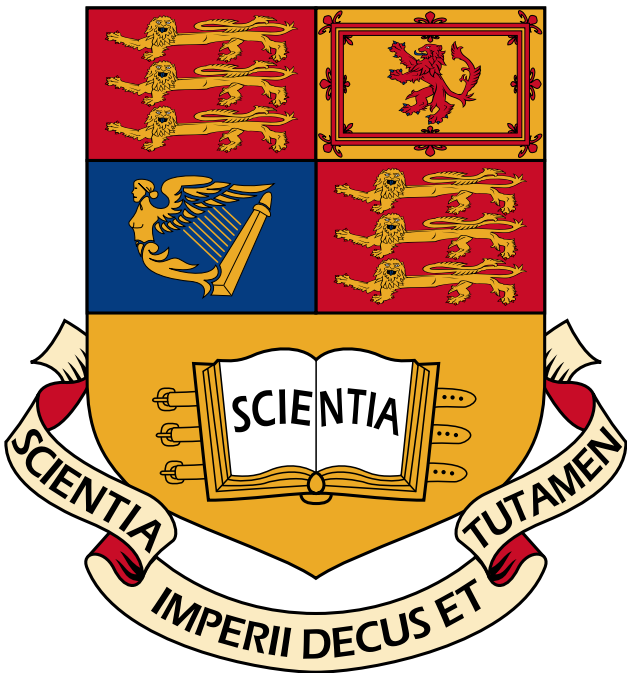
Opportunistic Networking in the London Underground

Thomas Przepiorka

Supervisor: Professor Julie McCann
Second Marker: Dr Anandha Gopalan



Department of Computing
Imperial College London
June 16, 2015

Abstract

We present an investigation into the mobile networking opportunities available on the London Underground transport network. We study the network conditions available on the Underground by developing a signal tracking application that locates users when travelling and allows everyday passengers to crowd-source data. We also present DeepOpp, a context-aware mobile system that can pre-fetch content using opportunistic signals based on measurements from SignalTracker. DeepOpp also integrates an optimized approach to determine which social contents should be cached based on signal data and phone state information. DeepOpp is implemented as an Android application and tested on the London Underground. We also built a device to device Android application which creates ad-hoc networks to disseminate content on deep-line trains on the Underground.

Travelling on the Underground consumes a significant amount of time for many commuters. Tunnels, the lack of traditional networking infrastructure, and moving trains means that it is a challenging environment to construct networks and share information. Using our innovative techniques we have explored approaches covering both sub-surface and deep-line trains. Our DeepOpp middleware offers significant improvements over previous implementations by using **2.5 times less power** for each media item retrieved.

Acknowledgements

I would like to thank Julie McCann who provided guidance and support throughout this project. She also provided me with opportunities to present my work to researchers and industry and introduced me to relevant to the research group. I'd also like to thank Dmitri Arkhipov, Lampros Lamprinos, and Di Wu who provided me with feedback and expertise. A thank you to my friends and family who were there to support me throughout.

Contents

1 Introduction

1.1 Motivation

Walk down a busy street and the wild success of social media and networks is clear. Accessing sites such as Facebook, Twitter, and LinkedIn has become a ubiquitous part of people's daily routines to manage social interactions. Media sites are increasingly offering up-to-minute bulletins as they seek to distribute information and provide live coverage of developing events. Growing mobile network coverage and speeds, combined with decreased costs mean that users are ever more likely to access this content on their mobile devices. Facebook, for example, has over a billion active mobile users and accounts for a fifth of all mobile traffic[1]. This level of interaction demonstrates how people have become accustomed to accessing content through their mobile devices and see it as a key way to receive updates and interact with people.

Coverage maps show that major cities, such as London, are fully blanketed in high speed mobile network coverage. However they neglect to account for the Underground train network that forms the backbone of the city's public transport network. London's Underground network carries millions of workers, tourists, and students everyday underneath one of the busiest global cities in the world. Each year, over two billion journeys on the Underground are completed and the number of hours spent travelling on the Underground each day is enough to complete 180 MEng Computing degrees [2]. The Underground speeds through tunnels deep below ground as well as tunnels near the surface that navigate through roads, parks, and even under rivers. These conditions make relying on existing mobile networking infrastructure extremely difficult. On sub-surface lines with intermittent signal, passengers have trouble knowing when they can use their phones. Users may constantly attempt to refresh content or simply give up trying to update app data both of which are poor user experiences. On deep line trains passengers assume they have no access to content at all.

No public data is available regarding the exact coverage and bandwidth available across the Underground. Projects such as OpenSignal[3] rely on GPS to collect, aggregate, and publish signal strength data, but the reliance on

[1] `http://qz.com/301011/facebook-is-simply-crushing-it-in-mobile/`

[2] Based on ECTS estimates (`http://www.doc.ic.ac.uk/internal/teachingsupport/ects/ects-meng.htm`) and TfL Data [12]

[3] `http://opensignal.com/`

satellites for location is not feasible on a subterranean transport network. We aim to build a system that is user friendly and can be deployed on the Underground to collect mobile network data. With a crowd-sourced app it should be possible to build a complete picture of networking opportunities at each station.

Existing social media apps pay limited attention to the different media types (text, image, and video) that can be retrieved. We can build an intelligent caching system that uses our aggregated network data to decide when to download content and which types of media can be efficiently be fetched dependent on our network situation.

A commute during rush hour or nearly anytime within Central London is likely to be uncomfortable with growing numbers of passengers crowding onto the tube every year. This density and flow of travellers can be used to provide fresh content to passengers in even the deepest tube lines. Combining principals introduced to build and manage wireless sensor networks and infrastructure-less regions, we can build a system that keeps the bustling metropolis connected where it was not previously possible.

1.2 Objectives

1.2.1 Signal Mapping

Build a mobile app that can build a map of the network conditions on the Underground. It should be able to scale to map the entire network and include relevant data.

Signal mapping and data collection will provide us with knowledge of what opportunities are available and what to priortise in the later stages of the project. This is the primary focus of Section 3.

1.2.2 Caching Middleware

Develop an application that can target usage on the Underground to pre-fetch and cache social media content, offering improved battery and data usage. The app should integrate novel optimization techniques to select what content should be downloaded.

This work will allow individual devices to have better access to information on the Underground by building on existing state-of-the-art optimizations techniques. The work should provide fresh content while saving battery,

memory, and data usage.

1.2.3 Device to Device Networking

Design and implement a protocol that can establish ad-hoc networks and use passengers to bring data into the network from the surface.

We aim to tackle the final sections of the Underground that do not even have intermittent coverage by designing and implementing a mobile ad hoc network protocol to be used by mobile phones. The implementation and protocol will need to be fast and adaptable given the constraints imposed by moving trains.

1.2.4 Real World Data

Throughout the project we will test on the Underground to provide accurate results using real hardware.

We aim to collect data in the real world which will provide us with greater confidence in the usefulness and correctness of our solution.

1.3 Contributions

- **SignalTracker** - A SignalTracker Android application designed for recording and submitting crowd-sourced data measurements. An intuitive interface that provides users with direct feedback of their recordings makes it accessible for a non-technical user base. SignalTracker was able to record over 20,000 signal readings submitted by six users from a range of backgrounds.

- **Underground Location Service** A location database and mapping scheme which does not rely on GPS. The scheme can locate a phone to station-level accuracy within seconds of entering a platform. Over 33 stations were mapped using this scheme using the SignalTracker application. The location service is used by both SignalTracker and DeepOpp.

- **Data collection** - Using controlled supervised collection and crowd-sourced data we have collected the first available mobile signal data on the Underground. In addition, both 3G bandwidth and latency readings were collected on a portion of the Underground.

- **DeepOpp** - A mobile prefetching and caching system that runs on Android, integrating optimization techniques to selectively download different types of media and content, providing savings of up to 52% on data transferred. A scheduling system, that relies on historic signal data locating passengers and their direction of transport can reduce the amount of power needed to download social media items by 2.5 times in comparison to baseline techniques.

- **MetrOpp** - A protocol is proposed that allows for a mobile ad hoc network to be created that can disseminate information between phones using WiFi Direct. An implementation application on Android has been created to demonstrate the protocol with a real use case scenario. The protocol allows phones to discover, connect and share updated state information within 6 seconds.

- **Research Paper** - Collaborated on the *DeepOpp: Context-aware Mobile Access of Social Media with Opportunistic Connectivity in the London Underground* which is to be submitted to *The 35th Annual IEEE International Conference on Computer Communications*.

- **ICRI Presentation** - Presented initial findings and methods from our Underground data collection at the Intel Collaborative Research institute (ICRI) Sustainable Connected Cities workshop in London.

1.4 Structure

The report is divided into three dedicated sections for SignalTracker, DeepOpp, and MetrOpp. Each of these is discussed in their section in full with distinct designs, implementations, evaluations, and conclusions.

2 Background

It is important to consider and understand background information on the London Underground network as it forms the motivation for the project. We will discuss some patterns and information regarding the specifics of the Underground and then proceed on to consider related works in section 2.7.

2.1 London Underground

London Underground consists of 11 different lines making up 403km of tracks [16], that covers Greater London. Seven of these lines are considered *deep-lines*, the remaining four are *Sub-surface*. Lines may travel both underground and above ground depending on where the train is. For example, the Northern line is a deep line train but has 14 of its 50 stations above ground. For lines like the Piccadilly line, the above ground sections tend to occur further away from Central London.

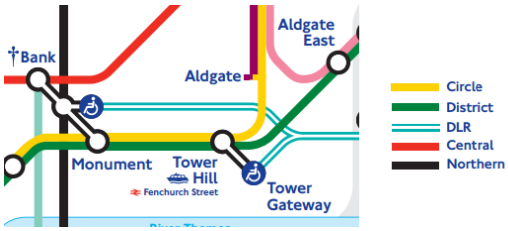


Figure 1: Tower Hill junction [14].

Lines on the tube often share tracks and platforms with certain stations offering the chance to interchange between lines. Figure 1 shows a an example of the interaction between lines and stations. The Circle line and District lines share the route from Monument station to Tower Hill station. At Monument station both lines stop on the same platform. After Tower Hill station, going East, the lines diverge with the Circle line continuing to Aldgate and the Distict line going to Aldgate East. There are several types of ways a passanger may interchange between lines:

Same Platform At Tower Hill to switch between the Circle and District lines a passenger can exit a train and wait on the same platform for the next District line train to continue their journey.

Different Platform At Bank to switch between the Northern line and the Central line, a passenger will stay in the same station but must go to a different platform. For some stations these platforms are located in different area. For example, at South Kensington station, Platforms 1 and 2 (District and Circle lines) are above ground and Platforms 3 and 4 (Piccadilly line) are deep below ground.

Pedestrian Subway The standard map shows an interchange between Monument and Bank. To make this change a passenger will go through a pedestrian underground tunnel. This is similar to changing platforms within a station, but passengers actually end up in a different station. The passenger doesn't pass through any exit gates and may not be aware that they have even moved to a new station.

Above Ground Walk A passenger at Tower Hill could change from a Circle line train to the Docklands Light Railway (DLR) by exiting the station and going above ground to a nearby station (Tower Gateway).

Passengers may depart from the tube directly to other means of transport without exiting a station above ground. This includes stations that offer connections to airports, rail, or shopping centres.

According to Transport for London (TfL) in 2009/2010 the average Londoner spent 9.3 minutes per day on the tube [13]. There are over 2 billion journeys made on the Underground network each year. TfL partnered with telecommunications provider Virgin Media to bring WiFi to the tube [15]. Currently they provide service to 150 of the 270 stations that the Underground serves. Customers of Virgin Media, EE, Vodafone, O2, and Three networks can access the WiFi at no extra cost. For passengers that are not customers of the previously mentioned networks, prices range from £2.00 for a day pass to £15.00 for a two month pass [35]. In Zone 1 all but one (Tottenham Court Road) station has activated WiFi [17], with outer zones generally having less access.

According to Virgin Media [36] 2.5 million people have registered for WiFi.

TfL publishes a number of feeds that are publicly available. There is a live feed for tube arrival time predictions, giving an estimate for when a given train will arrive on a station platform.

2.2 Mobile Networks and Signal Measurement

2.2.1 Network Structure and Standards

This project focuses on the mobile network and operators in the United Kingdom. There are four main mobile network operators in the UK: EE, Vodafone, O2, and Three. Additionally, BT Group and UK Broadband own mobile spectrum licenses. The amount of spectrum varies greatly between operators. Figure 2 shows the mobile spectrum holdings of these companies. The spectrum frequency used and the locations of base stations will affect the quality of the link.

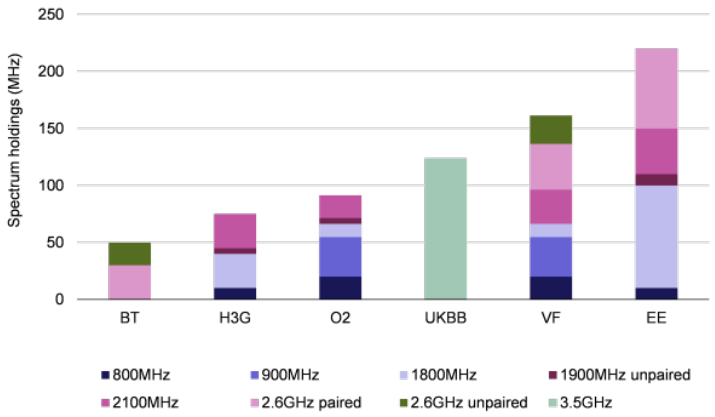


Figure 2: UK mobile spectrum holdings in MHz after the 2013 4G auction [5]

Table 1 shows the technologies that each operator utilizes in their network. The theoretical down-link speeds are the highest theoretical speeds for each technology, but variations in implementations and specification development for each technology means they may differ fbetween networks. 3G commonly refers to EDGE, UMTS, and HSDPA. 4G can refer to DC-HPSA+ and LTE. These classifications are set up the United Nations' International Telecommunications Union (ITU), but marketers and phone signal indicators may use different classifications. We can see that the type of connection we have greatly affects the possible speeds that we can attain. According to OfCom [38], in mid 2014 the average London HTTP download speeds on 4G networks were 13.1Mbps and 4.1Mbps for 3G networks. We refer to the Ofcom Research Document [38] for a detailed breakdown of mobile network

performance.

Table 1: Mobile Operators and Network Technology

Technology	**Operators** [39]	**Downlink Speed**
GPRS	EE, O2, Vodafone	0.171Mbps [21]
EDGE	EE, O2, Vodafone	0.384Mbps [21]
UMTS (W-CDMA)	EE, O2, Vodafone, Three	0.384Mbps [21]
HSDPA	EE, O2, Vodafone, Three	21Mbps [23]
HPSA+ (DC-HSPA+)	EE, O2, Vodafone, Three	168Mbps [28]
4G LTE-A	EE, O2, Vodafone, Three	300Mbps [22]

2.2.2 Measuring Cellular Data Signal Strength

Measuring signal strength is important to this project and also if techniques like Bartendr[42] are to be used. Tan, Lam, and Lau [44] provide various measurements taken under different network conditions to asses the stability and bandwidth related to measuring signal strength. Their work shows that downlink data bandwidth can vary depending on the level of network activity. At an urban ferry pier there was a difference in downlink bandwidth of only 0.67% between a *moderately busy* network and a *near idle* network. The paper indicates a great difference in network bandwidth when the network activity becomes *very busy*. Data collected at an urban shopping center shows a drop in bandwidth of 48.53% between *near idle* and *very busy* network levels. Unfortunately, the authors don't go into detail about how they categorized the network activity levels making it hard to relate back to Underground data. The research does highlight the potential issue of bandwidth changing significantly in the same location during times like rush hour when the Underground can get extremely crowded. The paper also notes that even though signal strength may be stable at a location, the bandwidth could still change by up to 54%.

The research also indicates many differences between network operators and the impact that is has on data transmission. The authors conclude that the network configurations vary substantially between operators for measurements such as TCP retransmissions and guaranteed minimum bandwidth allocations.

Spirent Communications [4], a telecommunications testing company that Ofcom uses, found that there is significant differences in measured download speeds between mobile devices running under the same conditions [9]. They attribute this difference to issues ranging "from firmware issues to antenna issues to poor design or IP configuration" [9]. Ofcom describe their methodology for measuring mobile network statistics in their research report [38]. Ofcom ensured fair readings amongst network operators by taking simultaneous readings, using identical handsets, rotating SIMs between devices, and others outlined in their report. Both research and industry experience indicates that several considerations have to be taken into account when measuring and recording mobile network data.

2.2.3 Signal Strength in Android

This section gives a brief overview of the options and APIs available in Android for retrieving signal strength and other relevant cellular data. There are two techniques that the Android API exposes to get signal strength. The first is the event handler, *onSignalStrengthsChanged*. The event handler provides a SignalStrength object exposing various methods to get different types of signal strength. We are interested in the GSM signal it provides [5] as values 0-31 and 99. These values are outlined in the ETSI (European Telecommunications Standard Institute) Technical Specification [26]. RSSI values from 0-31 correspond to range from -113dBm to -5dBm and an RSSI of 99 representing an unknown or undetectable signal.

Starting in Android API level 17 (4.2 Jelly Bean) Google introduced the *getAllCellInfo()* call in TelephonyManager which can be called on demand. This returns a list of CellInfoGsm, CellInfoCdma, CellInfoLte and CellInfoWcdma with each object corresponding to the relevant network technology (see Table 1). Each object has a method to get the signal strength which can be easily retrieved in a variety of measures such as dBm or ASU. The ASU measure conforms to the relevant ETSI specification 10.3.0 [27], and for our purposes is the same RSSI as described above with regards to conversion to dBm. While this API is documented there are several devices that do not implement it and an open bug report exists with Google regarding

[4] http://www.spirent.com/

[5] The *getCDMADbm()* refers to the CDMA standard, a GSM competitor, not to be confused with W-CDMA which can be used with GSM.

this [6].

2.3 Mobile Data Collection

Collecting data is a crucial part of understanding signal patterns on the Underground. There are several existing frameworks and tools that can be used to collect necessary data such as WiFi access points, cellular signal strength, and device information.

2.3.1 Funf

The Funf Open Sensing Framework [7] is an open source and "extensible sensing and data processing framework for mobile devices." Funf targets Android smartphones and offers a built-in set of data probes providing sensor information that is then stored and can be analyzed. The framework uses a resilient storage solution and has been deployed for long periods of time by its developers at the Massachusetts Institute of Technology (MIT). While there is no built-in probe for retrieving signal strength, the framework can be extended to accommodate this. The developers offer an off-the-shelf app called *Funf in a box* which requires very little configuration and setup to use. Funf in a Box does not provide sensing for getting mobile signal data.

2.3.2 Open Data Kit

The Open Data Kit [8] is an alternative data collection project for Android. The open source app offers a set of data forms that can be used to collect manual data or interact with the phone sensors. No pre-built forms are able to capture the cellular signal data we require, but it would be possible to create a new one. To automate the collection process the source code would likely need to be modified. Funf is targeted more at the automated sensor collection we require and includes a more comprehensive pipeline allowing easier data analysis.

[6] `https://code.google.com/p/android/issues/detail?id=60430`
[7] `http://www.funf.org`
[8] `https://opendatakit.org`

2.3.3 Spirent Datum

The Spirent Datum [9] tool was used by Ofcom in their 2014 testing of mobile broadband [38]. The tool is available for free download on all three major mobile app stores. Its comprehensive tests cover various use cases such as downloading data while in a call, web page load times, and HTTP speeds for various size files. The app with default settings takes over a minute to conduct a test. The tool is likely too slow to give us the granular data that we need whilst on a moving train. It is also limited to collecting either just WiFi or just mobile network signal, but we want to scan for multiple data at the same time. The tool may be useful in helping measure the accuracy of other sensing apps, like Funf. If we require stationary measurements (for example at platforms) then we could use the Datum app - but we must be cautious to calibrate the readings to match ones from Funf.

2.3.4 LiveLab

While the previous two data collection solutions are based on the Android mobile OS, LiveLab [43] runs on iOS. The application is also able to collect data from more sources than just physical sensors or user input. For example, it can collect detailed usage information for a user's apps such as the number of launches and time spent in a given app or app category. This could be particularly useful for understanding passenger app behaviours when on the Underground. The authors seem to limit their reach by writing the code for iOS and requiring users to 'jailbreak' their devices in order to gain root access on the phone required for the data collection. While the project website lists a section for the source code, it doesn't appear to be currently available. The data they collected showed that Facebook was one of the most popular communication apps used by the study participants.

2.3.5 Wireless Sensor Networks and Data Collection

Adeel, Yang, McCann[1] propose a sensing network that can collect under both real-time and delay tolerant scenarios. In their Mobile Urban Sensing System (MUSS) phones can either submit collected data directly or though multiple other phones communicating using short range radios. With large amounts of data it is not always feasible to rely on cellular networks to transport this data. Instead the authors capture on the increasing prevalence of short range communications available on mobile phones in order to efficiently

communicate these large data packets. Their work outlines a neighbour discovery scheme as well as techniques for the MUSS to self organise and handle heterogeneous data types with differing characteristics and restrictions. Another approach by the same authors in [50] improves routing by using social awareness to optimize the use of having people transport and offload sensing data in an urban environment.

CrowdWiFi [49] proposed by Wu et al. is specifically targeted at mapping WiFi access points (APs) for vehicles using crowdsourcing. Their system builds roadside AP distribution estimates to help travelling vehicles intelligently manage their connectives based on AP profiles and location. CrowdWiFi operates using two components on the phone and server which help assemble and aggregate information to build these estimates. As vehicles are often fast moving it relies on Compressive Sampling [8] to build a more accurate measure of signal strength of WiFi APs. CrowdWiFi can take significantly less signal readings to build an accurate measurement and provide improvements of over 80% compared to existing technologies.

2.3.6 Incentivisation in Data Collection and Relay

When any agent is required to perform a task or sacrifice resources it is important to consider their motivation and ways to incentivise them to participate in large scale data collection or distributed network. MUSS[1] provides an economic framework allowing each agent to set a *data selling price* which can be translated into a monetary value using a server set parameter. By using their neighbour discovery scheme each node can calculate the neighbour that will generate the most profit when forwarding data to be offloaded. In [50] an economic network for data production and transport is used allowing trading between people who act as carriers of sensing data.

2.4 Device to Device Communication

2.4.1 Wireless Communication Standards

As part of the project we will be interested in device to device communication between Android phones and so provide some background and research into the offerings available. The key characteristics when we make a decision about which wireless standard to use will depend on the range, bandwidth, energy consumption, as well as support for the standard. We look at each of

these aspects for different standards.

2.4.2 Bluetooth

Bluetooth was originally created to replace data cables and enable short range transfers between devices. It operates between 2.4 to 2.485 GHz on unlicensed spectrum. Several classes of Bluetooth radios exist that can provide differing ranges, with class 2 radios found in most phones providing a range of 10 metres. The advertised throughput of Bluetooth is 1-3Mbps using 100mW of power. Bluetooth has been part of the Android OS since its first release and the recommended API libraries have been included since 2009 [9]. Bluetooth supports service discovery so that devices can find peers that specifically work with a given service. Bluetooth supports multiple different protocols for various tasks (service discovery being one of them). Radio frequency communication (RFCOMM) provides a data stream similar to TCP for transferring data and also supports several other protocols that can accomplish similar tasks such as File Transfer Protocol (FTP) and Object Push Profile (OPP).

