Sunil Shah

Distributed Twitter

Part II, Computer Science Tripos

Fitzwilliam College, 2009

Proforma

Name:	Sunil Shah
College:	Fitzwilliam College
Title:	Distributed Twitter
Examination:	Part II Computer Science, June 2009
Word Count:	11, 337 approximately
Project Originator	Nishanth Sastry
Project Supervisor:	Nishanth Sastry

Original Aims of the Project

To implement and simulate a Pocket Switched Network. The implementation will be on an existing mobile device. The simulation will use pre-existing test data to evaluate different strategies for routing, and to find the most ideal strategy in terms of cost and performance. Finally, to develop a routing algorithm that passes messages based on interests that a user subscribes to in an optimum way.

Summary of Work Completed

A mobile application was successfully implemented. A simulation program was used to develop a new parameter controlled routing algorithm that meets the requirements specified to route messages based on interests. The routing algorithm was evaluated against a large dataset, and against extracted different geographically distributed communities. Its performance was measured against several existing routing strategies.

Special Difficulties

None.

Declaration of Originality

I, Sunil Shah of Fitzwilliam College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pocket Switched Networks (PSNs) are a type of delay tolerant ad-hoc network created between mobile devices[1]. In this way, it is possible to carry data from node to node without utilising infrastructure, allowing data forwarding paths where none had previously existed. This type of communication could be beneficial in several different situations where for varying reasons, infrastructure is not immediately available to communicating parties. Such a network is already used to pass messages between reindeer farmers in the Arctic Circle [1].

This project was an investigation into the routing methodologies employed by Pocket Switched Networks, using existing data of human-human contacts (collected through a project at MIT [7]) to evaluate various routing strategies. Two main applications were programmed: an implementation of a PSN that allows short messages to be sent, and a simulation application that allows routing algorithms to be tested on the MIT data. Through the use of the simulation application, a novel parameter controlled routing algorithm was created that routes messages purely based on hashtags. This algorithm was evaluated for its performance against data pulled from Last.Fm [26], a popular music social networking website and mapped to the MIT data. Various scenarios were considered where communities within the test data were geographically local or geographically dispersed. The routing algorithm has shown itself to be efficient in routing data by reducing the number of messages sent for an increased delay in the message delivery time.

1.0.1 Key Terms

A brief summary of the key terms relating to PSNs which may be unfamiliar to the reader.

Hash Tag A '#' character followed by a word. Used to demark interest. E.g., '#compsci'.

Tagged Message A message that contains a hashtag. E.g. 'message #compsci'.

Node A node may be any sort of device which can send or receive data on the network. For the purposes of this project, nodes will be considered to be mobile phones.

Community A community is a group of nodes who regularly interact with each other.

[1]Mobile devices which are stored in pockets. Hence 'Pocket' Switched Network.

Geo-local A community may be distributed geo-locally if the nodes it contains are geographically close to each other.

Geo-dispersed A community may be distributed geo-dispersed if the nodes it contains are geographically distant to each other.

Cost of Delivery The cost of delivery for a particular message is the number of copies of a message that are sent over the network before reaching any recipients.

1.0.2 PSN Example

This trivial example (figure 1.1) with four users shows how messages are routed across a simple network using interests. Alice and Xavier are both interested in *compsci* but rarely meet. Alice, Bob and Charlie meet regularly and Charlie is friends with Xavier.

1. Alice composes a message publicising a talk for all those interested in *compsci*.
2. Alice encounters Bob. Her mobile device transmits a copy of the message to Bob because there exists a route to *compsci* subscribers (Bob has previously met Charlie).
3. Bob has a copy of the message.
4. Bob encounters Charlie. His mobile device transmits a copy of Alice's message to Charlie because there exists a route to *compsci* subscribers (Charlie has previously met Xavier).
5. Charlie has a copy of the message.
6. Charlie encounters Xavier. His mobile device transmits a copy of Alice's message to Xavier because Xavier is interested in *compsci*.
7. Xavier has received the message from Alice.

1.1 Project Motivation

A PSN is a specialized form of a Delay Tolerant Network (DTN) in which the nodes are mobile devices [19]. DTNs have many potential uses including underwater communication between submarines, delivery of data to rural areas, communication in crisis hit areas, and mobile sensor networks. I will be considering a specialized case of such a network based on interactions between mobile phones.

Routing Algorithm

The foremost concern with a PSN is how to route messages. Standard routing algorithms (distance vector and link state) used on connected networks cannot be used on mobile ad-hoc networks such as a PSN. Routing on a PSN must take into consideration several factors including:

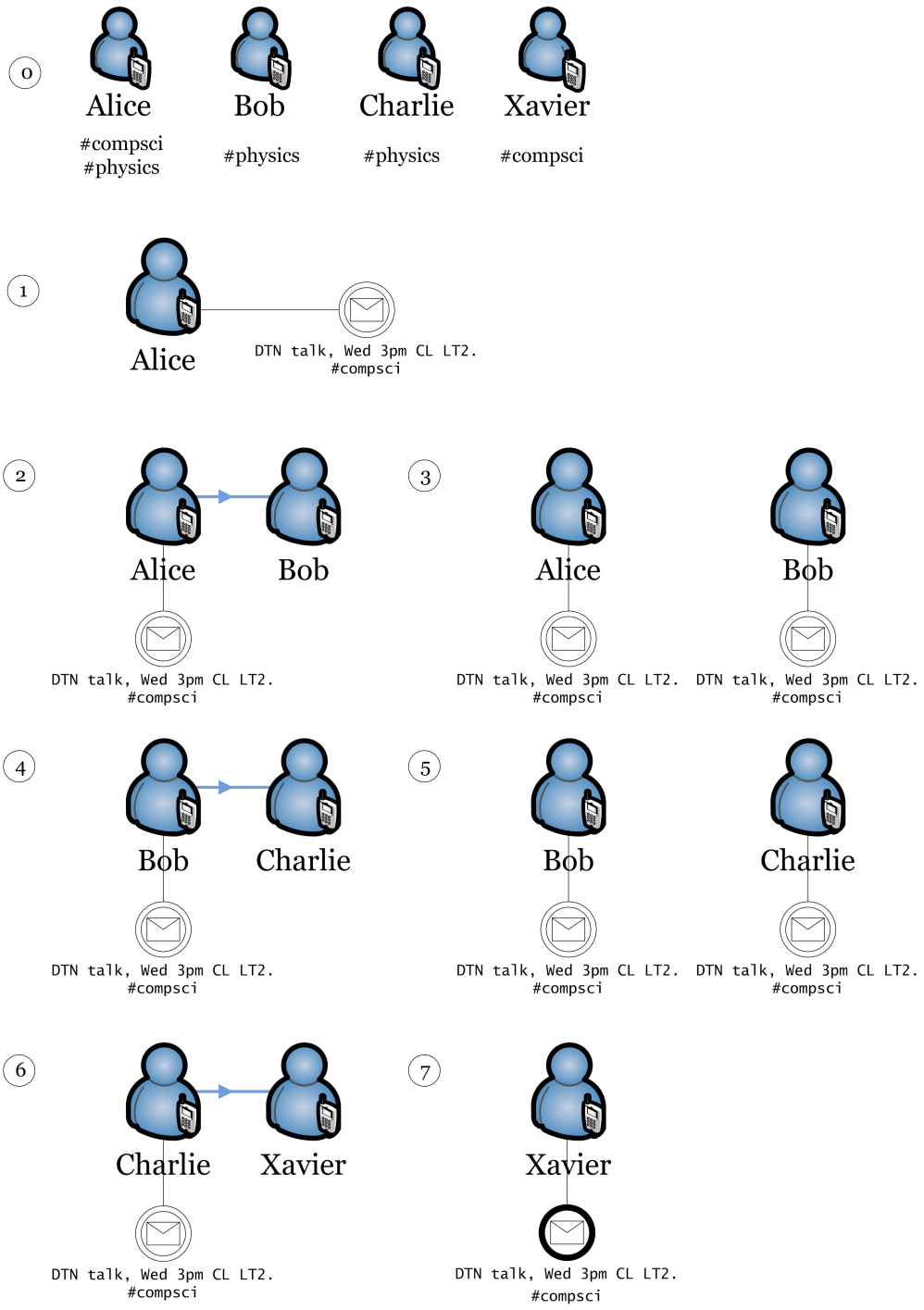


Figure 1.1: An example transmission across a PSN.

- *Mobility* of nodes. Some nodes are more mobile than others - they will encounter a greater variety of other nodes, including those further afield. Exploiting mobility can reduce cost of delivery.

- Availability of *network resources.* Nodes have limited storage and power resources with which to transmit. Reducing resource consumption increases the utility of each node - larger messages and those transmitted frequently increase consumption.

In this situation, power and storage considerations will affect design choices for the smartphone implementation of a PSN. I will investigate and use the mobility of nodes to create a routing algorithm which can more efficiently transfer messages.

The benefit of an adapting routing algorithm based on node mobility is that messages will reach their destination quicker whilst minimising costs incurred due to message transmission on the network. This is desirable since mobile phones are power constrained and transmission uses power.

Interest Based Routing

Twitter is a micro-blogging website that allows users to send out short messages of 140 characters in length to a network of followers. These messages are sent out to everyone but may be demarked with a particular username (conveying a message intended for a particular user) or a hashtag. Hashtags are used for assigning topics to a message, for example, messages sent from the midst of the recent G20 protests in London were demarked with '#g20'. These allow users to subscribe to a particular topic, either using a Twitter client or search.twitter.com.

Twitter can be thought of as utilising a publish/subscribe architecture; users *publish* updates and other users can *subscribe* to these. It is notable for its substantial amount of downtime [5], collapsing frequently under load due to its centralised architecture. A PSN which routes tagged messages to those who subscribe to particular hashtags can thought to be a distributed publish/subscribe model where there is no single point of failure. This is desirable since it is infrastructure free and hence considerably more resilient.

Imagine that there is a less well known ball taking place in a particular Cambridge college. Traditional methods of selling tickets may involve sending out advertisements which effectively point to a website. Instead, if every student carried a mobile phone which was part of a PSN, the organisers could simply send out a single tagged message[2]. This would be routed gradually to as many interested students as possible via interactions that took place in lecture halls, social events and so on.

Smartphone Application

Creating a smartphone application implementation of a PSN has already been done[3]. However, this was implemented in C++ and for a different smartphone operating system to what was originally considered for this project.

This application consists of several interesting challenges; developing for a mobile device has its own peculiarities and subtleties. In addition, wireless communication between

[2]E.g., 'Tickets for college ball available now! http://www.collegeball.com. #mayball'

[3]As part of the Haggle project. http://www.cl.cam.ac.uk/research/srg/netos/haggle/

devices was required to be seamless - users cannot be expected to manually transfer messages between devices as they meet. This implementation must also be able to utilise the routing algorithm developed in the simulator.

Simulation Application

Implementing a simulation application was necessary to test the efficiency of various routing algorithms (including the one developed). The MIT data available represents over 350,000 hours of data collected from 100 mobile phones carried by participants of the study over 9 months [6]. The simulation must be able to run through this data and simpler test cases to validate and verify the new algorithm.

1.1.1 Divergences from Proposal

The project proposal (section D) mentioned that some method of correlating interests using a social networking website would be used in routing. However, early on, I decided to focus on routing without use of infrastructure which made correlating interests highly difficult. Real world interests were still used (section 3.4.3) for simulation.

1.2 Related Work

PSNs are very much a current research topic, centred largely on how to improve the efficiency of routing messages. There have been several approaches considered already, most of which are covered briefly below. Routing strategies can be categorized by how much knowledge of the network is available to each node [15]. I will consider only those strategies which assume incomplete knowledge since we are considering stochastic networks for which it is not possible to know the topology dynamics in advance.

1.2.1 Random Walk

Also known as *First Contact*. This is where the message is forwarded each time two nodes meet. A single copy of the message exists at any time. This happens until the message reaches its intended recipient. Used for comparison [15].

1.2.2 Wait for Destination

This is a simplistic model where a node simply waits to encounter the intended recipient for a message. Again, used for comparison [11] .

1.2.3 Epidemic Routing

Epidemic routing[4] is a form of routing that seeks to maximise message delivery and minimize the latency associated with delivery [10]. In epidemic routing, each message is passed on to all encountered nodes if the recipient node has not yet received the message.

[4]Also known as flooding.

This occurs until the recipient node is the intended recipient of the message, or until the message expires (also known as a hop count). The message expires after a certain number of exchanges or *hops* to prevent unnecessary congestion on the network.

Epidemic routing offers the best case delivery scenario since no decisions are taken during routing not to pass the message on (assuming a sufficiently large hop count), and the message will reach its destination in the quickest way possible. Messages are certain to reach their destination assuming it exists.

However, there is considerable overhead associated with message delivery since a copy of the message is transferred at every subsequent interaction by nodes carrying it. No effort is made to limit superfluous transactions of messages where transferral of the message will not lead to successful message delivery. This is undesirable for PSNs where power usage is of concern.

1.2.4 Osmosis

Osmosis routing is a concept analagous to osmosis in Biology where users with devices are equated to cells [11]. Osmosis in nature refers to the movement of water molecules from an area of high concentration to an area of low concentration. In a PSN, this concentration can be used to determine which path to send messages along so that they only visit nodes which have a low probability of having seen the message previously. Performance is better than epidemic routing, much less data is transferred albeit with a slightly lower success rate. In [11], osmosis is used for file sharing over a PSN.

1.2.5 RAPID

Resource Allocation Protocol for Intentional DTN routing (RAPID) is a protocol designed to minimise either average delay, missed deadlines or maximum delay [16]. It seeks to minimise both bandwidth and storage requirements, using a heuristic based approach due to the uncertainty of DTN environments. The heuristic is found by calculating the utility to the network of storing and forwarding each packet. In this way, it is able to discard packets that would result in an increased average delay, missing a delivery deadline, or increasing the maximum delay. In addition, a control channel is used to transfer information between peers, including expected meeting times with nodes and a list of packets delivered amongst others.

1.2.6 PRoPHET

Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) is a routing protocol that uses frequency of node encounters to route messages [23]. Nodes exchange summary vectors when they meet. These vectors contain *delivery predictability* information that describe how likely it is that the other node will meet any of the other nodes. A threshold value is used to determine whether to forward messages or not. In the simplistic implementation considered, this is fixed such that messages are only passed to nodes with a higher probability than the current node. In the test set considered, PRoPHET has a lower communication overhead and sends less messages across the network than epidemic routing. The delivery rate and delay in message delivery is similar

to epidemic routing.

1.2.7 Spray and Wait

Spray and Wait is a replication based routing strategy which limits the number of copies and transmissions of a message [17]. Spray and Wait consists of two phases:

Spray In the spray phase, L copies of a message are spread to L peers by the source node which originated the message.

