A Proximity-Based Framework for Mobile Services

Justin Lewis Salmon Rong Yang
Department of Computer Science and Creative Technologies
University of the West of England
Bristol, United Kingdom
Email: justin2.salmon@live.uwe.ac.uk
rong.yang@uwe.ac.uk

***Abstract*—Peer-to-peer proximity-based wireless networking can provide improved spatial and temporal semantics and independence over alternative wireless topologies that rely on static network infrastructure, and can potentially enable new classes of mobile applications. However, the difficulties of setting up such ad-hoc connections has thus far been a development barrier. There is currently a need for an abstraction tool to allow application developers to exploit the potential advantages of such networks with minimal knowledge of the underlying connectivity. We present Proxima, a framework for the Android platform, which employs ad-hoc device-to-device connections and proactive mesh routing for a decentralised topology with solely proximity-based rich content dissemination. The framework is designed to be developer- and user-friendly with minimal configuration effort, lightweight, reusable and hardware independent. After compilation, the size of its binary distribution is only 6MB. We have further developed a real-life application, named TuneSpy, based around sharing music with local peers. The development of TuneSpy has produced two positive outcomes. Firstly, it strongly demonstrates the ease of writing proximity-based applications with Proxima. Secondly, it has served as a testing platform for all framework functionality.**

***Keywords*-Networking; proximity-based; mobile; ad-hoc; peer-to-peer.**

I. Introduction

Location-based applications have had several successes in the mobile arena. These applications have usually relied on traditional infrastructure networks such as Wi-Fi or cellular towers for timely publication of the location information to a centralised server, where the location information is determined by the users current GPS location. Some location-based applications, such as Foursquare [1], have traditionally oriented around the mobile device user arriving within proximity of a particular fixed location, while others have also oriented around the location of other nearby devices.

A. The proximity-based paradigm

This paper aims to explore a different paradigm, whereby reliance is solely upon mobile users being within proximity of one another, in a peer-to-peer orientation. Publication of information is also solely to proximal neighbours, and thus does not rely on additional networking infrastructure. We refer to this paradigm as the *proximity-based* paradigm throughout the remainder of the paper.

The concept of purely local, ephemeral, decentralised information dissemination has several apparent advantages. The ability to sense and connect with those around us allows purely real-time and real-space communication and collaboration. It gives us additional spatial and temporal semantics, i.e. a sense of *"here-and-now"*, on which to build applications upon [2]. The lack of infrastructure dependence also enables communication in a far wider range of locations.

There are unique potentials for mobile applications that might necessitate or otherwise exploit these *"here-and-now"* semantics, such as social, multimedia, crisis management, and gaming. There are some scenarios where dissemination onto the wider Internet is not desired, not useful, or simply not necessary due to the inherent importance of the temporal aspect.

B. A scenario

To illustrate the idea of proximity-based computing, let us imagine a scenario where this paradigm might apply. Consider, for example, a hypothetical train journey. Two people, Alice and Bob, are both passengers on this journey. Bob happens to be using his smartphone to listen to a track by his favourite music artist through a pair of headphones. Alice, who also has a smartphone, notices Bob, and begins to wonder what he might be listening to. She wishes that there were some way for her to tune-in to what Bob is currently listening to (assuming that Bob has pre-authorised this type of sharing behaviour), as if Bob were able to broadcast his current track to the people around him. Assuming that this train does not have a Wi-Fi access point available, how is Alice able to listen to Bob? How might their smartphones be able to connect directly to each other, simply based on the fact that they are in proximity to each other? The advanced capabilities of their devices should surely allow this behaviour.

The *"here-and-now"* semantics of this encounter are clear. Alice must be spatially close to Bob, and the information she is able to retrieve is temporally dependent, i.e. Bob will only be listening to a particular track for a certain period of time; after that time, the information is irrelevant and

unobtainable. Such is the ephemeral nature of the proximity-based paradigm.

C. The problem

As it stands today, there is a lack of a reusable platform on which to build applications that can make use of the aforementioned functionality. This is in part due to the complexity and difficulty of setting up such peer-to-peer connections, and the various underlying technologies that could potentially be used. Additionally, unlike traditional infrastructure connections, issues such as IP address assignment and service discovery do not have standard precedents in peer-to-peer networks.

