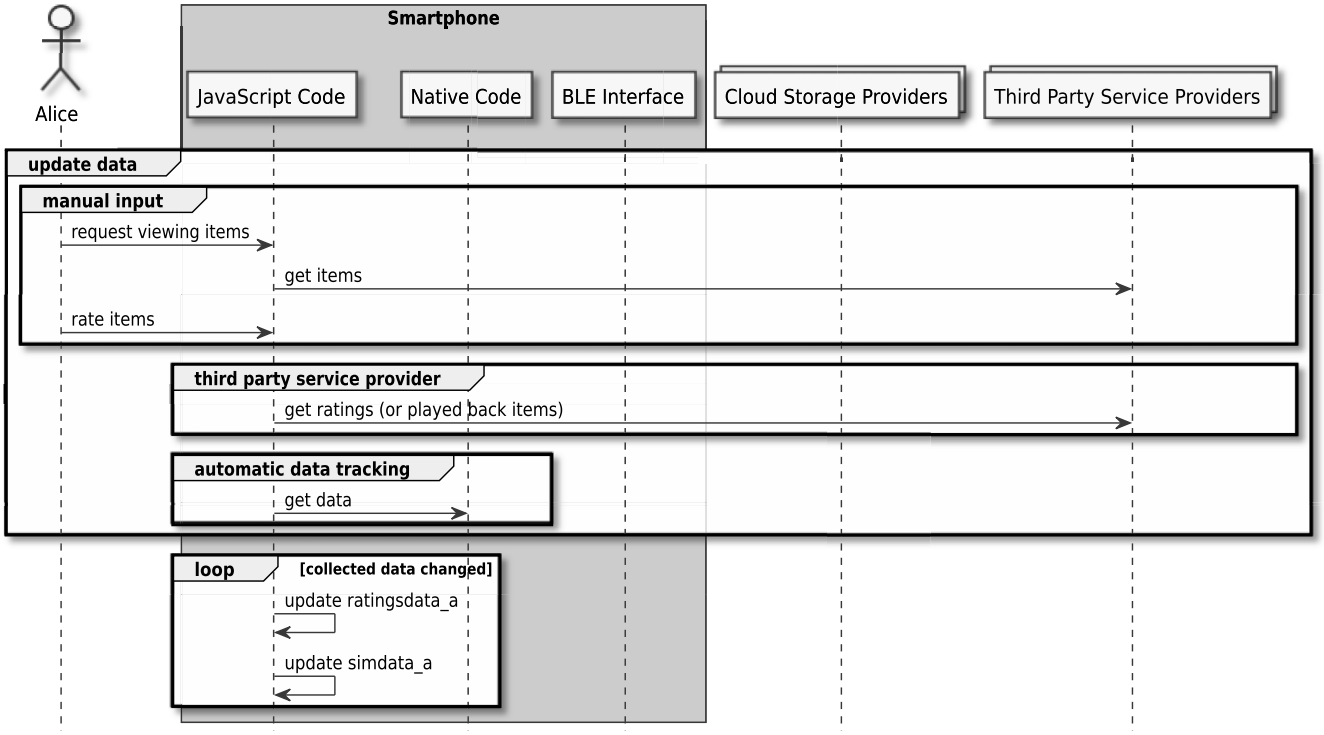

FIGURE 6. Data collection in MobRec.

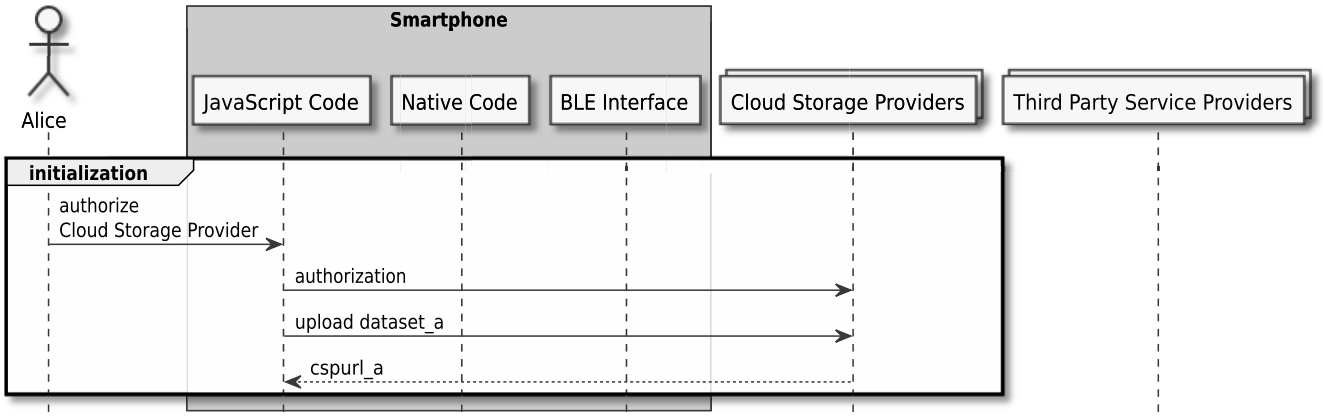

FIGURE 7. Initialization of the device-to-device communication.