Wait If the message has not been delivered in the spray phase, each of the L nodes and the origin enter the wait phase. They will now forward it only to the final destination.

It has similar performance to epidemic based routing but at a reduced cost of delivery per message. An alternative scheme exists, called Binary Spray and Wait, where when a node is in the Spray phase, it hands over half of the number of messages it has. The receiving node is then also responsible for Spraying the message. Once each node only has one copy of the message left, it switches to the Wait phase.

The choice of the L value in both variations is important - it is difficult to estimate this value correctly and this has a large effect (depending on the topology of the network) on message delivery. Spray and Wait is beneficial when there is limited bandwidth because fewer copies of messages are transmitted.

Summary of Algorithms

Generally the more information about the network that is available, the better performance of a routing strategy [15]. However, the DTN routing problem in general is NP hard to solve optimally so the use of a heuristic is recommended [16]. Replication based strategies appear to be generally favourable [16]. Most replication based strategies cut the copies of messages sent to save bandwidth and storage despite possibly reducing the number of successful deliveries and increasing delay.

Chapter 2

Preparation

2.1 Prior Research

2.1.1 Mobile SDK

It was necessary to investigate the various mobile software development kits that were available for use. They vary in terms of usability, integration with a suitable IDE and support for wireless transmission. Most SDKs do not support simulation of Bluetooth transmission, so real life testing was necessary. Hence the availability of devices was also a key consideration.

Google Android

The inital project proposal suggested Google's Android operating system. However, it was found that in August 2008, Google removed the Bluetooth API from the Android SDK [2].

Android does support Wi-Fi, and this was a potential option. A game is in development called *Wi-fi Army [3]* which utilises Wi-Fi in a similar fashion to what is necessitated by this application - to provide seamless communication. This game had not been released at this point and no other examples were evident.

Although the Android SDK offers excellent integration with Eclipse, a Java development environment, the emulator does not support testing of Wi-Fi communication and two devices would have been necessary.

iPhone OS

The iPhone operating system is a popular option for developers, due to the large base of users and the opportunity to create profitable applications. For the purposes of this project it is not suitable since the SDK restricts access to hardware and does not provide a Bluetooth API. Finally, the SDK is not free and would have required learning how to program in the Objective C language.

J2ME

J2ME is a form of the Java runtime optimized for mobile devices. It is found on nearly all mobile phones and allows development in Java using the Netbeans IDE. It provided adequate access to Bluetooth APIs but access to Wi-Fi is restricted and assumed that the host operating system had already set up a connection.

A J2ME application would have almost universal compatibility - testing could have utilised many more devices. With hindsight, there is a Wireless Toolkit which allows emulation of applications which use Bluetooth - since the actual application ended up using Bluetooth, the lack of a low-level Wi-Fi API was not actually a hindrance.

Symbian OS

Symbian was disregarded since it only supports C++. Additionally, there were few devices available to use for testing.

Windows Mobile

Windows Mobile provided a fairly well documented SDK which allowed applications to be written in C++ or any .NET language which compiles to the .NET Compact Framework (CF). The library features offered by the .NET CF are limited compared to those offered by the desktop .NET environment but comprehensive enough. They offer Bluetooth and Wi-Fi support through APIs, complemented by a popular 3rd party Bluetooth library.

The emulator does not provide Bluetooth or Wi-Fi support but enough devices were available to ensure sufficient testing when necessary. The SDK is free to download and integrates well with the Visual Studio 2008 IDE.

Windows Mobile was chosen as the development platform of choice.

2.1.2 Wireless Medium

Once the development platform was chosen, it was necessary to decide upon which wireless medium to approach first: Wi-Fi or Class 2 Bluetooth. The choice was made based on several metrics.

Data Transmission Rate

Wi-Fi on mobile devices is generally the 802.11b standard offering 11Mb/s. Class 2 Bluetooth offers a gross data rate of roughly 1Mb/s. However, messages are short in length - this will make little difference in real life.

The maximum message length is 140 ASCII characters, along with other metadata, which I will assume to be roughly the same size as the message. Add to this the overhead of the container format for the message (approximately 20 bytes) and the maximum message size in bytes can be estimated to be 300 bytes.

Time taken via Wi-Fi:

$$\frac{300}{1.15343\,10^7} \times 1 = 2.60 \times 10^{-5} s$$

Time taken via Bluetooth:

$$\frac{300}{1048576} \times 1 = 2.86 \times 10^{-4} s$$

Bluetooth takes 10 times as long but since this is less than a millisecond, it will make little practical difference.

Battery Life

It is well established that Bluetooth has much lower power consumption than Wi-Fi. Bluetooth was designed with portable devices in mind and has a limited range whereas Wi-Fi has an extended working range. Bluetooth uses approximately a quarter of the current in transmission and reception that Wi-Fi does [4]. First hand observation suggests that whereas enabling Wi-Fi would limit battery life to several hours, Bluetooth would limit battery life to an entire day.

Past Examples & Availability of Libraries

The Haggle project mentioned earlier was implemented using Bluetooth. However, similar work had been done using Wi-Fi in the Android game *Wi-Fi Army*. Looking at the specifications for each standard, it was clear that Bluetooth explicitly supports the sort of behaviour required by a PSN, whereas Wi-Fi is meant to emulate wired networking. Bluetooth offers several profiles which could be of use, including the Personal Area Network profile.

There were also considerably more documentation and examples implementing Bluetooth communication (provided by Microsoft). A 3rd party library called 32Feet.NET offers simplified Bluetooth support and was a possible option.

Bluetooth was used initially for implementation.

2.1.3 Programming Language

Windows Mobile supports use of either C++ or any language that compiles to the .NET CF. Code examples are given in both C++ and .NET languages, although more are available in C++. However, due to past experience in using C# (and its relative similarity to Java), it was chosen for implementation.

2.2 Requirements Analysis

This is what the agreed functionality of each part of the project should offer based on the project proposal and discussions with my supervisor.

Routing Algorithm

- Delivery latency must be less than a random walk routing algorithm (described in 1.2.1).

- Cost of message delivery must be less than an epidemic routing algorithm (described in 1.2.3).
- Routes messages based on hashtags.
- Can exploit different mobility patterns (e.g. geo-local or geo-dispersed).

Smartphone Application

- C# application running on Windows Mobile.
- Transfers messages between devices seamlessly (i.e. without user interaction).
- Suitable data format for messages.
  - Allows suitable storage of destination information and useful metadata.
  - Supports any extensions
- Able to use routing algorithm from simulation application (using adapter classes if necessary).

Simulation Application

- C# Desktop application.
- Uses MIT Reality Mining Data to simulate interactions.
- Shares common structure with smartphone application so routing algorithm can be reused.
- Imports interest[1] data from a social networking website to use in simulation.

2.3 Project Planning

As the requirements analysis shows, this project broke down into two applications. As per the project proposal, it was suggested that both be worked upon in parallel. However, there was an emphasis on the smartphone application for the first 6 weeks, followed by work on the simulation application and routing methodologies.

2.3.1 Preparatory Learning

It was necessary to learn the features offered by the Windows Mobile 6 Professional SDK - this was done partially before beginning the implementation. Furthermore, my knowledge of Pocket Switched Networks was limited and required additional reading. Several papers were recommended by my supervisor [12, 13]. Additional papers were read in the process of implementation, these are all referenced in the Bibliography.

[1] For example, a list of tags that a user is interested in.

2.3.2 Proposed Architecture

It was clear from the requirements analysis that it was necessary to organise packages such that the routing algorithm is easily transferrable between the two applications. It was proposed the applications be packaged separately to the algorithm to allow the code to be shared easily.

2.3.3 Development Model

Since there was no proposed end user of this system, there was no need at any stage for prototyping. At the same time, there were several milestones that had to be accomplished during development. A waterfall model was used in development of the smartphone application since it was relatively simple. Once requirements were specified, the application was implemented and then tested on several devices.

Development of the routing algorithm was also a simple waterfall process where a design was constructed and implemented. It was then tested once the simulator had reached a sufficiently advanced stage.

However, for the simulation application, a more flexible approach was necessary since changes were being made constantly to the underlying model. An iterative model of development was used, centered around project progress meetings with my supervisor. These iterations were derived from the Plan of Work outlined in the Project Proposal.

Each iteration would seek to add greater depth to the simulation, for example:

Iteration 1 Adapt the MIT data into a form that was suitable for simulation.

Iteration 2 Import of MIT data from MySQL into C#.

Iteration 3 Basic routing using MIT data to simulate devices.

Iteration 4 Import of social network data into simulation.

Iteration 5 Advanced routing using social network and MIT data.

Once the feature was implemented, it would be tested before demonstration. Finally, the simulation was used to test the routing algorithm extensively, both on simple test cases and the MIT dataset.

2.3.4 Tools Used

Microsoft Visual Studio 2008 was used to develop both the smartphone application and simulation application. In addition the Windows Mobile 6 Professional SDK was used to develop the smartphone application.

A MySQL server installation was necessary to use and manipulate the MIT dataset, whilst MySQL Query Browser assisted with database interaction.

2.3.5 Contingency Measures

As recommended by the writers of the Project Briefing Document and in the briefing lecture, suitable backup provisions were made. The source code and scripts necessary to recreate alterations to the MIT database, were stored in a remotely hosted Subversion repository. The same server was used to host an electronic version of my project notebook in the form of a TiddlyWiki[2].

[2]http://www.tiddlywiki.com

Chapter 3

Implementation

3.1 Overall Structure

Two Visual Studio *solutions* were used to separate the different applications. Within these, there were a number of projects that were used to contain various related classes. I will document the implementation of the routing algorithm, the smartphone application and simulation application separately. The projects and their names are shown in figure 3.1 grouped together by solutions. This allowed the routing algorithm to be shared easily. A separate project was created to encompass all the code used to acquire and map social network data from Last.fm (section 3.4.3).

3.2 Smartphone Application

The smartphone application was attempted first because of its relatively straightforward goal.

3.2.1 User Interface

The user interface for this application is extremely basic since it is purely proof of concept. It consists of a list of potential recipients, a textbox to enter a message (hashtags are extracted automatically from the message) and a send button. Figure 3.2 shows the UI.

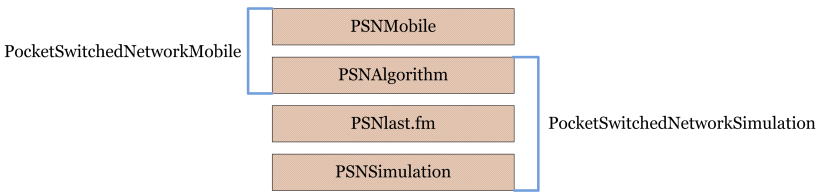


Figure 3.1: Project and solution structure.

Figure 3.2: Screenshot of forms in Windows Form Designer.

3.2.2 Bluetooth Communication

When the application is started, it enables the Bluetooth radio if needed and spawns two threads used for scanning and serving.

32Feet.NET

32Feet.NET [34] is a shared source 3rd party Bluetooth library for .NET applications that offers considerably better Bluetooth support than Microsoft's own library functions. Initially I attempted using Microsoft's own Bluetooth functions based on several code samples but it was not evident there was a way to transfer messages between unpaired devices[1]. Since pairing requires user intervention, this would violate one of the key requirements that this application operates seamlessly and so 32Feet.NET was used instead.

3.2.2.1 Scanner

The scanner runs every 20 seconds and detects any Bluetooth devices in the nearby vicinity. It attempts to connect to each device using a pre-generated GUID[2]. If it is able to connect, the other device is running this application and the sender's interaction code (as described in 3.3) is begun.

20 seconds was chosen because it is the minimum necessary to allow the discovery scan to finish and provides the most frequent update rate.

3.2.2.2 Server

This creates a service record using the pre-generated GUID. It yields when attempting to accept a Bluetooth client and is reinvoked once a client attempts to connect. It runs the recipient's interaction code as in 3.3.

[1]Devices in Bluetooth usually need to be paired as a security feature. However, it is possible to send messages without pairing - e.g. http://en.wikipedia.org/wiki/Bluejacking

[2]Globally Unique Identifier

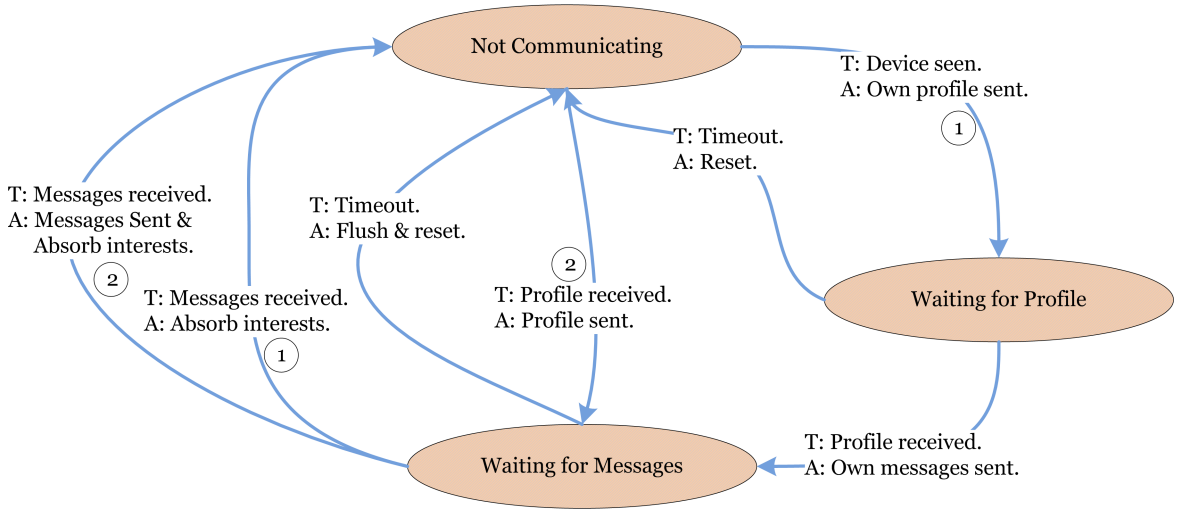


Figure 3.3: State diagram of Bluetooth communication.

3.2.2.3 Manager

The routing algorithm requires each node to send both a profile and a collection of messages to the other node. The profile contains some routing information. Since the list of messages is dependent on the profile, communication takes place in a variety of stages. For a more involved description of how this works see section 3.3.2. A list of each device encountered and the state of communication with it is kept in a Dictionary object, this list is managed by a static Manager class. A state diagram of the process is shown in figure 3.3, where (1) is the path if the device initiated communication and (2) is the path if another device has initiated. T is the *trigger* which causes the state change and A is the *action* that is taken.

3.2.2.4 Message Transfer

It would be ideal to send messages in a binary form to save on filesize and transfer quicker. Unfortunately the .NET CF does not support binary serialization, so messages are serialized to XML before being transferred.