2.4.3 Bluetooth Low Energy

Bluetooth Low Energy (or Bluetooth Smart) is a power friendly version of Bluetooth targeted towards devices running with very small batteries for long periods of time. It aims to match the ranges of Bluetooth, but with considerably less power usage. It has a maximum application layer throughput of 236.7 kbps[20], although actual transfer rates can be much lower than this. Gomez et al. found in experiments that their throughput rate was only 58.48 kbps.

2.4.4 WiFi Ad-hoc

WiFi provides an ad-hoc mode which can create networks without infrastructure. Ad-hoc mode is able to chain up connections and doesn't rely on a central group owner or access point meaning that the range of the network can be quite large and there is much less reliance on the presence or location of a single node. It is also quite dynamic to topology changes in a network

[9] `http://developer.android.com/reference/android/bluetooth/package-summary.html`

and offers true peer to peer networking. WiFi ad-hoc mode has no explicit requirements for security either giving the developer more flexibility with how they want to authenticate users and control access to data. It possible to create an ad-hoc network without any encryption at the networking layer and instead rely on higher layers to provide the data encryption mechanism. This mode is not supported by Android phones, unless they have been rooted. This appears to be a deliberate omission by Google as Thinktube.com[10] have attempted to merge support for WiFi Ad-hoc into the Android Open Source Project (AOSP), but had their patch rejected. They have included support for this mode in CyanogenMod, an open-source mobile OS based on Android. It doesn't appear that this mode will be available in any versions of Android and so will not have widespread support.

2.4.5 WiFi Direct

WiFi Direct is a new specification designed for device to device communication built on top of WiFi and managed by the Wi-Fi alliance[11]. Devices proceed through two stages in forming groups - device discovery and group formation. In device discovery devices scan through multiple WiFi channels to find existing clients. There are multiple procedures for group formation, which can occur unilaterally by a single node or through a negotiation when forming a group with a neighbouring node. In [10] Conti et. al measure the times for group discovery and group formation, displayed in Figure 3. We can see that under autonomous unilateral group creation the process finishes within five seconds and can take longer for the other two processes. The worst case is a time of 27 seconds to complete the process.

WiFi Direct does rely on the Group Owner (GO) to be present and will require new group formation if that GO drops from the network and in this sense is not a true peer to peer network system. It also means that all devices must be within range of the GO. The range of WiFi direct can be up to 200 metres[12], but most devices support a signal range of 100 metres. WiFi Direct also supports service discovery. Throughput speeds can be up to the regular WiFi speeds of 250Mbps.

[10] `http://www.thinktube.com/component/content/article/19-technicalinformation/46-android-wifi-ibss`

[11] `http://www.wi-fi.org/discover-wi-fi/wi-fi-direct`

[12] `http://www.wi-fi.org/knowledge-center/faq/how-far-does-a-wi-fi-direct-connection-travel`

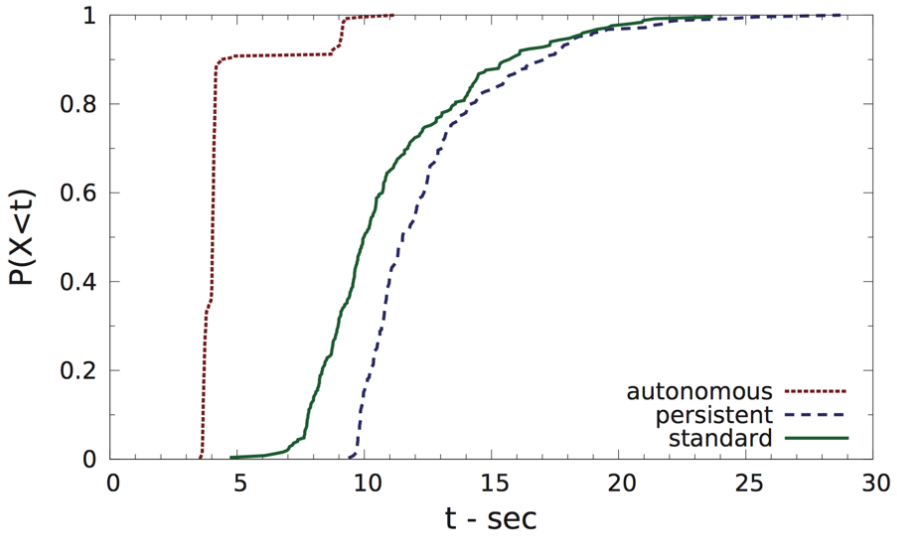


Figure 3: CDF Discovery and Group Formation[10].

2.4.6 WLAN-Opp

WiFi-Opp is a proposed standard put forth by Trifunovic et. al in [45] and implemented as WLAN-Opp on Android in [46] to tackle the shortcomings of some of the aforementioned standards. The authors cite WiFi ad-hoc's lack of support on mobiles and Bluetooth's short range and pairing process as being too limited for real world mobile opportunistic networks. The initial system design was published in September 2011 and Google only released the first version of Android supporting WiFi Direct in October of the same year [13], although the authors believed it would not deliver the desired feature set. WLAN-Opp can either connect devices without existing infrastructure or through open access points. The authors detail how phones can connect to open WiFi access points and use this infrastructure to send peer to peer messages. They claim this approach can also work on some paywall access points which still allow the node to node communications as long as they do not make any external requests to the internet. This is an innovative approach to save on battery life by not requiring any of the participating

[13]urlhttp://developer.android.com/about/versions/android-4.0-highlights.html#UserFeatures

phones to act as an access-point which is a power intensive operation. In order for phones to join the same open network they will cycle through and attempt to connect to available networks.

If no access points are available then the phones can communicate through a single phone that acts as an access point. If no available networks are found then a phone will create an access point with a probability of p^{AP}. Access points are created by using hidden Android APIs and enabling the internet tethering features available on most phones. p^{AP} is dependent on the number of recent neighbour nodes CNR encountered which is defined as

$$C(N_r) = \begin{cases} N_r & \text{if } N_r > 0 \\ 2 & \text{otherwise.} \end{cases} \tag{1}$$

p^{AP} is then set to

$$p_{on}^{AP}\left(t_{off}^{AP}, N_r\right) = \begin{cases} \frac{1}{C(N_r)} & \text{if } t_{off}^{AP} > t_{on,min}^{AP} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

where t^{AP}_{off} $t^{AP}_{on,min}$ determines whether sufficient time has passed since the device was last in access point mode.

The authors claim that this is the ideal trade-off between finding the next available access point quickly and not creating two disjoint networks. Trifunovic et al. presents a compelling new system that can indeed improve upon many of the limitations in existing wireless communication standards. The authors were correct that WiFi Direct does not provide a comprehensive solution and still suffers from cumbersome pairing processes. Their solution is impressive and novel, but does rely on using hidden APIs which are undocumented and may change at anytime without notice. They leave some minor UI problems as WLAN-Opp is making use of tethering features to implement WiFi access points on Android.

Both Kalic, Bojic, Kusek [29] and Friedman, Kogan, Krivolapov [18] have published results comparing energy consumptions and throughput of WiFi and Bluetooth communications on Android phones. Friedman et al. finds that WiFi communication is much more vulnerable to the interference created by other WiFi transmissions than Bluetooth with respect to the interference created by Bluetooth transmissions. The authors note that the specific devices used were able to handle interference differently. Friedman et al.'s research shows that sending a file through Bluetooth was about 2.92 times

more energy efficient than sending it over WiFi. Work in [29] shows this ratio to be a more modest 1.32. According to Kalic et al. for larger file downloads WiFi becomes more efficient per byte transferred. The throughput ratio for WiFi over Bluetooth was found to be 2.78 in [29].

These results are going to differ based on the exact devices used and software installed. We can see that in the differing values obtained between the two papers and it is important to note that while both papers were only published two and three years ago, Android has gone through multiple new OS releases since then.

2.5 Routing Protocols

Routing protocols for delay tolerant networks aim to transport and route data throughout a network from a source to a destination. Routing may involve unicast messages (addressed from one node to another arbitrary node), multicast (one to many), broadcast (one to all) and others. We will look at some of the routing protocols that are important and relevant to delay tolerant networks (DTNs) and Mobile Ad Hoc Networks (MANETs) and some of the specific schemes set up using these protocols.

2.5.1 Epidemic and Gossip Routing

Epidemic routing uses "random pair-wise exchanges of messages among mobile hosts [to] ensure eventual message delivery"[48]. The goals are to maximize delivery rate and minimize latency and power consumption. Each node communicates with nodes that it intermittently comes into contact with. To prevent duplications, the nodes maintain a 'summary vector' which is a hash table indexed by a unique identifier for the messages it is holding. When nodes come in contact with each other they exchange summary tables. Then each node requests copies of the messages it has not seen. A hop count limits the maximum number of times a message will be copied to reach its destination. Optional acks provide confirmation on delivery. The authors of [48] used the Monarch simulator to implement this protocol and found that for radio ranges of 25m and over there was a 100% delivery rate.

PRoPHET [34] and MaxProp [7] both seek to improve on the efficiency of epidemic routing by only eliminating redundant message transfer. PRoPHET uses non-randomness of real world movement to generate probabilities that passing on a message will increase the chance of it being delivered. MaxProp

does copy all messages like in epidemic routing, but uses an ordered queue to manage which messages get copied first and which ones get dropped first.

Allavana, Demers, and Hopcroft present a gossip protocol in [2]. In their protocol nodes maintain lists of views which contain subsets of the overall group membership. In each round the two lists are updated with nodes picked at random based on a reinforcement weighting factor.

Epidemic and gossip protocols are simple and can easily propagate information. Their shortcomings are related to their power consumption and efficiency in sending redundant information in the hope that it will reach its destination. In an information dissemination scenario this is not such a problem.

2.5.2 Trickle

In [33] Levis, Patel, Culler, and Shenker propose Trickle which is a system to provide code updates in wireless sensor networks. Dealing with the power constraints imposed by these networks they seek to minimize maintenance while allowing rapid propagation and scalability. A node in Trickle will periodically transmit metadata if it has not heard other motes send this same information. An example is if node A has code version ϕ_i and node B has version ϕ_{i+1}. It does not matter which of these two nodes transmits their version first. If A transmits first then B will know that A needs a newer version of the code. If B transmits first then A will realize a newer version is available. The protocol has been used as part of the Deluge protocol and on Berkeley motes.

Trickle is specifically designed for information dissemination and updating nodes to their most recent versions. It keeps messages to a minimum by only broadcasting periodically and remaining silent if known nodes are on the same version of metadata as they are. This is an efficient system for managing such data and could potentially be used in other contexts outside of low power motes in wireless sensor networks.

2.6 Network Simulators

2.6.1 NS-3 and NS-2

Ns-3 is an open source discrete-event network simulator. It allows a variety of routing algorithms and models to be run on top of it and is specifically

designed to be open and extensible to target the research community. It supports a real time simulator and can be integrated with existing networking infrastructure to provide simulations based off WiFi IP and non-IP protocols. According to [31] the NS-2 simulator is the most popular simulator used in papers submitted to the ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobi-Hoc) conference.

2.6.2 The ONE Simulator

The Opportunistic Network Environment (ONE) simulator[30] is designed to test and simulate delay tolerant networks. The main features of ONE are:

- generating node movement
- routing messages between nodes using DTN routing algorithms
- visualizing these simulations in a graphical user interface.

The simulator is Java based and specifically tailored towards testing delay tolerant networks. Different scenarios are setup that can test the performance of opportunistic networks depending on node mobility, density, routing algorithms, range of transmission technology among others.

The simulator's website[14] hosts the source code of the project which allows developers to extend the project to include new message creation types and routing algorithms. The compiled project is run alongside a configuration file that contains all of the parameters for the simulation. We will not go into detail about the parameters here, but more information can be found in the README and by browsing the javadocs for the simulator.

2.6.3 Accuracy of Simulators

Kurkowski, Camp, and Colagrosso [31] study the accuracy, believability, and repeatability of simulators used in papers submitted to ACM's Mobi-Hoc conference. They find that less than 15% of the published MobiHoc papers are repeatable and only approximately 12% of the MobiHoc simulation results appear to be based on sound statistical techniques. They outline several pitfalls that researchers succumb to when they run simulations. These include mistakes such as:

[14] `http://www.netlab.tkk.fi/tutkimus/dtn/theone/`

- Not changing the pseudo random number generator (PRNG) seed between trials.
- Relying on a single set of source data or not stating the number of simulations run.
- Not discussing statistical analysis methods or providing confidence intervals.

The paper surveyed papers submitted between 2000-2005, so hopefully the more current results are more accurate and repeatable.

2.7 Related Works

2.7.1 Location Aware Networking

There are several projects and research that has gone into using location and previous signal strength history to predict future movements and coordinate networking activities around these predictions. BreadCrumbs [42] uses a second-order Markov model to predict future movements. Locations are grouped into discrete areas and predictions for future movement can be made for varying times in the future. Their results are promising and provide better than 70% accuracy for looking one step in the future. By allowing for location prediction mistakes where the network conditions between the prediction and actual future location are similar the accuracy is 90% for one step predictions and 80% for predictions six steps in the future. The paper's methods were specifically designed to take into account the low energy and computing requirements that mobile computing imposes. BreadCrumbs uses Place Labs [32] to predict locations. Place Labs is able to estimate locations even when direct GPS measurements are not available. It does this by mapping fixed beacon (WiFi access points, bluetooth connections, and cell base stations) locations to the beacon's unique identifier. This technique is tested and proved quite accurate in indoor locations, providing accuracy to 20.5 metres in an urban setting. Deshpande, Kashyap, and Das use a similar technique when attempting to predict availability of future WiFi access points in moving vehicles [11]. Their technique generates an *RF fingerprint* which consists of even more information than Place Labs uses. The RF fingerprint includes the signal to noise ratio, MAC address, SSIDs, access point security information, and network layer data such as the default gateway and

DNS server. The authors use some of this additional information to provide a more detailed picture of what access points a device may be able to connect to.

Place Labs uses the wigle.net [15] database to obtain additional WiFi to location mappings. The service offers over 170,000,000 WiFi network mappings. The service also offers an API and access to the data. An Android application is also available that allows automatic mapping and uploading. Unfortunately this app must rely on the GPS coordinates provided by the phone's OS in order to get the location. This will prove problematic on the Underground when no GPS signal is available.

Bartendr [42] builds upon some of the previous work from the BreadCrumbs and Place Labs. Crucially the authors find that "pattern of variation in signal strength is quite stable when location is coupled with direction of travel." By using prior tracks of signal data Bartendr is able to "predict signal strength 800 s in the future without a significant increase in error." By introducing the idea of measuring and predicting based off of signal tracks we can get an accurate estimate of what our signal strength will be over ten minutes in the future. This recording and measuring method is highly suitable to the Underground network where we have a well defined track we are measuring against.

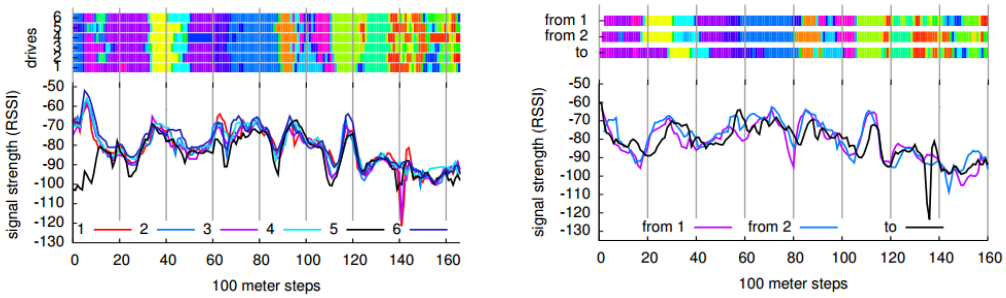
The authors of the Bartendr paper also measure the difference in signal strength when traveling in one direction and when traveling in opposite directions. Their experiment is conducted by traveling on a road for a length of 17km. Figure 4 shows the differences which the authors conclude that "it is clear that there is less correlation when traveling in opposite directions" [42]. This will be particularly relevant to take into account when measuring signal strengths when traveling in opposite directions on the tube.

2.7.2 Contextual Information in Opportunistic Networking

We can incorporate additional contextual information to help better determine how we will manage inter-device communication. Boldrini, Conti, Delmastro, Passarella present a context and social aware middleware for opportunistic networks [6]. They define three types of contexts which are summarized as:

User Context The name, location, timetable, and other personal informa-

[15] `https://www.wigle.net`



(a) Signal Strength over a number of drives in the same direction (Schulman et al., 2010)

(b) Signal Strength over a number of drives in different directions (Schulman et al., 2010

Figure 4: Differences in measuring signal strength depending on the direction of travel

tion of the user. We can use this to build a picture of the user's social network.

Service Context The interests of the user. Could be obtained by manual input or from the user's social network profiles.

Device Context Sensor information, battery profile, and other device data that can be used to help manage resources.

The service context is used to determine the interests that users will have in particular pieces of content. For our purposes this could be more easily related to the benefit function and utility value described by O^2SM[51]. O^2SM describes a concrete algorithm and process for obtaining a numerical value to represent this which can be used and is especially powerful as it more closely relates to our problem of providing efficient access to a user's personal social media news feed.

The goal of using the context and socially aware middleware is to "exploit traveler nodes to establish opportunistic communications between separate communities, making all the users able to share and disseminate contents even with those users that they will never get in touch with" [6]. We can expand the definition of a community to include the general internet as a community. Traveler nodes will be people entering and leaving tube carriages as they offer the opportunity to transfer data between different unconnected communities (isolated train carriage members and the wider internet). The

paper also introduces the concept of the weighted value for how willing a participant is to cooperate with neighbour nodes. This is assumed to be 1.0 for a node and itself. We need to be able to balance a participant's own needs, but also help spread inter-community information.

2.7.3 Smart Caching

Smart caching techniques allow for data to be retrieved at optimal times.Prefetching data allows data access when no connectivity is available and can help reduce energy consumption by only accessing data at times of high signal strength. O^2SM outlines a system architecture for an Android app called oFacebook. The authors introduce motivation for providing access to social media on mobile devices with 55% of smartphone users access Facebook on their phones. Their paper focuses on a general approach to providing offline access to social media and identifies three main motivations for this: sporadic network availability, bandwidth limitations, and high access costs. In London, the focus of this project, these concerns are not particularly relevant. Coverage of 3G and 4G is comprehensive across most of London. Figure 5 shows mobile network operator EE's 4G coverage map in London with 4G coverage in turquoise [16].

Coverage is very comprehensive, although the map is not perfect and indoor conditions will vary.

One of the author's main contributions is to rank content based on the probability that a user will view and enjoy it. Facebook continually invests in their algorithms to "show the right content to the right people at the right time" [4]. Twitter has begun a similar rollout of a ranking feature that displays posts in a ranked order rather than just chronologically [41]. We can rely on Facebook's ranking (which is exposed through its API) to get this ordered set of posts rather than implement it ourselves.

The system design involves a middleware that runs as a service on Android shown in Figure 6. This middleware handles fetching content, profiling, and content ranking. Individual apps then rely on the middleware to provide them with the content they can display. This approach allows for a common middleware platform for multiple social media services and keeps the main ranking logic separate from the individual social app. An alternative to this that is not discussed in the paper is to host the middleware on a server. The

[16] `http://ee.co.uk/ee-and-me/network/4gee/coverage-checker`

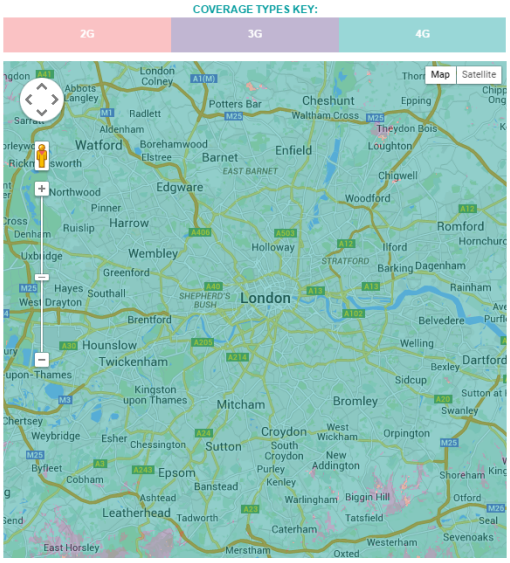


Figure 5: Mobile operator EE's 4G coverage map for London according to their website.

app could interact with this server directly. The server could allow for more computing power, apply compression, be easily adaptable to multiple mobile platforms (iOS, Windows Phone), and more easily allow synchronization across multiple devices. Using a server for this would also introduce privacy concerns of storing user data, require transmitting all profiling data, and introduce the cost and maintenance of the server. As a follow-up to O^2SM, [52] proposes using a broker/proxy system architecture to achieve similar goals. Their prime motivation was that resources are wasted to check if there are updates when no new content has been processed, and in some cases also requires retrieving multiple data items and analyzing them. Their work using the broker/proxy infrastructure offered improvements of up to 9.1 times when on cellular data. The update service was intended to run throughout the day, something not necessary when focusing on just tube journeys. Limiting the scope will hopefully allow us to reduce extraneous update checks.

In order to pre-fetch data Zhao et al. propose a scheduling algorithm targeted specifically towards mobile devices. There is a lot of research regarding caching social media data at Content Delivery Network sites, but these do not take into account the energy and processing constraints of cell phones. The scheduling algorithm is shown in Figure 7. We will briefly

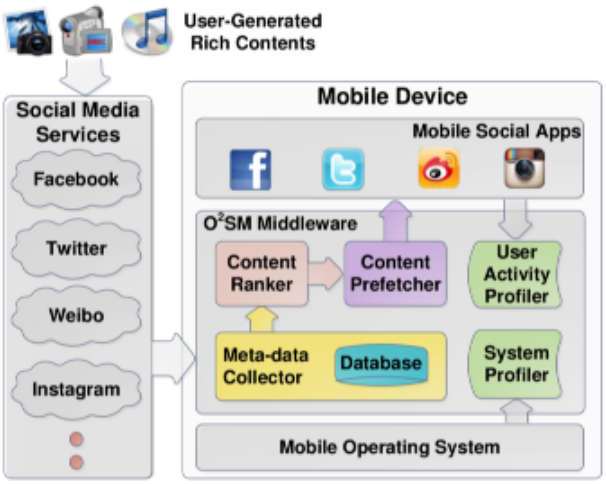


Figure 6: Middleware architecture proposed by Zhao et al. for a social media stream of data on a mobile device. [51]

cover the Cost/Benefit and O^{2}SMPS techniques referenced in the scheduling algorithm.

Step 1-Read Unviewed Ranked Contents:
read the list of unviewed ranked contents output from the Content Ranking component.
Step 2-K Slot-ahead Forecast:
forecast the network conditions and user context for the next K slots.
Step 3-Offline Online Social Media Prefetch Scheduling (O^{2}SMPS):
formulate an O^{2}SMPS problem using a cost/benefit analysis to decide which contents to download in each of the next K slots.
Step 4-Contents Download:
sequentially download contents that are scheduled for the current slot.