This paper aims to tackle these issues, using modern tools to create a platform that is capable of operation regardless of infrastructure availability, and reaches as close to zero-configuration as possible. We present a reusable platform to allow future application developers to exploit this temporal and spatial information. Due to the myriad of available mobile devices and their varying levels of compatibility and interoperability, we will initially build on the Android platform, whilst still aiming to support as wide a range of devices as possible.

The rest of the paper is organised as follows. The next section discusses the background to the paper and related work. Section 3 describes our work, the from the top level view to implementation details. Section 4 discusses testing and evaluation. Finally we conclude with Section 5.

II. Background and Related Work

This section analyses the technologies that one might use to implement peer-to-peer mobile connectivity and the rationale behind the selections that have been made for our implementation. Some existing solutions are outlined and compared with our implementation.

A. Background

Two major technologies that allow wireless connection between mobile devices are Bluetooth and Wi-Fi. The former is standardised by the Bluetooth Special Interest Group, and the latter standardised by the IEEE. Both are available on most modern mobile phones, tablets, laptops, and other electronic devices.

Bluetooth uses short-wavelength radio waves; the data transmission speed is comparatively slow. It is best suited to low-bandwidth peer-to-peer applications. Wi-Fi can achieve much higher transmission speed and wider range than Bluetooth. Therefore Wi-Fi technology has dominated for most high-bandwidth internet-based applications.

However, we know that under standard Wi-Fi (otherwise known as Basic Service Set (BSS) for single access points or Extended Service Set (ESS) for multiple access points), the connection usually relies upon being within range of a fixed wireless network access point. This is something against our

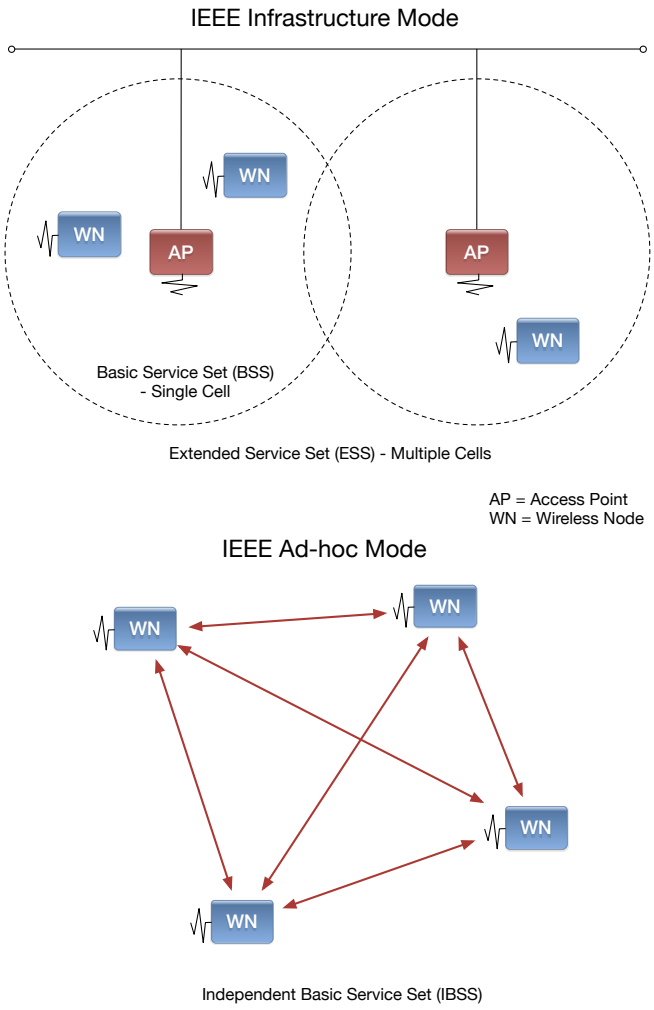

Figure 1. IEEE 802.11 operational modes.

proximity-based paradigm. For a modern mobile platform, we need a technology like Bluetooth which allows direct device-to-device connection but with the speed and operable range benefits of a technology like Wi-Fi.