3.2.3 Other Concerns

Naming Naming of devices is an issue with PSNs [21] since most schemes require some sort of centralised name service, which an unconnected device would not have access to. However, since the routing algorithm designed here considers only hashtags, it was not necessary to implement a naming scheme.

Deployment Deployment to an actual phone for testing purposes took place initially using the Windows Mobile SDK which allowed the application to be deployed to a device instead of an emulator while debugging. However, this quickly became limiting when it was necessary to test multiple devices since the device must remain connected. CAB files are executable archive files used on Windows Mobile to package applications and files

that they are dependant on. A CAB project was set up in Visual Studio and used to create a CAB file which could be manually installed on multiple devices at a time.

3.3 Algorithm

Designing and writing my own routing algorithm was a substantial part of this project. A description of how it works and the rationale behind design decisions follows.

3.3.1 Design

Having considered various approaches, it seems that heuristic based methods are the best for PSNs due to the large inherent uncertainty in the topology of a network.

Traditional link state and distance vector algorithms would never have a complete view of the network. There are variants developed, such as Delay Tolerant Link State Routing [9] but there is an overhead associated with sending link state announcements between nodes; this is not suitable for a PSN.

Goal It is important first to consider the goal of this algorithm. It is intended to provide an efficient (lower cost of delivery) method to route messages containing hashtags. The aim is to deliver relevant messages to as many nodes subscribed to a particular hashtag as possible. It also needs to provide a method of adjusting routing to suit different geographical topologies (local/dispersed).

3.3.1.1 Heuristic Choice

The choice of heuristic is tightly coupled to the intended use for a network. PRoPHET [23] considers message delivery between nodes and uses a metric called delivery predictability - the probability that each node will see each other node.

Since the intention for this algorithm is to route messages based on hashtags, the natural choice for a heuristic is to the consider actual tags and the probability of seeing nodes which subscribe to them again.

Route stability refers to the availability of the same routes to a particular destination. As is clear in related literature, route stability is not guaranteed. However, the tags or nodes a node has seen in the past are likely to form a good indication of what the node will see in the future. So instead of considering fixed routes, the heuristic will be an exponentially weight moving average of the probability that each tag will be encountered in the future. Using an average probability exploits recurrence to build an approximate model of the network [18].

3.3.1.2 Initialising Probabilities

Probabilities for all tags that the node subscribes to itself are set to 1. All other probabilities are set to 0 (by excluding them from the list of seen probabilities altogether).

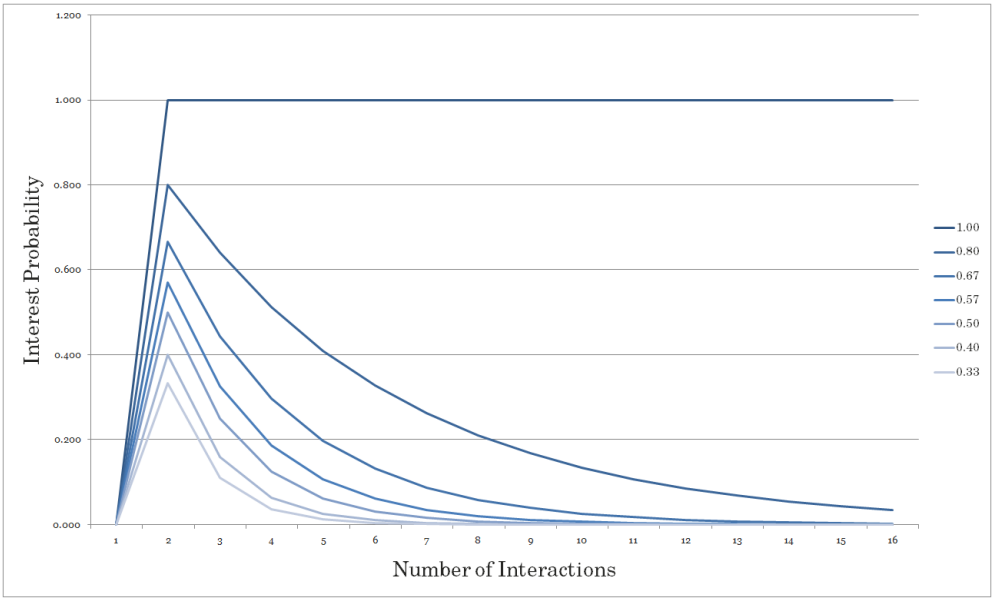


Figure 3.4: Effect of different α values

3.3.1.3 Aging Probabilities

All tags that the node subscribes to remain at 1. The rest are aged each time another node is encountered using the following formula.

$$P'_I = \alpha \times (P_S + P_I)$$

P_I is the current probability for a particular interest I, P'_I is the new probability and P_S is the probability for that tag that a recently seen node has given. α is the smoothing factor applied and determines how long that probability is remembered for.

Figure 3.4 compares some different α values and how the probability changes with each successive interaction based on seeing a device with $P_S = 1$ on the second interaction.

3.3.1.4 Threshold Value

A threshold value is used to decide whether or not to forward messages to encountered nodes. This value defines the minimum probability for each interest that a node should have before it is passed messages containing that interest. This value, in addition to the α value, provides the capability to tweak the routing behaviour of the algorithm. The effects of changing these values on routing are investigated in the evaluation.

3.3.1.5 Control Channel to Improve Knowledge

Knowledge improves routing decisions made in DTNs and is generally desirable. It is used to improve the performance of the RAPID protocol [16] using a control channel on which nodes transfer information about the rest of the network. However, [19] describes how the distribution of contact duration between contacts follows an approximate power law, i.e. there are many contacts with a short contact time, rapidly decreasing as contact time increases.

This suggests that it may be preferable not to include a control channel in order to reduce transmission time. Omitting a control channel also makes implementation simpler.

3.3.1.6 Storage / Bandwidth Constraints

As mentioned in [16], most algorithms do not take into account storage of messages. It is sometimes necessary for devices to *queue* messages that they are holding in transit. However, I decided this would not be an issue since these messages are very small in size and most modern smartphones have several megabytes of memory - enough for hundreds of messages. There is no limit to the number of messages a phone can store in transit.

3 folders exist on each device, Inbox, Transit and Sent. Transit is hidden from the user and contains messages that are queued to be sent onwards.

3.3.1.7 Count To Infinity

To prevent absorption of probabilities which are based on a route which passes through the current node (i.e. routing loops), each interest stores a history of addresses in the form of a stack which it passes through. When absorbing an interest, it is rejected if the route includes the current node's address.

When a new route is encountered for a particular interest, the route with the highest probability is picked (irrespective of whether this is the old or the new route).

3.3.1.8 Message Metadata

Some other useful metadata is stored for each message.

Transit Expiry Messages might have some time constraint for delivery after which they cease to be informative.

Last Node Address To prevent messages being sent back to the same node.

Counter To store how many nodes the message has passed through.

Interests A list of interests is extracted from hashtags in the message when it is sent.

Time Sent Time and date that the original message was sent.

With all replication based networks, it is necessary to be able to uniquely identify messages so that duplicate messages received can be discarded. It is sufficient in this case to compare the content and the time sent.

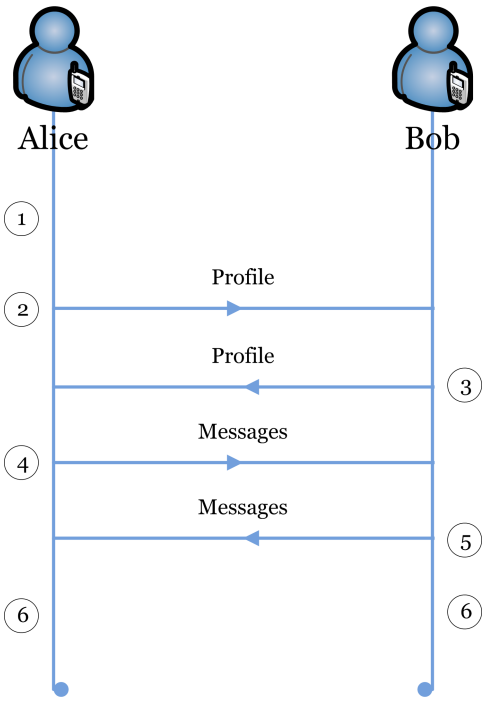


Figure 3.5: Diagram of routing algorithm in action.

3.3.2 In Action

Routing messages between two devices takes place in a number of stages shown in figure 3.5 and described below.

1. Alice initiates communication with Bob.

2. Alice transmits profile to Bob. The profile consists of her address and her dictionary of seen interests which contains probabilities for each interest.

3. Bob transmits his profile to Alice.

4. Based on Bob's profile, Alice transmits a batch of messages to Bob. Messages are only included if the probability that Bob will see that interest is greater than the threshold value. (Before messages are sent they are also *trimmed* to remove any expired messages.)

5. Based on Alice's profile, Bob trims messages in transit and then transmits a batch of messages to Alice.

6. Alice and Bob absorb each other's list of seen interests into their own dictionaries using the method for aging probabilities described in section 3.3.1.3. If any of the messages they have received contain interests subscribed to by the user then these are copied to the Inbox.

Algorithm 3.1 RunInteraction

```
RunInteraction (Address, InterestList):

for each Message in Transit
  if (Message.LastNodeAddr is not Address)
        if (InterestList.Intersect(Message.Interests) > 0)
        if (InterestList.Probability(Interest) > Threshold)
                  MessagesToSendList.Add(Message);
                break;
          end if
        end if
  end if
next

return MessagesToSendList;
```

3.3.2.1 Important Methods

RunInteraction Used to select which messages to transmit (algorithm 3.1). Messages sent on if there are common interests and they meet the threshold value.

AbsorbInterests Used to absorb the newly received probabilities into the existing dictionary of interests (algorithm 3.2).

3.3.3 Testing

Several simple scenarios were concocted in order to extensively test correct functioning of the algorithm. Table 3.1 shows a summary of the tests run and a description of their purpose. Figure 3.6 shows the topology and interactions used for each test (encircled numbers are interactions).

Testing was carried out using the simulation application and an Excel spreadsheet in which the results and state of each node after each interaction were calculated manually. The result of the simulation was confirmed against the corresponding state in Excel. Some example calculations are included in the Appendix (section B).

3.4 Simulation Application

Initially the simulation application was built as a graphical application. This turned out to be too slow to easily simulate the thousands of interactions that took place in the MIT data and was rewritten by abstracting all simulation operations to a separate class. This allowed simulations to be run either via the console or a GUI (when it is beneficial to be able to inspect state).

A class diagram of the application and a screenshot of the GUI are included in the Appendix (section A.2).

Algorithm 3.2 AbsorbInterests

```
AbsorbInterests
        (NewInterestList,
        ExistingInterestList,
        UserSubscribedInterests,
        ThisAddress):

for each Interest in NewInterestDictionary
  if (ExistingInterestDictionary includes Interest)
        if (NewInterestDictionary.History(Interest)
                does not include ThisAddress)
          if (NewInterestDictionary.Probability(Interest)
                > ExistingInterestDictionary.Probability(Interest))
                  ExistingInterestDictionary.History(Interest) =
                NewInterestDictionary.History(Interest);
          end if
      ExistingInterestDictionary.Probability(Interest) =
          (NewInterestDictionary.Probability(Interest) +
          ExistingInterestDictionary.Probability(Interest);
        end if
  else
        ExistingInterestDictionary.Add(Interest);
  end if
next

for each Interest in ExistingInterestDictionary
        ExistingInterestDictionary.Probability(Interest) =
        ExistingInterestDictionary.Probability(Interest) * alpha;
next

for each Interest in UserSubscribedInterests
        ExistingInterestDictionary.Probability(Interest) = 1.0;
        ExistingInterestDictionary.History(Interest) = ThisAddress;
next
```

Test	Diagram	Description
Basic (Unidirectional)	1)	Tests that messages are routed - assuming all nodes subscribe to the same interests.
Basic (Bidirectional)	2)	Tests that messages are routed - assuming all nodes subscribe to multiple same interests.
Probability (Unidirectional)	1)	Tests that information about seen interests is routed through the network - with one interest.
Probability (Bidirectional)	2)	Tests that information about seen interests is routed through the network - with multiple interests.
Probability (Interactions To Zero)	3)	Tests that aging of seen interest probabilities happens as expected.
Probability (Threshold Based Routing)	4)	Tests that messages are routed according to threshold value.
Probability (Larger Test Case)	5)	Larger test case which tests probabilities filter through the network properly and age as they should and correct routing of messages.
Probability (Larger Test Case 2)	6)	Larger test case which tests that interest address history is being stored and used correctly.
Message Expiry	1)	Tests that messages are being removed once expired.

Table 3.1: Summary of tests to show correct functioning of algorithm.

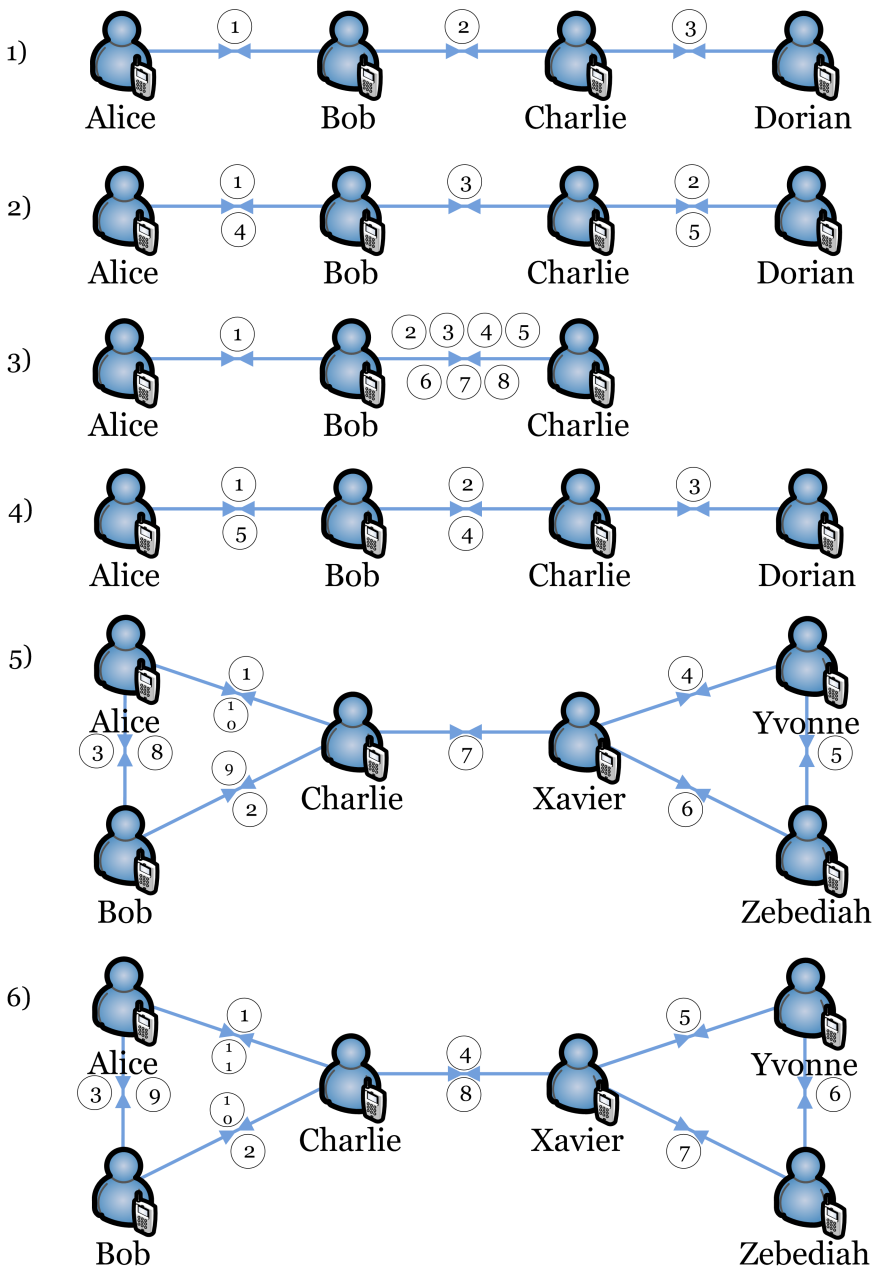


Figure 3.6: Diagram of topologies used in testing.