Figure 7: Pre-fetching schedule proposed by Zhao et al. for a social media stream of data on a mobile device. [51]

The cost/benefit model builds up a value $C(i, t_pre)$ which is the cost of pre-fetching content i at slot t_pre. This cost is a function of the weigted energy and data costs to download the content i. A benefit value $C(i, t_pre)$ is the benefit to pre-fetch content i if the user navigates to the social content stream in time period t_nav. This benefit function considers the energy benefit which is modeled on models from PowerTutor [17]. A similar benefit value for

[17] `http://ziyang.eecs.umich.edu/projects/powertutor/`

cell data is factored into the benefit as is the viewing performance benefit.

The O^{2}SMPS problem aims to maximise the total gain (benefits minus costs) and develop a schedule that achieves this. As presented in the paper we build up a two dimensional matrix representing whether we will download content i at time k. The total benefit and costs equations for a given matrix Z are shown in Figure 8. This can be represented as a Generalized Assignment Problem.

$$\begin{aligned} Benefit(Z) = & \, p_{nav}(1) \times \textstyle\sum_{i=1}^{|I|} (B(i,1) \cdot y_i(1)) \\ & + (1 - p_{nav}(1)) p_{nav}(2) \times \textstyle\sum_{i=1}^{|I|} (B(i,2) \cdot y_i(2)) \\ & + \cdots + \textstyle\prod_{k=1}^{K-1} (1 - p_{nav}(k)) p_{nav}(K) \times \sum_{i=1}^{|I|} (B(i,K) \cdot y_i(K)), \end{aligned}$$

(a) Benefit function of a scheduling matrix Z (Zhao et al.)

$$Cost(Z) = \sum_{i=1}^{|I|} \sum_{k=1}^{K} C(i,k) \cdot z_{i,k}.$$

(b) Cost function of a scheduling matrix Z (Zhao et al.)

Figure 8: Cost and Benefit functions of the O^{2}SMPS problem.

2.7.4 Data Offloading

Data offloading allows data intended for one network to reach its destiation by passing through others. Data offloading can take many forms, the MADNet[24] architecture allows mobile users to transfer data between each user's phones and then when one of the phones has connection to a WiFi hotspot or cellular data it can transmit messages on as part of the wider network. Their data indicates that constantly scanning for WiFi access points can bring the battery life of a phone down from 300 hours to only five hours. The authors measure two variables that help reduce energy consumption: scanning less frequently and scanning in areas with few available WiFi access points.

Helgason, Yavuz, and Kouyoumdjieva propose a middleware architecture for a mobile peer-to-peer content distribution system[25]. The structure of contents is based on a publish/subscribe model where individual nodes in the network can choose different data feeds to follow and communicate with nearby nodes relaying information related to these data feeds. Users who are

on the edge of the peer-to-peer network and are able to communicate with the internet act as gateways to the ad-hoc mobile network. The system is best-effort based with no guarantees on the order of delivery. To transfer larger amounts of data the authors suggest using *chunks* which allow incomplete data to be easily transferred. Their recommended size for these chunks is 16kB. Like in MADNet, battery life is significantly reduced from about 20 hours to five. The authors postulate that 802.11 wireless draws much more energy than Bluetooth, but Bluetooth has limitations such as a "long and inefficient discovery process" [25].

In an opportunistic network nodes may not always have a direct link to their destination and node location and state will be dynamic. In a tube carriage we can see that this will be the case as people and their phones enter and exit trains at each platform. As the network is constantly changing global knowledge of the network topology is not necessarily straightforward, but can greatly help in making routing decisions. PeopleRank [37] is a novel approach proposed that relies on user's social graph interactions. Unlike network topology which is dynamic and would be changing every few minutes on an underground car, social information is more stable and so can provide more efficient routing decisions. PeopleRank's approach is based off the PageRank technique used to rank web pages. The algorithm relies on existing social network information that is available on sites like Facebook or LinkedIn. A PeopleRank value is calculated as shown in (3) taken from the PeopleRank paper where $N_1, N_2, ..., N_n$ are the nodes, $F(N_i)$ is the set of neighbours that links to N_i, and d is is the damping factor which is defined as the probability, at any encounter, that the social relation between the nodes helps to improve the rank of these nodes [37].

$$PeR(N_i) = (1 - d) + d \sum_{N_j \in F(N_i)} \frac{PeR(N_j)}{|F(N_j)|} \tag{3}$$

One of their datasets and tests rely on building relationships based on shared common interests from Facebook profiles. At a delay time of 10 minutes the decentralized PeopleRank algorithm has a normalized success rate of over 86% compared to only 58% achieved by a degree-based algorithm. This method must rely on users opening up their social accounts to the routing infrastructure, which is not a limiting factor given we will be developing specifically for a social network application. It also does not deal with nodes which don't have any social profiles or presence. Given the diverse set of

people and number of users on a public transport system such as the London Underground it is not likely there will be significant direct social relations between passengers, but common information such as shared interests or place of work could be valuable in helping to determine routing decisions.

Their results across a number of datasets suggest an optimal dampening factor of about 0.8.

Zhuo et al. outline a model for providing incentives for users on a saturated network to rank and prioritize specific data to be offloaded. The incentives in the paper comes from network operators providing discounts for users who are willing to allow a greater delay in their data being transmitted. The TfL could also be a potential source of incentives for users to offload data. The paper details an auction and bid algorithm allowing the incentives to be distributed and attempts to accomplish an efficient allocation of limited bandwidth between users who are waiting to offload data. The algorithms also make use of the fact that not all users have the same bandwidth available (dependent on mobile devices and network operators) so we should try make use of those devices that have more bandwidth available to them.

3 Part I - Measuring Networking Characteristics on the Underground

We aim to develop a data collection system that can measure mobile network characteristics on London Underground train carriages. Before beginning to design methods of improving data access we need objective measurements and an understanding of what opportunities we have for building these improvements. There is little public data about mobile phone network characteristics on the Underground.

3.1 Design

A mobile phone based solution was chosen for the data collector. Desktop solutions running on a traditional operating system allow for full network analysis programs to be run[18], but mobile phones provide accurate and realistic measurements. Professional equipment may have also been an option, but by using commodity hardware we allow for data to be crowd sourced and incorporated into subsequent portions of this project. The available measurement frameworks and tools are presented in Section 2.3. The Funf framework was chosen due to its existing probes, open source code, and customizability. Rather than writing a measurement application from the ground up, Funf provided mechanisms for data persistence and backup.

The specific requirements for the app to be successful are:

- Collect data for: Mobile signal strength, WiFi access points, and mobile network bandwidth.
- Persist data so it can be easily retrieved from devices and analysed.
- Continue to work in the face of unreliable and constantly changing network conditions.
- Run at a high enough frequency to provide insights for sections that are covered by a rapidly moving train.
- Run in the background when the app is not directly open.

[18]Wireshark - https://www.wireshark.org/ and Google Resource Timing API - https://developer.chrome.com/devtools/docs/network among others

- Consume a reasonable amount of battery so that multiple iterations of data could be collected without having to charge a phone.
- A GUI to allow monitoring and operation by non-technical smartphone users.

The Funf framework provides some of the basis of these requirements, specifically:

- Existing code design to allow new data probes to be added and integrated into the application workflow.
- Data persistence and resilience. Funf includes the ability to post results to a simple server.
- A scheduling system that relies on some of Android's timing services to configure and run probes at given intervals.
- Background service setup.
- An extremely basic GUI giving users the ability to toggle data collection, run scans on demand, and persist data to an SD card.

To meet our goals we would need to extend the application to provide the following missing functionality:

- Probes for mobile signal strength and bandwidth that work in unreliable conditions on the Underground.
- Build a useable GUI providing more information and control.

We need a way of mapping the results of the readings to the location on the tube network. Getting an accurate GPS signal did not work when tested, as much of the journey is in a tunnel or on a platform below ground. Actual results were either delayed, inaccurate, or non-existent using location frameworks. Train timings could be used to estimate where we were on the route and relate the readings back to the scheduled location. Like any transport system, the timings would not be accurate enough to give locations as trains often stop for different amounts of times and are held on platforms and in tunnels. A difference of a minute could mean assigning high signal

strength to a tunnel instead of the platform it was actually recorded. Taking enough samples could help minimize these drifts, but we still want to achieve a higher degree of location accuracy.

We chose an approach that relies on the available WiFi access points installed at nearly all of the Central London stations. Each access point contains a unique MAC address, which can be mapped to the station. This method assumes that the physical access points are stationary and aren't moved around frequently. These mappings would give us station/tunnel level accuracy of our data. While there are many services that provide WiFi to location mappings, none of them provided coverage on the Underground with mappings to stations. Measuring bandwidth also required a new probe. The solution chosen was to begin the download of a file and measure the progress that was made and amount of data transferred. This would mean that even if the download took a large amount of time or needed to be reset we could still obtain measurements.

3.2 Implementation

The probes in the Funf framework are built on top of the provided Probe.Base class which allows the probe to fit into the lifecycle shown in Figure 9 [19]. Each probe begins disabled and then enters an enabled (but stopped) state when the user chooses to run the app. Each time the probe runs its data collecting code it is in a running state and upon completion returns to the enabled state until it is scheduled to run again.

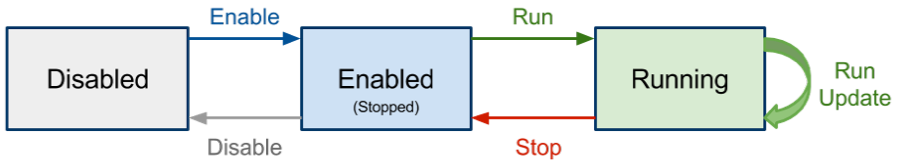


Figure 9: Funf Probe Life-cycle .

3.2.1 Cell Signal Probe

As outlined in Section 2.2.2 there are two available methods to get cellular signal strength readings on Android. We attempt to use the `TelephonyMan-`

[19] `http://www.funf.org`

ager.getAllCellInfo() call which provides more detailed information, but if it returns null then we assume the device does not support the API and fall back to using the onSignalStrengthChanged() listener. If the phone was in flight mode it could report that it did not support the getAllCellInfo() call, but by running the check each time the probe is enabled we will be able to use the best information as soon as it is available.

Depending on the type of network we are connected to (LTE, WCDMA, etc.) different sets of information is available. For example, if a WCDMA network is available then we can record the 'Primary Scrambling Code', a value described in the specification for WCDMA. For each of the different network types we provide overloaded methods to extract and store all of the provided information.

3.2.2 Bandwidth Probe

The bandwidth probe was created to incorporate bandwidth estimates into the SignalTracker app and record them through the Funf framework. The android.app.DownloadManager class in Android provides functionality to start a download easily and then monitor its progress. A test download file was chosen from Think Broadband [20] which hosts several different sized test files that can be downloaded over HTTP. The DownloadManager runs the download in the background and we query how much data was transferred in the given time period. If the download completes then the download manager will restart it again keeping track of the data downloaded between downloads. Using HTTP and existing download managers means an easy setup and will rely on the same networking protocols that actual data would likely use in an application.

In practice the bandwidth probe did not provide reliable readings. An alternative method was used to get these measurements. The popular Speedtest.net [21] Android application was used to make bandwidth downlink, uplink, and latency estimates. As the test could take upwards of a minute to complete the experiment was designed so that if a mobile signal was registered by the time the train carriage doors opened then I exited the train and took three measurements on the platform of that station before continuing the journey. Upon the train leaving the platform a test was begun, although in most instances the signal was too low for the test to complete between

[20] http://www.thinkbroadband.com/download.html
[21] http://www.speedtest.net/

stations. Three trials were run for each of the measurements. All of the tests connected to the same server helping to achieve consistent measurements.

3.2.3 UI Features

An important goal of this part of the project is gathering enough data to make conclusions about significant portions of the Underground. It is possible for one person to collect data on a subset of the lines, but ideally people could regularly carry out this task as they go about their day. Our goal for this part of the project is to streamline the usage of the app so that someone can contribute to our signal data without being burdened with a deep understanding of the app or any technical knowledge. The app should be simple enough to be used and follow existing design conventions recommended for Android applications.

With these requirements in mind we have produced an interface that can provide a user with information about what data is being collected and provide an intuitive way for new station mappings to be added.

3.2.4 Top Control Buttons

The top control buttons allow users to easy control the usage of the application. These are presented on the main view of the app to allow one click disabling, saving, or forced scanning.

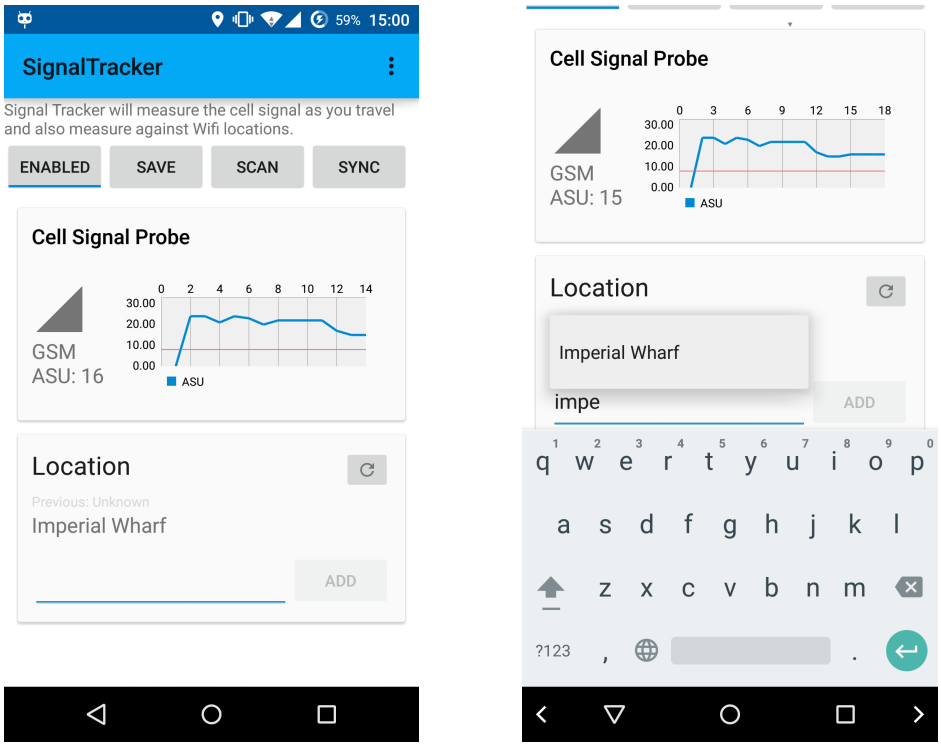
3.2.5 Card Interface

Card views were introduced into Android's Best UI Practices with the introduction of Google's Material Design language[22]. They provide an easy way to convey separate pieces of information on cards. Each different set of information is displayed on a separate card.

3.2.6 Live Measurement Information

If users are running the application voluntarily they are likely to be interested in the results reported from SignalTracker. We provide a card that shows the exact ASU measurements as the app progresses and an enlarged icon of the signal strength. Users are also presented with a graph of their

[22] `https://developer.android.com/training/material/lists-cards.html`



(a) SignalTracker enabled and collecting data.

(b) Adding a new station mapping for the first time.

Figure 10: The SignalTracker app

measurements over time which can be of interest to them. This graph can also help identify any bugs in measurements and allow users to match up recorded measurements to their experiences and expectations of signal on their journey.

3.2.7 Station Information

Users can see what station they are currently at. This helps to identify any mis-mapped stations.

3.2.8 Experimentation

The experimentation consisted of riding on the Underground and collecting data in a real world scenario. Multiple journeys were carried out and this section goes into detail about the experimentation conditions used.

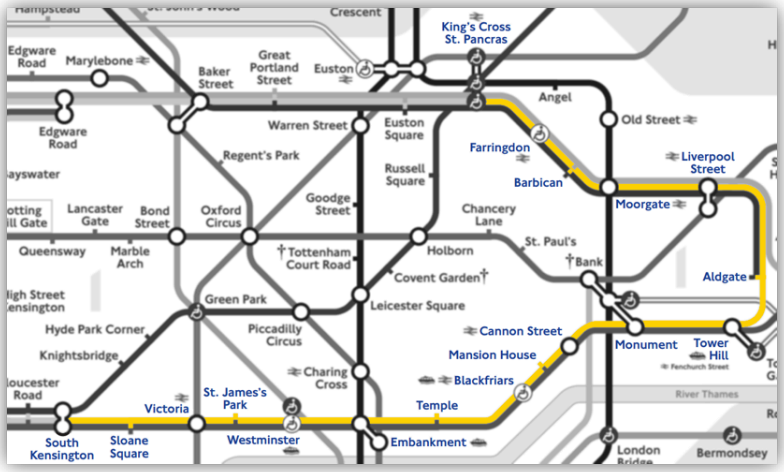


Figure 11: The Route Covered.

The trials were carried out on the Circle Line between the stations South Kensington and King's Cross St. Pancras anti-clockwise. This set of 18 stations covers areas that are both above and below ground and ones that have a potential for having any cellular signal strength at all. It also runs through Central London and some of the most used stations on the Tube network. Additionally, all of the stops on this route are shared with at least

one other line making the data applicable to other routes that travel through the same stations.

Data was collected across three different networks: EE, O2, and Vodafone. Three different phones were used with each one having a SIM card from one of the chosen operators.

The following controls were kept:

- Phone location Phones were kept in separate pockets of a polyester jacket.
- Carriage location / platform All data was collected on the centre carriage on a seat facing the inside of the tracks.
- Time The data was all collected outside of rush hour from 20:35 22:15 when carriages were generally empty.
- Initial Battery The batteries of all the phones were fully charged at the start of the data collection.

During the data collection for this portion there was a single unexpected stop on one of the trains, but this was not likely to affect any of the readings or results.

3.2.9 Initial WiFi Mappings

The initial WiFi MAC address mappings needed to be created so that we could later use the access points to determine our location. For the first round trip on the route we recorded a timestamp of when we entered a station on the train. This was determined as the point when the seat passed the beginning of the platform. Exiting the platform was determined as the point when the seat passed the end of the platform. The WiFi Probe marked each set of available WiFi access points with a timestamp.

The station entry and exit time intervals could then be mapped to the WiFi MAC addresses that appeared at the given platform. In this stage of the mapping we only considered WiFi networks with the SSID 'Virgin Media WiFi'. This helps eliminate personal hotspots that may travel between stations and allows us to focus on the networks maintained by TfL/Virgin Media. Every MAC address was unique to a station. There were two MAC addresses that appeared only in tunnels which were left as unknown and may have been results of a delayed scan. A total of 108 access points were mapped

this way. Two stations out of the 18 did not have any Virgin Media WiFi access points available: Farringdon and Moorgate. TfL lists these stations as having WiFi available, but it possible that they do not provide the service on the Circle Line platform or they were temporarily down (Farringdon station was undergoing construction work for the Crossrail project).

3.3 Results

Here we present the results from our signal tracker measurements.

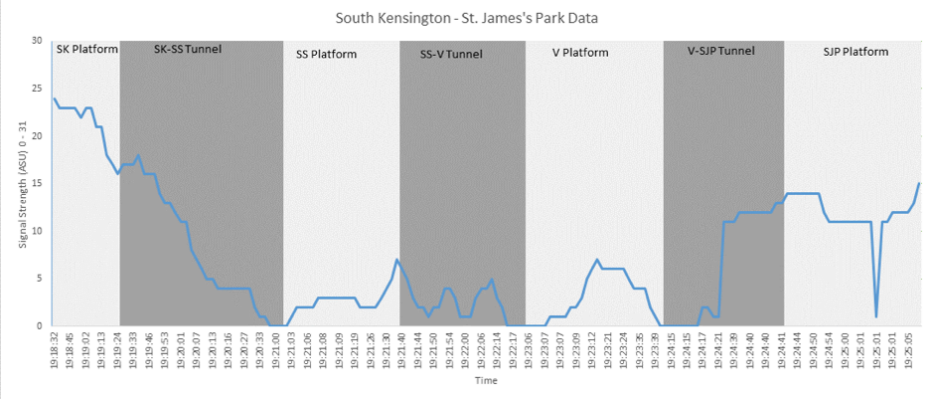


Figure 12: A single journey between South Kensington and St. James's Park.

Figure 12 shows the data from one single run of a subset of the route covered and was collected outside of the main data collection period. Areas shaded in dark grey are in tunnels and areas in white grey are at the platform. The timings were taken manually for this set of data. We can see the high sampling rate of data gives us a picture of the many changes in signal strength during one trip. We can see drops that take place upon entering a tunnel section and the variation between the stations. There is a delay between entering a platform and gaining signal. Upon entering the tunnel between South Kensington and Sloane Square we continue to get signal for a portion of the journey and see a steady decrease in signal.

There is a mixture of both steady, slow changes in signal strength (SK-SS tunnel) and more drastic changes (Victoria to St. James's Park tunnel). We see changes before we even leave platforms and delays in coming into platforms. There is also an outlier recorded at the platform at St. James's Park which is likely an error in reporting. This data stresses the difficulties

and unreliable nature of working with cellular networks on the Underground. There are patterns specific to each location so understanding what station we are at will be the only way we can make reasonable assumptions and predictions about the connectivity available.

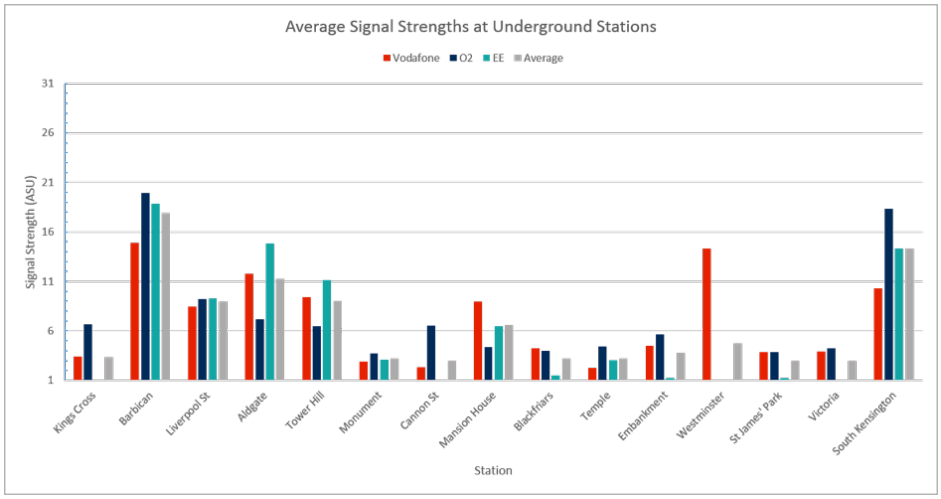


Figure 13: Average signal strengths by station and network operator.

Figure 13 shows the average signal strengths taken at each individual station, separated by the network operator. The stations are ordered according to their location on the route. No operator has a clear advantage across all of the stations. At most stations where one operator can provide a signal above 8 ASU the others can too. Vodafone had recorded signal strength above eight ASU in seven stations, EE in five stations and O2 in three stations.

The strong signal between adjacent stations suggests that we can get longer stretches of good signal for which we can transfer data. However, the signal may be completely gone in the tunnels between these highly connected stations. The distribution of stations with signals does open the opportunity to selectively pull down and cache data at each area. For example, if travelling from South Kensington to Liverpool Street we could pre-fetch data at South Kensington which could be used on the journey through stations without any signal.