The proposed solution is to use Wi-Fi ad-hoc mode, otherwise known as Independent Basic Service Set (IBSS), which is an alternative operation mode allowing connection without an access point. With Wi-Fi ad-hoc mode, we can achieve device-to-device communication like Bluetooth yet at a high efficient Wi-Fi speed. See Figure 1. for an illustration of infrastructure Wi-Fi vs ad-hoc Wi-Fi. This mode of operation is supported by most network interface drivers. It is, however, not supported by default on Android devices. Some configuration is needed to enable it, including gaining root access to the device.

This is exactly what we need for achieving a real-time mobile platform. However, Wi-Fi ad-hoc mode only provides the basic connectivity infrastructure. In order for it to be

useful for complex applications, more software is needed to build on to achieve functions such as automatic network configuration, routing and service discovery. These problems have burdened mobile application developers wishing to write applications leveraging the potential benefits of Wi-Fi ad-hoc mode. This is largely factored by the tediousness of configuring such networks, and by the diversity of competing strategies and protocols.

Our goal was to build a general framework on top of ad-hoc Wi-Fi, which allows people to enjoy the benefits of ad-hoc Wi-Fi without worrying about any low level issues. Our aim is to make the framework user-friendly, hardware-independent, powerful, lightweight and reusable.

B. Related Work

Wi-Fi Direct [3] is a piece of software which does not use ad-hoc Wi-Fi. It provides a good interface allowing people to develop mobile applications. The main idea of Wi-Fi Direct is to create a software access point to active connectivity without a real access point. It is a well established software and available only on newer mobile devices. So Wi-Fi Direct is hardware-dependent, that is against one of our initial aims. Moreover, Wi-Fi Direct is not capable of multi-hop routing, and requires a manual setup procedure (Wi-Fi Protected Setup). These are the major differences from our work.

The Serval Project [4] aims to bring infrastructure-free mobile communication to people in need, such as during crisis and disaster situation when vulnerable infrastructure like phone cell tower and mains electricity are cut off. This is also one of our aims. We want to our framework to be more generic and reusable, while the Serval Project is only dedicated to this special purpose.

Commotion Wireless [5] is an open-source communication tool that uses mobile phones, computers, and other wireless devices to create decentralised mesh networks. We share one common aim that is decentralisation. However we are focusing more on a framework to allow people to develop applications rather than just communications.

Open Garden [6] is another free piece of software which creates an overlay mesh network using Bluetooth and Wi-Fi connections across a range of mobile devices, from smartphones to tablets to laptops and desktops. The aim of Open Garden is to provide connection to internet without an Wi-Fi access point, while our aim is more general than just accessing the Internet.

Project SPAN [7] is an Android API for ad-hoc networks. While actively developed, it does not provide any mechanisms for automatic IP configuration, service discovery or name resolution.

The findings from our field study show that the existing work on proximity-based wireless networks is either application-specific, i.e. based around a specific use case; or it is not quite functionally adequate to meet our requirements for a useful framework for proximity-based computing.

III. Proxima–A general framework for proximity-based communication

In this section we describe the Proxima framework from an application developers viewpoint, in terms of how to integrate proximity-based functionality into an Android application. We then describe some of the low-level detail and design decisions that were made during development.

A. Developer Viewpoint

The Proxima framework provides a fully asynchronous, thread-safe API and can be used by multiple client applications on a single device. It acts like a ”neighbours and resources finder” server. It is not necessary for API users to worry about how to discover neighbors and what services are available; the framework will take care of these low-level details.. Communication with the framework is done using asynchronous method calls with user-supplied callback functions. Users are sent broadcast intents when changes occur, such as a new device coming into proximity.

Registering a client application and retrieving a list of neighbors is demonstrated in Listing 1. The client-facing API takes inspiration from the Android Wi-Fi P2P API. This is a simplified version; the calls to initialize() and discoverNeighbors() both take nullable callback arguments to notify the user of success or failure. These calls are usually placed into the onCreate() method of an Android activity.

The framework also sends periodic broadcasts to all registered client applications when there is a neighbor change, so clients should implement a broadcast receiver to listen for these specific broadcasts. Neighbors can be retrieved at any point once the discovery process has begun.