Simulation A main simulation class is used to run different scenarios. Simulations are run continuously on a separate thread. This allows the GUI to remain responsive while the simulation is run. A configuration class specifies which source to take the reality graph from - this can either be one of the generated test cases (as in section 3.3.3) or the Reality Mining data from the database.

Initially it retrieves a Hashtable of Device objects - these encapsulate all information to do with a single device. The key for each value in the HashTable is the address of the device, used for faster lookup. It then loads a series of interactions and runs them on each device involved.

MessageTracker Message tracking is necessary to provide statistics for evaluating each algorithm's performance. A list of references to messages that are sent as tracked messages is stored throughout the simulation. When queried, the message tracker calculates several statistics using the referenced messages and the state of each device. At a specified interval (not every interaction, since this reduces the simulation speed), the simulation class can query the message tracker and save statistics to a CSV file.

Database This class contains all code used to interact with the database. Data is retrieved in two ways:

1. Device data, the information used to initialize information about each node is retrieved as the result of a single select statement.

2. Interaction data. Each row represents an interaction between two devices. It is not necessary to have all of this information at once, nor is it ideal for performance since there are several thousand rows of data. Instead, a DataReader is used which fetches only the next row. Public variables in this class hold the last interaction that was fetched.

Algorithm In order to share the routing algorithm and data structure with the mobile application, the implementation of the algorithm was stored in a separate project. An interface was implemented that specified the methods each algorithm should support. This allows other algorithms to be used and was used for comparison. A class diagram of the Algorithm project can be found in the Appendix (section A.4).

3.4.1 Reality Mining Data

The MIT data from the Reality Mining project was supplied in a form that could not be used immediately in the simulation. It was necessary to write SQL to adapt the data into a more suitable form. A custom created database (figure A.1) was used to store an adapted version of the data specific for this application.

Interactions from Device Spans Each device involved in the study was rigged to regularly scan its surroundings and log any Bluetooth devices present. These scans logged devices which were not involved in the study and some users also had more than one device. These devices and interactions with them were removed from the dataset.

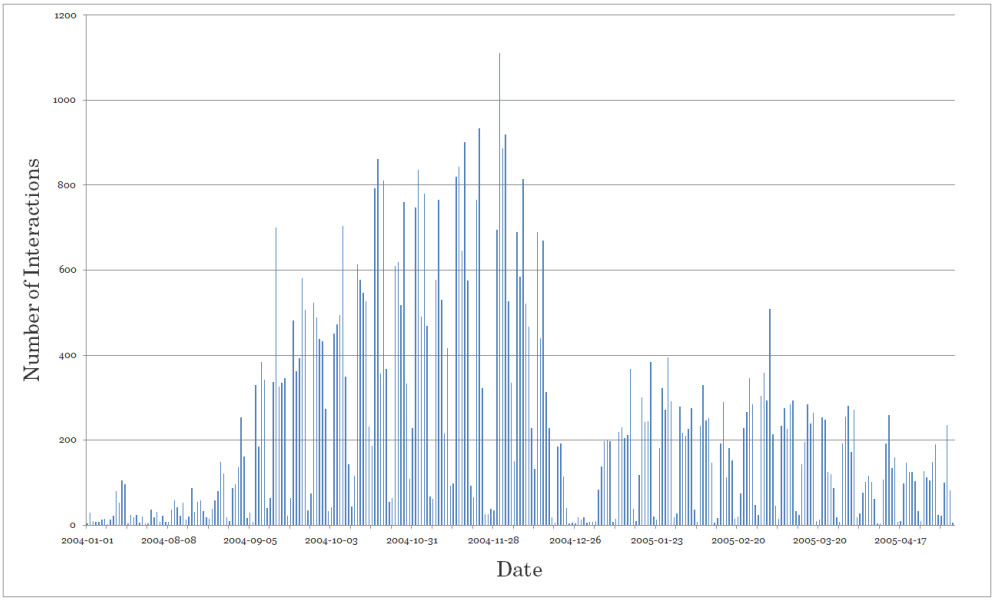


Figure 3.7: Interactions occuring over time.

This gave 114,046 interactions, of which many were momentary (and too short to be useful in real life). Those interactions which had a time span of zero (i.e. the device was seen just once) were discarded to give 64,731 interactions in total. Figure 3.7 shows how these interactions progressed over time.

3.4.2 Building the Graph

In order to map the interest data to the Reality Mining data, it was necessary to represent the entire network in a programmatically understandable form for which a graph is a natural choice.

Nodes of the network are considered to be vertices, edges between the vertices are used to store the frequency of interaction between them. This is a similar model to that used in [15]. An undirected graph was chosen since the frequency of interaction applies bilaterally to a pair of vertices. Tagged edges were used to store a floating point number that represents the frequency of interaction.

First the number of contacts between any two nodes was counted and saved as the frequency score. The scores were then normalized so that they were between 0 and 1. This was helpful when mapping interest data (section 3.4.4) and extracting communities (section 4.1.1.1).

3.4.2.1 Displaying Graphs

A useful extension that was easily added to the project was to display graphs. The QuickGraph library for .NET [31] offered several options, including exporting to the popular dot format and through the use of the Microsoft GLEE tool [32] (available to download as Automatic Graph Layout [33]). Both were used since GLEE worked well for smaller graphs but struggled on larger graphs. This helped in the testing of basic scenarios and to create the diagram shown in figure 4.1.

3.4.3 Acquiring Interest Data

It was necessary in order to simulate hashtag based routing that genuine interest data be attached to the Reality Mining data.

Choice of Social Network

The data required is a list of interests for each node. There were several social networking websites that provide this information about users. Initially, the social bookmarking website del.icio.us [25] was suggested - it allows users to bookmark websites, and categorize them by tags. but inspection of the API offered uncovered that it was not suitable since it did not provide information about friendships with other users.

Last.Fm is a music social networking website [26] which categorises the type of music users listen to through the use of various reporting clients. It uses this data to provide aggregate statistics, such as the most popular artists and songs, as well as recommending similar artists using collaborative filtering [4]. The API for Last.Fm is considerably more open and there exist several libraries to interact with it.

There are two libraries for C# available: Last.fm Lib.Net [29] and lastfm# [30]. After some experimentation, lastfm# was chosen for its superior stability.

Acquiring Tags

Tags on Last.fm can be applied to albums, artists and chosen by the user. Those chosen by the user would be ideal but it was found that many users have not tagged anything, resulting in an empty set of user tags.

Instead, the top tags for a user were acquired by looking up their top five artists. It was possible to get a list of the tags applied to each artist sorted by popularity - from these, the top five tags were added into a list for the user such that the list contained only unique tags.

Tags were *cleaned* before being added by removing spaces, case, punctuation and symbols since there were many cases where this prevented the same interest from being identified. E.g. 'hiphop' compared to 'hip-hop' or 'Hip Hop'.

Finally, it was necessary to merge several categories so that the interests were not completely disparate between nodes. This was achieved by listing each interest distinctly and specifying an alternative,more general tag. A Visual Basic script in Excel was then used to generate SQL to change these so that there were now only 22 different interests as opposed to 351 initially. Some examples are shown in section A.3.

Walking the Graph

In order to find a suitably large network, a random group[3] with many members was chosen. In order to rule out the possibility of choosing a group where users would have similar musical interests, a group based around a desktop application (a plug-in for the iTunes media player) was chosen. All data for this group was downloaded and saved in the form of an undirected graph. Each edge between users represented a friendship and was weighted by a score that Last.fm assigns which specifies the similarity between the

[3]A group on Last.Fm is a collection of users who can share music and partake in discussions.

two. This score is a number between 0 and 1, where two users who share the exact same music taste will have a score of 1.

3.4.4 Mapping Interest Data

Mapping was necessary in order to assign the downloaded data from Last.Fm to the Reality Mining data. Three mappings are considered to see the end effect this has on routing.

3.4.4.1 Random Mapping

Users were randomly chosen from the downloaded data and their interests were assigned to devices in the Reality Mining data.

3.4.4.2 Close & Distant Mapping

It may be the case that people who meet often in real life share interests (a fairly reasonable assumption to make). In order to evaluate the effect this might have on routing, two further mappings were performed based on two premises: that users who meet often have similar interests, and that users who meet often have dissimilar interests.

Interests were mapped by comparing the friends of each user on Last.Fm with the most commonly encountered devices of each node in the Reality Mining data. The most commonly encountered device was given the interests of either the most similar friend (in terms of musical compatibility) or the most dissimilar. These mappings are referred to as close and distant respectively.

Mapping a subgraph from one graph onto another is a known NP-complete problem and it was necessary to use a *best effort* approach whereby each path was mapped until either the Last.Fm user or Reality node ran out of friends who had not already been assigned. The device and user were then both reset to a random unassigned vertex in their respective graph and mapping continued again.

3.4.5 Community Extraction

In order to effectively evaluate the efficiency of this algorithm when routing for differently geographically distributed communities, it was necessary to extract several communities from the Reality Mining data.

This was achieved using the k-clique community extraction algorithm outlined in [35] and used for a similar purpose in [36]. The definition, as given in [36], is: "a union of all k-cliques (complete subgraphs of size k) that can be reached from each other through a series of adjacent k-cliques, where two k-cliques are said to be adjacent if they share k - 1 nodes."

There was no library available for C# which implemented this (the only example found was written in Python). Initially I implemented the algorithm as detailed in the supplementary material for [35]. However, the last stage involved Principal Components Analysis (PCA), a statistical technique used to extract patterns from data. After investigating this in some depth, it seemed unfeasible to attempt implementation. Fortunately, during this investigation, a program called CFinder [37] came to light - it is an exact Java

implementation of the k-clique method. Several communities were then extracted from the Reality Mining data; an overview of the extracted communities is in the Evaluation (section 4.1.1.1).

The algorithm itself uses 3 stages.

1. Finding all cliques in the underlying graph.
2. Preparation of a clique overlap matrix.
3. Extraction of k-cliques from matrix.

The first two stages only need to be done once, since the same overlap matrix can be used to find k-cliques for any k. The graph of the Reality Mining data (as described in 3.4.2) was used, except with the consideration that all edges which had a weight[4] of less than 0.1 (approximately equal to 2 hours worth of total contact) were not included. This has the effect of ensuring greater cohesion between nodes in found communities since they spend more time together.

Clique finding is a known NP-complete problem - an efficient implementation is through the use of a union-find algorithm. This iterates through descending numbers of clique size whilst recursively walking the graph. Two sets are kept, the first set (A) containing nodes that are all connected to each other, the second (B) containing nodes connected to nodes in the first. Nodes are moved from B to A, assuming they are connected to all nodes in A. Each time, all nodes in B that are not connected to all nodes in A are removed. This continues until either B is empty or A has reached the specified clique size (the nodes in A form a clique). Everytime a new clique is found, it is checked to see if it is already contained in any larger cliques already found.

The clique overlap matrix simply maps cliques against cliques and specifies the number of overlapping nodes between each (the diagonal is the size of the clique). To extract k-cliques from the matrix, all diagonal matrix elements which are either equal or greater than k, or all off-diagonal elements which are equal or greater than (k-1) are marked. The marked elements form the k-clique communities and need to be extracted using PCA.

Cell Towers Finally, it was necessary to assign celltowers to each community so that geographically dispersed and local communities could be identified. This utilises the assumption that communities which span a greater number of cell towers are dispersed and those which span fewer are local, and that cell towers are uniformly geographically distributed[5]. This was achieved primarily by using SQL to aggregate the amount of time spent at each celltower by each user from the cellspan data provided. By analysing the distribution of these times, a suitable cut-off point was chosen (3 hours). Cell towers where the user had spent more than this amount of time were considered as part of the community. In actual fact, this time has little effect since there remain noticeable differences in the number of cell towers between communities regardless of the cut-off time.

Communities with a similar number of nodes but considerably different numbers of cell towers were chosen for Evaluation.

[4]This is also known as the threshold weight, w*.

[5]Since this data was collected at the MIT campus, in Cambridge MA - an urban area, it seems reasonable to assume this to be the case.

Chapter 4

Evaluation

In order to evaluate the success of this project, I will evaluate the routing algorithm against the requirements specified in section 2.2 through the use of the simulation application. My routing algorithm is hereupon referred to as HashPRoP[1].

In order to assess the effectiveness of HashPRoP compared to the random walk and epidemic routing algorithm, I have compared the performance of each algorithm on the entire data set in section 4.2. To assess its effectiveness in exploiting different mobility patterns, I have compared performance of each algorithm on selected communities in section 4.3.

The smartphone application is evaluated separately in section 4.5.

4.1 Test Data & Metrics

An overview of the data and performance metrics used to evaluate the algorithms.

4.1.1 Reality Mining Data

Average Interaction Time (minutes)	55.7
Average Number of Interactions Per Node	727.3
Standard Deviation of Number of Interactions Per Node	771.7
Average Number of Interests Per Node (Random Mapping)	7.6
Total Number of Contacts	64,731

Table 4.1: Reality Mining data statistics.

Table 4.1 shows some statistics pertaining to the data used for evaluation. Figure 4.1 shows the Reality Mining network. Nodes are placed in a consecutive order in a circular fashion (attempting automatic placement crashed GraphViz) and lines are drawn between nodes that come into contact with each other. It shows the distribution of contact between nodes. Near the perimeter of the diagram, where the lines are darker represent where nodes are in more frequent contact with their neighbours. Lighter areas are where nodes are in less frequent contact.