Signal strength gives us one indicator of the usability of mobile data networks, but more telling is the estimated bandwidth available. In Figure 14 we only consider the stations, which had usable bandwidth available and measure on the Vodafone network. All of the download bandwidths, with

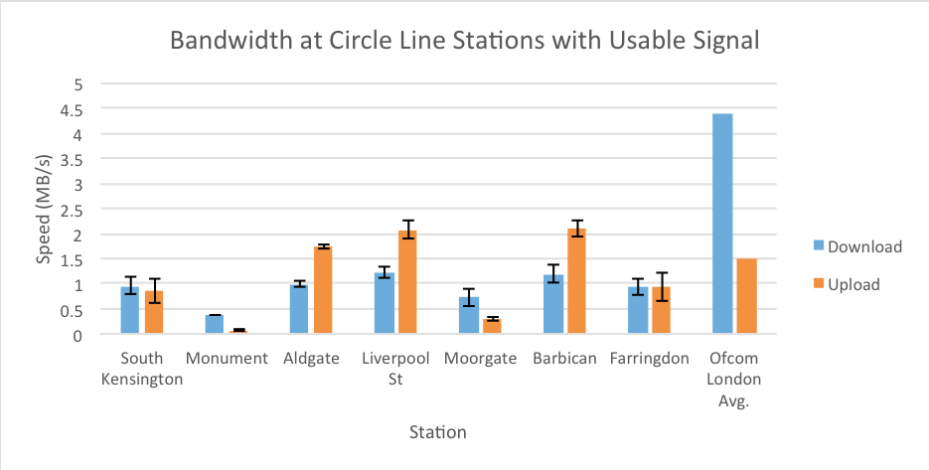


Figure 14: Average bandwidth measurements at each station that had usable signal. Standard error bars are displayed. Ofcom London averages [38] are displayed for comparison.

the exception of Monument station, sit between 0.5 to 1.5MB/s. The upload bandwidths can reach higher speeds up to 2.0 MB/s.

These are all usable speeds to transfer small amounts of data. Interestingly, the upload speeds are all relatively high. This makes offloading through the cellular data a more attractive prospect. According to Ookla [23] the average mobile download speed at the time of writing in London is 15.1Mbps and upload speed is 8.8Mbps. The published Ofcom average speeds are much lower for London though and can be seen in Figure 14.

In Figure 15 we see that across all of the stations measured the latency sits between 40 to 60 ms. The variation of the latency varies greatly between stations. We can see that this value is inline with Ofcom's London averages.

Figures 16 and 17 show the same data, but for the tunnels that have usable signal. The downlink bandwidths are quite low in two of the tunnel sections and the Farringdon to Kings Cross tunnel have a relatively high latency of 122 ms. There is a particularly strong difference between the download and the upload bandwidths in these tunnel sections suggesting they should be treated differently. It's also important to note that there are only three tunnel sections that have signal high enough to conduct the bandwidth test compared to the seven platforms in Figures 14 and 15.

[23] `http://www.netindex.com/mdownload/4,105023/London,-GB/`

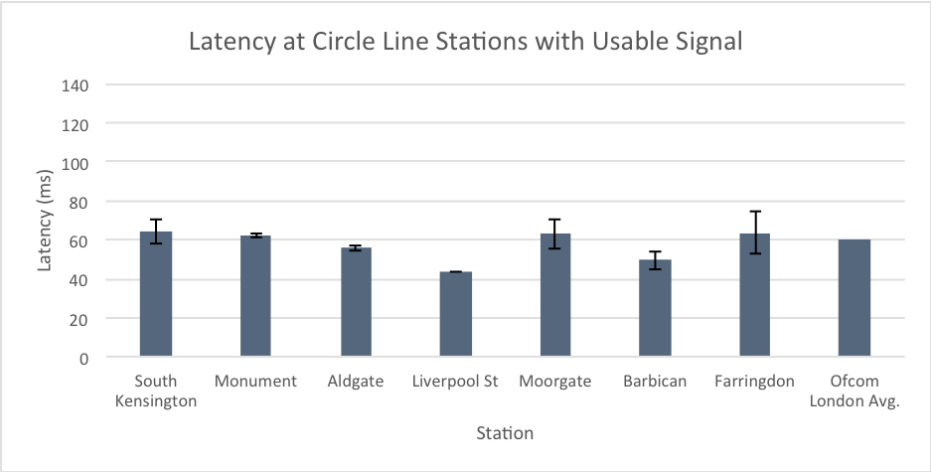


Figure 15: Average latency measurements at each station measured. Standard error bars shown. Ofcom London average [38] is displayed for comparison.

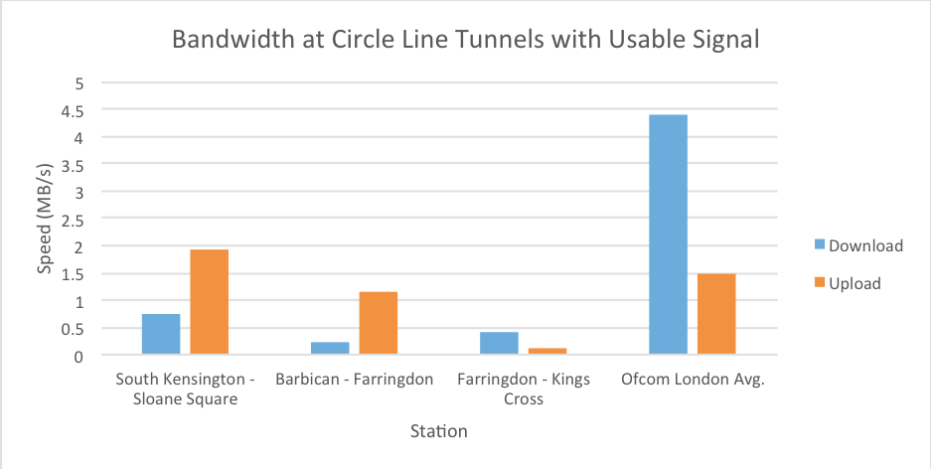


Figure 16: Average bandwidth measurements at each tunnel measured. Standard error bars shown. Ofcom London average [38] is displayed for comparison.

3.3.1 Accuracy and Comparison of Station Mapping

We now consider the accuracy of the WiFi MAC address scheme discussed previously. GitHub user 'benjojo' has published[24] an incomplete list of MAC

[24] `https://github.com/benjojo/TubeWifi`

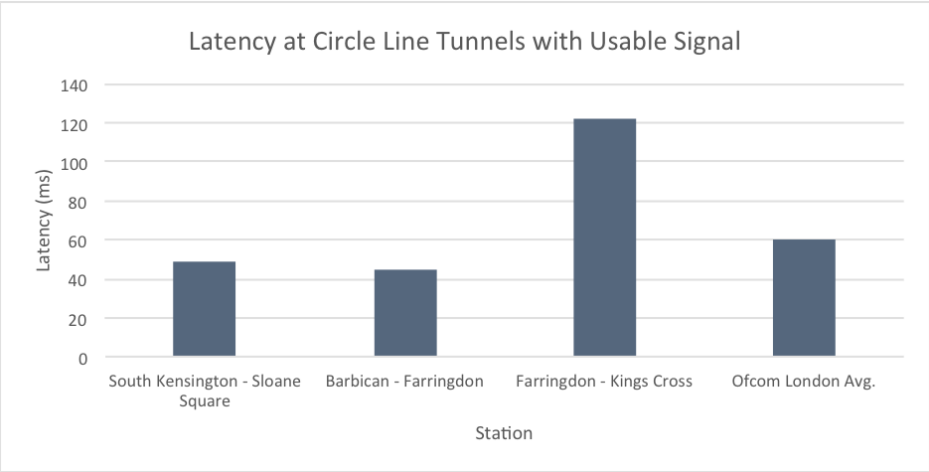


Figure 17: Average latency measurements at each tunnel measured. Standard error bars shown. Ofcom London average [38] is displayed for comparison.

address mappings to stations. None of the MAC addresses matched between our data. However, the user's methodology is not known and we don't know if they covered the same platforms or areas that the Circle line passes. The Westminster data was added April 10, 2014 (although it is not clear on what date the data was actually collected) so it could be out of date. If it is out of date it could indicate that the access points change regularly at each station (due to upgrades, maintenance, etc.). This would mean it is important to automate the mapping and consider that access points may change during this process.

During later experimentation the results of the station mapping could be checked during further data collection. The station mapping was accurate and in nearly all cases managed to identify the station before the train had come to a stop on the platform.

3.4 Optimizations

The data gathered from the dedicated collection runs gives us a good indication as to the levels of signal strength at various stations. However, there is only so much data that one person can collect and we can see variances in the sample size of the data from the limited number of trials. After the

initial data collection was carried out the app was extended to allow the collection of data from any number of phones. One of the limiting factors of the previous design was collecting the data and parsing it for analysis. While the app's UI allows for easy interaction getting the actual data into a workflow was more laborious.

In [19] Gallacher et al. present three challenges of crowd sensing data. They are:

Accessibility Engaging less technical competent individuals.

Incentivisation and Sustained Participation Need to consider rewards and incentives for participation. The authors claim that many existing devices do not provide any visual feedback of what is being sensed making it hard to engage the user.

Deployment Practicalities For dedicated sensors it must be possible to deploy without causing alarm or suspicion. This is less relevant to our mobile application.

We seek to address these concerns in our optimizations.

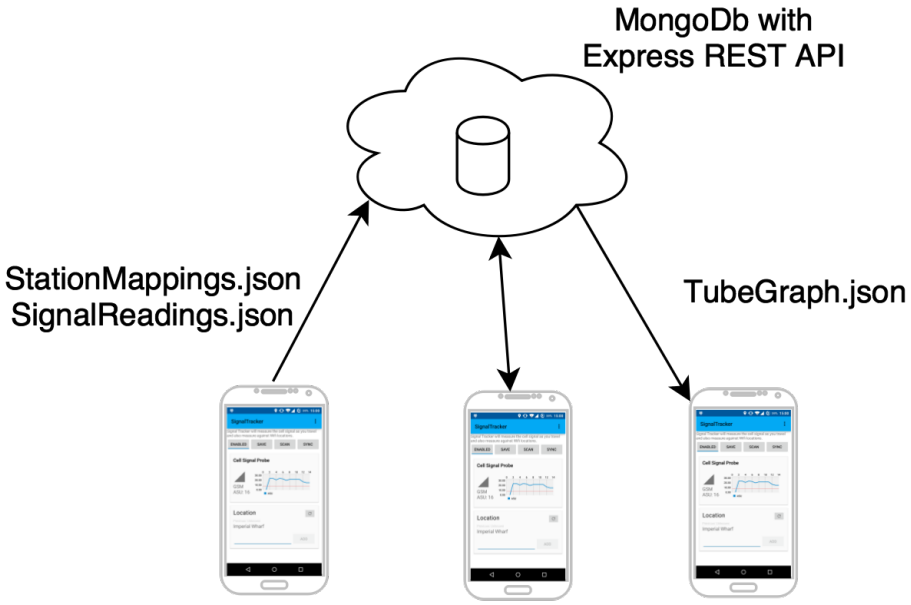


Figure 18: System structure with Signal Tracker Web service

Introducing a server to collect data allows us to easily get access to signal readings and station MAC address mappings and also allows the app to update with the most recent version of the tube graph. The structure and basic interactions of the system are displayed in Figure 18 with client devices sending updates to the web server which provides updated tube graphs to each client on synchronization.

A combination of Node.js, Mongodb, and Express were chosen to build the backend infrastructure of the server. These technologies all integrate well together as part of the MEAN stack [25] and can be set up with little overhead using Yeoman [26]. This quick setup as well as prior familiarity with the tools meant the functionality could be added relatively quickly allowing for further data and evaluation to be performed on the app. Mongodb was a logical choice for storing the document data as the app was already using JSON objects to represent the tube graph and could store these in Mongo's native format.

Selecting a REST API means it is supported well by Android libraries and can be integrated in a future workflow if needed. The REST interface sits between the Android application and the data store. This means that we can better manage authorization to the data and do not have to store any database credentials on the distributed application. This layer also means we only have to maintain one codebase to do the processing of the data before adding it to the database.

The implementation of the server exposes three API endpoints which the app can communicate with. These are listed in Table 2 alongside a description of each endpoint.

Table 2: API Endpoints for SignalTracker Server

End Point	**REST Method**	**Description**
/api/stations	POST	Send updated station MAC mappings
/api/signals	POST	Send updated station signal readings
/api/tubegraph	GET	Retrieve a new copy of the tube graph

[25] `http://mean.io`
[26] `http://yeoman.io/`

We use the Retrofit [27] library to create a Java interface (`SigTrackWebService`) out of these API endpoints. When we collect either signal strength data or MAC address mappings we update our local tube graph to represent these changes. In addition we also persist these in the form of JSON files to the phone's storage. By doing this we ensure that any data collected is not lost if the app is restarted and keep memory usage low. When the app is ready to synchronize, these files are POSTed to the server and upon successful acknowledgement we can clear our local cache. We also request a new tube graph representation that should take into account the changes that we have sent as well as any new data from other phones. This new tube graph comes directly as JSON and replaces the previous tube graph on storage and is reloaded into the TubeGraph class.

This server addition means the app could be adopted by a large number of people all contributing to the same set of data with processing and storage all being taken care of on the server side.

3.5 Evaluation

To evaluate the optimisations of the SignalTracker app we will look at the additional data generated as well as some user experience studies to understand if the app actually provides a seamless experience that can enable large scale crowd-sourced data collection on the Underground.

3.5.1 Experimentation

In order to test out our enhancements to the SignalTracker application we deployed it to a small set of people. These users were given the app and ran it either on their own phone or on a phone provided to them. They were instructed as to the basics of the application - it's intended purpose and the how to enable and disable the data collection. During their daily journeys we requested they collect data and map as many stations as possible, but were not instructed to travel additional routes. The data collected by them was all submitted to our web service and usage analytics was automatically logged to the MixPanel[28] service. Data was also collected by me on my daily commutes, but no additional trips were taken to expand coverage. A total of six people collected data. All were in their mid 20's and included a range of

[27] `http://square.github.io/retrofit/`
[28] `https://mixpanel.com/`

people. The participants include fellow Computing students, a professional in the financial services sector, and one participant who has a background in fashion studies. This range of participants was a valuable opportunity to develop an app that can target the general public.

3.5.2 Testing on the Play Store

Developing and testing the application with real users meant having to manage and deploy and update SignalTracker when bug fixes and new features were added. At first, with only one other tester, this was done by sending the application .apk installation file to them and informing them to update the application as necessary. This started to become cumbersome and time consuming for both of us involved and would not scale well to more testers.

The Google Play store offers a Beta channel[29] which allows application updates to be deployed to a private listing in the store and updates are managed by the Google Play store app on a user's phone. The benefit of using the Beta channel feature is controlling exactly who uses the application. In order to grant access to the private store listing the developer creates a group on the Google+ social network and invites people to this group which is linked to the store's developer account. This process was setup and used amongst early testers. It was found that the procedure to get people onto Google+ was not easy with none of testers (and myself) familiar with Google+. There is also several hours delay between a user joining the group and the permissions propagating which made it difficult to communicate with them and get the app on their phone quickly.

When the app was ready to be rolled out to more people it was published on the Play store as shown in Figure 19. This made the installation process much easier. The app currently has six active devices (not all of the test devices I provided use the Play store version of the application). The app was downloaded on some devices but never used as potential participants did not take the tube.

Finding users highlighted some limitations of the design. Within our University entry year, the number of students who had Android phones was lower than the overall market share would have suggested. The app had originally been designed to be used by myself on the given test devices and as such targeted Android 4.4 KitKat. This version, as of June 1st 2015, has

[29] `https://support.google.com/googleplay/android-developer/answer/3131213?hl=en-GB`

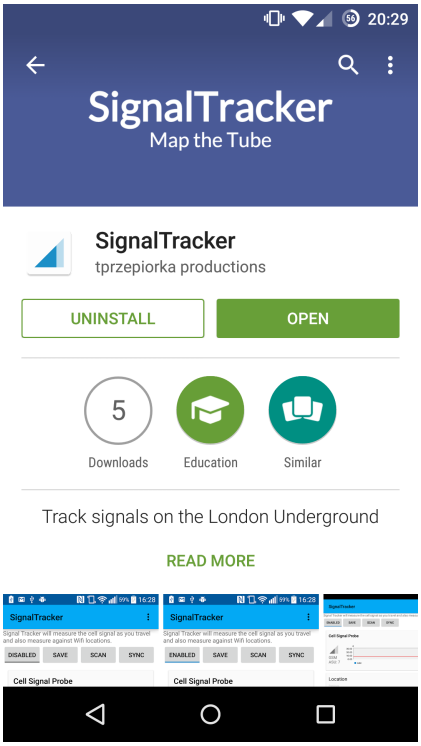


Figure 19: Google Play store listing for SignalTracker seen from a phone.

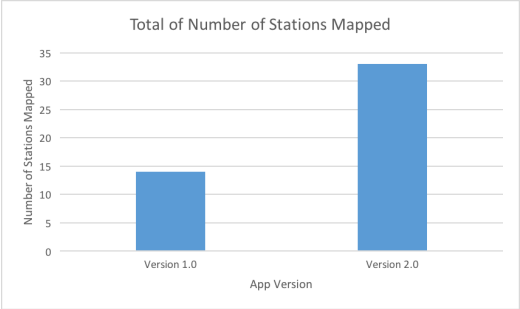
51.6% market share[30] and some users did not have phones capable of running the app. There is also a large number of people who do not regularly take the Underground and who rely on buses, bikes, or walking to get into University.

3.5.3 Number of Stations Mapped

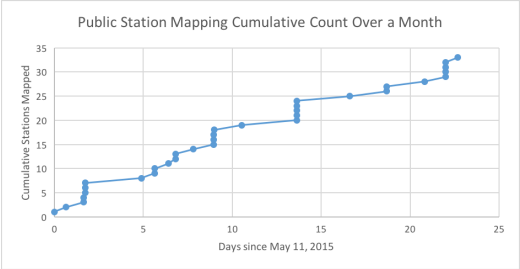
By spending a considerable amount of time manually collecting data during the first trials outlined earlier we obtained data for 14 stations on the underground. Figure 20a shows we were able to over double the number of stations mapped to 33.

Figure 20b shows the cumulative number of mapped stations over 23 days. The increase is steady with an average rate over the period of 1.46 stations per day with a total of seven devices used. Some days show jumps of multiple

[30] `https://developer.android.com/about/dashboards/index.html?utm_source=suzunone`



(a) The total number of stations mapped before and after usability improvements.



(b) Station mapping over time by multiple users.

Figure 20: Number of station locations mapped using our scheme.

stations which is likely when a route was being mapped for the first time. Some of these devices were used by multiple people.

Data collected through the app covered all four mobile network providers, including Three for which no data was collected during the manual trials.

3.5.4 Data by Phone

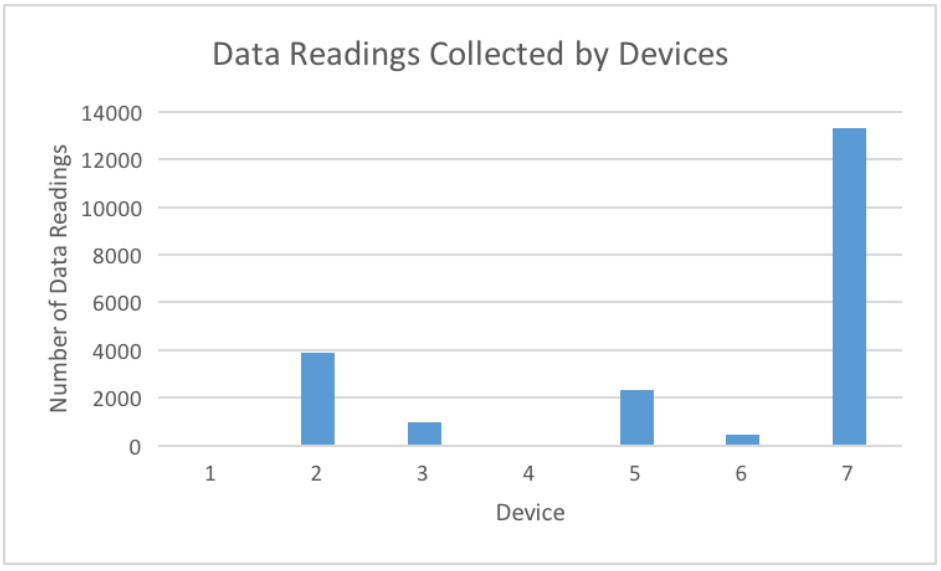


Figure 21: Amount of data collected by phone device.

In Figure 21 we can see the distribution of the total number of data readings by phone. There is certainly a discrepancy between the amount that each phone was used to collect data. Some devices were only used for a few readings (the minimum being 10 on one phone).

3.5.5 Comparison of Results

We compare three of the stations that recorded data using both the controlled experiment and through the second version of the SignalTracker app.

Taking readings from EE at South Kensington from both the publicly pooled data and the originally collected data we see that there is a difference in the average signal reading. Running a t-test on the two data sets shows that there is a statistical difference. The differences in averages is shown in Table 3.

Station	**Signal Strength**		**Difference**
	V1.0	**V2.0**	
South Kensington	8.86	11.92	-3.06
Victoria	0.75	5.28	-4.53
St. James's Park	1.26	1.23	0.03

Table 3: Comparison of results using two different measurement methods.

Both Victoria and St. James's park which also had data collected by both versions of the app with operator EE also have differing values for the average signal strengths as shown in Table 3. If we were using a threshold of eight ASU then we would see that there would be no difference between the two data sets.

There are several factors which could account for the differences in data. These are the control variables we used, the most notable being: phone model, seat positioning, and time of day. The data was also collected months apart when the actual network configuration may have changed at these stations. The collected data from users may be more representative of actual usage habits.

Moving forward it would be good to combine the values which are collected under controlled conditions with those from crowd-sourced data. In a deployment scenario to the mass public the amount of data collected would be much more valuable. Insights into the particular behaviour of certain

phone models, trips during times of day and over time are cannot feasibly be collected by one person dedicated to riding the tube for a few hours. From logs submitted by the crowd-sourced version of SignalTracker it is a straightforward process to generate these statistics using aggregate functions over the data logs stored in our web hosted database.

3.5.6 Auto-Sync

Once a few testers were using SignalTracker it became problematic to remind them to manually synchronise the data with our server. As a result the automatic sync functionality built on Android's sync framework was added to the app on May 21st. Figure 22 shows the resulting number of syncs made. As soon as the user had the syncing handled for them the number of times they manually sync'd data dropped. The total number of syncs also increased meaning the user was more likely to hold an up-to-date copy of the tube graph and we would have improved access to their submitted readings. The data for this was collected through the MixPanel analytics service that we integrated into SignalTracker.