Listing 1. Example usage of the Proxima API.

```
ProximityManager mProximityManager
    = ProximityManager.getInstance();

// This must be the first call into the API
Channel mChannel = mProximityManager
    .initialize(this);

// Begin the neighbor discovery procedure
mProximityManager.discoverNeighbors(mChannel);

// Retrieve the list of neighbors
mProximityManager.retrieveNeighbors(mChannel,
    new NeighborListListener()
{
    @Override
    public void onNeighborsAvailable
        (NeighborList neighbors)
    {
        // ...
    }
});
```

Each neighbour in the retrieved list represents a device detected by the routing daemon. Prior to passing this list to the user via the callback, the framework queries the a service on each neighbour and retrieves metadata such as the user-specified device name, the device GPS coordinates and the available services (Proxima applications) provided by the device. The metadata service is described below.

In summary, as a mobile application developer, to use the Proxima framework, all you need to do is to register the application with the framework as a client at the beginning. Then, the framework will send the devices found in the neighbourhood to you, and any changes in the network will be updated via self-defined callback function. In Section IV, we will present a real-world application which was developed on top of the Proxima framework. The swift and successful development of the application has demonstrated that the framework is very simple and easy to use.

B. Design

This section gives brief details on the design of the framework. We begin by analysing the requirements from the point of view of the stakeholders who will use the framework. Figure 2 shows an overview of these requirements, given in the format of a standard UML use case diagram. Note the provision of requirements for service registration and discovery. Also note the role of the mobile device end-user in the diagram; this is due to the requirement that mobile devices must be identifiable on the network with a human-readable name, therefore this name must be user-specifiable.

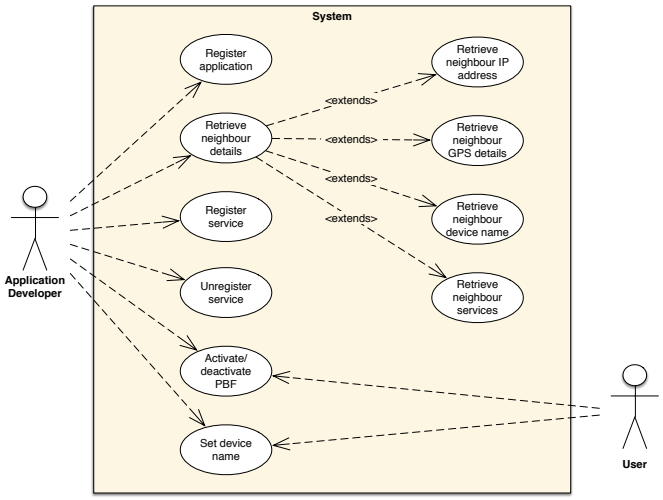

Figure 2. UML use case diagram.

Figure 3 shows a compact version of the UML class diagram for the framework. In this design, all client interaction is done via the ProximityManager class, which communicates with the background ProximityService via a number of Channel objects (one per client application). This client interaction is handled asynchronously with the use of callback listeners. Listing 1 reflects this client interaction pattern.

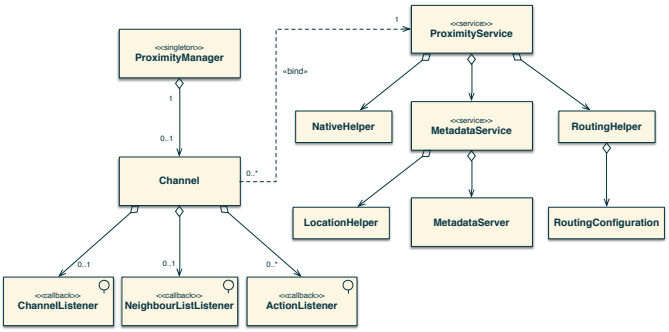

Figure 3. Proxima compact UML class diagram.

To illustrate the asynchronous client interaction pattern further, we can use UML sequence diagrams. Figure 4 is an example of such, illustrating the use case whereby a client application registers itself with the framework. Note the half-head arrow, which represents an asynchronous method call. The ProximityManager can be seen taking control of binding a Channel object for the ProximityService for the purposes of future inter-process communication. Once bound, the client callback listener is invoked and the client is thereby notified that his/her application has been successfully registered with the framework.

This interaction pattern is kept consistent throughout the entire Proxima API.