Once a message, i.e., broadcast, is received, its content is handed to the platform-independent JavaScript code and processed there. First, the received *simdata_b* is compared to the phone's user's *simdata_a*. If the similarity comparison meets a predefined threshold, *dataset_b* is downloaded via *cspurl_b*. This means that we avoid downloading data from other users if the pre-defined similarity threshold is not met, cf. bottom of Fig. 8.

D. RECOMMENDATIONS

Instead of implementing our own recommender systems, for the MVP, we used external third party service providers. For music recommendations, we utilize Spotify. Their API can return music recommendations based on up to five so-called seed tracks entered.[50] For movie recommendations, we utilize the TMDB API. Given a movie or TV show, other items are recommended.[51] Based on these APIs, we recommend new items to Alice based on Alice's own preferences and based on the preferences of similar people that Alice met.

[50] https://developer.spotify.com/documentation/web-api/reference/browse/get-recommendations/, accessed 2020-07-20

[51] https://developers.themoviedb.org/3/movies/get-movie-recommendations and https://developers.themoviedb.org/3/tv/get-tv-recommendations, both accessed 2020-07-20

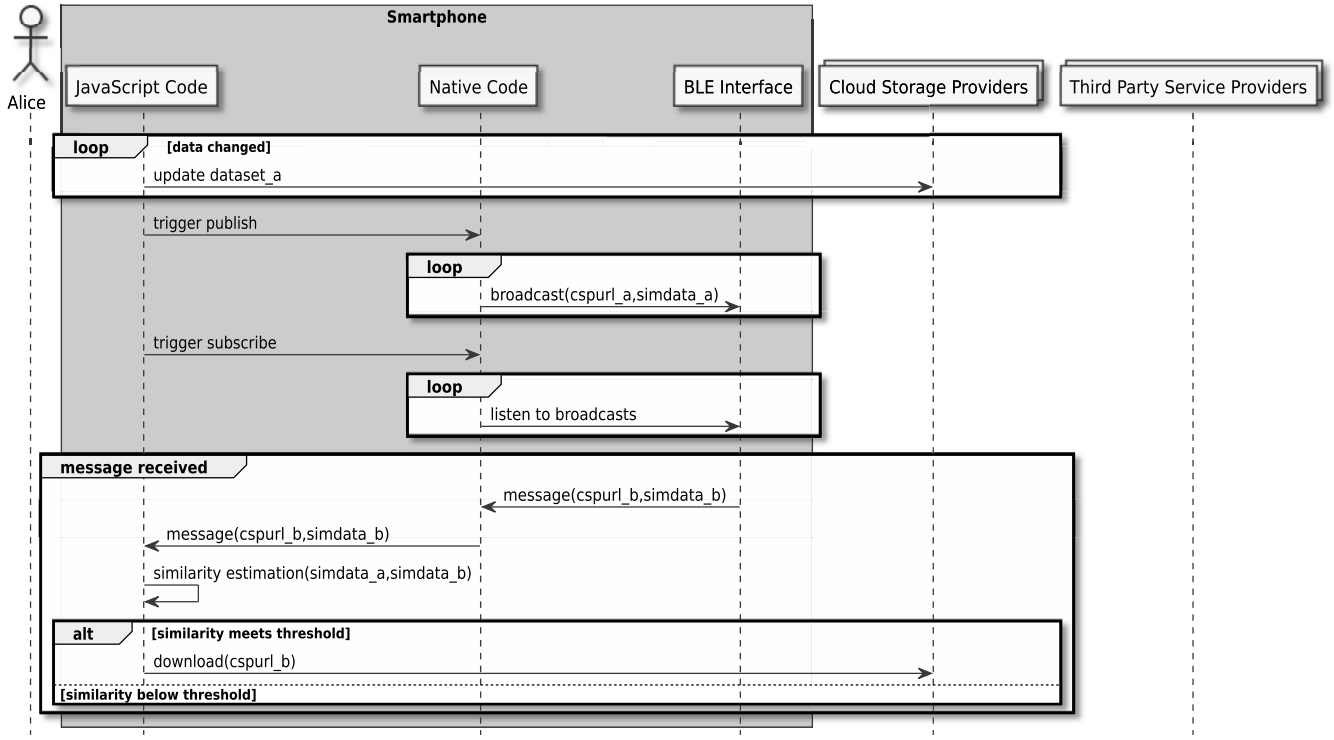

FIGURE 8. Device-to-device communication.

E. MINIMUM VIABLE PRODUCT

Fig. 9, 10, and 11 show screenshots of our MVP. Fig. 9 shows music recommendations based on the user's own listening history, and Fig. 10 shows music recommendations based on similar users met in proximity. Fig. 11 shows how movie recommendations are displayed. Each row indicates to the user why the recommendations are being displayed, some based on own preferences, some based on users previously met.

VII. EVALUATION

In this section, we will evaluate our system, focusing on data exchange via device-to-device communication. Based on the concept of users exchanging data with other users, we analyze different scenarios in order to develop a concept of how to conduct the evaluation. During the design of MobRec, we already accounted for requirements R1–R4. There is no user interaction necessary for data exchange (R2), and we worked around limitations on payload size (R4). In this section, we investigate how well the data exchange works in the background (R3) and if there are any differences observable for Android and iOS (R1).

Imagining the average users, most time is probably spent at home or at work. In those cases, the distance to other users will be very short and the time spent in proximity is rather long, be it during a meeting or while sleeping. Furthermore, chargers will likely be ubiquitously available.