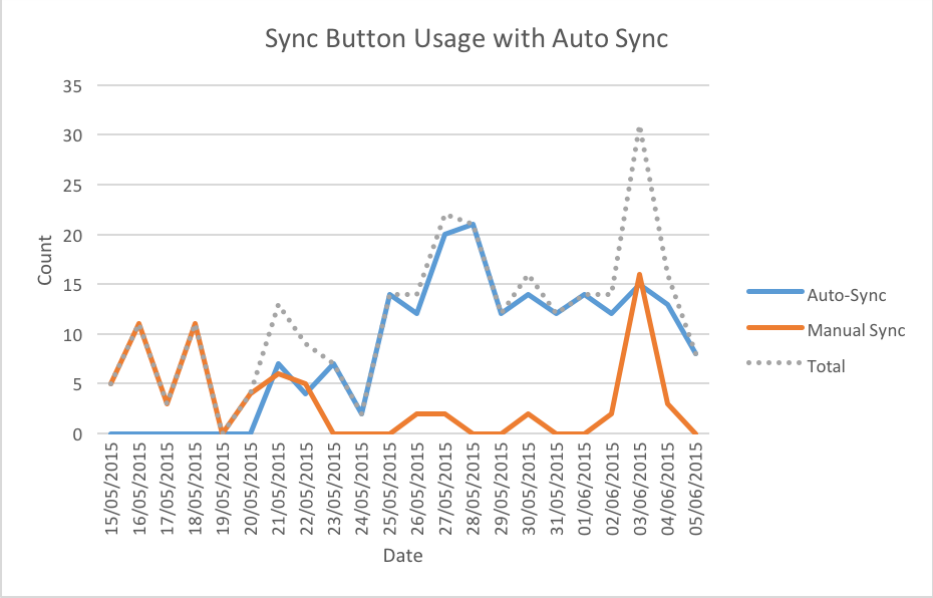


Figure 22: Sync Button and Auto-Sync usage in SignalTracker

3.5.7 User Retention

Date	People	The number of days later your users were retained.										
		< 1 day	1	2	3	4	5	6	7	8	9	10	11
May 27, 2015	6	100.00%	100.00%	66.67%	83.33%	66.67%	66.67%	50.00%	66.67%	66.67%	50.00%	33.33%	
May 28, 2015	6	100.00%	83.33%	83.33%	50.00%	66.67%	50.00%	66.67%	66.67%	50.00%	50.00%		
May 29, 2015	5	100.00%	100.00%	60.00%	80.00%	60.00%	80.00%	80.00%	60.00%	60.00%			
May 30, 2015	6	100.00%	66.67%	66.67%	50.00%	66.67%	66.67%	66.67%	33.33%				
May 31, 2015	3	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	66.67%					
Jun 1, 2015	4	100.00%	75.00%	100.00%	100.00%	75.00%	75.00%						
Jun 2, 2015	4	100.00%	75.00%	75.00%	75.00%	50.00%							
Jun 3, 2015	5	100.00%	80.00%	80.00%	40.00%								

Figure 23: User retention for the SignalTracker app.

Using MixPanel analytics the user retention was recorded for the SignalTracker application with a significant number of users starting on May 27th. This is displayed in Figure 23 from MixPanel's dashboard. It shows the percentage of users retained after a number of days since the date in the first column. The data was put together on June 6th, explaining the limited number of days displayed. At the end of each period there were still users using the tracking application. On May 27th there were six users and a full six days later half of those had interacted with the application. Ten days later only two had still interacted with the application. This data suggests that users will continue to use the application past the first day they install it which is a significant achievement. However, they likely won't maintain a sustained usage over a long term. This may be due to them already feeling they have mapped their daily route, forgetting about the application, or deciding that it was too resource intensive on the phone. In the future we could focus on developing ways to get users to maintain usage over longer time spans. This could be done by implementing an 'auto-enable' feature similar to the one that is outlined in Section 4.2.7. Alternatively focus could be spent on marketing the app to a larger number of users. The high turnover would not be a massive problem if new users are constantly joining the platform. This focus on new users would also likely bring in a more diverse set of routes and help expand the coverage of our readings, whereas focusing on user retention would improve the depth and reliability of our data. With only a limited

number of participants it would be best to expand the current studies to include more users as six unique users is quite limiting.

3.5.8 User Feedback

Discussions were held with some of the test users of SignalTracker. The main users said that they enabled the application on about 70-80% of their tube journeys. All found the tagging process intuitive and straightforward. Testers were able to understand the purpose of the application and understood that their data was being collected to map signal patterns on the Underground. Particularly being able to see the graph of signal data and the current station was well received and likely contributed to more usage. Providing a usable application is important to be able to gather sufficient information to deploy solutions on the Underground and users all felt that SignalTracker was a good enough app to do so.

3.5.9 Limitations and Future Work

SignalTracker performs well at meeting the challenges imposed when collecting signal data on the Underground, but it is important to recognise the limitations and aspects that are not captured by the app. We present some of the limitations of the application and include suggestions to overcome these which act as the future work for SignalTracker.

We place a large amount of trust with the users of our app and the devices they have. There is currently no easy interface to correct incorrectly mapped stations and everyone's input data is treated equally. We rely on average recordings which means if one person has a faulty device or malicious intent they can influence our collected data. While all data and its source is logged making it possible to revert any of these attempts it is not possible to automatically identify these faulty readings. Research has been made into successfully identifying accuracy and trust in wireless sensor networks by Asmare and McCann [3] and some of these techniques could be applied to our data collection.

When attempting to deploy the application and find testers the number of users running compatible Android devices was lower than anticipated. While rolling out SignalTracker to a wider audience would mean that the iPhone bias would diminish, but it would still be wise to consider a cross-platform

application which could be developed using a tool such as Xamarin[31].

The location scheme using Virgin Media WiFi access points proved to be effective in the areas covered. However, outside of Zone 1 many stations do not have Virgin Media WiFi available. Some may have other WiFi from nearby buildings APs which could be used to map a location. To handle these it is important to eliminate any non-stationary access points. Ideally the mapping would rely on no infrastructure.

User engagement was strong in the first few days following installation of SignalTracker, but fell off quickly after this. Introducing gamification using profiles and a scoring scheme could allow users to see how much they have contributed to the project and encourage them to make more measurements over a longer period. Giving the public access to the collected data through an internet site or the app itself would help them feel like they are participating more.

The design for SignalTracker included a plan and outline of how bandwidth data could be collected using the application. Bandwidth data is more useful than signal data by itself, but the technique used was not successful. Devising a way to calculate bandwidth would be beneficial. Including transfer speed data collection in the other parts of this project (namely DeepOpp) could be a source of this data.

An important aspect of data collection is privacy of user details. SignalTracker has access to users' location, but does not currently take explicit steps to protect this confidential information. A privacy policy is included in the app outlining the collection and use of data, but more could be done to prevent the identification of specific users and to encrypt and transfer all data to the server securely.

3.6 Conclusion

In Part I we presented the challenge of building up an objective measure of network characteristics on the London Underground. The lack of available data means that we have to take an initiative to collect real world data on one of the world's busiest public transport networks. The nature of the Underground means that we were faced with several challenges that network mapping presents. We require a precise measure of a phone's location, but GPS was not available as a solution. Moving train carriages mean

[31] `http://xamarin.com/`

that the mobile connections are constantly changing and network availability is unreliable along many routes. We have developed a robust solution to provide the required information. Initial trials riding the Underground built a preliminary view of signal patterns along the Circle Line in Central London. A location mechanism was built which can figure out with station level accuracy where a phone is on the Underground. An app was built that exposes this data collection platform to everyday users who could help to expand our knowledge of the changing conditions of the Underground. The app proved to be intuitive, reliable, and greatly expanded our coverage by crowd-sourcing the collection and aggregating submitted data automatically through our server structure. While signal strength readings will always have inherent shortcomings due to changing external conditions we have built a relatively comprehensive understanding of the network patterns in parts of Central London.

4 Part II - DeepOpp: Middleware Caching

The data collected in Part I provides evidence that there are opportunities to exploit network condition differences between stations in order to provide an intelligent data retrieval mechanism.

Passengers on the Underground want to access the internet on their journey, but as we saw in Part I, data access is only available in certain stations. Having the user constantly check their phone to refresh content is a drain on limited phone resources and leaves the user frustrated. Many apps are designed with constant internet access assumed. Apps that do cache data often only do so a few times a day leading to stale content. Apps, such as Facebook, appear to make no distinction between the type of content displayed when there is little bandwidth available.

In this section we utilize the data from Part I and present a design, implementation, and evaluation of a mobile phone pre-fetching caching system targeted towards use on the London Underground.

4.1 Design

4.1.1 System Design

As travellers make a journey on the London Underground they may pass through stations, some of which have access to a mobile network connection. The system design presented here takes advantage of this variation between stations to pre-fetch and cache data. The goal of the middleware is to fetch, cache, and make content available to a client application. This is shown in Figure 24. While at the station in 1 the DeepOpp middlware will fetch and store contents even if the user is not currently using a client application. When underground, without signal, in 2 the user can use the client application and consume the data downloaded in 1. This process repeats as we download data again in 3 for use later.

4.1.2 App Separation and Design

The middleware is separated from the client application and has been built as a separate Android application. The roles of the DeepOpp middleware are to:

- Provide access to the internet and APIs of client applications

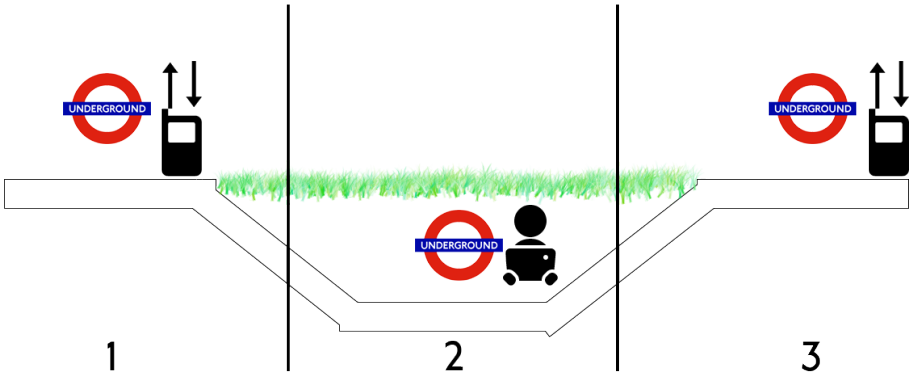


Figure 24: DeepOpp overall design to use stations with good signal to download and cache contents.

- Fetch data from the internet at scheduled intervals
- Store the data in a structured format that can be retrieved
- Provide user defined settings and preferences
- Enable and disable the middleware

The user interacts with their content through a client application. Keeping the middleware separate shows how it could be integrated in an actual scenario, as part of the networking APIs provided by the Android OS. It also shows how the techniques and algorithms are generic and can be applied to multiple client apps.

4.1.3 Schedulers

We propose three schedulers which are to be used as part of the DeepOpp middleware. Each of these defines the process for how often and under which conditions the middleware will fetch and obtain new data to store in the cache.

Basic Scheduler This baseline scheduler runs on a fixed interval and will make a fetch at this interval.

One Dimensional Scheduler This runs on a fixed interval scheduler and will make a fetch if the current signal strength is above a set threshold.

Location Scheduler This uses the location scheme and data collected in Part I to reschedule fetches only at stations that we know have usable signal strength. It determines the current location and an estimate of the direction of travel to reschedule at the next station that has a signal higher than a given threshold. See Section 4.2.8 for a more detailed discussion.

4.1.4 Middleware Design

There are two possible implementations that could be used for the caching logic in the middleware.

Server Side Hosting the caching and pre-fetching code on an intermediary server has several advantages. We can host the cache on this server and include a content ranker and optimizer. The app middleware can provide the profiling information and get the relevant cached data from the server.

This approach means that the bulk of the optimization and processing can be done on a powerful server that is not constrained by the limited battery available on phones. It can also be adapted and used on multiple platforms. We can also compress and optimize media before sending it to the phone.

Phone Based Application Storing all of the logic and middleware on the phone has several advantages over using a server side approach. We can save data by not having to transmit profiling state and instead only retrieve contents meta-data. There are less privacy concerns as users will not be sending their data and content through a third party host. It is also a more scalable approach that doesn't require the maintenance of an external middleware service.

For these reasons we chose to use a single phone side application to provide the caching functionalities.

4.1.5 DeepOpp Dataflow

Figure 25 shows the flow of operation of DeepOpp. In (a) DeepOpp is enabled by the user which starts a given scheduler. Once started the scheduler will rerun at set intervals depending on the type of scheduler and provided

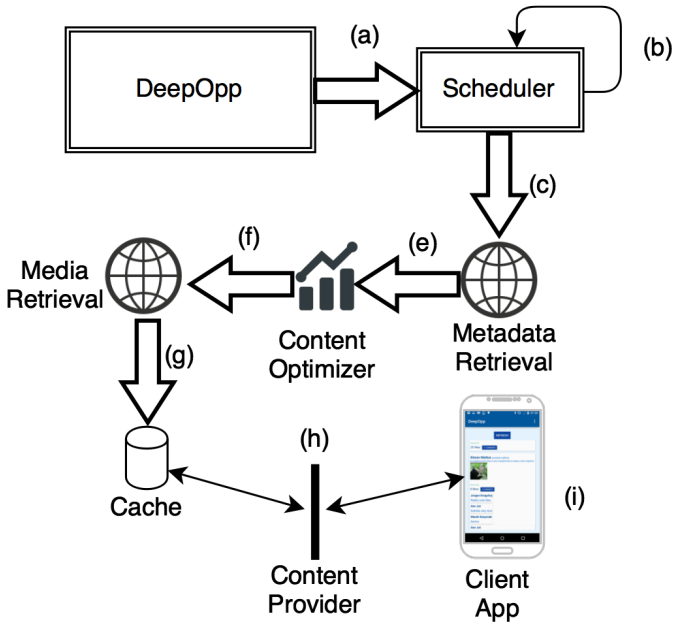


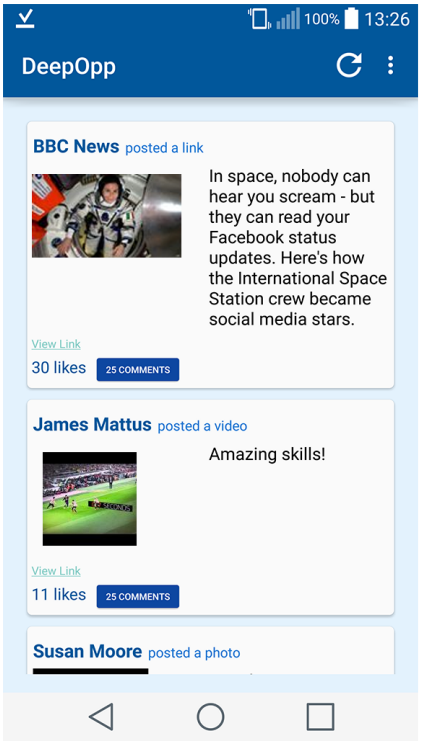
Figure 25: DeepOpp operational flow.

conditions as shown in (b). When a scheduler starts a fetch of data in (c) metadata is retrieved from an external remote service and passed to the optimizer in (e). The Optimizer filters potential media items based on phone profile conditions. The remaining content to be downloaded in (f) is retrieved from the remote service and stored in the DeepOpp cache (g). When the cache receives a request for data it provides it to a client app in (h). The client app can then display the content to the user as appropriate in (i).

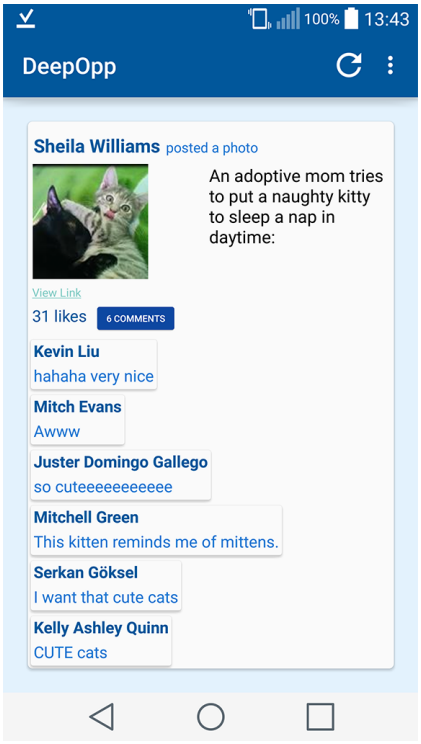
4.2 Implementation

4.2.1 Facebook Client Application

It would be ideal to modify the existing Facebook application to support the middleware, but the application is closed source and not easily modifiable. Intercepting the app's requests would be a possibility and allow us to rely on the existing application. Facebook encrypts its data making this problematic. Reverse engineering their app to provide this would require substantial work when the front-end application is not the focus of this project. A custom app

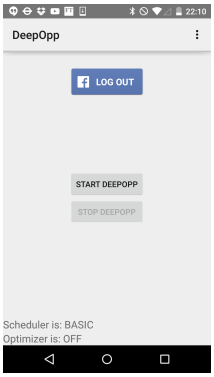


(a) Posted link with comments.

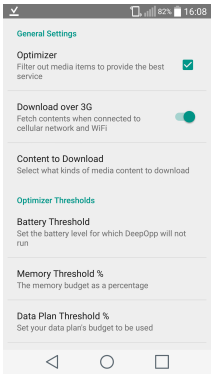


(b) Posted photo with comments.

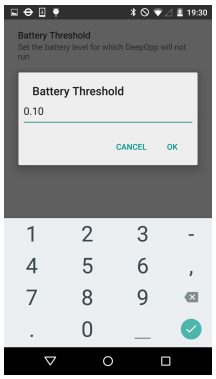
Figure 26: The Faceboop client app for the DeepOpp middleware.



(a) The main screen of the DeepOpp middleware control app.



(b) The user settings available.



(c) Changing the battery threshold of the DeepOpp middleware.

Figure 27: The DeepOpp app

that supports the basic content display would suffice in demonstrating the abilities of the middleware. The app created, Faceboopp, supports displaying user posted statuses, links, and photos. Users can see the number of 'likes' a post has gotten and see comments on a post. There is the option to manually refresh the content displayed.

4.2.2 DeepOpp Application and User Interface

The DeepOpp application provides some controls that the user of the phone can manage. If the middleware were integrated into the Android OS then the features and controls could be found in the phone's main settings. Figure 27a

shows the user interface presented to the user when they open the application. A native Facebook login button is shown allowing them to add their Facebook and authenticate through Facebook as described in Section 4.2.4. Buttons are shown that allow the user to start or stop the background middleware service, overriding the automatic geofencing feature. There is information about whether the optimizer is enabled and which scheduler is being run at the bottom of the screen. The user also has access to settings shown in Figure 27b letting them control specific thresholds for the optimizer. Figure 27c shows how a user can intuitively change settings using native Android controls.

4.2.3 Inter-app Communication

As discussed in the Section 4.1.2 of the design, the middleware is built around a base app providing the caching functionalities and interacting with individual client applications that can display the cached content to users through a custom interface such as the one created in the Faceboopp application. In order for the two components to interact, a form of inter-app communication was implemented. The first attempt in doing so used Android's Bounded Service Messenger interface[32]. This interface defines Message objects that can be sent between different processes and handled in the way necessary. However, when sending media items the predefined size limits of 1Mb [33] were too restrictive. Android ContentProviders were used to provide this functionality. ContentProviders are a lot more flexible and make sending larger media items possible.

4.2.4 Facebook SDK and API

Facebook provides an official SDK and API to access their content. The Facebook login and authentication button[34] will interact with the native Facebook application if it is installed or open a web browser page to enable login by the user. An 'app' was created on the Facebook Developer Platform. The user is presentedwith the option to authenticate with the application

[32] http://developer.android.com/guide/components/bound-services.html#Messenger

[33] http://developer.android.com/reference/android/os/TransactionTooLargeException.html

[34] https://developers.facebook.com/docs/facebook-login/android/v2.3

and see the Facebook permissions (specifically read_stream) required by the app. After this a session token is created and managed by the Facebook SDK library which can be retrieved from within the DeepOpp application context.

Figure 28: The Json metadata is used to build the Faceboop UI.

The only endpoint used is at `/me/home`[35]. This provides the metadata needed and will select the relevant ranked content of a user's news feed. An example of the metadata is shown in Figure 28. Important to note is any media that is part of the post can be accessed through the provided url. This makes it possible to download a list of these metadata objects and then select which media we want to download. According to the Facebook documentation [36] this API endpoint is not designed to be used on platforms such as Android that already have existing Facebook applications. By keeping the Facebook Platform application in development mode and assigning each user as a tester for this app we are able to proceed to test using real Facebook metadata.

4.2.5 Data Storage

Caching data is a core responsibility of the DeepOpp middleware and there are several possibilities of how to store Facebook feed data. A SQLite

[35] `https://developers.facebook.com/docs/graph-api/reference/v2.3/user/home`

[36] `https://developers.facebook.com/docs/graph-api/reference/v2.3/user/home`

database was an obvious choice for this task as it provides many of the advantages of a relational database in terms of structure and is well supported and integrated into the Android ecosystem. The Json response data from the Facebook API could be handled natively by Json-based databases like MongoDB or by relational databases such as PostgreSQL which provide Json data types. Using these options would rely on third party libraries and the additional complexity in an Android app would not provide much benefit. Querying through supported classes also helps the apps integrate well with the ContentProvider pattern used for inter-app communication.

Three tables are created and maintained by DeepOpp:

Feed Pull Table Stores a reference to each unique fetch of data we have made, allowing grouping of individual posts together.

Feed Item Table Each row is a single post from the Facebook timeline. Each of these has a reference to the pull from which it originated, the type of post, and some additional information such as the object identifier generated by Facebook.

Media Table Contains blobs of media items and relations back to the items from which they originated. Each record also holds the size of the media, type, and other meta information.

In order to provide a cleaner syntax in Android's Java code additional wrapper classes have been created to facilitate interaction with the database. These are contained in the `storagetypes` package with a corresponding helper class for each of the data tables described. The `DataSource` classes contain methods for adding new items through standard Java syntax. Each `DataSource` returns a `Cursor`[37] object which can be passed natively by the ContentProvider to the Faceboop app.

4.2.6 Media Caching

Facebook text posts and statuses are relatively small and are, by default, contained in the metadata result returned by the Facebook API. Text posts are the lightest possible content so we do not focus on optimizing or caching these and include them in the metadata. Users can post pictures and videos

[37] `http://developer.android.com/reference/android/database/Cursor.html`

to their Facebook feed and we aim to filter potential media items so that we may cache these items in an optimal manner.

If a post contains a media item then it will be marked as such in the meta data retrieved from Facebook and a link provided to fetch the full item. If a post has not been filtered out then we will proceed to add it to the local SQLite `MediaTable`. We check to see if the media is already in the table if it is then we don't require downloading it again. If we do proceed to download it then we use an `ImageLoader` to download the image and store the blob in the `MediaTable`. As the `MediaTable` exposes a Cursor to the ContentProvider, the Faceboop client application can directly request a specific media item which is returned on demand by the ContentProvider.

4.2.7 Geofencing

Even though one of the aims of the DeepOpp middleware is to reduce battery drain by limiting unnecessary network calls there will be overhead in running the middleware application. Ideally we would like to only run the background service when the user is on the Underground. Requiring the user to manually enable and disable the service would break the seamless experience of connectivety that DeepOpp hopes to achieve. By using the user's location and the Geofencing libraries we can manage the lifecycle of the service automatically without any user interaction.