[1]Hashtag Probability ROuting Protocol

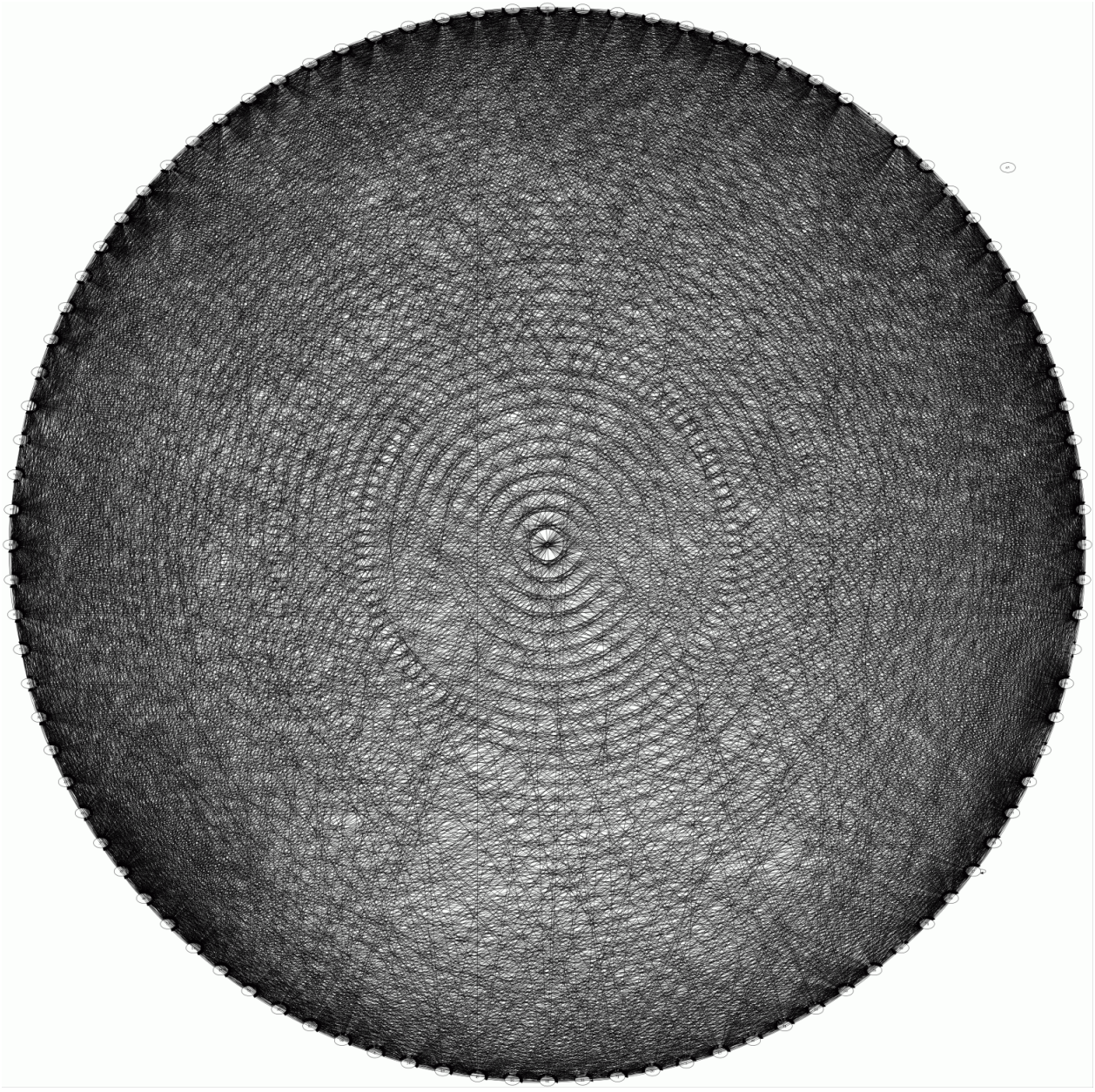


Figure 4.1: Diagram of Reality Mining network created by GraphViz.

4.1.1.1 Communities

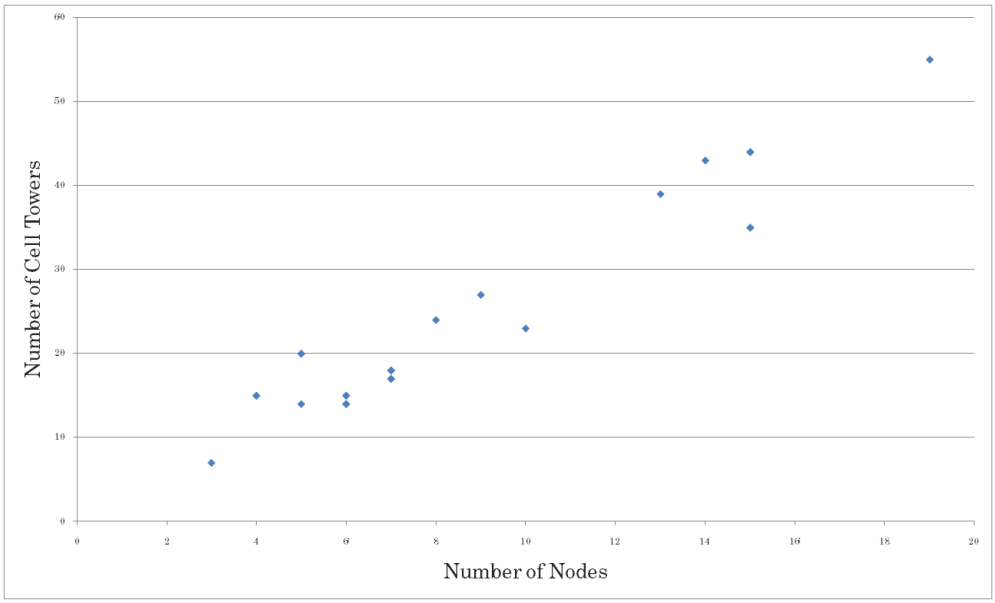


Figure 4.2: Scatter graph of Number of Nodes against Number of Cell Towers for extracted communities.

Using the community extraction methods outlined earlier (section 3.4.5), several communities of varying geographical distribution were extracted. Figure 4.2 shows how the number of cell towers within a community increase as the size of the community increases. For the purposes of evaluating performance based on the geographical distribution of each community, it is necessary to identify communities which are geo-local and geo-dispersed.

Since I am assuming that communities which have a similar number of nodes but different numbers of cell towers are differently geographically distributed, those two communities which have 15 nodes will be considered (from k=4), as well as those two which have 5 and 6 nodes (from k=3).

A full summary of the extracted communities and the numbers of celltowers in each can be found in the Appendix (table C.1). Figure 4.3 shows the extracted communities that will be used graphically. Local communities are on the right, dispersed on the left.

4.1.2 Performance Metrics

A couple of statistics are used to gauge performance quantitively.

Message Delivery Ratio (MDR) This ratio represents the percentage of all nodes subscribed to a particular interest represented in a message who actually receive it. The final value is average of the message delivery ratio for each interest.

Sent Messages This is a counter of the total number of messages sent on the whole network. Incremented for every copy of a message that is sent during each interaction.

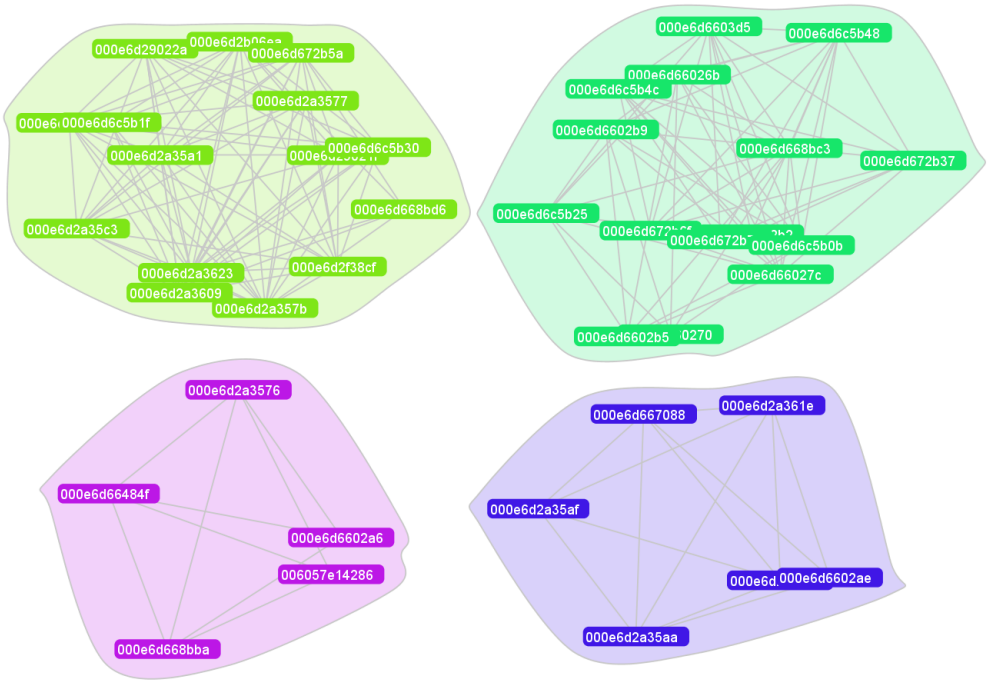


Figure 4.3: A diagram showing the structure of the extracted communities used.

4.1.3 Terminology

Subscription Ratio The proportion of devices out of the total number of devices that subscribe to a particular interest. (I.e., they want to receive messages about it).

Threshold Value Threshold value used for HashPRoP to determine whether to forward messages (as described in section 3.3.1.4). A higher threshold value should limit messages only to be sent to nodes which encounter an interest more frequently.

α Value Alpha value used for HashPRoP interest history smoothing (as described in section 3.3.1.3). A higher α means that nodes will remember interests they have seen for longer.

Latency Latency is the time it takes for the Message Delivery Ratio to converge on a fixed value. The faster this happens, the lower the latency of message delivery.

4.1.4 Other Algorithms

The requirements specifically mention the performance of the developed algorithm to best the Random Walk and the Epidemic routing algorithms for latency and cost of delivery respectively. It is necessary to evaluate the algorithm's performance against these. Furthermore, I have decided to implement Binary Spray and Wait in addition to provide additional comparison since it is the current optimal solution for routing in most scenarios. Some adaptations were necessary to route messages by interest.

Random Walk (Section 1.2.1)

Messages undergo a random walk of the network until they encounter their destination. Since the destination is not one node but the set of all nodes that subscribe to a particular interest, messages in my implementation continue to walk the network even after encountering a subscribing node. However, only one copy of the message exists on the network at a time.

Epidemic (Section 1.2.3)

No adaptations were necessary. All messages are sent to all nodes.

Binary Spray and Wait (Section 1.2.7)

Implemented as described in [17]. This was adapted so that when a node with the message to be sent is in the wait phase, it forwards the message on to nodes which subscribe to any interest in the message (ordinarily, this would be transmitted to the destination node). The L value (which determines how many messages are sent from the original node) chosen was a power of 2 large enough to let it achieve a similar MDR to the epidemic case.

4.2 Performance on Whole Network

4.2.1 Workload

In order to test performance, a suitable workload[2] had to be devised. As table 4.1 shows, the standard deviation for each node is quite high indeed. The most popular node (with the most contacts) interacts with other nodes 3975 times while the least popular only interacts 4 times (histogram in appendix, figure C.1). There are 97 nodes in the entire set. I shall assume that approximately a tenth of these wish to send messages - 10 nodes. These nodes are picked to proportionally represent their popularity (in terms of number of interactions).

In order to determine which interests test messages should be tagged with, a test message from the most popular node was sent with each interest to observe what impact the popularity has on routing. There is not much variation between interests in the time that it takes for the MDR to converge (this can be observed in appendix figure C.2). As a result, I decided simply to pick 10 tags, interspaced the same distance apart from the list of tags sorted by their subscription ratio. This was repeated for each mapping since the differently assigned interests will cause the subscription ratios to differ.

These tags are assigned to nodes based on the assumption that those with fewer interactions are less likely to share a popular interest. Preliminary tests have shown this does not affect results. For example, the most popular node will send a message tagged with '#acoustic' for random mapping. Its subscription ratio suggests 70% of users wish to receive these messages. Appendix table C.3 shows the nodes and the interest that each sent message will be tagged with. For the purposes of this evaluation, sending a message tagged with just one interest is sufficient.

The message expiry will be fixed to a time where Epidemic routing is shown to have converged to a MDR since this is the fastest that messages can be delivered. Other algorithms can them be evaluated based on their performance in this time period.

4.2.2 Choosing Parameters

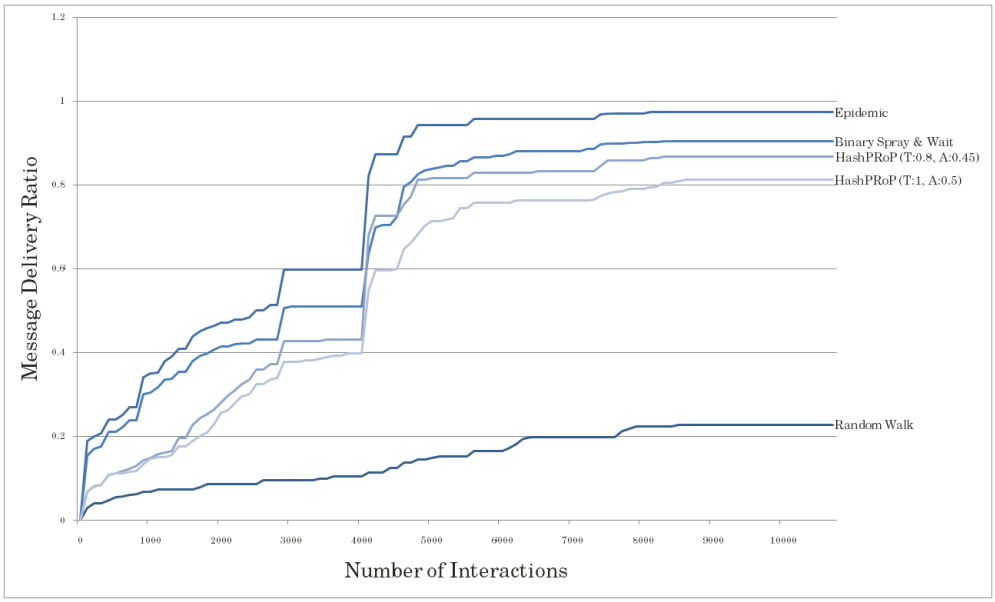
It was necessary to evaluate various threshold and α values in order to pick the optimum values for the dataset at hand. This will allow for better performance. These were picked for each scenario by evaluating a range of different parameter combinations and choosing two which offered either increase message delivery at higher cost, or decreased message delivery at lower cost.