FIGURE 9. Music recommendations based on the user's favorite tracks.

Physical distance, the time it takes to discover nearby devices and exchange data, and battery consumption thus are not critical in this scenario. At busy workplaces, there could be interferences if there are a multitude of devices present though. We assume that Internet connectivity, required in order for Google Nearby Messages to work, is available in

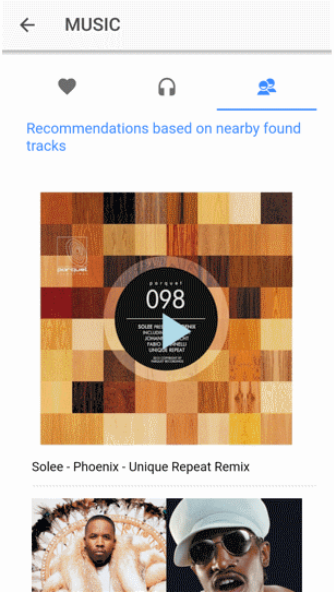

FIGURE 10. Music recommendations based on people met.

FIGURE 11. Movie and TV show recommendations based on user's favorites and based on people met.

almost all home and work scenarios. In order for our concept to work properly, users need to meet new people to exchange data with though. Home and work location will thus not be the crucial situations where users exchange data.

Another scenario is to spend time together at some public or private place. This could be some event like a restaurant visit, a music show, or any other leisure time activity. Here, the time window might be shorter than at home or at work, but is probably still at least around one hour. The distance between users probably ranges from a few to around 50 meters. Depending on the location, Internet connectivity might not be as good as in the previous scenario.

The third scenario is just passing other users, for example when commuting via public transport. The time window of being in proximity might be rather short, e.g., waiting for the metro for a few minutes. We assume the distance to be short, from a few to around 20 meters. Internet connectivity might be bad or at worst non-existent.

We also made tests regarding the physical distance between devices. As our system uses Google Nearby Messages with BLE, we assumed the distances between devices to be unproblematic. We confirmed this with tests in both indoor and outdoor situations. Details about the distance tests are omitted here.

This leaves the following aspects to consider, which we cover in the following sections:

- **Multiple devices**. We will check whether it is feasible to exchange data with multiple devices and whether the presence of multiple other devices has negative effects on the data exchange.
- **Discovery time**. The time needed for successful discovery of present devices in proximity and data transfer. The transfer here just refers to receiving the braodcast data, *simdata* and *cspurl*, as downloading the *dataset* from the CSP is not time-critical and can be done later.
- **Internet connectivity**. We will analyze to what extent bad Internet connectivity influences the data exchange.

In Section VII-D, we will summarize the evaluation and propose optimizations for battery consumption.

A. MULTIPLE DEVICES

In our test setup, we used two iOS devices (both of them iPhone 6, iOS 12.2; in the following, we distinguish the two devices with (a) and (b)) and two Android devices (Xiaomi Mi A2 with Android 9 and LG K8 with Android 7). All devices were placed next to each other, and started broadcasting and scanning at the same time. We recorded the timestamps for the start of broadcasting and for receiving the messages from the other devices. The test was conducted in a busy restaurant. This way, we simultaneously tested the feasibility of sending/receiving data from multiple devices at the same time and potential interferences by other nearby devices. We repeated the test five times. The results of the tests are shown in Table 2. The time given in the table is the time between start of broadcasting/scanning until receiving all three messages from the other devices. Additionally, we show the average time. The LG K8 was the slowest to receive all messages in all test runs. However, the maximum was only 6.2 seconds. Overall, this test indicates that even with multiple devices and in busy places, all messages are received reliably in a matter of seconds. While the whole system might not scale indefinitely, we regard this test as evidence that the data exchange between multiple devices works well.

TABLE 2. Time in seconds until devices received messages from all other devices. Tests conducted in busy restaurant.

Run	LG K8	Xiaomi Mi A2	iPhone 6 (a)	iPhone 6 (b)
1	3.00 s	1.40 s	1.94 s	0.06 s
2	6.20 s	2.00 s	1.00 s	1.00 s
3	4.30 s	2.60 s	1.40 s	1.34 s
4	5.70 s	1.50 s	1.03 s	0.09 s
5	2.30 s	2.10 s	2.10 s	1.10 s
Avg.	4.30 s	1.92 s	1.49 s	0.72 s

B. DISCOVERY TIME

One crucial factor for the evaluation of our system is the time it takes for devices to send and receive broadcast messages, i.e., finding nearby devices and receiving the Google Nearby Messages payload. In order to evaluate this, we consider three binary variables:

- The devices can already be *in proximity* or *move into proximity*.
- The app start can be *now* (started at the beginning of the experiment) or in the *past* (i.e., the app is already running for some time).
- The device to be found is already known or not; i.e., a broadcast message from that device was already received in the past or not.

Table 3 gives an overview of all possible combinations C1 to C5. Three binary variables yield eight overall possible combinations. Three combinations are not possible: Two devices *moving into proximity* cannot be combined with app start *now*. The app already has to be running when moving into proximity. This leaves out two cases (with device known *yes* and *no*). Additionally, when two devices are in proximity and the app start is in the past, then it is not possible that the devices do not know each other already.