Android supplies a set of Geofencing APIs [38]. These allow the programmer to define geographical areas which can trigger actions within applications. Using this and a list of station coordinates[39] we can generate and register geofence areas for all of the stations on our chosen route. Upon entering the geofence area we activate and enable the tube middleware by starting an Intent.

In practice the background location technique used by the geofencing services was not highly accurate and through preliminary testing it often took up to half an hour before presence in the area was registered. In order to maximise the likelihood that the service was enabled upon entering a station area, this radius of the geofence was increased. With a larger area we increase the possibility that the user inadvertently enabled DeepOpp when travelling past a station entrance even though they have no intention to travel on the Underground.

[38] `https://developer.android.com/training/location/geofencing.html`
[39] `http://www.doogal.co.uk/london_stations.php`

We also want to disable the middleware when a user exits the Underground. Simply relying on the geofencing areas would trigger exits between every stop we travel on. The geofencing methods become even more unreliable on the Underground likely due to the lack of GPS. To achieve automatic disabling the middleware tracks of the amount of time since the phone last encountered a known tube location. After a set interval of not encountering a station we disable the middleware. Using Java's thread management Handler class we can easily manage this with little overhead. In practice we set the time limit before disabling to ten minutes. This handles both cases where a user may simply walk by a station and when the user finishes their journey on the Underground. With enough usage and trace data of passenger journeys on the Underground and their travels near stations it would be possible to optimize the time limits before disabling DeepOpp.

An alternative to relying on the geofencing libraries could be to only rely on the Underground location scheme discussed earlier in this report. By default many Android phones perform background WiFi scans and a lightweight background service could run which checks these identified WiFi MAC addresses to our tube graph dataset and enables the middleware upon finding a known station location. This solution burdens us with the overhead of running the background service and will only be able to run at stations that we have mapped manually. It would be a good source for a future extension.

4.2.8 Location Scheduler

Three schedulers were presented in Section 4.1.3. The Basic Scheduler and One Dimensional were covered in sufficient detail given their complexity, but the implementation of the Location Scheduler merits additional discussion. The scheduler works by locating the user on the tube and scheduling a fetch over the network for when the user is at the next station that has sufficient signal strength.

Figure 29 shows an example situation where the scheduler can be used. If we are currently at station S1 and travelling towards S4 we have several stations where we do not have any signal (S1, S2, S3). The location scheduler will find our current direction and location as S1 travelling towards S4. It then adds up the total time between these stations in order to schedule the next fetch. In the example in Figure 29 this would be six minutes.

The location scheduler uses the same tube graph used by the Signal-

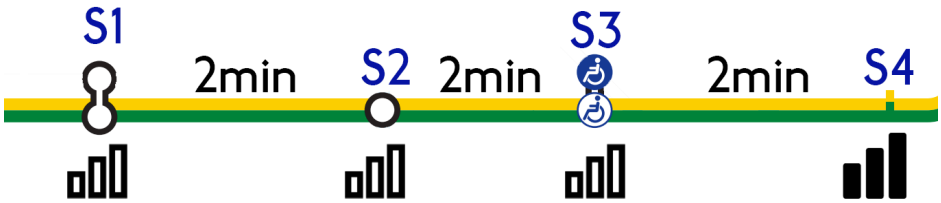


Figure 29: An example scenario where the Location Scheduler can be used.

Tracker in Part I. This common Json representation means we can rely on the same web service to retrieve updates and easily utilize the signal data collected through SignalTracker. Again, we build up an in-memory JsonObject containing all of the station objects and their neighbouring stations.

To identify the direction of travel we record the user's previous station and can infer direction from this. Once we have the direction of travel it is possible to traverse the graph of stations to find the next station that is likely to have sufficient signal. The graph also includes the estimated travel times to each of these stations and this gives us a time for when we should schedule the next fetch attempt.

4.3 Evaluation

4.3.1 Evaluating the Optimizer

Experimentation Conditions The conditions for the evaluation of DeepOpp is aimed to measure the performance and improvement that the middleware provides in real world scenarios. One of the key focuses of these investigations has been relating them directly to the real world and working with the actual conditions of the London Underground which can prove to introduce high levels of unreliability and variable network conditions as examined in Part I. For this reason we continued to experiment on the Underground rather than rely on simulators for data.

A total of about eleven hours worth of data was collected at various times riding on the Circle Line route used throughout this report. The significant amount of time spent travelling on the tube gives us confidence in our results and means that we can break down readings by time of day.

Trial	Direction	Start	End	Time
1	Clockwise	29/04/2015 07:31	29/04/2015 08:11	Morning
1	Anti-Clockwise	29/04/2015 08:15	29/04/2015 08:49	Morning
2	Clockwise	29/04/2015 08:52	29/04/2015 09:44	Morning
2	Anti-Clockwise	29/04/2015 09:45	29/04/2015 10:19	Morning
3	Clockwise	29/04/2015 12:32	29/04/2015 13:06	Afternoon
3	Anti-Clockwise	29/04/2015 13:08	29/04/2015 13:39	Afternoon
4	Clockwise	29/04/2015 16:44	29/04/2015 17:23	Evening
4	Anti-Clockwise	29/04/2015 17:25	29/04/2015 18:04	Evening
5	Clockwise	30/04/2015 07:29	30/04/2015 08:11	Morning
5	Anti-Clockwise	30/04/2015 08:22	30/04/2015 09:02	Morning
6	Clockwise	30/04/2015 09:12	30/04/2015 09:47	Morning
6	Anti-Clockwise	30/04/2015 09:55	30/04/2015 10:31	Morning
7	Clockwise	30/04/2015 12:30	30/04/2015 13:04	Afternoon
7	Anti-Clockwise	30/04/2015 13:05	30/04/2015 13:40	Afternoon
8	Clockwise	30/04/2015 16:42	30/04/2015 17:19	Evening
8	Anti-Clockwise	30/04/2015 17:24	30/04/2015 18:01	Evening

Table 4: Data collection trial journeys with times.

Each trial consists of a round-trip. Start and end times are broken up into the direction travelled. Four morning trials were conducted during rush hour and two trials were conducted during the evening rush hour. During these trials the train carriages ranged from busy to very packed and near capacity. Another two trials were carried out during the afternoon when the train carriages were generally quite empty with seats available. Specific timings are displayed in Table 4.

The data was collected on two LG G3 smartphones running Android 5.0 Lollipop with EE sims and data plans. On each trial one phone ran the DeepOpp middleware with the optimizer enabled and one with the optimizer disabled. Both phones ran the one dimensional scheduler at an interval of thirty seconds.

4.3.2 Data Collected Summary

The data was logged and written to files using the 'logback-android' library[40]. These files were retrieved at the end of data collection and parsed using python scripts and then imported into an SQLite database.

Battery The optimizer will have some overhead in order to be able to run, but it also aims to save battery by filtering out media items based on current phone state. By measuring battery we can compare the effectiveness of the optimizer. In order to do so we record the battery percentage throughout each of the trials.

Database Size Storage space is another resource consumed by DeepOpp. DeepOpp will cache pre-fetched items so this metric will give us an indication of the overall amount of space that is consumed by the middleware. We can also compare the difference in data generated and stored by the optimizer. SQLite stores a single database file on the Android filesystem that we use for our cache. We monitor the size of this database file and measure it.

Filtering This measures how much of the content is filtered out by the optimizer code. We record the number of media items to be downloaded as a proportion of the total items available for download. For the phone not running the optimizer code no content is ever filtered.

Network Traffic One of the goals of the optimizer is to reduce the amount of network usage and we can measure both of the amount of data transmitted and received independently. We can also compare this with the amount of items that were filtered by the optimizer. To measure this we rely on Android's TrafficStats[41] library which comes as part of the API. It allows us to retrieve the transmitted and received data for the app process.

Fetches In order to compare and benchmark the two phones we also measure each time that we make an attempted fetch of data from Facebook's API.

[40] `http://tony19.github.io/logback-android/`
[41] `http://developer.android.com/reference/android/net/TrafficStats.html`

4.3.3 Results

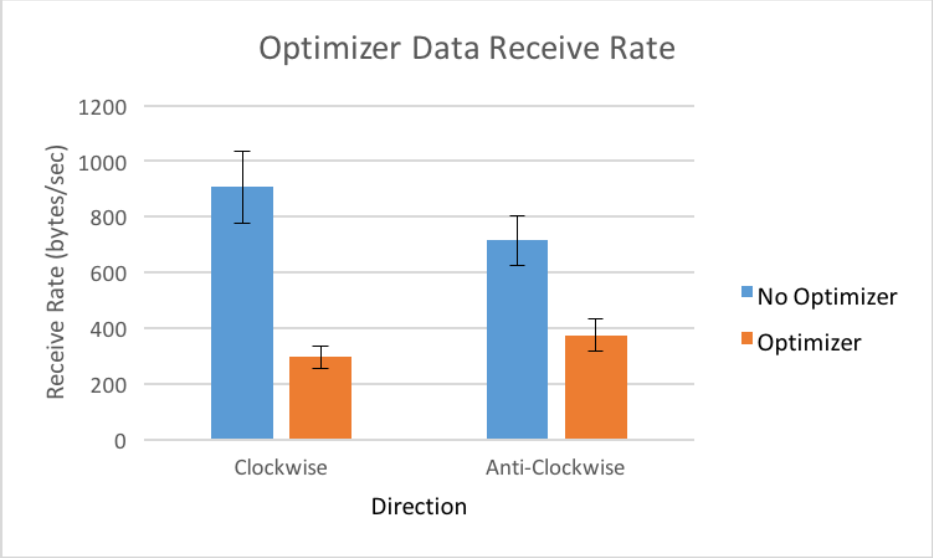


Figure 30: Average download rate for DeepOpp with standard error.

In Figure 30 we can see that the optimizer is dramatically reducing the amount of data received. We can see that the optimizer is actively filtering out contents meaning that we are seeing about 33% of the data received on the clockwise direction and approximately 52% of the data on the anti-clockwise direction. The t-test data shown in Table 5 shows that both direction are statistically different.

Direction	T-Stat	Critical Value
Clockwise	4.24	2.31
Anti-Clockwise	3.02	2.18

Table 5: T-test results for recorded download rates of DeepOpp.

4.3.4 Battery

Figure 31 shows the power used by DeepOpp during one of the trials in the anti-clockwise direction. The data displayed is the moving average over the previous 100 seconds. The data for this battery measurement has been

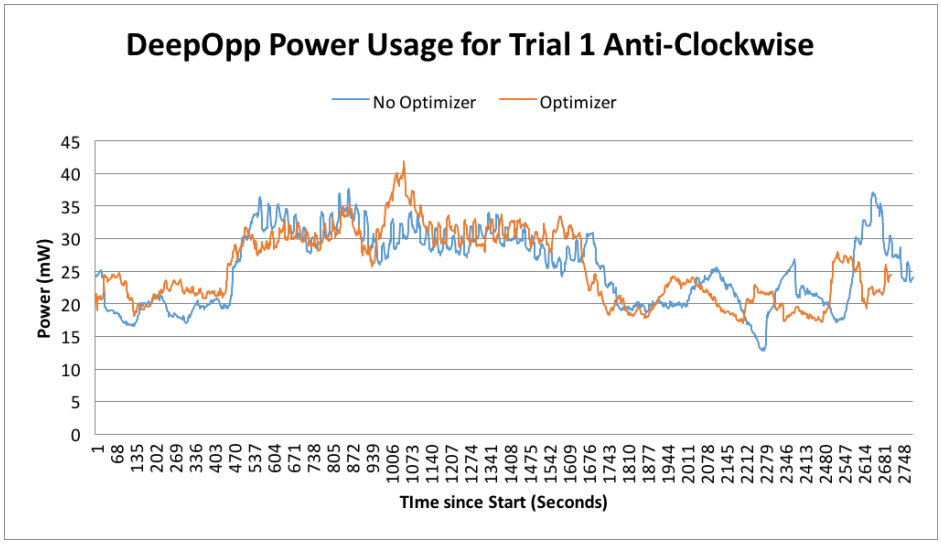


Figure 31: Power consumption of DeepOpp over the course of a single run. Data recorded with PowerTutor.

collected by the PowerTutor[42] application which can give breakdowns for specific applications running on an Android phone and used in [51]. Despite the middleware's aims it appears that the power used by the application does not differ significantly between running DeepOpp with the optimizer enabled and with it disabled. The overhead in running the optimizer negates the savings in only running at any given time.

4.3.5 Fetches and Content Received

We can see in Figure 32 that the size of the our cache databse grows much slower when running the optimizer code. This result agrees with our results for the data transmitted shown in Figure 30.

4.3.6 Comparison to Existing Solutions

The O2SM OfflineFacebook provides a prefetching and caching design that aims to reduce battery drain and network bandwidth. Their efforts are targeted towards ranking content, but their O2SMPS problem is also formulated

[42] `http://ziyang.eecs.umich.edu/projects/powertutor/`

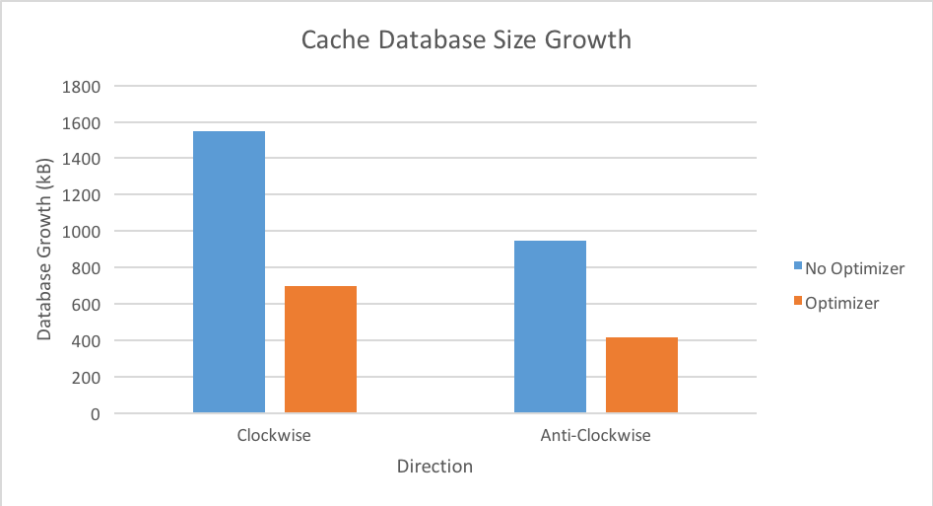


Figure 32: Cache Database size on disk.

in terms of battery usage and phone state. We would like to compare our middleware caching implementation to theirs. The authors' app is not currently available for download[43], so it is not possible to compare their implementation directly. Their algorithm used to solve the O2SMPS problem they present is non-trivial to implement. In order to provide a measure of how well DeepOpp performs we run a trial comparing our LocationScheduler to the basic scheduler which has no knowledge of our position and direction on the Underground. This will provide the best estimate as to the improvements DeepOpp delivers.

The trial was run on the same section of the Circle Line used throughout the report and signal strength data is based on a combination of the signal data collected manually with SignalTracker and the additional crowd-sourced data gathered as part of the optimization to SignalTracker. The same two LG G3 phones were used under the same experimentation conditions previously discussed. DeepOpp makes requests to Facebook and then additional media item requests.

Tables 6 and shows some of the metrics collected comparing the basic scheduler to the location scheduler. The number of requests was significantly more for the basic scheduler and it provided over 30 times as many success full retrievals of metadata. Figure 33a shows the percentage of successful

[43] `http://www.ics.uci.edu/~dsm/oFacebook/`

Table 6: Comparison of DeepOpp Schedulers

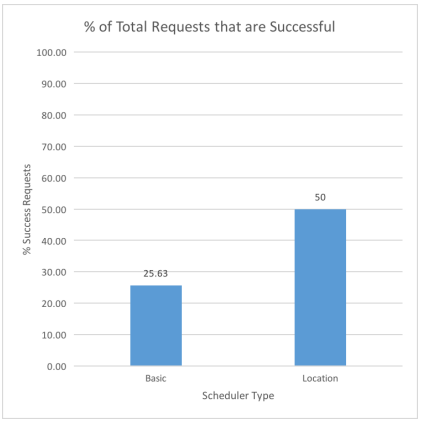
	Scheduler	
	Basic	**Location**
Total Requests	359.00	6.00
Successful Requests	92.00	3.00
Total Power (mW)	405053.00	5118.00
Power per Successfull Request	4402.75	1706.00
Power per Media Item	442.20	176.48
Power per Byte	0.04	0.01

requests. While the Basic Scheduler made many more requests, a larger proportion of these requests failed. Reasons for failure were likely due to network errors as reported by the Facebook SDK. The Location Scheduler was nearly twice as likely for one of its requests to be successful.

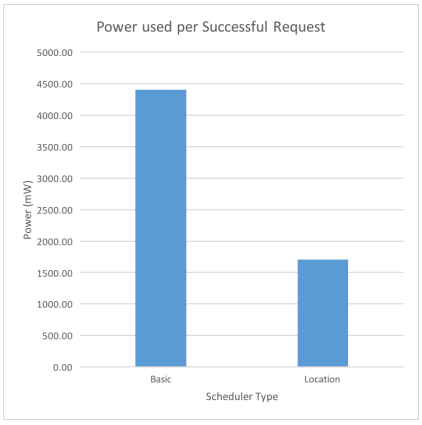
The total power used by each scheduler is shown in Table 6. The power data was collected using PowerTutor. Due to the significant number of requests made by the Basic Scheduler it consumes much more power. The total power consumed will depend on the interval at which the Basic Scheduler runs and so is not representative by itself. The amount of power per successful request displayed in Figure 33b shows that the Location Scheduler needs 2.58 times less power for each successful request of data. This is due to the wasted power the basic scheduler uses on failed attempts at making requests.

Figure 33c shows the total amount of power used by each app divided by the number of media items retrieved. This is similar to results in Figure 33b as most feed items from Facebook contain some media item. In our trial each metadata request contained ten feed items, which is why the power for each media item is approximately a tenth of that for each metadata request. Measuring the media items, the Location Scheduler is 2.51 times as efficient at getting media items successfully.

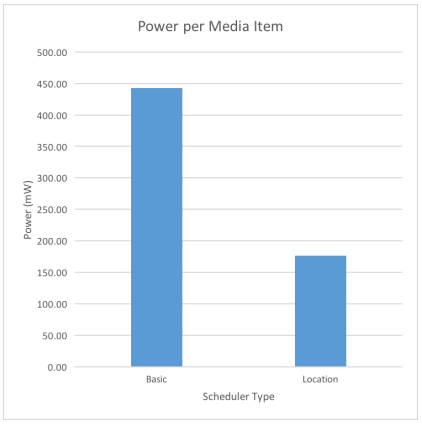
Finally, we look at the efficiency of data downloaded compared to the power used by each of the two Schedulers. Figure 33d shows the total power used divided by the total bytes downloaded over the course of the experiment. The Location Scheduler is a lot more efficient for the amount of data used. The Basic Scheduler is going to schedule downloads at points with little signal strength where the download may begin, but fail due to the low signal strength in that location. Each time this happens the data downloaded



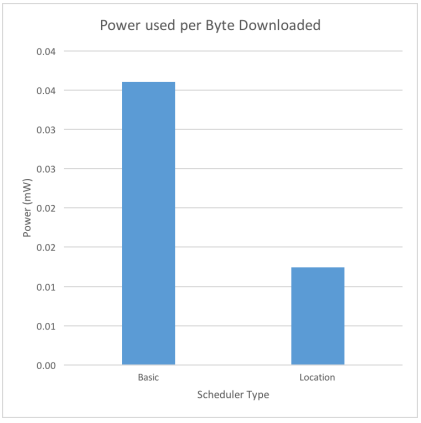
(a) Percentage of Successful requests.



(b) Power used divided by number of successful metadata requests.



(c) Power used divided by media items requested.



(d) Power used divided by number of bytes downloaded.

Figure 33: Evaluating the Location Scheduler against the Basic Scheduler.

is wasted, but it still contributes to a user's data allowance and consumes energy on their battery. The overhead in making these failed requests will mean more used power.

Each of the metrics shows that the Location Scheduler offers a significant improvement over a naive scheduler. The Basic Scheduler is much more likely to waste data and energy on failed requests than the Location Scheduler we have designed and implemented. For each amount of content retrieved we using much less power with the Location Scheduler.

4.3.7 Working with the Research Team

As part of the development of the tube middleware, I worked as part of a team with four researchers. The team consisted of Lampros Lamprinos, Julie McCann, and Di Wu[44] from Imperial College and Dmitri Arkhipov[45] of University California, Irvine. I maintained communication with them through in person meetings, Skype calls, emails, and instant messenger. The code for the optimizer found in `CacheOptimizer.java` was written by Dmitri. Working remotely with this experienced team provided a unique opportunity to participate as part of a research team at a high level.

Contributions to the Team The work discussed in this section covers the contributions I made to the team and the work we set to accomplish. My main contributions are:

- Design and implementation of the middleware architecture. The design and classes discussed are all my own work and were mainly driven by decisions I undertook. Dmitri wrote the optimization class which was integrated as part of my work.

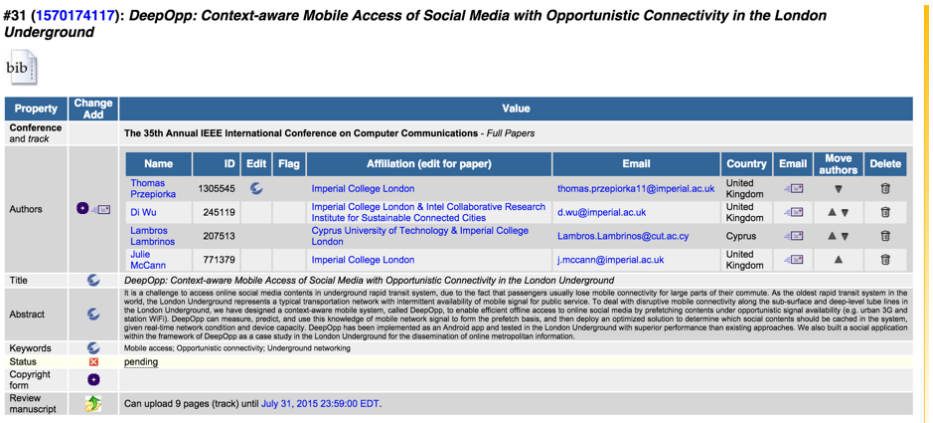
- Faceboop client application. All architectural design, code and UI design was implemented by myself.

- Testing and data collection. Infrastructure to support data collection, testing in the real world, and riding on the Underground to get real results. Initial analysis and parsing recorded data into usable formats. All analysis and results presented in this report.

[44] {l.lamprinos, j.mccann, d.wu}@imperial.ac.uk
[45] darkhipo@uci.edu

- Location Scheduler design, implementation, and testing.
- Discussions and contributions. I was involved in discussions and many design decisions with the team with my input being considered, valued, and factored into the project.
- Additional contributions. Preparing screenshots and writing up sections of the paper among other smaller tasks.