Comparative Performance

Performance will be compared for each mapping individually.

[2]A workload is the load placed on the network used for evaluation. In this case, this is the messages that are sent across the network during testing.

a)



b)

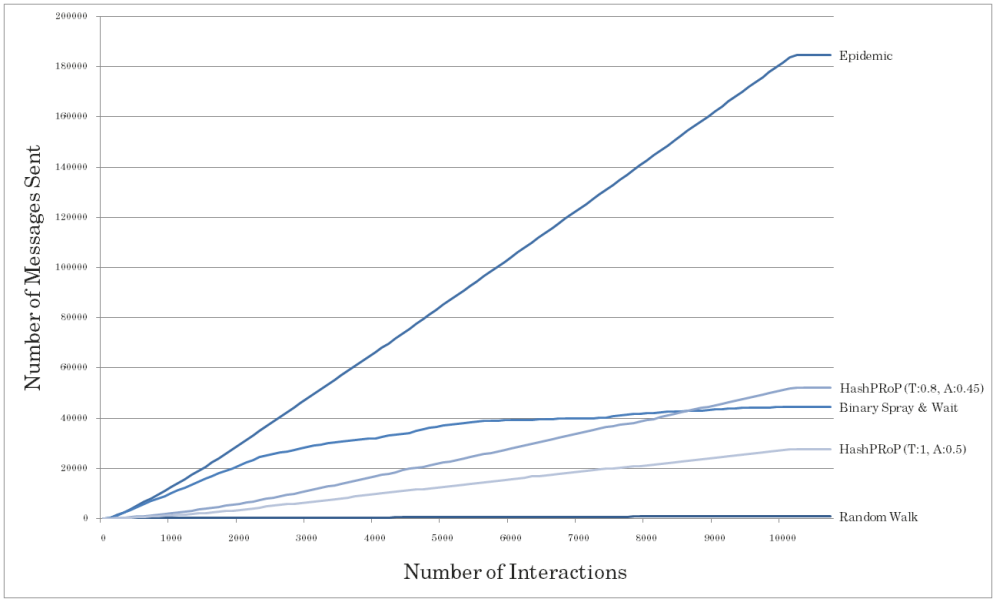


Figure 4.4: Performance of different algorithms over entire dataset using a random mapping.

4.2.3 Randomly Mapped

Figure 4.4 shows the performance of each algorithm when using a random mapping of interests to devices. The graphs are truncated once it was clear that the MDR had converged and was unlikely to increase further.

Observations

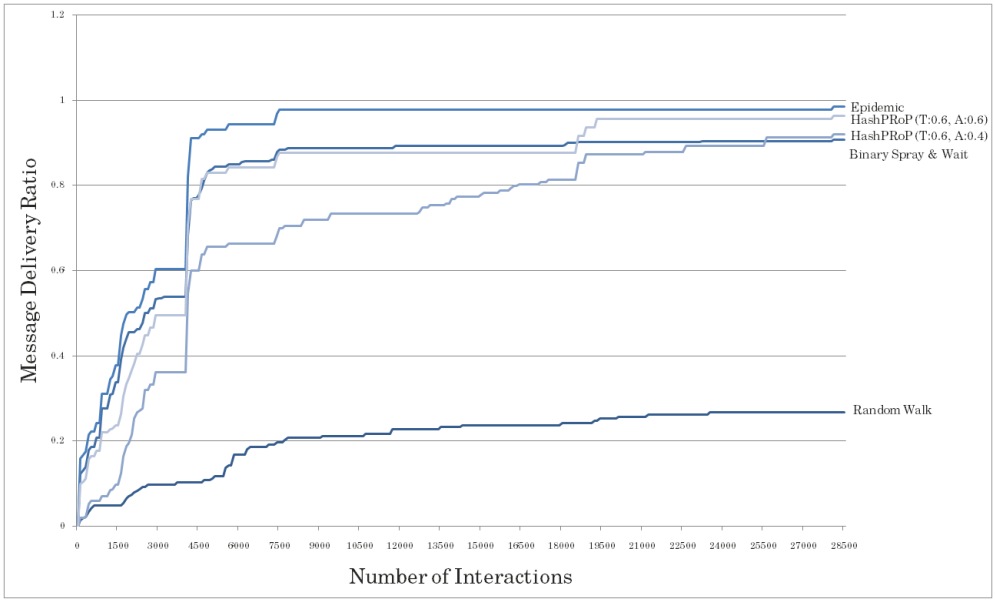
a) MDR:

- Random Walk achieves an extremely poor delivery ratio.
- HashPRoP, Binary Spray & Wait and Epidemic all achieve a similar ratio. Epidemic converges on the highest ratio but this is to be expected (it is the best case).
- There is a clearly observable spike just before 4,000 interactions. This most likely corresponds to an event where all nodes were present (this can be observed on figure 3.7, where there is also a noticeable spike).
- The largest gain appears to be made early on - within the first 100 interactions. Another large gain in delivery ratio is achieved near the spike where HashPRoP (T:0.8, A:0.45) closes the gap between Binary Spray & Wait.

b) Number of Messages Sent:

- Random Walk sends less than 1,000 messages. The most conservative replication based strategy sends almost 30 times as many messages. Epidemic routing sends approximately 200 times as many messages.
- Epidemic and Random Walk routing send a constant number of messages at each interaction. HashPRoP and Binary Spray and Wait show some visible fluctuations - these represent routing decisions being made whether or not to forward messages.

a)



b)

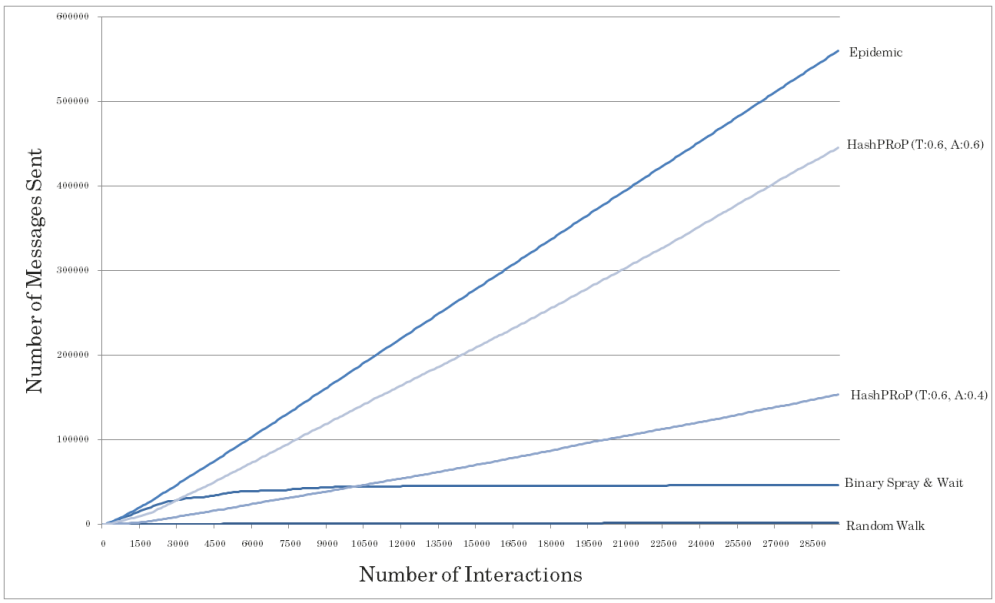


Figure 4.5: Performance of different algorithms over entire dataset using a close mapping.

4.2.4 Closely Mapped

Figure 4.5 shows the performance of each algorithm when using a close mapping of interests to devices, i.e. mapping such that people who encounter each other often share interests too.

Observations

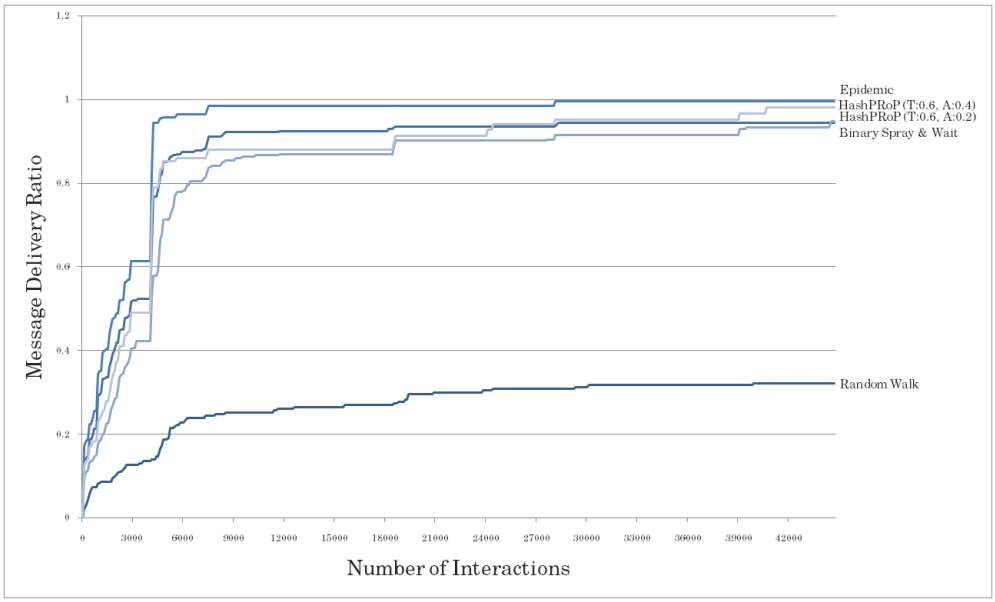
a) MDR:

- Random Walk achieves an extremely poor delivery ratio as with random mapping.
- HashPRoP, Binary Spray & Wait and Epidemic all achieve a similar ratio.
- HashPRoP bests Binary Spray & Wait for both versions.
- HashPRoP (T:0.6, A:0.4) catches up with the delivery ratio after a bad start. Some significant improvement is observable with HashPRoP (T:0.6, A:0.6) at about 18,000 interactions. The other algorithms do not seem to benefit from this.

b) Number of Messages Sent:

- Random Walk sends the fewest messages, followed by Binary Spray & Wait. Each of the HashPRoP series does better than Epidemic routing but worse than Binary Spray & Wait.
- Binary Spray & Wait sends less than a 10th of the number of messages sent by Epidemic but achieves a MDR within 10% of the Epidemic algorithm.

a)



b)

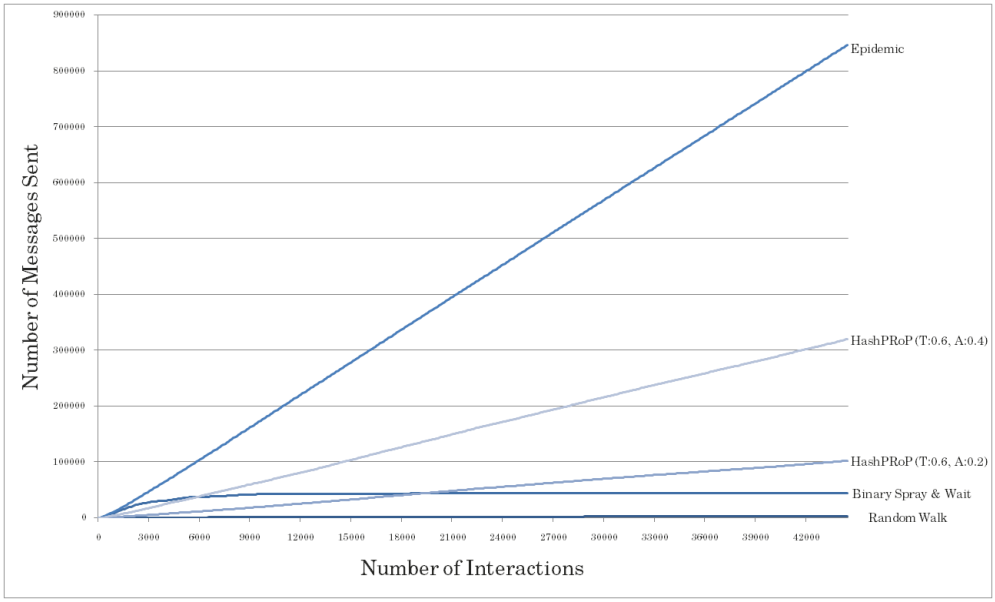


Figure 4.6: Performance of different algorithms over entire dataset using a distant mapping.

4.2.5 Distantly Mapped

Figure 4.6 shows the performance of each algorithm when using a distant mapping of interests to devices, i.e. mapping such that people who encounter each other often don't share interests.

Observations

a) MDR:

- HashPRoP, Binary Spray & Wait and Epidemic all achieve a similar ratio above 0.9.
- HashPRoP bests Binary Spray & Wait as with Closely Mapped interests. There is less of an observable gain but both versions do increase in the latter stages of the simulation whereas the other versions do not.
- It takes many more interactions for the MDR to converge for all algorithms (but primarily HashPRoP) compared to the closely mapped case. This is most likely because it takes longer for a history of interest probabilities (and hence knowledge of routes to then) to be created.

b) Number of Messages Sent:

- Similar result to the closely mapped community, except that the worst HashPRoP case does twice as well (by sending half as many messages) as the Epidemic algorithm.
- Binary Spray & Wait again sends far fewer messages. Since the simulation runs for a longer time before convergence, it sends approximately a 20th of the number of messages sent by the Epidemic case.

4.3 Performance on Geographically Varied Communities

To investigate the performance of HashPRoP on differently geographically located communities, it will be run on 4 different communities as described earlier. Each data set is smaller and at most consists of 15 interacting nodes. This causes some problems since there is not much scope for algorithms to route intelligently due to the limited routes available. Each case will be considered separately (local and dispersed). Randomly mapped interests were used.

4.3.1 Workload

A similar methodology to the whole dataset is employed here. For each community, a tenth of the number of nodes is used (and rounded up to the nearest integer). Since these nodes will be meeting often over the course of the whole interaction period, one or two of the least popular tags (based on subscription ratio) are chosen. Likewise, the least popular nodes are chosen to send the messages in the hope that this will favour intelligent routing. Appendix table C.3 lists the test messages.