TABLE 3. Experiments conducted regarding devices finding each other. Shows the five possible combinations C1 through 5.

	Proximity	App start	Device known	Results
C1	in p.	now	no	Table 2 line 1
C2	in p.	now	yes	Table 2 lines 2–5
C3	in p.	past	yes	Table 4
C4	moving into p.	past	yes	same as C3
C5	moving into p.	past	no	Table 5

The results for C1 and C2 are already given in Table 2. All devices find each other in a matter of seconds, regardless if the devices have received messages from each other before or not. In these experiments, the app's start was at each test run's start. The test from Section VII-A indicates that if the app's start is *now*, discovery time is at most a few seconds.

With C3, we test the time between messages received from the same device. Here, the app's start lies in the past and broadcasting and scanning runs continuously in the background. For this test, we used one Android device (Xiaomi Mi A2, Android 9) and one iOS device (iPhone 6, iOS 12.2). We assume that our test results are still generalizable, as the implementation only differs between different platforms, not different devices of the same platform. We let both devices broadcast and scan for test periods of five hours and recorded when messages were received. Table 4 shows the average time between received messages for four test runs, as well as the average time between messages. On Android, the other device was found at least once per hour, whereas on iOS, the time between messages was around 10 minutes. Thus, if two devices already exchanged messages before, subsequent messages are received in a lower frequency.

TABLE 4. Average time between messages in minutes (C3).

	Android	iOS
Avg. of test run 1	47.13 min	8.00 min
Avg. of test run 2	29.92 min	8.19 min
Avg. of test run 3	57.77 min	11.19 min
Avg. of test run 4	49.23 min	9.20 min
Average	46.01 min	9.14 min

C1, C2, and C3 consider scenarios where the devices are in proximity. In the following, we consider scenarios where two devices move into proximity. In this case, the app start always lies in the past. We distinguish between devices that already exchanged messages before (C4) and those that did not (C5).

In C4, the devices already exchanged messages before. We let the devices move into proximity of each other and recorded the time until messages where received. Repeating the test five times, we got roughly the same results as for C3. This indicates that when the app start lies in the past and the devices already exchanged messages before, it does not make a difference if the devices are already in proximity or move into proximity during runtime.

We note the significant difference between first message (app start *now*, C1/C2) and subsequent messages (app start *past* and device known *yes*, C3/C4) – few seconds vs. several minutes (iOS) / up to one hour (Android). A possible reason for this could be that if two devices already have exchanged messages before, the token for the message was already exchanged and is not sent again until it is renewed. When re-starting broadcasting/scanning, the token might be renewed and thus, messages are received immediately on both sides after starting the scanning. The exact internal mechanisms of Google Nearby Messages are not public and we are not sure when exactly tokens are renewed. We assume that Android and iOS work differently, either regarding the token or regarding the BLE interface or implementation provided by the OS. This would explain the different results for the different platforms. A possible workaround for long time intervals between messages from the same devices could be to re-start broadcasting/scanning in a pre-defined frequency or depending on some other factors like location changes.

The last case to evaluate is C5. It is the same as C4, only that the devices have not exchanged messages before.

We performed the test five times. The results for C5 are shown in Table 5.[52] For both Android and iOS, all messages were received in a time of less than or equal to 10 minutes. On average, messages were received after approximately three to four minutes after devices were in proximity. This is a longer time compared to the results when the broadcasting/scanning was just started and messages were received after a few seconds (C1/C2). It is also significantly less compared to C3/C4. The results from C5 show that when broadcasting/scanning is already running in the background, messages are received after a longer time even though no messages have been exchanged before.

TABLE 5. Time in minutes until message is received after moving into proximity (C5).

Test run	Android	iOS
1	4 min	4 min
2	10 min	7 min
3	2 min	2 min
4	1 min	1 min
5	2 min	2 min
Average	3.8 min	3.2 min

A possible reason for this could be that when re-starting broadcasting/scanning, tokens are exchanged immediately for the first time and then, only in a specific interval of around 1–10 minutes. Battery optimizations by the OS could lead to the inconsistent times for each test run. A possible workaround for this could also be re-starting broadcasting/scanning.

C. INTERNET CONNECTIVITY

Our system utilizes Google Nearby Messages, which requires an Internet connection in order to facilitate the actual message exchange between devices. In this section, we evaluate to what extent this message exchange is influenced in situations where connectivity might be bad, e.g., inside of some underground metro stations.

In order to consistently and reproducibly simulate bad Internet connectivity, we used the iPhone's built-in "Network Link Conditioner." It can simulate different network conditions including "very bad network," which we used in this experiment. It constraints the speed to 1000 kilobyte per second and simulates a packet loss of 10%.

We let one device broadcast messages and then let the iPhone scan for messages while being constraint to "very bad network" conditions. We logged the time it took to receive a message. The experiment was repeated 10 times. As a means of comparison, we repeated the same experiment with LTE connectivity.

Android does not have a similar built-in feature to simulate network conditions. In order to perform the experiment under the same conditions as with the iPhone, we set up a WiFi hotspot on an iPhone, given the "very bad network" constraint and let the Android device (Xiaomi Mi A2) connect to it. Here, again, we conducted 10 test runs with both "very bad network" and LTE.