Submission to IEEE INFOCOM We are proud of the results of the of the middleware caching layer and what it accomplishes and the progress it has made in the field and have been working diligently to publish the results. As of writing we are aiming to publish the results at the IEEE INFOCOM 2016 Conference on Computer Communications in San Francisco [46]. We are confident that we can get our work included at this prestigious conference.



#31 (1570174117): *DeepOpp: Context-aware Mobile Access of Social Media with Opportunistic Connectivity in the London Underground*

bib

Property	Change Add	Value
Conference and *track*		The 35th Annual IEEE International Conference on Computer Communications - *Full Papers*
Authors		Name / ID / Edit / Flag / Affiliation (edit for paper) / Email / Country / Email / Move authors / Delete Thomas Przepiorka 1305545 — Imperial College London — thomas.przepiorka11@imperial.ac.uk — United Kingdom Di Wu 245119 — Imperial College London & Intel Collaborative Research Institute for Sustainable Connected Cities — d.wu@imperial.ac.uk — United Kingdom Lambros Lambrinos 207513 — Cyprus University of Technology & Imperial College London — Lambros.Lambrinos@cut.ac.cy — Cyprus Julie McCann 771379 — Imperial College London — j.mccann@imperial.ac.uk — United Kingdom
Title		*DeepOpp: Context-aware Mobile Access of Social Media with Opportunistic Connectivity in the London Underground*
Abstract		It is a challenge to access online social media contents in underground rapid transit system, due to the fact that passengers usually lose mobile connectivity for large parts of their commute. As the oldest rapid transit system in the world, the London Underground represents a typical transportation network with intermittent availability of mobile signal for public service. To deal with disruptive mobile connectivity along the sub-surface and deep-level tube lines in the London Underground, we have designed a context-aware mobile system, called DeepOpp, to enable efficient offline access to online social media by prefetching contents under opportunistic signal availability (e.g. urban 3G and station WiFi). DeepOpp can measure, predict, and use this knowledge of mobile network signal to form the prefetch basis, and then deploy an optimized solution to determine which social contents should be cached in the system, given real-time network condition and device capacity. DeepOpp has been implemented as an Android app and tested in the London Underground with superior performance than existing approaches. We also built a social application within the framework of DeepOpp as a case study in the London Underground for the dissemination of online metropolitan information.
Keywords		Mobile access; Opportunistic connectivity; Underground networking
Status		pending
Copyright form		
Review manuscript		Can upload 9 pages (track) until July 31, 2015 23:59:00 EDT.

Figure 34: Submission entry for IEEE INFOCOM.

4.4 Limitations

The DeepOpp middleware provides intelligent pre-fetching and caching on the Underground. The integration of the optimization scheme has reduced data consumption by 52% and the location scheduler provided 2.5 times gains in power efficiency by limiting failed fetch attempts. The scheduling

[46] `http://www.ieee.org/conferences_events/conferences/conferencedetails/index.html?Conf_ID=30862`

for DeepOpp's fetches still resulted in a 50% failure rate of retrieving content through the network. More accurate ways to identify the location of the phone and signal strengths would help reduce this failure rate. Incorporating bandwidth measurements as discussed in Section 3.5.9 would be helpful in determining accurate measurements. The geofencing technique to enable DeepOpp did not operate quickly and accurately. Relying on our WiFi mappings instead of Android's GPS based geofencing library could improve the accuracy of this feature. In predicting our path we oversimplified the model to include one route when in fact the Underground is a complex series of interchanges and lines that share routes and split into separate directions. Research and implementations exist that can use historical travel data to provide more accurate predictions of the paths and destinations of users. DeepOpp only downloads data when most applications will allow users to post data back to a service (comments, statuses, photos, etc.). DeepOpp could be expanded to include this functionality and incorporate some of the same techniques to schedule these uploads at stations that have high upload bandwidth available. Current signal data is used as an overall average, but research from Tan[44] shows there are a number of factors that influence the effective bandwidth that can be used on a mobile network. With a larger set of data that can be broken down to specific times our predictions would be stronger.

4.5 Conclusion

The DeepOpp middleware provides intelligent pre-fetching and caching of social media data on the Underground where users suffer from interruptions in their data collection. The nature of travelling on an Underground transport system means that there is constantly variation in the network availability and we may be presented with only limited opportunities to fetch data. DeepOpp provides a seamless experience where client apps rely on a background middleware that understands location, network conditions and a profile of the phone to schedule updates to content. Integrating a novel optimization technique means that the middleware will selectively filter out content to provide a positive user experience and maximise the amount of relevant items to be displayed to the user. Our evaluation shows a reduction of over half of network traffic for providing this content to the user. The Faceboop client app is a usable Facebook client application that can authenticate with Facbook to display a user's main feed with comments.

5 Part III - MetrOpp: A Protocol for Disseminating Ordered Content on the Underground

5.1 Introduction

DeepOpp introduced a novel caching scheme and implementation that provided promising results by targeting specific content under constrained situations. It operates on the surface level lines, but relies on intermittent individual network connectivity. In Part III we introduce MetrOpp which provides the final piece of network coverage on the Underground by targeting the deep lines. MetrOpp relies on peer to peer connections to introduce data into temporary networks created deep below the ground-level to create a delay tolerant network that can provide users with a stream of information. We will discuss the design of the system, implementation details, and evaluate the protocol and implementation.

5.2 Design

Our goal is to be able to share streams of data on deep line carriages of the Underground that do not have access to the internet. The data should be continuous, linear, and update frequently. At each stop on the Underground passengers enter and leave a train. Some of these may be entering from other deep line trains and some will be entering from either surface level trains or directly into the station through a street-level entrance. MetrOpp will use passengers entering from surface level to introduce new and fresh content to an ad-hoc network on a train carriage and disseminate this data throughout participating phones on the network. MetrOpp will need to deal with a highly dynamic network as passengers are constantly entering and leaving a carriage every few minutes.

5.2.1 Choosing a Wireless Access Standard

Device to device opportunistic networking has been approached before on Android phones. The WLAN-Opp platform (cf. Section 2.4.6) provides peer to peer communication using a combination of WiFi and Bluetooth. The theory and design behind the system would be a suitable fit for the aims

of this part of the project. WLAN-Opp claims to offer fast and power efficient communication on unrooted Android phones by piggybacking off of the internet-sharing tethering feature to create pseudo-ad-hoc WiFi networks. This method means that the user is never presented with any pairing dialogues and we can connect quickly to an available network of phones. This offers a major UI advantage over using either Bluetooth or WiFi Direct which require a pairing process to authenticate between any two devices for the first time.

In practice WLAN-Opp was not reliable and despite significant effort in getting the basic platform and demo chat app working on the devices available, it could not reliably find peers, maintain a connection, or deliver messages. Problems may have occurred due to different devices and OS versions being used or due to other unknown issues.

After attempting to build MetrOpp on top of WLAN-Opp we moved focus to deciding a new medium for data transmission. The three considered here are Bluetooth, WiFi Ad-hoc, and WiFi Direct (also referred to as WiFi P2P in Android's implementation[47]). Each of these is a realistic possibility to use, so we can compare them and justify our choice.

In Section 2.4.1 we looked at the wireless communication standards available on Android. The range of Bluetooth is particularly limiting as all of the London Underground rolling stock trains are over 100 metres (with the exception of the Waterloo and City line trains). WiFi allows us to cover a much larger area of the train carriage and makes it more likely we can set up a single network that covers the entire carriage. Bluetooth's bandwidth also limits the extensibility of the protocol to support multimedia images. Ad-hoc wireless isn't supported on un-rooted Android phones so is not a realistic solution for deployment. It would be an ideal choice given it's resilience to dynamic network topology changes and lack of reliance on a single access point or node. WiFi Direct by itself offers the best solution for achieving our goals.

Support for WiFi Direct is available on all four of the test devices we were using as part of the project and is supported on many other modern smartphones. As the technology is relatively new many previous attempts at creating mobile ad hoc networks have not been able to use the technology so this is a good opportunity to further explore its applicability in the context of a user application of a MANET running on smartphones. The listed range of

[47] `http://developer.android.com/guide/topics/connectivity/wifip2p.html`

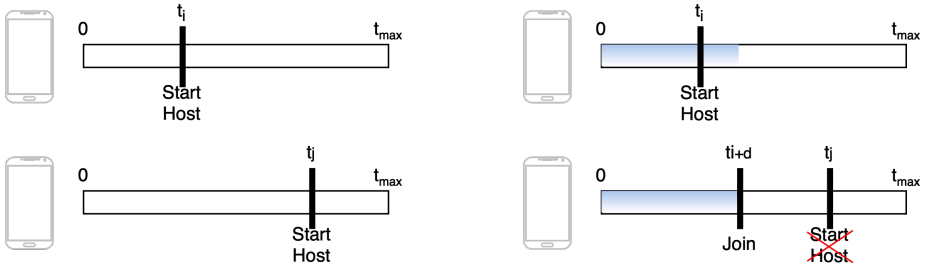
100 metres can cover most of a train carriage even if the Group Owner (GO) is not in the middle carriage. The high data bandwidth means it is a suitable medium for future extensions when transfer of media items such as pictures, animated GIFs, or videos could be shared through the network. WiFi Direct has a reliance on a single access point (the GO) which means if this one node drops from the network we will lose the network. We hope to overcome this limitation imposed by the underlying protocol by building MetrOpp's higher level protocol around this. While there are several supported authentication modes, the least intrusive requires the user to accept a dialogue prompt allowing a connection from a previously unknown device for the first time. In a real deployment scenario this would be a very large problem as the protocol is designed to connect to strangers and this authentication would result in a poor experience for the user and potentially open up security issues if the user becomes accustomed to accepting new connection requests. According to posters online it is possible to automatically bypass this user prompt, but requires the phone to be rooted [48]. We will not address this UI issue here, but it is a limitation of the chosen technology. Instead we will focus on an efficient protocol that is designed to be effective in the space and time requirements imposed by an underground mass transport system.

There are several routing protocols that have been studied and implemented as part of delay tolerant networks (DTNs) (cf. Section 2.5). The particular topology constraints of WiFi Direct and information flow required by MetrOpp will dictate how we define our communication protocol. It is important that we are not sending or routing data to any particular destination, but rather disseminating it amongst all participating nodes. Gossip protocols are effective at this task and essentially flood the network, but with certain constraints. One of the main determining factors is our use of WiFi Direct which only acts as a pseudo-P2P network. We must still have a single GO in the networks we form, but we can support creating networks with many different devices that are clients. For this reason we can use the GO to coordinate communication on our application level protocol.

5.2.2 Roles

In the MetrOpp protocol we propose the following roles and rules.

[48] `http://stackoverflow.com/a/18629705/1257925`



(a) Potential hosts schedule host creation.

(b) A host has been created and other nodes cancel scheduled host creation.

Figure 35: MetrOpp Host Creation

Host The host is the Group Owner of the WiFi Direct Group. The host broadcasts their service using the Service Discovery protocol available on Android phones and communicates with each of the devices. The host maintains a list of the connected clients in the current group along with their IP addresses and ports.

Clients Clients connect to host WiFi Direct groups and share their content information and subscribe to get updates about any new content made available to the group.

5.2.3 Host Creation

We want to minimize the number of hosts on a train carriage. The ideal situation is that there is one host and no disjoint networks of nodes. On initialisation the node will search for any service broadcasts. If it finds a service broadcast from a host advertising the MetrOpp service then it will join that host's WiFi Direct group. If no service discovery is found then we want the client to assume the role of the host.

It is especially likely in our context that we may have multiple nodes that, at the same time, search for an existing host and find none. This will happen when a GO disconnects. We want exactly one of these nodes to become the host. To handle this we use a probabilistic method to create the host based on a technique used in WLAN-Opp. In the first stage each node schedules a timeslot to set it self up as a host as shown in Figure 35a. In this Figure the first phone randomly gets a slot at t_i and the second phone at t_j. At t_j

the first phone assumes the role of a host. After a service discovery delay (d) at time t_{i+d} the host is visible to neighbour nodes as shown in Figure 35b. At this point the second phone joins the first's network and cancels its own host creation slot. WLAN-Opp uses a probability that is proportional to the inverse of the number of neighbours that each node has. This means that nodes that have more neighbours they can reach directly will have a lower probability of assuming the host role. This is the opposite of what we want as the ideal new host will be within range of the most nodes. For this reason we select the number uniformly based on a threshold (t_{max}).

If we have a case where multiple hosts are created then we will allow this continue. We are assuming a highly dynamic topology and so it will not be a long amount of time before the additional hosts depart the tube carriage.

5.2.4 Finding and Joining a Host

When nodes scan and find a host advertising the MetrOpp service through service discovery they will join them. If multiple hosts are available then they will join the first one they find in the search. The approach used in WLAN-Opp was to randomly choose a host with uniform probability. But rather than waiting to find all service discovery broadcasts we simply join the first one speeding up the connection time and maximising the time spent as part of a network.

5.2.5 State

The state is the content that we wish to be able to distribute through MetrOpp. States are strictly ordered. We say that:

$$S_i < S_j \tag{4}$$

if S_j has newer information than S_i.

The state will be generated and updated from external sources. This is likely a web server publishing updated ordered content. This means that nodes may be in different locations, but still hold the same state. For these examples we assume that: $S_i < S_j < S_k$.

5.2.6 Protocol Messages

INITIAL JOIN(S_c) Upon connecting to a MetrOpp WiFi Direct group the client will send this message which indicates it has just joined the group

and includes its current state S_c^i.

OWNER_REPLY_UTD() If the host receives an INITIAL_JOIN message from a client S_c^i such that $S_c^i >= S_h$ then the host sends this message informing the client that their state is up to date and no further content exists.

OWNER_REPLY_NEW_CONTENT(S_h) If the host receives an INITIAL_JOIN message from a client S_c^i such that $S_c^i < S_h$ then the host sends this message informing the client that the newer state Sh is available and including it in the message to client.

NEW_CONTENT(S_c) Similar to OWNER_REPLY_NEW_CONTENT, but for general new content. The host sends this to all clients that need to be updated with new content that has become available to the host.

Similar to the Trickle protocol we do not manage any heartbeats as we don't expect any exiting nodes to be able to gracefully close their connection.

5.2.7 Scenarios

We now look at some scenarios to illustrate how the MetrOpp protocol works in propagating information.

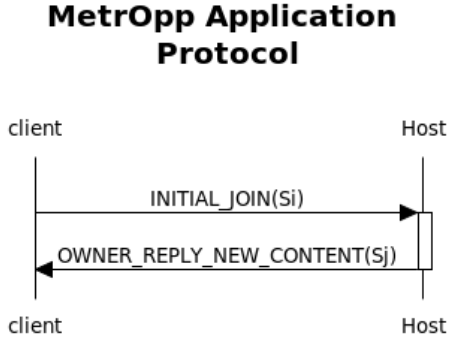


Figure 36: A client joins with a stale state.

Figure 36 demonstrates an example of a client joining a host where the host has a newer state than the client joining. We see the initial join allows

the host to determine if it should reply with OWNER_REPLY_UTD or as is the case in Figure 36 it replies with the new content.

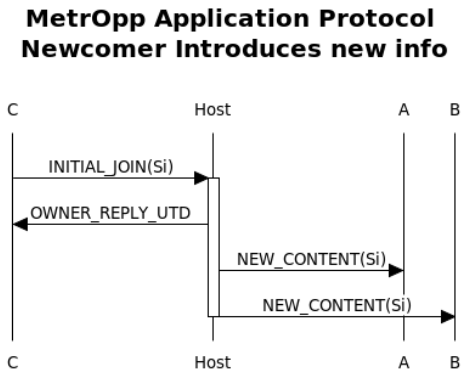


Figure 37: A client joins with a fresh state.

In Figure 37 a group involving the Host and clients A and B is already formed and has a state less than S_i. When C joins it sends S_i to the host which then proceeds to store it locally and forward this newest state to both A and B.

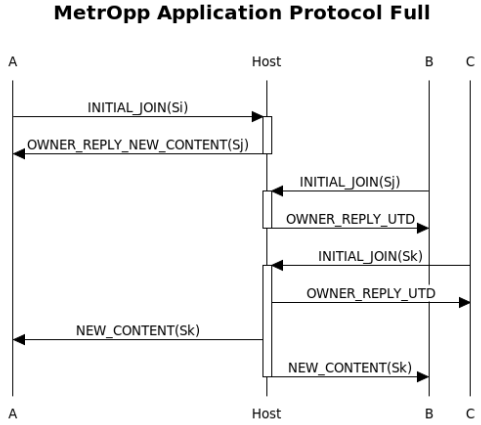


Figure 38: A more complete example of the MetrOpp protocol.

In Figure 38 we see a more complete example. A joins and sends its local state of S_i, but the Host replies with the more recent state S_j. B then joins

with S_j as well and the Host informs B that it is up to date. Then C joins with S_k which the Host stores and forwards on to A and B.

We look at a timed example of the exchange in Figure 38 in the context of the Underground. Assume we are only considering the deep tunnel sections of the lines.

15:00 Alice enters the Piccadilly line from above ground.

15:05 Harry enters the Northern line from above ground and Bob enters the Victoria line from above ground.

15:10 Alice switches to a Northern line train that Harry is on and joins his network. Alice gets updates up until 15:05.

15:15 Bob switches to the Northern line train and joins the group. No one's state is updated.

15:15 Carol enters the Northern line from above ground and joins the group. She sends updates until 15:15 to Harry who then sends these onto Alice and Bob.

At the end of this exchange all parties have the most recently available state.

5.3 Implementation

We have introduced the problem of providing fresh content on deep line trains that lack signal. We have then proceeded to discuss the MetrOpp protocol, it's specific design decisions, and chosen the core technology to base it on. We now focus on the implementation of a prototype application that demonstrates how WiFi Direct can be used in a mobile ad hoc network and show how the MetrOpp protocol can be used to provide a valuable service for users.

5.3.1 Demonstration Scenario

To demonstrate the abilities of MetrOpp we have built an Android application that provides users with live text commentary for football matches. With fans not able to follow each match in person or on television many will use a service that provides them with up to the minute updates about what is going on during a live match. Almost all major sports websites will

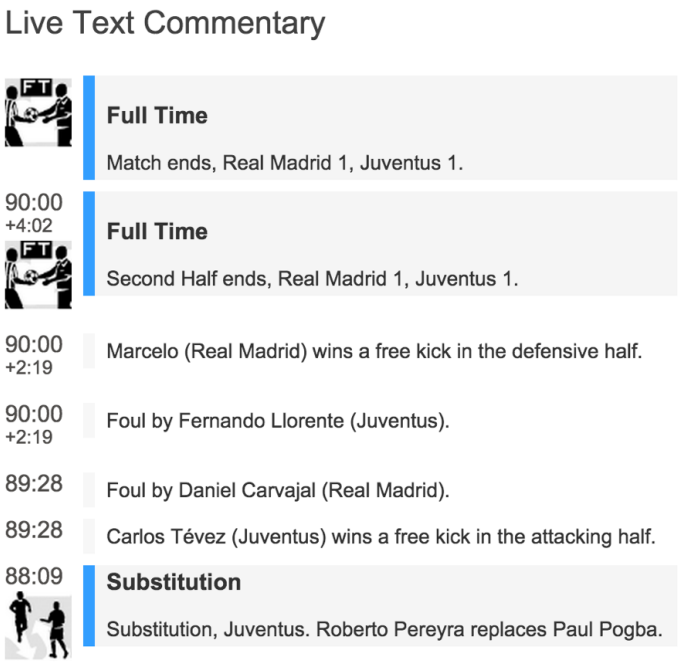


Figure 39: Live text football commentary on BBC Sport.

provide some level of this commentary and apps such as OneFootball[49] are amongst the most popular in the Play app store. An example feed from BBC Sport's website can be seen in Figure 39. The results and actions of a match are highly time sensitive as fans want to stay as up to date as possible as new commentary is coming in every minute. There are a limited number of matches played at a single time which means fans are often interested in the same games. The rapid number of updates that serve a large audience makes it an ideal problem that can be solved by MetrOpp. There are other use cases that could also be used as part of MetrOpp such as disseminating public emergency information, breaking stories stories, public Twitter feeds and others.

5.3.2 Managing WiFi Direct and MetrOpp Protocol

Our application will provide an interface for users to view each event item. It also provides all of the WiFi Direct connection and management logic.

[49] `http://www.onefootball.com/`

The WiFiP2PManager class included in Android's API helps us manage the key interactions and lifecycle of our WiFi Direct connections. Using ActionListeners we can get the results of the various searching and connecting requests we make with the P2P manager.

On the start of the application we add a service request and listen for any broadcast services that match the String 'MetrOpp'. Once a client finds a MetrOpp service broadcast it initiates a connection request using WPS Push Button Connection to authenticate. This is the least intrusive authentication method available through WPS and displays a dialogue box on the remote device to accept the connection. Other WPS methods involve displaying and entering PINs. The WiFiP2PManager will notify us upon a successful connection and we proceed to set up a server which can handle incoming connections.

The WiFiP2P classes provide the functionality to discover peers and establish connections and manage the connection lifecycle. Once we have established a connection we rely on Java's Socket classes to transfer data and send protocol messages between devices. Once we establish a connection we start a ServerSocket. The ServerSocket uses a blocking call while waiting for connections so we run the server in a separate thread using FileServerAsyncTask(Object, Long, ProtocolMessage). It does not take in any parameter objects or post any status updates. Upon completion of the socket we return the message to our main thread for processing. We immediately start a new FileServerAsyncTask to allow us to receive subsequent messages.

Each socket communicates over a network port. To avoid clashing with ports used by system processes or other apps available on the phone we use automatic provisioning to obtain an available port. For the host node we include this port in the service broadcast. For service broadcasts we rely on the DNS Service Discovery which includes format rules for TXT Records. The DNS TXT records provide additional information about an instance of a service discovery in a key/value pair format. We use these TXT records to include the port that the host is listening on. Each message from a client also includes the port that the host can reply to. Device addresses are included in both the WiFi Direct and Socket communications so we don't have to add any worthwhile consideration to obtaining these.

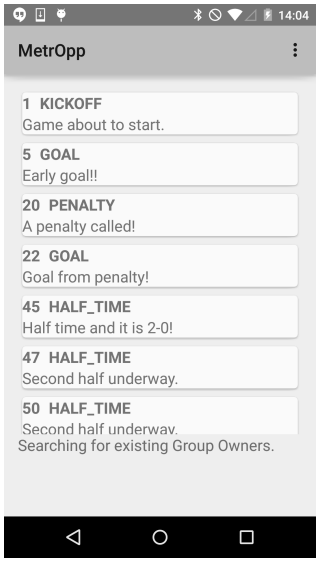


Figure 40: The MetrOpp app UI.

5.3.3 App UI

Figure 40 shows the main screen of the demo application. A stream of match commentary is displayed to the user. As soon as new information becomes available it is added to this stream for the reader to consume.

5.4 Evaluation

We have outlined the design of a delay tolerant network protocol targeted at the London Underground and presented an implementation that runs on Android phones using WiFi Direct. We now proceed to evaluate the possibility of running a delay tolerant network on the Underground and how MetrOpp could be used to disseminate information.

Figure 41 shows the average connection and state update phases for a client phone running the MetrOpp application. The initialisation phase before the searching begins takes a negligible amount of time. The overall average time for a client who joins an existing network to search, connect, and received the most recent state is 5.13 seconds. This is sufficient to handle the entrance and exits of people on a journey.