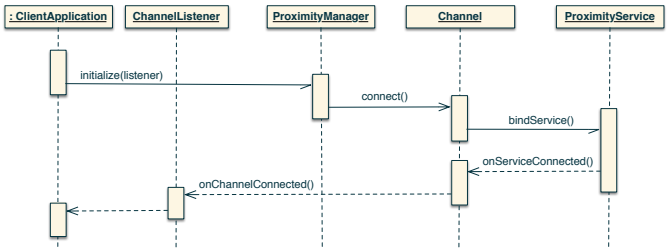

Figure 4. UML sequence diagram for the RegisterApplication use case

C. Implementation Details

As mentioned in the background section, ad-hoc Wi-Fi is used for the underlying connectivity. To enable ad-hoc Wi-Fi mode on Android, it is necessary to have root access to the device. This is because the network interface must be switched from infrastructure to ad-hoc mode. Other parameters, such as IP address and channel must also be configured in this manner. Native binaries required for this were compiled for the ARM processor and packaged with the framework. Core binaries include ifconfig, iwconfig, olsrd and dnsmasq. Utility/testing binaries include busybox and tcpdump.

1) Routing: For the routing mechanism, we chose to use the Optimized Link State Routing (OLSR) protocol [8]. It is widely tested, scalable, reliable, and is open source. It is a proactive routing protocol, as opposed to reactive or hybrid routing protocols [9]. It allows multi-hop routing, and is fully decentralised and self-healing. It also allows routing of traffic onto the Internet via gateway nodes if desirable; this is not, however, one of our key requirements. The olsrd implementation of OLSR has been used [10] , following recompilation of the code for Android.

The olsrd implementation features a plugin system. The jsoninfo plugin [10] is used to interact with the olsrd daemon. It listens on the localhost interface on port 9090 and takes a URL of the form: `http://127.0.0.1:9090/command` where `command` is the command to be given to the plugin. Many commands are available. To get a list of nearby devices, the links command is used, i.e. `http://127.0.0.1:9090/links`. Multiple commands can be supplied at once, e.g. `http://127.0.0.1:9090/links/routes`. The Jackson JSON library is used to parse the output from the olsrd daemon. A configuration file packaged with the framework specifies the plugins to be used and wireless interfaces to be used.

2) Name resolution: DNS-like name resolution was initially handled using a combination of the olsrd nameservice plugin and the dnsmasq program. The olsrd plugin modified the /etc/hosts and /etc/resolv.conf files and sent a SIGHUP signal to the dnsmasq process when there was a change in local devices. Each node was then effectively a DNS server; it would resolve its own hostname when asked, and kept a cache of hostnames for other nodes. Other such solutions exist, such as described in [11], [12], but the functionality was already in the OLSR implementation and is open-source and extendable. However, we found this solution to be too brittle. For example, if a node changed IP address, then it would take a while for each node to update its DNS cache, causing inconsistencies. Additionally, this solution only allows names composed of the small subset of characters allowed by the DNS protocol, which does not include Unicode characters or even whitespace characters. Following these discoveries, name resolution has been absorbed into the metadata service, which is explained below.

3) Service discovery: We have decided to incorporate UPnP-like service discovery into the metadata service. Service discovery could potentially be implemented using an extension to the olsrd nameservice plugin, and having additional framework API methods for registering/searching/unregistering services. However, this puts additional reliance upon the particular routing protocol, making any future changes more difficult. Other methods for service discovery in ad-hoc Wi-Fi networks are described in [13], [14], [15].

4) Geolocation: Previous approaches to GPS coordinate sharing in ad-hoc networks have tried to modify the OLSR protocol to pass GPS coordinates, such as in [16]. However, we feel that this approach is disruptive to the protocol and hence becomes less cross-compatible. Geolocation information is distributed on request via the metadata service.

5) IP autoconfiguration: DHCP-like IP autoconfiguration is currently done in a very rudimentary way. The subnet ID is simply randomly generated (i.e. 192.168.1.x where x is randomly generated from 1-254). Will need to have a collision resolution strategy. Several such strategies currently exist, such as [17]. These solutions will be explored in future iterations.

In the next subsection, we propose a solution for the name resolution, geolocation and service discovery problems. Our approach is conceptually simple; we embed a HTTP server on each device which acts as a metadata server.