The results of the test are shown in Table 6. Note that with respect to the three binary variables introduced in Section VII-B, this is an experiment with combinations C1/C2. For bad network conditions, the time until a message is received is significantly higher. But still all messages in the test run were received with a maximum discovery time of 9:41 minutes. On average, each message was received almost instantly via LTE (confirming the results from Table 2). The average delivery time for bad connectivity was 1-2 minutes.

TABLE 6. Time in minutes until message is received with different network conditions.

	Android		iOS	
Test run	bad conn.	LTE	bad conn.	LTE
1	0:40	0:02	2:00	0:01
2	1:45	0:02	0:10	0:01
3	0:06	0.02	1:00	0:01
4	2:00	0:02	0:47	0:01
5	0:52	0:02	3:12	0:01
6	0:41	0:02	9:41	0:01
7	0:57	0:02	0:33	0:01
8	1:31	0:02	0:04	0:01
9	0:14	0:02	0:02	0:01
10	0:14	0:02	0:04	0:01
Average	0:54	0:02	1:45	0:01

D. EVALUATION SUMMARY AND BATTERY DRAIN OPTIMIZATIONS

We summarize the key results of our evaluation as follows:

- Messages from multiple devices in busy scenarios are sent and received without issues within seconds.
- (Re-)starting the broadcast/scan mechanism makes the device receive message in a matter of seconds.
- New devices in proximity can be discovered in 3–4 minutes.
- Discovery of devices met before is slow – on average 9 minutes (iOS) and 46 minutes (Android).
- Bad Internet connectivity will introduce an overall negligible delay in discovery time of around 2 minutes.

Looking back at the scenarios we described for exchanging data between devices, most of them are realizable. The discovery time of new devices of 3–4 minutes might lead to some missed opportunities of data exchange in quickly moving scenarios like waiting at the metro station. The longest time window was between messages from the same device. In those cases, the user would receive the same data anyway, which would not help to improve the performance of the recommender systems. Even if we assume new data is present, there is a simple workaround: the data that is transferred is the

[52] Note that we do not give seconds-accuracy here in order to account for the small inaccuracies introduced by not measuring the time it took to move into proximity.

cspurl, which does not change when the *dataset* is updated. We can just check if the *dataset* changed and re-download from the users previously met. This way, each user would have only to be met once. On the other hand, this reduces the recognition of meeting the same user multiple times – which could indicate similarity. Also, changes in *simdata* would be missed.

Regarding battery drain, permanently running broadcast/scan in the background accounts for roughly 5% (Android) to 10% (iOS) of battery consumption per hour. Such a battery drain is not acceptable for real-world deployment. However, significant improvements for both average discovery time and battery drain could be easy: re-starting the broadcast/scan mechanism depending on specific times and locations will improve both aspects at once. Consider the following naive optimization. We assume that the time of each smartphone is running in sync, as they usually use online servers to sync their time. Then, we can let our app turn on broadcasting/scanning at the exact same time on every phone for two minutes. Our evaluation shows that two minutes is enough to reliably find most devices in proximity, even during bad network connectivity. We could let the app broadcast/scan for two minutes every 15 minutes, as long as there has been a location change. We assume that on average at least during 16 hours of the day, there won't be location changes (sleep and work). This leaves eight hours, each of which has four 15-minute intervals. Multiplied by two minutes of broadcast/scanning, this yields 64 minutes of running in the background instead of 24 hours. This would reduce the battery drain to less than 5% of its original value, and likely still produce a lot of the data exchanges that would happen during permanent broadcasting/scanning. While we have not tested this, the implementation of this optimization should be possible with Ionic's background mode.[53] If that fails and native code is necessary, in Android, the Alarm Manager[54] can fire events at exact times. In iOS, a workaround might be necessary, for example by utilizing media playback to keep the app from being suspended.[55]

VIII. CONCLUSION

Current recommender systems often exhibit lock-in effects. Recommendations might be biased according to the interests of the providing platform and are often bound to the items available through the platform. We proposed a decentralized mobile architecture for recommender systems, MobRec, that leverages the preferences/ratings from users that are, or have been, in proximity. The introduced system runs on the users' smartphones and utilizes existing external third-party service providers. It is built on the general concept that similar people like similar things.

MobRec consists of three main modules, *data collection*, *data exchange*, and *recommender system*. We highlighted that while short-range wireless transmission technologies are implemented on all modern smartphones, exchanging larger amounts of data in the background without user interaction on a system available for off-the-shelf Android and iOS devices remains a challenging task. We proposed a solution based on Google Nearby Messages that let's users broadcast a URL of their data on a cloud storage provider. The evaluation of our MobRec prototype shows that the discovery time – the time needed to find other devices and exchange data – is just a few seconds when the broadcasting/scanning mechanism was just started. Overall, new devices in proximity are discovered within 3–4 minutes on average. Devices previously met are discovered again at a much slower rate, from around 10 minutes (iOS) to around 46 minutes (Andrdoid) on average. Because Google Nearby Messages requires an Internet connection, we also evaluated the influence of bad Internet connectivity and found that it introduces a delay of about 1–2 minutes on average. Battery drain remains an issue with constant broadcasting/scanning. We proposed the simple optimizations of only broadcasting/scanning for messages in fixed time intervals. While our MobRec prototype relays the recommendation to external service providers, we pointed out the challenges and potential solutions for local recommender algorithms, including finding similar users.