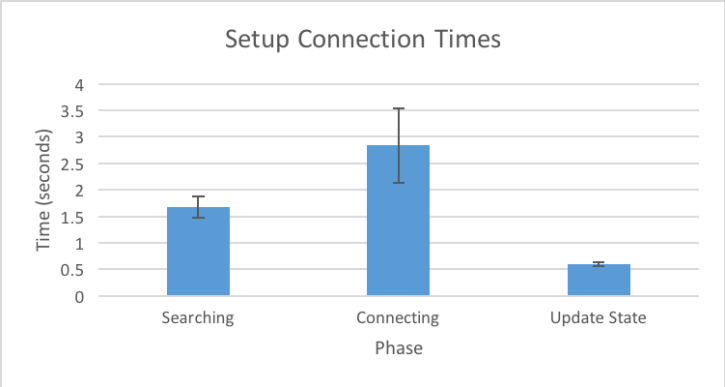


Figure 41: MetrOpp connection and state exchange timings.

5.4.1 Connection Times

One of the key concerns for the MetrOpp app on Android is whether it can discover, connect, and exchange state within the time constraints imposed by the Underground. We have tested the various connection timings for a client connecting to a host and going through the WiFi Direct discovery and MetrOpp state sharing phase.

5.4.2 The ONE Simulator

The ONE Simulator was selected due to its focus on delay tolerant networks. In the context of the ONE Simulator a host refers to any node that is part of the network.

5.4.3 Geographical Representation of the Underground

The simulator has many different movement models for nodes. One of the supported modes allows for a map of route data to be imported and have nodes follow these paths. This is the movement mode chosen for this project as we can simulate the tube routes and move nodes along these paths. The coordinates for the route definitions are provided in Well Known Text (WKT) files. WKT files define lines and polygons by specifying individual line segments, or groups of line segments. For example, a line from (0, 1) to (2, 4) is defined as:

LINESEGMENT (0 1, 2 4)

The coordinate system used by the ONE Simulator was not stated in the documentation and the standard geographical latitude and longitude coordinates were not accepted. After research into coordinate systems and checking example files it was determined that the best coordinate system to use is the Universal Transverse Mercator (UTM) coordinate system. UTM uses a 2-dimensional coordinate system making it easy to represent on a grid like area used by the ONE simulator. Coordinates are given with the longitude first then latitude.

Geographical coordinates for the tube lines was obtained from Doogal[50] in Keyhole Markup Language (KML) format. These were parsed to extract each line segment as a pair of coordinates. An online conversion tool KML Tools[51] was used to convert these coordinates to UTM format. Some of the results were spot checked to ensure they were reasonable.

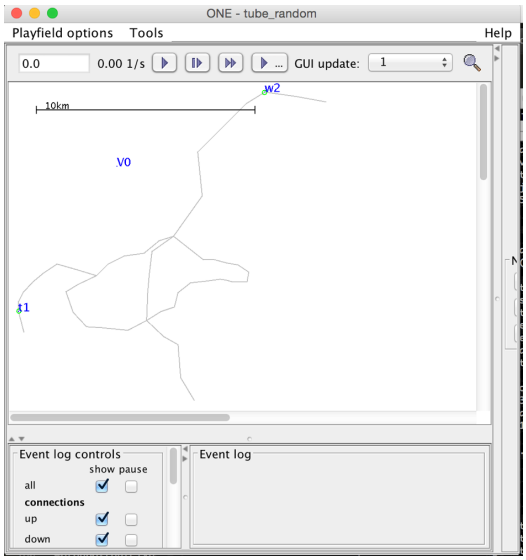


Figure 42: The Circle Line and Victoria Lines loaded into the ONE Simulator.

Each tube line had its segments added to a WKT file for that line. The resulting routes are shown in Figure 42 in the ONE simulator GUI for the Circle and Victoria line data.

[50] `http://www.doogal.co.uk/london_stations.php`
[51] `http://kmltools.appspot.com/geoconv`

5.4.4 Groups

A group is a set of nodes that share movement and routing module settings. Each group we create represents movement along a single tube line. We present our configuration options for a group and justify each choice.

Group2.bufferSize = 50M The message buffer size for each node. 50MB is the default and sufficient for our purposes of representing a text commentary for a football match.

Group2.movementModel = ShortestPathMapBasedMovement The ShortestPathMapBasedMovement uses Dijkstra's algorithm to find the shortest path between two points in the graph provided in the route file.

Group2.routeFile = data/circle_line_movement.wkt Here we define the route which this set of nodes can move along. In this case this represents the Circle line nodes. We take actual journey information based on the routes within the Circle line made by passengers[52]. The data is from midday on 2nd December, 2013. We process this data to generate a shuffled WKT file. Each route should be chosen by the simulator with a probability reflected from actual TfL journeys.

Group2.routeType = 1 A route type of 1 will send the nodes in a circular route of the track. A route type of 2 sends nodes back and forth between two points.

Group2.waitTime = 90,150 The time each node spends waiting at points. As the points represent stations we enter values of 90 and 150 seconds. The simulator will uniformly randomly choose a value in this range. This value corresponds the the amount of time a station is stopped at a platform.

Group2.speed = 7, 10 In m/s the speed of nodes. The average speed of an Underground train is 33km/h [47] which corresponds to 9.17m/s, but we lower the range of possible speeds to 7m/s to account for slowing down and speeding up when leaving platforms.

[52] `https://www.tfl.gov.uk/info-for/open-data-users/`

5.4.5 Other Simulator Conditions

The first investigation into the feasibility of the MetrOpp protocol will involve looking at the scale that the Underground presents. The Android app proves that the protocol can work with the given functionality needed. We want to know how many people would need to be using the app for it to work on a public transport network the size of the Underground. We will look at the environment conditions to evaluate the feasibility of the app.

Our first experiment runs on just the Circle line. Adding more lines made the simulator run prohibitively slow.

Message generation is done using the OneToEachMessageGenerator. The generator creates one message and delivers it to a list of hosts. We set 20% of hosts to generate messages and attempt to deliver to all of the nodes in our simulation.

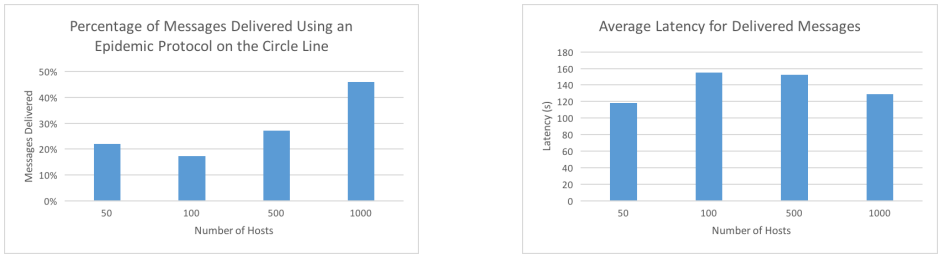
The nodes use an epidemic routing protocol. This is similar to how MetrOpp works as it attempts to deliver its messages to all other nodes (set by our message generator) and does so by contacting all nodes it comes in contact with.

We run the simulation for 9450 seconds (157.50 minutes). A football match (90 minutes) with half-time (15 minutes) runs over 105 minutes. Injury time in the Premier League averages 6 minutes[40]. Commentary typically starts 30 minutes before a match when team sheets are released. With time left over for post-match discussion this brings us to our 157.50 minutes to run the simulation.

These conditions and parameters make up our scenario. We use 10, 50, 100, 500, and 1000 nodes. The numbers used in this scenario are the total number of hosts on the entire line. For comparison, according to TfL data from Saturday, November 12th, 2012 from 15:00 to 15:15 there were 962 passenger entrances into South Kensington alone. Most football matches in the Premier League kick-off at 15:00 on a Saturday.

5.4.6 Simulator Results

Here we consider the results of running the Epidemic Routing protocol included in the ONE Simulator on the Circle Line. The simulator was run with 10, 50, 100, 500, and 1000 nodes. These are based on real passenger journeys on an accurate representation of the Circle line route. The simulator run with 10 nodes did not deliver any messages so is not included in the figures.



(a) Average number of messages delivered.

(b) Average latency for delivered messages.

Figure 43: Metrics for an epidemic protocol running on the Circle line.

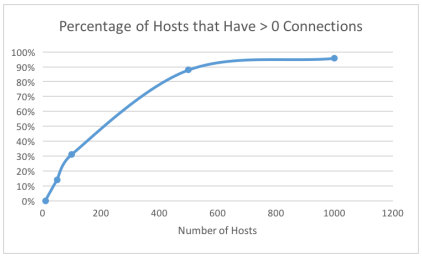
Figure 43a shows the percentage of messages delivered throughout the simulation for each of the different number of hosts. The trial with 1000 hosts is able to deliver a much higher percentage of the messages created. Figure 43b shows the average latencies for delivered messages. All are on par with each other and offer very reasonable latencies for the data considered.

The above simulation results assume hosts are constantly travelling on journeys for the entirety of the simulation. In reality host journey times will be much shorter. To account for this we use the connection data output from the previous trial and parse it to limit the host's active time on the network. Each host is given a 20 minute to be active and we eliminate all connections created when either host in a connection was in this active state.

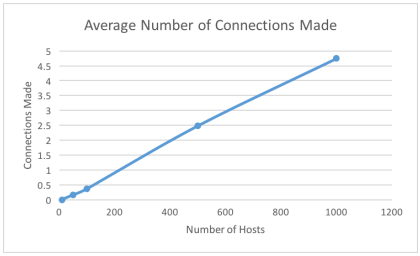
Table 7: Connection Statistics for Time Limited Nodes.

	Number of Hosts				
	10	**50**	**100**	**500**	**1000**
Number Connected	0	7	31	439	956
% Connected	0%	14%	31%	88%	96%
Average Connections	0	0.16	0.36	2.48	4.748

Table 7 shows connection statistics for each number of hosts. Here the hosts is the total number of hosts present on the entire trip and each node is active for 20 minutes. Figure 44a highlights how the increase in participating nodes has a dramatic effect in increasing the percentage of hosts that are able to establish a connection. In Figure 44b we see that the average number of connections made by hosts grows linearly as the number of hosts increase.



(a) Percent of nodes that made at least one connection.



(b) Average number of connections made.

Figure 44: Connection metrics for a simulation with a 20min node lifetime.

5.4.7 Summary of Evaluation and Limitations

We have gathered preliminary data from running the MetrOpp app on Android phones. The app requires large quantities of participants to test in the field, which means we were not able to test the app in a real world scenario. In practice the app was able to connect and find hosts quickly and updating state between phones was near-instant.

We have provided some preliminary results as to feasibility of running a device to device ad hoc network on the Underground using the ONE Simulator. Several assumptions and simplifications have been made in the models used in the simulator. The simulator did not scale to handle loading and running simulations on all of the tube lines, so we limited our scope to only consider the Circle line. The protocol was not fully implemented in the simulator and a similar epidemic protocol was used in its place. We did not trace the introduction and dissemination of messages on the network, but rather just the number of connections established between hosts that were active. These limitations mean that it is not possible to draw any definitive conclusions. However, this work has meant we have established a base for testing the app in the future. We were able to locate and parse tube map data to provide an accurate representation of the route we would deploy along. The passenger trace data was based on actual journey data published by TfL.

Future work can involve expanding the simulation to directly include an MetrOpp implementation. The protocol can also be expanded to handle media items exchange. Adding media items will mean that the size of the entire state is not negligible. using hashes of the media items means that they can be exchanged as part of the initial state transfer and specific request

messages could be sent separately when requested.

MetrOpp has shown the technical feasibility of developing an app that can use the relatively new WiFi Direct protocol to create and disseminate information across a delay tolerant network. The initial simulator results are promising and provide justification for further experiments before designing a large scale experiment.

6 Conclusion

6.1 Applications Outside London

Our work has shown success at providing improved access and efficiency when making mobile network connections on the London Underground. Targeting one of the most iconic and busiest transport systems in the world provided focus and the ability to control our results. A true test of achievement would be in understanding how our insights can be applied to the rest of the world. Wikipedia lists 160 metro systems across 55 countries[53] and the *World Metro Database* lists 195[54]. Even if only a small fraction of these exhibit similar network opportunities as the Underground the reach and applicability of our work has the potential to be used by many more users. In DeepOpp we saw how we could understand network patterns of intermittent connectivity while on a moving train to improve the efficiency and effectiveness at which we can retrieve content. Network rail has some of these similar properties but on a much larger scale. Rural communities around the world lack the ubiquitous access to network coverage that is available in a global city like London. Adapting both DeepOpp and MetrOpp to target this use case could provide benefits outside of set rail transport networks.

6.2 Conclusion

Our objectives set out a range of targets to understand and use opportunistic networks to improve access to content on the London Underground. Our SignalTracker application showed that it is possible to gather station level mobile signal strength data in an unreliable environment. Although it is not able to gather bandwidth data its results were proven when the data was used as part of the DeepOpp middleware. DeepOpp was able to integrate an efficient optimization algorithm that can help determine what media items to fetch based on current phone state conditions. A novel scheduling system designed was able to reduce the power required to fetch social media items by 2.5 times. While the location scheduler reduced failed fetch attempts it can still be greatly improved and future work could focus on providing more content at the same levels of efficiency we have achieved. Finally, the MetrOpp

[53] `https://en.wikipedia.org/wiki/List_of_metro_systems`
[54] `http://mic-ro.com/metro/table.html`

protocol has been a demonstration that it is technically possible to use newer WiFi Direct to set up mobile ad-hoc networks on the Underground and use passengers entering and leaving carriages to transport data. The three parts of this project have been developed as separate applications. With confidence in each of the parts they could be become part of one system. The SignalTracker abilities could be run in the background and provide analytics and constant updates to changing network characteristics. DeepOpp could run alongside the MetrOpp protocol providing a hybrid approach to accessing fresh data and allow a seamless experience for the user as they move between sub-surface and deep-line trains. Targeting network improvements on a Metro system is challenging, but can provide utility to millions of people each day. We have shown, by building on previous research, that tangible improvements can be made by relying on opportunistic networking.

References

[1] Usman Adeel, Shusen Yang, and Julie A McCann. Self-optimizing citizen-centric mobile urban sensing systems this paper is included in the proceedings of the. *11th International Conference on Autonomic Computing*, pages 161–167, 2014.

[2] André Allavena, Alan Demers, and John E. Hopcroft. Correctness of a gossip based membership protocol. *Proceedings of the twenty-fourth annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing - PODC '05*, page 292, 2005.

[3] Eskindir Asmare and Julie A McCann. Lightweight sensing uncertainty metricincorporating accuracy and trust. *IEEE SENSORS JOURNAL*, 14(12):4264–4272, 2014.

[4] Lars Backstrom. News feed fyi: A window into news feed. `https://www.facebook.com/business/news/News-Feed-FYI-A-Window-Into-News-Feed`, 2013.

[5] Philip Bates. After the uk 4g auction, which uk mobile operator has the most valuable spectrum portfolio? `http://www.analysysmason.com/About-Us/News/Insight/UK-4G-auction-Mar2013/`, 2013.

[6] C. Boldrini, M. Conti, F. Delmastro, and a. Passarella. Context- and social-aware middleware for opportunistic networks. *Journal of Network and Computer Applications*, 33(5):525–541, 2010.

[7] John Burgess, Brian Gallagher, David Jensen, and Brian Neil Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks.

[8] E.J. Candes and M.B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, 2008.

[9] Spirent Communications. User experience analytics system for data services. `http://www.spirent.com/~/media/Datasheets/F4L/Spirent_Datum.pdf`, 2014.

[10] Marco Conti, Franca Delmastro, Giovanni Minutiello, and Roberta Paris. Experimenting opportunistic networks with wifi direct. *IFIP Wireless Days*, 2013.

[11] Pralhad Deshpande, Anand Kashyap, Chul Sung, and Samir R. Das. Predictive methods for improved vehicular wifi access. *Proceedings of the 7th international conference on Mobile systems, applications, and services - Mobisys '09*, page 263, 2009.

[12] Transport for London. ltds-workbook-2013, 2011.

[13] Transport for London. Travel in london, supplementary report: London travel demand survey (ltds). `http://www.tfl.gov.uk/cdn/static/cms/documents/london-travel-demand-survey.pdf`, 2011.

[14] Transport for London. Standard tube map. `https://www.tfl.gov.uk/cdn/static/cms/documents/standard-tube-map.pdf`, 2014.

[15] Transport for London. Station wifi. `https://www.tfl.gov.uk/campaign/station-wifi`, 2015.

[16] Transport for London. TfL Key Facts. `https://www.tfl.gov.uk/corporate/about-tfl/what-we-do/london-underground/facts-and-figures`, 2015.

[17] Transport for London. Wifi on the london underground. `https://www.tfl.gov.uk/cdn/static/cms/documents/map-of-tube-stations-with-virgin-media-wifi.pdf`, 2015.

[18] Roy Friedman, Alex Kogan, and Yevgeny Krivolapov. On power and throughput tradeoffs of wifi and bluetooth in smartphones. *IEEE Transactions on Mobile Computing*, 12(7):1363–1376, 2013.

[19] Sarah Gallacher, Christian Jetter, Vaiva Kalnikaite, Julie A McCann, David Prendergast, and Jon Bird. Investigating the challenges of crowd sensing : Lessons from zurich. 2014.

[20] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors (Switzerland)*, 12(9):11734–11753, 2012.

[21] 3GPP Specifications Group. Grps & edge. `http://www.3gpp.org/technologies/keywords-acronyms/102-gprs-edge`.

[22] 3GPP Specifications Group. Lte. `http://www.3gpp.org/technologies/keywords-acronyms/98-lte`.

[23] Fourat Haider, Erol Hepsaydir, and Nicola Binucci. Performance analysis of a live mobile broadband-hsdpa network. *2011 IEEE 73rd Vehicular Technology Conference (VTC Spring)*, pages 1–5, 2011.

[24] Bo Han, Pan Hui, and Aravind Srinivasan. Mobile data offloading in metropolitan area networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 14:28, 2011.

[25] Ór Helgason and Ea Yavuz. A mobile peer-to-peer system for opportunistic content-centric networking. *on Networking, systems* , page 21, 2010.

[26] European Telecommunications Standard Institute. Etsi ts 127 007 v8.5.0. `http://www.etsi.org/deliver/etsi_ts/127000_127099/127007/08.05.00_60/ts_127007v080500p.pdf`, 2008.

[27] European Telecommunications Standard Institute. Etsi ts 127 007 v10.5.0. `http://www.etsi.org/deliver/etsi_ts/127000_127099/127007/10.03.00_60/ts_127007v100300p.pdf`, 2011.

[28] Klas Johansson, Johan Bergman, Dirk Gerstenberger, Mats Blomgren, and Anders Wallén. Multi-carrier hspa evolution. *IEEE Vehicular Technology Conference*, 2009.

[29] Goran Kalic, Iva Bojic, and Mario Kusek. Energy consumption in android phones when using wireless communication technologies. *MIPRO, 2012 Proceedings of the 35th* , pages 754–759, 2012.

[30] Ari Keränen. The one simulator for dtn protocol evaluation. *Proceedings of the Second International ICST Conference on Simulation Tools and Techniques*, page 55, 2009.

[31] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. Manet simulation studies. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50, 2005.

[32] Anthony LaMarca, Yatin Chawathe, Sunny Consolvo, Jeffrey Hightower, Ian Smith, James Scott, Timothy Sohn, James Howard, Jeff Hughes, Fred Potter, Jason Tabert, Pauline Powledge, Gaetano Borriello, and Bill Schilit. Place lab: Device positioning using radio beacons in the wild. *Pervasive Computing*, 3468:116–133, 2005.

[33] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. *Proceedings of the First Symposium on Networked Systems Design and Implementation*, pages 15–28, 2004.

[34] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19, 2003.

[35] Virgin Media. London underground wifi. `http://my.virginmedia.com/wifi/index.html`, 2015.

[36] Virgin Media. Wifi on london underground reaches 150th station. `http://about.virginmedia.com/press-release/9457/wifi-on-london-underground-reaches-150th-station`, 2015.

[37] Abderrahmen Mtibaa, Martin May, Christophe Diot, and Mostafa Ammar. Peoplerank: Social opportunistic forwarding. *Proceedings - IEEE INFOCOM*, 2010.

[38] OfCom. Measuring mobile broadband performance in the uk. `http://stakeholders.ofcom.org.uk/binaries/research/broadband-research/mbb-nov14.pdf`.

[39] Commonwealth Telecommunications Organisation. Ict sector information - united kingdom. `http://www.cto.int/media/ICT-data/united%20kingdom.pdf`.

[40] Huw Richards. Soccer: A game of two halves - plus added injury time. `http://www.ft.com/cms/s/0/1623969e-2bd5-11df-8033-00144feabdc0.html`, 2013.

[41] Paul Rosania. While you were away... `https://blog.twitter.com/2015/while-you-were-away-0`, 2015.

[42] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. *Proceedings of the sixteenth annual international conference on Mobile computing and networking - MobiCom '10*, page 85, 2010.

[43] Clayton Shepard, Ahmad Rahmati, Chad Tossell, Lin Zhong, and Phillip Kortum. Livelab: Measuring wireless networks and smartphone users in the field. *ACM SIGMETRICS Performance Evaluation Review*, 38:15–20, 2011.

[44] Wl Tan, Fung Lam, and Wc Lau. An empirical study on the capacity and performance of 3g networks. *Mobile Computing, IEEE* , 7(6):737–750, 2008.

[45] Sacha Trifunovic, Bernhard Distl, Dominik Schatzmann, and Franck Legendre. Wifi-opp : Ad-hoc-less opportunistic networking. *Performance Evaluation*, (Section 2):37–42, 2011.

[46] Sacha Trifunovic, Maciej Kurant, Karin Anna Hummel, and Franck Legendre. Wlan-opp: Ad-hoc-less opportunistic networking on smartphones. *Ad Hoc Networks*, 25:346–358, 2014.

[47] Metro UK. London underground turns 150: Top 10 tube facts. `http://metro.co.uk/2013/01/09/london-underground-turns-150-top-10-tube-facts-3344227/`, 2013.

[48] Amin Vahdat and David Becker. Epidemic routing for partially connected ad hoc networks. *Technical report number CS-200006, Duke University*, (CS-200006):1–14, 2000.

[49] Di Wu, Qiang Liu, Yuan Zhang, Julie McCann, Amelia Regan, and Nalini Venkatasubramanian. Crowdwifi : Efficient crowdsensing of roadside wifi networks categories and subject descriptors. pages 229–240.

[50] Shusen Yang, Usman Adeel, and Julie a. McCann. Selfish mules: Social profit maximization in sparse sensornets using rationally-selfish human relays. *IEEE Journal on Selected Areas in Communications*, 31(6):1124–1134, 2013.

[51] Ye Zhao, Ngoc Do, Shu-ting Wang, and Cheng-hsin Hsu. O 2 sm : Enabling efficient offline access to online. (i):445–465, 2013.

[52] Xuejun Zhuo, Wei Gao, Guohong Cao, and Sha Hua. An incentive framework for cellular trafc ofoading. pages 1–15, 2013.