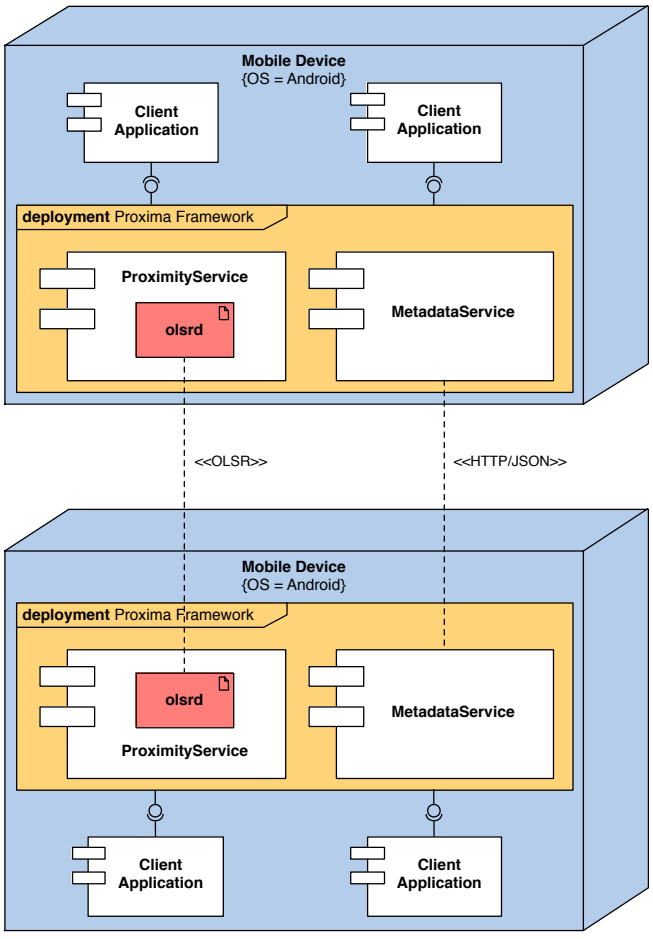

Figure 5. Proxima architectural overview.

D. The metadata service

The metadata service functions as a HTTP server and is responsible for serving information about its parent node,

such as GPS coordinates, and human-readable node identifiers. The metadata service also acts as a service discovery layer. Client applications can register their services, and can even specify custom per-application metadata to be served.

See Figure 5 for a high-level overview of the architecture of the Proxima framework, shown as a UML deployment diagram. Figure 5 shows the Proxima framework being used by multiple clients on multiple devices. It shows the communication between two metadata services, and the underlying OLSR daemon communication mechanism.

The Proxima API was written using the Android Java SDK. It is compatible with Android 1.6 (codename Donut, API level 4). It is deployed as an Android Library Project. It requires a rooted device. Development was done using the Eclipse IDE (Kepler) with the Android ADT plugin on a machine running Ubuntu 10.04 (Lucid Lynx) x86_64 with kernel version 2.6.32. The Java Process.exec() class is used to interact with the native system, to modify the network interface into ad-hoc mode, set the IP address and netmask, and to start/stop the olsrd and dnsmasq daemons. It is also used to make calls to the olsrd jsoninfo plugin.

IV. Results

We have successfully implemented a real-world application which uses all the functionality of the framework. The application demonstrates the ease of use of the Proxima framework and serves as a platform for functional testing. Development was done on the following devices:

- Samsung Galaxy S4, running Android version 4.3 (Jelly Bean)
- Samsung Galaxy Tab 10.1 running Android version 3.2 (Honeycomb)
- HTC Wildfire running Android version 2.2.3 (Froyo)

A. TuneSpy

Our sample application, named TuneSpy, allows users in proximity to one another to stream their current music track to each other. The application uses the Proxima framework to broadcast metadata about itself to surrounding neighbours. Upon request from a neighbour, the application will directly stream the actual track data. An illustration of this is given in Figure 6. The application was inspired by the initial scenario given above, and is designed to operate on a purely local, ephemeral, peer-to-peer basis.

Note that the TuneSpy application is designed for sharing music between individuals in a streaming fashion; no data is stored permanently on the listening neighbour. By using the application, users agree that sharing music via TuneSpy is not for commercial use.