Future work includes the deployment of the system with real users. Regarding the recommender system, future work includes the implementation of a mobile recommender engine operating on locally available data. A simulation with a real data set, for example collected from our previous research [57], [58], could help evaluate the quality of the recommendations that such a system can provide. Future work also consists of adapting MobRec for group scenarios: in an ad hoc manner, a group of users can use some device-to-device communication feature that exchanges data between the users in order to provide some service based on the shared data, for example group recommendations. In that case, R2 and R3, the broadcasting of data in the background without any user interaction, would not be applicable, making it possible to utilize a broader range of the device-to-device technologies.

ACKNOWLEDGMENT

We are grateful for the support provided by Tobias Eichinger, Axel Küpper, Robert Staake, Jan Pokorski, and Yong Wu.

REFERENCES

[1] F. Beierle, "Do you like what I like? Similarity estimation in proximity-based mobile social networks," in *Proc. 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Communications/ 12th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2018, pp. 1040–1047.

[2] T. Eichinger, F. Beierle, S. U. Khan, and R. Middelanis, "Affinity: A system for latent user similarity comparison on texting data," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[53] https://ionicframework.com/docs/native/background-mode, accessed 2020-07-20

[54] https://developer.android.com/guide/background#alarmmanager, accessed 2020-07-20

[55] https://developer.apple.com/documentation/avfoundation/media_assets_playback_and_editing/creating_a_basic_video_player_ios_and_tvos/enabling_background_audio, accessed 2020-07-20