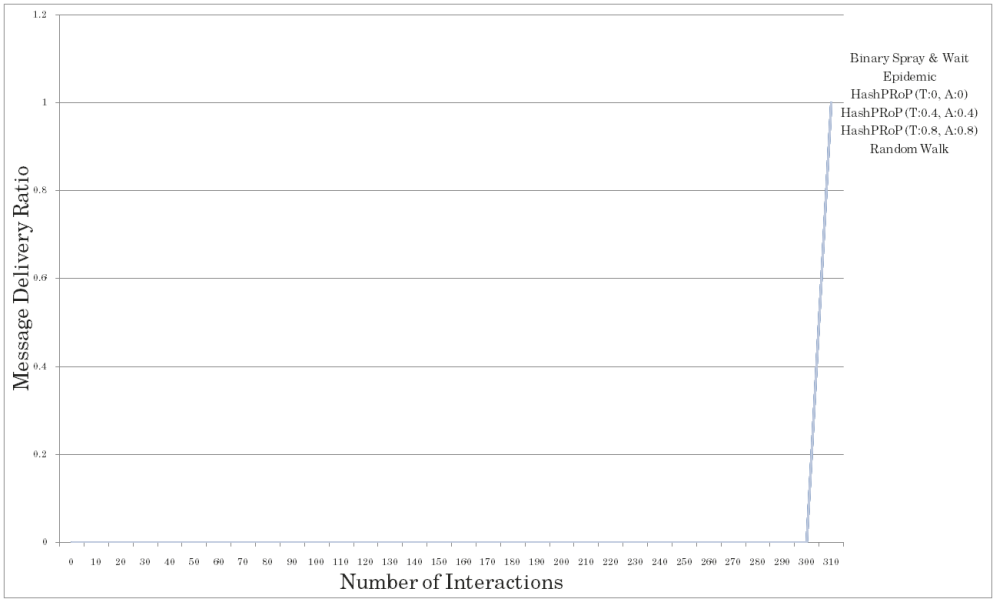
4.3.2 Choosing Parameters

Again, it was necessary to pick optimum threshold and α values for HashPRoP routing. The two geographical cases will be considered separately. MDR was not a useful statistic in this case since this always converged to 1 within the lifespan of the simulation. Instead, the number of interactions before the MDR converged was recorded since this measures the latency involved in message delivery. It is preferable most often to minimise this delay.

Figure C.3 shows the effect of different threshold and α values on the number of messages sent and the number of interactions before the MDR converged to 1 for 2 local communities. Figure C.4 shows the effect on 2 dispersed communities. As with the full dataset, there is no clearly tangible trend in each graph, except for the larger local community. It appears to that the effect of the parameters are largely dependent on the specific community and subscribed interests.

For the purposes of evaluation, 3 parameter combinations were chosen for each case. This allows the most optimum for each community to be chosen, in addition to a parameter value somewhere in the middle.

a) Smaller Community



b) Larger Community

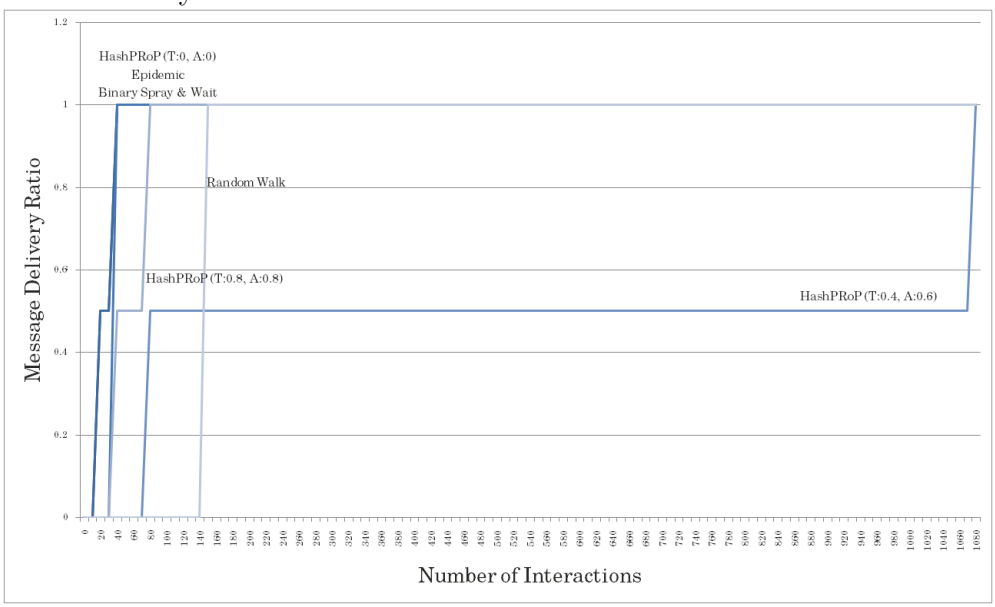


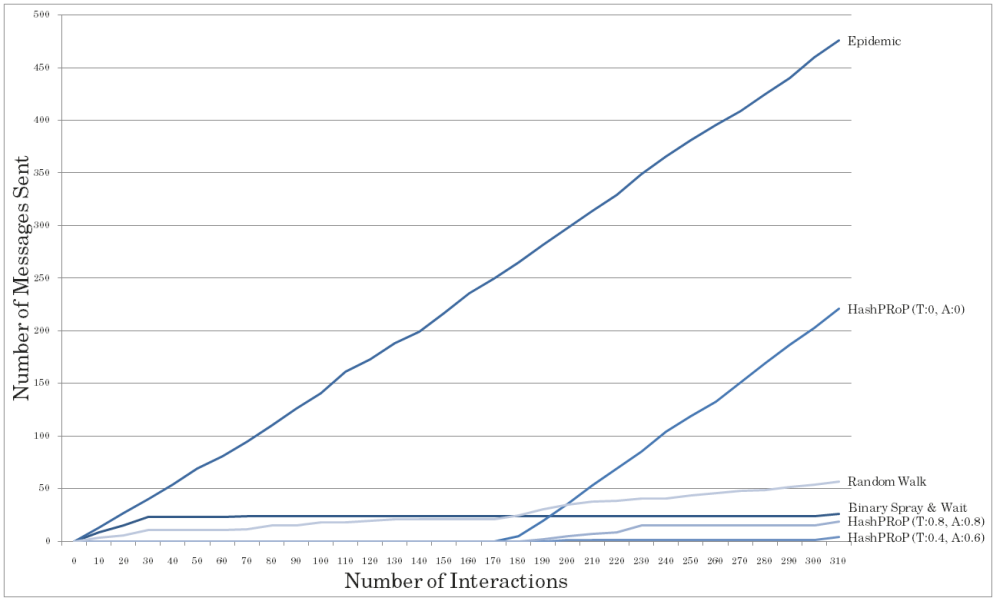
Figure 4.7: MDR against Number of Interactions for two local communities.

4.3.3 Local Communities

Message Delivery Ratio Figure 4.7 shows how each algorithm approached complete message delivery for each local community. Note that the *blocky* nature of the graph is due to the relatively small ratio of nodes subscribed to each interest, resulting in a large increase in MDR everytime a message is delivered.

- For the smaller community of 6 nodes shown in a), all algorithms converge at the same time. This suggests that there is only one possible route.
- The larger community shown in b) however shows some variance. HashPRoP with parameters set to 0, Epidemic and Binary Spray & Wait deliver all messages the quickest. HashPRoP (T=0.8, A=0.8) is quicker than Random Walk. HashPRoP (T=0.4, A=0.6) is the slowest by a large margin.

a) Smaller Community



b) Larger Community

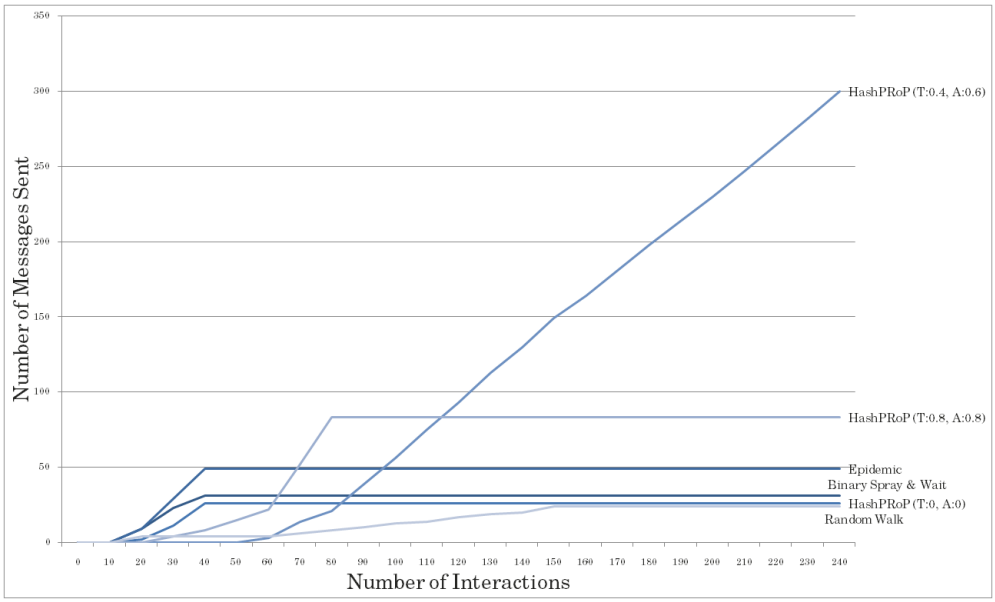


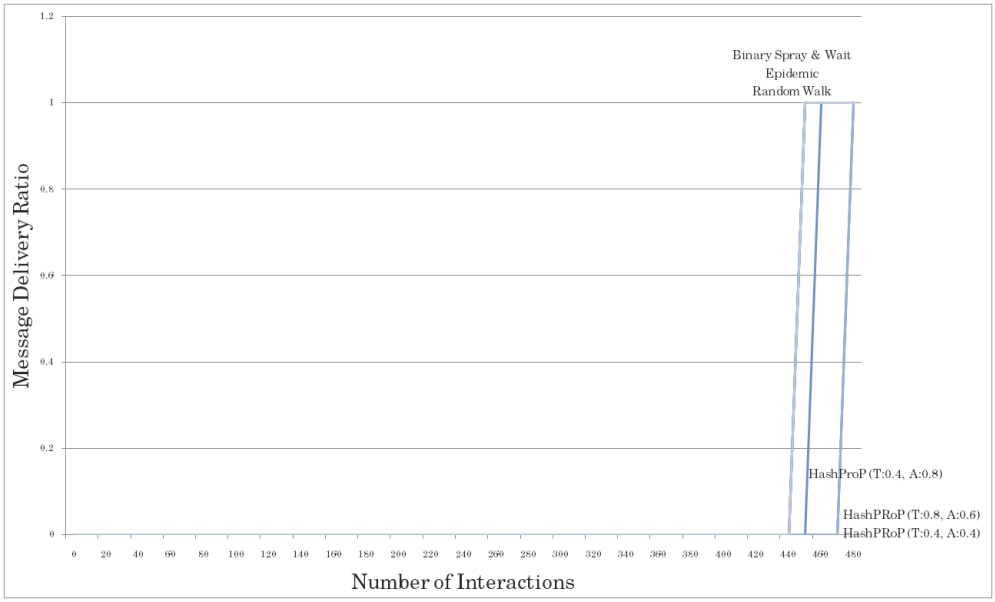
Figure 4.8: Number of Sent Messages against Number of Interactions for two local communities.

Number of Messages Sent Figure 4.8 shows how messages were sent over each local community.

- In the smaller community in a), there is a large difference between the number of messages sent out from Epidemic to HashPRoP (T=0.4, A=0.6), nearly 125 times more messages are sent. HashPRoP and Binary Spray & Wait are clearly more efficient given that the same MDR was achieved in the same time. Since there is only one route (as shown in the previous graph), there is less to be gained from flooding the network with messages as Epidemic does. This favours a smarter routing strategy.

- In the larger community in b), HashPRoP appears weaker. This graph has been truncated for visibility reasons but HashPRoP (T=0.4, A=0.6) sends out many messages. This is related to the lack of early MDR convergence - the other routing algorithms have been stopped earlier on. HashPRoP (T:0.8, A:0.8) sends out more messages than the Epidemic case for the same reason. Considering all the other faster algorithms, HashPRoP (T:0, A:0) achieves the second lowest number of sent messages, and is very close to that of Binary Spray & Wait.

a) Smaller Community



b) Larger Community

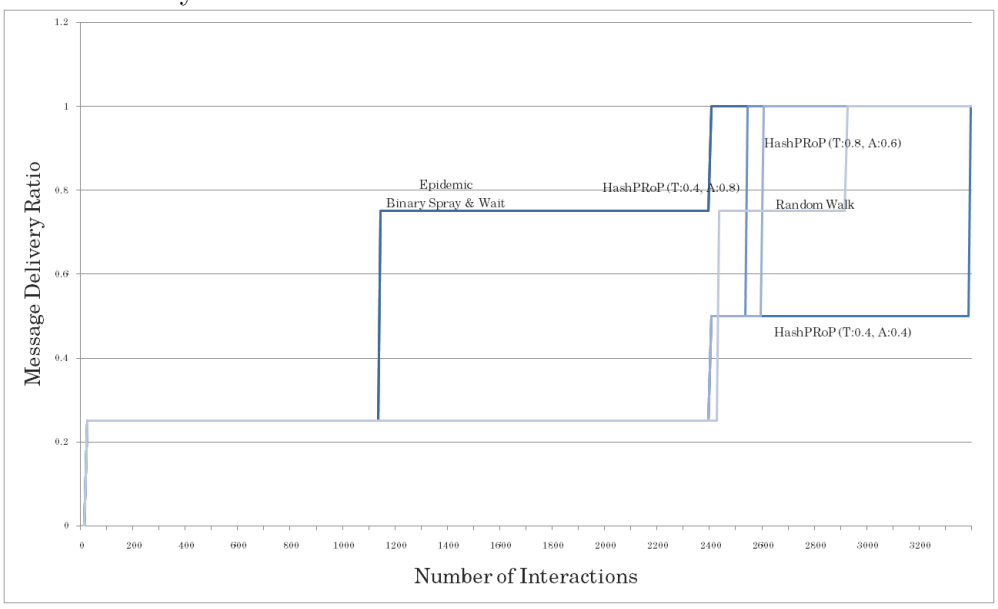


Figure 4.9: MDR against Number of Interactions for two dispersed communities.