B. Testing

The three development devices were placed in a triangular orientation, 30 metres apart from one another. They were then tested by continuous music streaming using the TuneSpy application. The streaming performance was uninterrupted at a distance of 30 metres. When the distance was increased to 40 metres, connectivity became variable as is nominally expected from ad-hoc Wi-Fi connections. Overall the performance of the application on top of the framework is stable and reliable at the designed operational distances.

C. Energy consumption

As with any software designed for mobile use, energy consumption must be taken into consideration. The energy consumption of the Proxima framework is equivalent to other software using OLSR routing, and scales with the number of network nodes [18]. The configuration parameters given to the OLSR routing daemon are relatively standard. From our testing work on TuneSpy, it is clear that compared with the power used for music streaming the energy consumption by the framework is negligible.

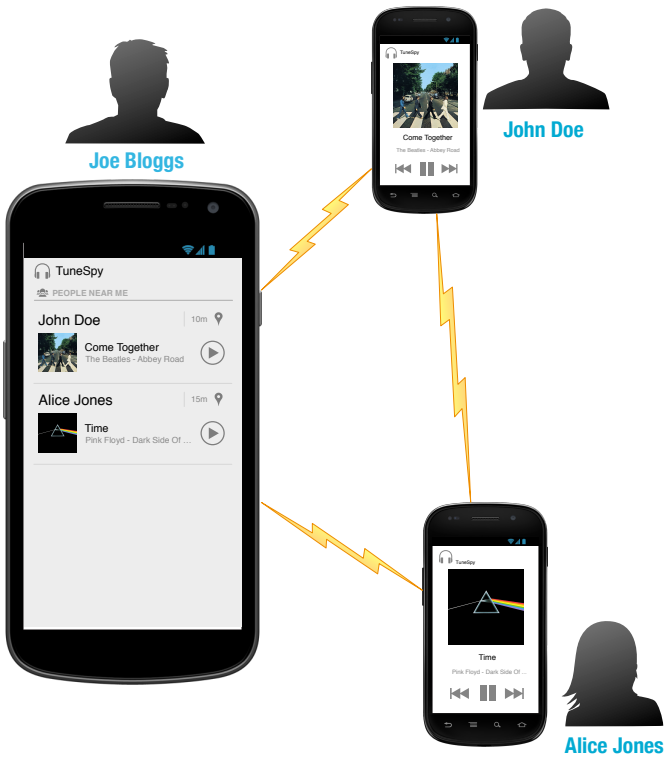

Figure 6. TuneSpy operational overview.

V. Conclusion and Future Work

A. Conclusion

We have created a useful abstraction tool for mobile peer-to-peer proximity-based networking, following our initial vision of the proximity-based paradigm. We believe that our work is a significant contribution to the collaborative mechanism and architecture. This belief is held for the following reasons.

Firstly, a framework such as this, with its unique semantics of purely-local information dissemination, could potentially allow application developers to incorporate this proximity-based functionality into their existing applications, or develop entirely new classes of unique mobile applications. Based on our current research, we have not found any other system with matching generality or functionality of the Proxima framework.

Secondly, Proxima provides a nearly zero-configuration interface. The only aspect that needs to be configured is the device/user name (similar to that of the Bluetooth device name) which can be achieved programmatically with a simple API call. This ease of configuration makes the framework both user- and developer-friendly.

Thirdly, our separation of underlying connectivity mechanism, routing functionality and higher-level services gives us flexibility to potentially change any one of these elements in the future. Since we are currently using the OLSR protocol as a routing backend, applications developed using the framework benefit from all the scalability and resilience of the protocol. The framework takes care of the transport mechanics, device specifics and configuration issues, leaving the application developer to focus solely on implementing their application.

Finally, the framework is very lightweight at only 6MB (including all necessary native binaries and compiled code), representing a small overhead for addition into applications.

B. Future work

We hope in the future to extend the framework, improving the IP autoconfiguration strategy and implementing improved security mechanisms, amongst other things.

One significant limitation is the lack of built-in support for ad-hoc mode on Android and hence the necessity for extensive modifications to gain root access to the device. It is not likely for this support to be added in the future. Thus we hope to explore alternative ad-hoc connectivity mechanisms that do not require extensive device modification but still endow us with the proximity-based semantics that we desire.

We will continue to test and support the framework on a wider variety of Android devices and larger networks. We also hope to implement a version of the framework for Apple iOS, and another version for desktop operating systems.