[3] M. Ahmed, Y. Li, M. Waqas, M. Sheraz, D. Jin, and Z. Han, "A survey on socially aware Device-to-Device communications," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2169–2197, 3rd Quart., 2018.
[4] W. Zhang, H. Flores, and P. Hui, "Towards collaborative multi-device computing," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2018, pp. 22–27.
[5] C.-N. Ziegler, "Semantic Web recommender systems," in *Current Trends in Database Technology–EDBT Workshops* (Lecture Notes in Computer Science), W. Lindner, M. Mesiti, C. Türker, Y. Tzitzikas, and A. I. Vakali, Eds. Berlin, Germany: Springer, 2005, pp. 78–89.
[6] R. Baraglia, P. Dazzi, M. Mordacchini, and L. Ricci, "A peer-to-peer recommender system for self-emerging user communities based on gossip overlays," *J. Comput. Syst. Sci.*, vol. 79, no. 2, pp. 291–308, Mar. 2013.
[7] L. Nunes Barbosa, J. Gemmell, M. Horvath, and T. Heimfarth, "Distributed user-based collaborative filtering on an opportunistic network," in *Proc. IEEE 32nd Int. Conf. Adv. Inf. Netw. Appl. (AINA)*, May 2018, pp. 266–273.
[8] F. Beierle and T. Eichinger, "Collaborating with users in proximity for decentralized mobile recommender systems," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2019, pp. 1192–1197.
[9] M. Falch, A. Henten, R. Tadayoni, and I. Windekilde, "Business models in social networking," in *Proc. CMI Int. Conf. Social Netw. Communities*, 2009, pp. 1–23.
[10] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, and K. Rzadca, "Decentralized online social networks," in *Handbook of Social Network Technologies and Applications*. Boston, MA, USA: Springer, 2010, pp. 349–378.
[11] A. Bleicher, "The anti-facebook," *IEEE Spectr.*, vol. 48, no. 6, pp. 54–82, Jun. 2011.
[12] G. Ruffo and R. Schifanella, "A peer-to-peer recommender system based on spontaneous affinities," *ACM Trans. Internet Technol.*, vol. 9, no. 1, pp. 1–34, Feb. 2009.
[13] C. Mettouris and G. A. Papadopoulos, "Ubiquitous recommender systems," *Computing*, vol. 96, no. 3, pp. 223–257, Mar. 2014.
[14] J. Manweiler, R. Scudellari, and L. P. Cox, "SMILE: Encounter-based trust for mobile social services," in *Proc. 16th ACM Conf. Comput. Commun. Secur.*, 2009, pp. 246–255.
[15] A. C. Champion, Z. Yang, B. Zhang, J. Dai, D. Xuan, and D. Li, "E-SmallTalker: A distributed mobile system for social networking in physical proximity," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 8, pp. 1535–1545, Aug. 2013.
[16] F. Beierle, K. Grunert, S. Gondor, and V. Schluter, "Towards psychometrics-based friend recommendations in social networking services," in *Proc. IEEE Int. Conf. AI Mobile Services (AIMS)*, Jun. 2017, pp. 105–108.
[17] W.-S. Yang and S.-Y. Hwang, "iTravel: A recommender system in mobile peer-to-peer environment," *J. Syst. Softw.*, vol. 86, no. 1, pp. 12–20, Jan. 2013.
[18] J. Shu, S. Kosta, R. Zheng, and P. Hui, "Talk2Me: A framework for Device-to-Device augmented reality social network," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Mar. 2018, pp. 1–10.
[19] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, R. Pryss, and J. Zimmermann, "TYDR–Track your daily routine. Android App for tracking smartphone sensor and usage data," in *Proc. IEEE/ACM 5th Int. Conf. Mobile Softw. Eng. Syst. (MOBILESoft)*, May/Jun. 2018, pp. 72–75.
[20] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, R. Pryss, and J. Zimmermann, "Context data categories and privacy model for mobile data collection apps," *Procedia Comput. Sci.*, vol. 134, pp. 18–25, 2018.
[21] H. Xiong, Y. Huang, L. E. Barnes, and M. S. Gerber, "Sensus: A cross-platform, general-purpose system for mobile crowdsensing in human-subject studies," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (UbiComp)*, Sep. 2016, pp. 415–426.
[22] K. Jayarajah, R. K. Balan, M. Radhakrishnan, A. Misra, and Y. Lee, "Livelabs: Building *in-situ* mobile sensing & behavioural experimentation testbeds," in *Proc. 14th Annu. Int. Conf. Mobile Syst., Appl., Services*, 2016, pp. 1–15.
[23] D. Ferreira, V. Kostakos, and A. K. Dey, "AWARE: Mobile context instrumentation framework," *Frontiers ICT*, vol. 2, p. 6, Apr. 2015.
[24] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot, "MobiClique: Middleware for mobile social networking," in *Proc. 2nd ACM Workshop Online Social Netw.*, 2009, pp. 49–54.
[25] N. Eagle and A. Pentland, "Social serendipity: Mobilizing social software," *IEEE Pervas. Comput.*, vol. 4, no. 2, pp. 28–34, Apr. 2005.
[26] A. Sapuppo, "Spiderweb: A social mobile network," in *Proc. Eur. Wireless Conf. (EW)*, Apr. 2010, pp. 475–481.
[27] Z. Yu, Y. Liang, B. Xu, Y. Yang, and B. Guo, "Towards a smart campus with mobile social networking," in *Proc. Int. Conf. Internet Things 4th Int. Conf. Cyber, Phys. Social Comput.*, Oct. 2011, pp. 162–169.
[28] J. Teng, B. Zhang, X. Li, X. Bai, and D. Xuan, "E-Shadow: Lubricating social interaction using mobile phones," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, Jun. 2011, pp. 909–918.
[29] Z. Yang, B. Zhang, J. Dai, A. C. Champion, D. Xuan, and D. Li, "E-SmallTalker: A distributed mobile system for social networking in physical proximity," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 468–477.
[30] N. Davies, A. Friday, P. Newman, S. Rutlidge, and O. Storz, "Using Bluetooth device names to support interaction in smart environments," in *Proc. 7th Int. Conf. Mobile Syst., Appl., Services (Mobisys)*, 2009, pp. 151–164.
[31] Y. Shafranovich, "Bluetooth data exchange between Android phones without pairing," 2015, *arXiv:1507.00650*. [Online]. Available: http://arxiv.org/abs/1507.00650
[32] O. Turkes, H. Scholten, and P. J. M. Havinga, "Opportunistic beacon networks: Information dissemination via wireless network identifiers," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2016, pp. 1–6.
[33] J. Teng, B. Zhang, X. Li, X. Bai, and D. Xuan, "E-shadow: Lubricating social interaction using mobile phones," *IEEE Trans. Comput.*, vol. 63, no. 6, pp. 1422–1433, Jun. 2014.
[34] J. Yang, C. Poellabauer, P. Mitra, J. Rao, and C. Neubecker, "BlueNet: BLE-based ad-hoc communications without predefined roles," in *Proc. IEEE SmartWorld, Ubiquitous Intell. Comput., Adv. Trusted Comput., Scalable Comput. Commun., Cloud Big Data Comput., Internet People Smart City Innov. (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, Aug. 2017, pp. 1–8.
[35] A. Sikora, M. Krzyszton, and M. Marks, "Application of Bluetooth low energy protocol for communication in mobile networks," in *Proc. Int. Conf. Mil. Commun. Inf. Syst. (ICMCIS)*, May 2018, pp. 1–6.
[36] D. J. Dubois, Y. Bando, K. Watanabe, and H. Holtzman, "ShAir: Extensible middleware for mobile peer-to-peer resource sharing," in *Proc. 9th Joint Meeting Found. Softw. Eng. (ESEC/FSE)*, 2013, pp. 687–690.
[37] D. Camps-Mur, A. Garcia-Saavedra, and P. Serrano, "Device-to-device communications with Wi-Fi direct: Overview and experimentation," *IEEE Wireless Commun.*, vol. 20, no. 3, pp. 96–104, Jun. 2013.
[38] Z. Lu, G. Cao, and T. La Porta, "Networking smartphones for disaster recovery," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Mar. 2016, pp. 1–9.
[39] T. Oide, T. Abe, and T. Suganuma, "Infrastructure-less communication platform for Off-The-Shelf Android smartphones," *Sensors*, vol. 18, no. 3, p. 776, Mar. 2018.
[40] Y. Wang, A. V. Vasilakos, Q. Jin, and J. Ma, "Survey on mobile social networking in proximity (MSNP): Approaches, challenges and architecture," *Wireless Netw.*, vol. 20, no. 6, pp. 1295–1311, Aug. 2014.
[41] Z. Mao, J. Ma, Y. Jiang, and B. Yao, "Performance evaluation of WiFi direct for data dissemination in mobile social networks," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 1213–1218.
[42] N. Aneja and S. Gambhir, "Profile-based ad hoc social networking using Wi-Fi direct on the top of Android," *Mobile Inf. Syst.*, vol. 2018, Oct. 2018, Art. no. 9469536.
[43] K. Kwan and B. Greaves, "FileLinker: Simple Peer-to-Peer file sharing using Wi-Fi direct and NFC," in *Proc. IST-Africa Week Conf. (IST-Africa)*, May 2019, pp. 1–9.
[44] S. Trifunovic, M. Kurant, K. A. Hummel, and F. Legendre, "WLAN-opp: Ad-hoc-less opportunistic networking on smartphones," *Ad Hoc Netw.*, vol. 25, pp. 346–358, Feb. 2015.
[45] Y. Wang, L. Wei, Q. Jin, and J. Ma, "Alljoyn based direct proximity service development: Overview and prototype," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng.*, Dec. 2014, pp. 634–641.
[46] H. Lokhandwala, S. M. Kala, and B. R. Tamma, "Min-O-mee: A proximity based network application leveraging the AllJoyn framework," in *Proc. Int. Conf. Comput. Netw. Commun. (CoCoNet)*, Dec. 2015, pp. 613–619.
[47] S. M. Kala, V. Sathya, S. S. Magdum, T. V. K. Buyakar, H. Lokhandwala, and B. R. Tamma, "Designing infrastructure-less disaster networks by leveraging the AllJoyn framework," in *Proc. 20th Int. Conf. Distrib. Comput. Netw.*, Jan. 2019, pp. 417–420.

[48] J. Rodrigues, E. R. B. Marques, L. M. B. Lopes, and F. Silva, "Towards a middleware for mobile edge-cloud applications," in *Proc. 2nd Workshop Middleware Edge Clouds Cloudlets (MECC)*, 2017.
[49] F. Beierle, K. Grunert, S. Gondor, and A. Kupper, "Privacy-aware social music playlist generation," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 5650–5656.
[50] J. Froehlich, M. Y. Chen, I. E. Smith, and F. Potter, "Voting with your feet: An investigative study of the relationship between place visit behavior and preference," in *UbiComp 2006: Ubiquitous Computing* (Lecture Notes in Computer Science), P. Dourish and A. Friday, Eds. Berlin, Germany: Springer, 2006, pp. 333–350.
[51] T. Eichinger, F. Beierle, R. Papke, L. Rebscher, H. C. Tran, and M. Trzeciak, "On gossip-based information dissemination in pervasive recommender systems," in *Proc. 13th ACM Conf. Recommender Syst.*, Sep. 2019, pp. 442–446.
[52] D. M. Marvin, "Occupational propinquity as a factor in marriage selection," *Quart. Publications Amer. Stat. Assoc.*, vol. 16, no. 123, pp. 131–150, Sep. 1918.
[53] L. Festinger, S. Schachter, and K. Back, "The spatial ecology of group formation," in *Social Pressure in Informal Groups*. New York, NY, USA: Harper, 1950, pp. 141–161.
[54] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.* (Proceedings of Machine Learning Research), vol. 54, A. Singh and J. Zhu, Eds. PMLR, 2017, pp. 1273–1282.
[55] A. Crossen, J. Budzik, and K. J. Hammond, "Flytrap: intelligent group music recommendation," in *Proc. 7th Int. Conf. Intell. User Interfaces*, 2002, pp. 184–185.
[56] B. N. Miller, I. Albert, S. K. Lam, J. A. Konstan, and J. Riedl, "Movielens unplugged: Experiences with a recommender system on four mobile devices," in *People and Computers XVII–Designing for Society*, E. O'Neill, P. Palanque, and P. Johnson, Eds. London, U.K.: Springer, 2004, pp. 263–279.
[57] F. Beierle, V. T. Tran, M. Allemand, P. Neff, W. Schlee, T. Probst, J. Zimmermann, and R. Pryss, "What data are smartphone users willing to share with researchers?" *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 6, pp. 2277–2289, Jun. 2020.
[58] F. Beierle, T. Probst, M. Allemand, J. Zimmermann, R. Pryss, P. Neff, W. Schlee, S. Stieger, and S. Budimir, "Frequency and duration of daily smartphone usage in relation to personality traits," *Digit. Psychol.*, vol. 1, no. 1, pp. 20–28, Jun. 2020.

FELIX BEIERLE (Member, IEEE) received the M.A. degree in media studies and American studies from the University of Marburg in 2009, the M.Sc. degree in computer science from the University of Hagen in 2014, and the Ph.D. degree in computer science from Technische Universität Berlin in 2020.

During his studies, he worked as a Software Engineer with Capgemini. He is currently a Postdoctoral Researcher with the Service-Centric Networking group, Technische Universität Berlin, and the Telekom Innovation Laboratories, Berlin, Germany. His research interests include ubiquitous computing, social networking, recommender systems, and mHealth.

SIMONE EGGER received the B.Sc. degree in computer science and media from the Technische Hochschule Nürnberg in 2015 and the M.Sc. degree in computer science from Technische Universität Berlin, Berlin, Germany, in 2020.

During her studies, she worked as a Mobile App Developer in an agency for cross-platform app development. She is currently with the Service-centric Networking, Telekom Innovation Laboratories, Technische Universität Berlin, Berlin, and an IT Consultant with Netlight Consulting, where she is involved in different client projects to deliver solutions, where IT is business critical.