ACKNOWLEDGMENT

The authors would like to thank the Computer Science and Creative Technologies department at the University of the West of England for funding this research. They would additionally like to thank the anonymous reviewers of MS2014 for their encouraging and constructive feedback on this paper.

REFERENCES

[1] Foursquare, "Foursquare web page [online]," Available: https://foursquare.com/ [Accessed 17 February 2014].

[2] B. Bostanipour, B. Garbinato, and A. Holzer, "Spotcast – a communication abstraction for proximity-based mobile applications," in *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, Aug 2012, pp. 121–129.

[3] WiFi Alliance, "Wi-fi peer-to-peer (p2p) technical 7 specification [online]," Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-direct [Accessed 17 February 2014].

[4] P. Gardner-Stephen and S. Palaniswamy, "Serval mesh software-wifi multi model management," in *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*, ser. ACWR '11. New York, NY, USA: ACM, 2011, pp. 71–77. [Online]. Available: http://doi.acm.org/10.1145/2185216.2185245

[5] A. Reynolds, J. King, S. Meinrath, and T. Gideon, "The commotion wireless project," in *Proceedings of the 6th ACM Workshop on Challenged Networks*. ACM, 2011, pp. 1–2.

[6] Open Garden, "Open garden web page [online]," Available: http://opengarden.com/ [Accessed 17 February 2014].

[7] J. Thomas, J. Robble, and N. Modly, "Off grid communications with android meshing the mobile world," in *Homeland Security (HST), 2012 IEEE Conference on Technologies for*, 2012, pp. 401–405.

[8] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L. Viennot, "Optimized Link State Routing Protocol (OLSR)," 2003, network Working Group Network Working Group. [Online]. Available: http://hal.inria.fr/inria-00471712

[9] M. Campista, P. Esposito, I. Moraes, L. Costa, O. Duarte, D. Passos, C. de Albuquerque, D. Saade, and M. Rubinstein, "Routing metrics and protocols for wireless mesh networks," *Network, IEEE*, vol. 22, no. 1, pp. 6–12, Jan 2008.

[10] A. Tonnesen, T. Lopatic, H. Gredler, B. Petrovitsch, A. Kaplan, and S. Turke, "Olsrd: An adhoc wireless mesh routing daemon [online]," Available: http://olsrd.org [Accessed 14 February 2014], 2008.

[11] S. G. Hong, S. Srinivasan, and H. Schulzrinne, "Accelerating service discovery in ad-hoc zero configuration networking," in *Global Telecommunications Conference, 2007. GLOBECOM '07. IEEE*, Nov 2007, pp. 961–965.

[12] X. Hong, J. Liu, R. Smith, and Y.-Z. Lee, "Distributed naming system for mobile ad-hoc networks," *contract*, vol. 14, no. 01-C, p. 0016, 2005.

[13] X. Shao, L. H. Ngoh, T. K. Lee, T. Chai, L. Zhou, and J. Teo, "Multipath cross-layer service discovery (mcsd) for mobile ad hoc networks," in *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, 2009, pp. 408–413.

[14] N. Le Sommer and Y. Mahéo, "OLFServ: an Opportunistic and Location-Aware Forwarding Protocol for Service Delivery in Disconnected MANETs," in *Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (Ubicomm 2011)*, X. P. Services, Ed., Lisbon, Portugal, Nov. 2011, pp. 115–122. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00663455

[15] U. Aguilera and D. López-de Ipiña, "A parameter-based service discovery protocol for mobile ad-hoc networks," in *Ad-hoc, Mobile, and Wireless Networks*. Springer, 2012, pp. 274–287.

[16] W. Anbao and Z. Bin, "Realize 1-hop node localization based on olsr protocol in ad hoc networks," in *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, Dec 2012, pp. 1475–1478.

[17] C. Bernardos, M. Calderón, and H. Moustafa, "Survey of ip address autoconfiguration mechanisms for manets," *IETF Internet Draft*, October 2007.

[18] A. McCabe, A. Cullen, M. Fredin, and L. Axelsson, "A power consumption study of dsr and olsr," in *Military Communications Conference, 2005. MILCOM 2005. IEEE*, Oct 2005, pp. 1954–1960 Vol. 3.