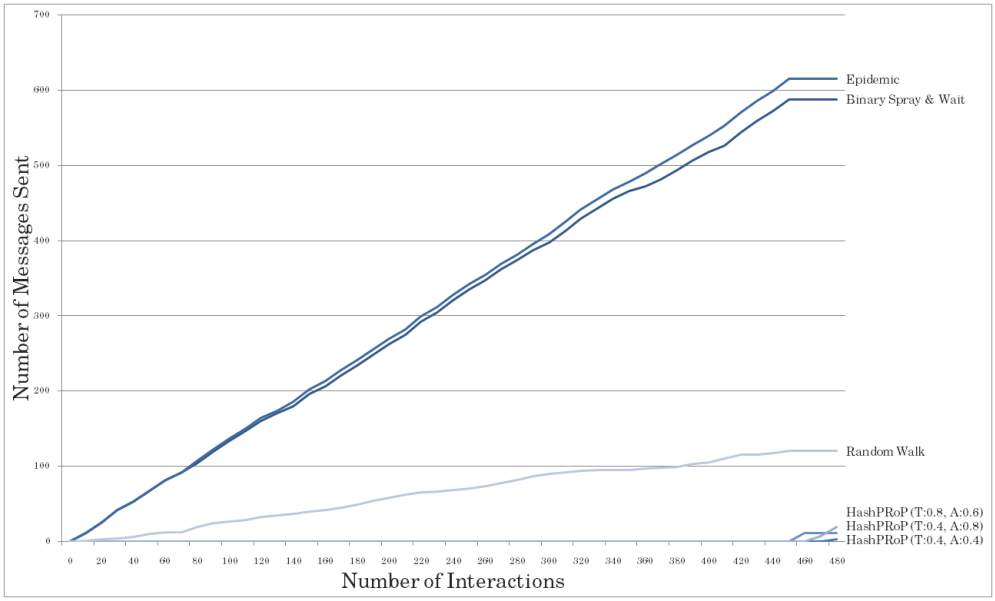
4.3.4 Dispersed Communities

Message Delivery Ratio Figure 4.9 shows how each algorithm approached complete message delivery. As with the local community case, there is little variance in the smaller community and more so in the larger community. This is most likely due to the increased number of routes available in a larger network.

- In the smaller community in a), Binary Spray & Wait, Epidemic and Random Walk are all faster than the HashPRoP algorithms. Not by a large margin, but this is significant nonetheless. The lack of variation between the number of interactions this takes is similar to the small local community - only a limited number of routes exist.

- In the larger community in b), Epidemic and Binary Spray & Wait are the fastest to converge. It takes almost double the number of interactions for the rest to converge - HashPRoP (T:0.4, A:0.8) and HashPRoP (T:0.8, A:0.6) beat Random Walk. HashPRoP (T:0.4, A:0.4) takes the longest (at 3 times the number of interactions that the fastest took).

a) Smaller Community



b) Larger Community

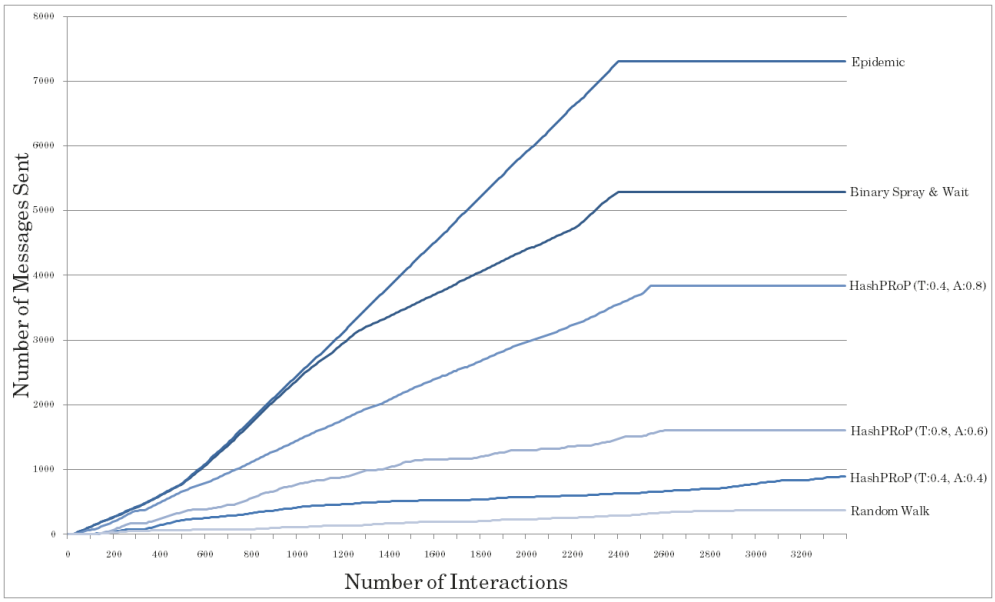


Figure 4.10: Number of Sent Messages against Number of Interactions for two dispersed communities.

Number of Messages Sent Figure 4.10 shows the number of sent messages for each dispersed community. Where the line flattens, the simulation was stopped for that algorithm.

- HashPRoP does extremely well in the smaller community in a), sending a minuscule number of messages compared to Epidemic and Binary Spray & Wait. It even manages to send fewer messages than Random Walk.
- In the larger community in b), it again beats Epidemic and Binary Spray & Wait. HashPRoP (T:0.4, A:0.4) which is the slowest to converge, also sends the fewest messages - at near to double what Random Walk sends (compared to 16x as many messages send by the Epidemic algorithm).

4.4 Remarks on Performance

The requirements state that delivery latency must be less than a random walk routing algorithm and that the cost of message delivery must be less than an epidemic routing algorithm. Performance on the entire dataset has fulfilled both of these requirements. It fulfills this requirement on all communities tested except for the smaller geographically dispersed community where latency is higher and for the larger geographically local community where more messages are sent.

HashPRoP works reasonably well on all communities, sending far fewer messages than the Epidemic algorithm in most cases whilst achieving a similar MDR. For all but the smaller dispersed community it achieves good latency at the benefit of a reduction in the number of messages sent. It works best over smaller communities - over the whole dataset delivery cost is always worse than Binary Spray & Wait.

4.4.1 Effect of Mapping

There was a noticeable difference in routing when using differently mapped communities. Although over time the MDR converged to a good level similar to that of the best case, all algorithms were much quicker to converge using a distant mapping. This suggests that a distant mapping (where frequent encounters are not with people who share the same interest) favours HashPRoP. Intuitively, this seems valid since this would lead to more diversity in the interests that a node has seen. More diversity means that paths are more likely to exist. However, it is difficult to draw any deeper inference without further evaluation over several different communities mapped in a similar way.

4.4.2 Effect of Geographical Distribution

HashPRoP sends fewer messages in general for the dispersed communities whilst achieving a latency close to Epidemic and Binary Spray & Wait (the best cases). For certain parameter values it is slower than Random Walk. It works less well on the local communities where latency is similar but for certain parameter combinations many more messages are sent. These results would suggest it works better over more dispersed communities but further evaluation is necessary.

4.5 Smartphone Implementation

The smartphone application was tested to be working on 3 Windows Mobile devices. Unfortunately, due to a lack of any more devices, it was not possible to test it on a larger scale. A few issues became apparent:

- Speed of Bluetooth scan and message transmission. Scans took up to 20 seconds, and more depending on the number of devices nearby. If a device is discovered then it takes even longer for the scan to complete. Scan time should not be fixed to allow for these possibilities (since the next scan would start immediately or interrupt the current scan).

- There were some odd issues with synchronicity where phones would try to send messages to each other at the same time and both fail. Perhaps this is because of the scan time being too short (if it was longer there is more chance that one device would be free when the other is trying to send). Perhaps adding a random interval into the time in between scans would help.

Comparing it against the requirements laid out in section 2.2, it is largely successful and able to send messages seamlessly using the data format and algorithm designed for the simulation application. Implementing it in Wi-Fi would also interesting since it is quicker and offers a larger range (this means that people would be 'in contact' for longer since their phones could still communicate with each other and that they would not have to actually come within 10 metres of each other).

Chapter 5

Conclusion

5.1 Result

All requirements laid out in section 2.2 have been successfully met. A working smartphone application has been created and provides adequate proof of concept for such a network. A simulation application was created that allowed extensive evaluation of different routing algorithms on a large set of test data.

HashPRoP, a routing algorithm based on interests was proposed and shown to be more efficient than several other routing algorithms when routing interest tagged messages. However, the results leave some further investigation to be desired. It is not clear what the specific effect of the two parameters is on the algorithm and the current method of selecting them is not at all ideal. Practically, devices would need to pick optimum threshold and α values on their own. I speculate that the values would have an increasingly graduated effect on larger communities but unfortunately there was not enough time nor test data readily available to experiment upon.

Finally, although HashPRoP achieves a decent delivery ratio, this is usually at the cost of a disproportionally large increase in the number of messages sent. Whilst it may be efficient in specific cases, Binary Spray & Wait remains the best all-round routing strategy.

5.2 Personal Thoughts

Pocket Switched Networks could provide a very useful additional message bus on which data could be sent. While message delivery is not quick enough to replace traditional infrastructure based methods of communication, it is tolerable for some applications where timing is less of an issue.

Mobile ad-hoc networks are very useful in circumstances where there is a lack of infrastructure. For example, the Delay Tolerant Link State Routing (DTLSR) protocol is a modification of existing link state routing protocols that allows link state announcements to persist over partitions in the network (breaks in the network are likely to be temporary) and to take into the account the probability that a link will return based on its history [9]. DTLSR has been tested in pilot projects in developing regions with positive results and has been considered for connectivity by the OLPC Project [24].

5.3 Future Extensions

Exploiting Path Explosion to reduce message delivery time

Path Explosions in PSNs is where once the first path is created, the number of other paths to the destination grows rapidly. To exploit this phenomenon, it is necessary to act to reduce the time taken to Path Explosion by forwarding messages deliberately towards 'high rate' nodes - those which come into contact with more other nodes most often [20]. It is clear in the larger local community evaluated that one of the HashPRoP parameter combinations results in many more messages being sent, presumably because it missed a forwarding path early on in the simulation. Perhaps a situation where the parameters change as the simluation progresses would reduce messages sent overall yet increase the probability of reaching desirable forwarding paths.

Bibliography

[1] *Even the dolphins use pocket switched networks!;* http://www.cs4fn.com/internet/dolphinnetworks.donphp; 2007

[2] *Google Android loses Bluetooth API, Bluetooth still lives;* http://blogs.zdnet.com/hardware/?p=2450; 2008

[3] *Wi-Fi Army;* http://www.wifiarmy.com

[4] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen; *A Comparative Study of Wireless Protocols: Blueooth, UWB, ZigBee, and Wi-Fi;* Information & Communications Research Labs, Industrial Technology Research Institude, Hsinchu, Taiwan; 2007

[5] *Twitter Downtime: As Bad As Everyone Says It Is;* http://www.businessinsider.com/2008/5/no-social-network-is-down-more-than-twitter; 2008

[6] Nathan Eagle and Alex Pentland; *Reality mining: sensing complex social systems;* Pers Ubiquit Comput; 2006

[7] Nathan Eagle; *Using Mobile Phones to Model Complex Social Systems;* http://www.oreillynet.com/pub/a/network/2005/06/20/MITmedialab.html; 2005

[8] Cecilia Mascolo; *Advanced Systems Topics Lecture Notes;* 2009

[9] Michael Demmer and Kevin Fall; *DTLSR: Delay Tolerant Routing for Developing Regions;* NSDR '07; 2007

[10] Amin Vahdat and David Becker; *Epidemic Routing for Partially-Connected Ad Hoc Networks;* Technical Report CS-2000-06, Department of Computer Science, Duke University; 2000

[11] Pan Hui, Jeremie Leguay, Jon Crowcroft, James Scott, Timur Friedman, Vania Conan; *Osmosis in Pocket Switched Networks;* First International Conference on Communications and Networking in China, Beijing, China; 2006

[12] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, James Scott; *Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding;* Technical Report 617, University of Cambridge, Computer Laboratory, 2005

[13] Pan Hui and Anders Lindgren; *Phase Transitions of Opportunistic Communication;* CHANTS '08; 2008

[14] Wenrui Zhao, Mostafa Ammar and Ellen Zegura, *A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks;* MobiHoc '04; 2004

[15] Sushant Jain, Kevil Fall, Rabin Patra; *Routing in a Delay Tolerant Network;* SIGCOMM '04; 2004

[16] Aruna Balasubramanian, Brian Neil Levine and Arun Venkataramani; *DTN Routing as a Resource Allocation Problem;* SIGCOMM '07; 2007

[17] Thrasyvoulos Spyropoulos, Konstantinos Psounis and Cauligi S. Raghavendra; *Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks;* SIGCOMM '05; 2005

[18] Dan Henriksson, Tarek F. Abdelzaher and Raghu K. Ganti; *A Caching-Based Approach to Routing in Delay-Tolerant Networks;* IEEE, 2007

[19] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft and Christophe Diot; *Pocket Switched Networks and Human Mobility in Conference Environments;* SIGCOMM '05; 2005

[20] Vijay Erramilli, Mark Crovella, Augustin Chaintreau and Christophe Diot; *Diversity of Forwarding Paths in Pocket Switched Networks;* IMC '07; 2007

[21] Pan Hui, Augustin Chaintreau, Richard Gass, James Scott, Jon Crowcroft and Christophe Diot; *Pocket Switched Networking: Challenges, Feasibility and Implementation Issues;* WAC '05; 2005

[22] Hoang Anh Nguyen, Silvia Giordano and Alessandro Puiatti; *Probabilistic Routing Protocol for Intermittently Connected Mobile Ad hoc Network;* IEEE, 2007

[23] Anders Ligren, Avri Doria, Olov Schelén; *Probabilistic Routing in Intermittently Connected Networks;* Department of Computer Science and Electrical Engineering, Luleå University of Technology, Sweden. Electronics and Telecommunications Research Institute (ETRI), Korea; 2004

[24] *OLPC Project;* http://www.laptop.org

[25] *delicious social bookmarking;* http://delicious.com

[26] *Last.fm;* http://www.last.fm

[27] *Graphviz;* http://www.graphviz.org/

[28] Muhammad Saleem; *Collaborative Filtering: Lifeblood of The Social Web;* http://www.readwriteweb.com/archives/collaborative_filtering_social_web.php

[29] *Last.fm Lib .Net - An open source Last.fm library;* http://lastfmlibnet.sourceforge.net

[30] *lastfm-sharp;* http://code.google.com/p/lastfm-sharp/

[31] *QuickGraph;* http://www.codeplex.com/quickgraph

[32] Lev Nachmanson, George Robertson and Bongshin Lee; *Drawing Graphs with GLEE;* GD 2007; 2007

[33] *Automatic Graph Layout;* http://research.microsoft.com/en-us/downloads/f1303e46-965f-401a-87c3-34e1331d32c5/

[34] *32feet.NET;* http://inthehand.com/content/32feet.aspx

[35] Gergely Palla, Imre Derényi, Illés Farkas and Tamás Vicsek; *Uncovering the overlapping community structure of complex networks in nature and society;* Nature vol. 435; 2005

[36] Pan Hui, Jon Crowcroft, Eiko Yoneki; *BUBBLE Rap: Social-based Forwarding in Delay Tolerant Networks;* MobiHoc '08; 2008

[37] *CFinder;* http://www.cfinder.org

Appendix A

Implementation Diagrams

A.1 Database

Database schema diagram of tables used to store adapted version of the Reality Mining data as described in section 3.4.1.

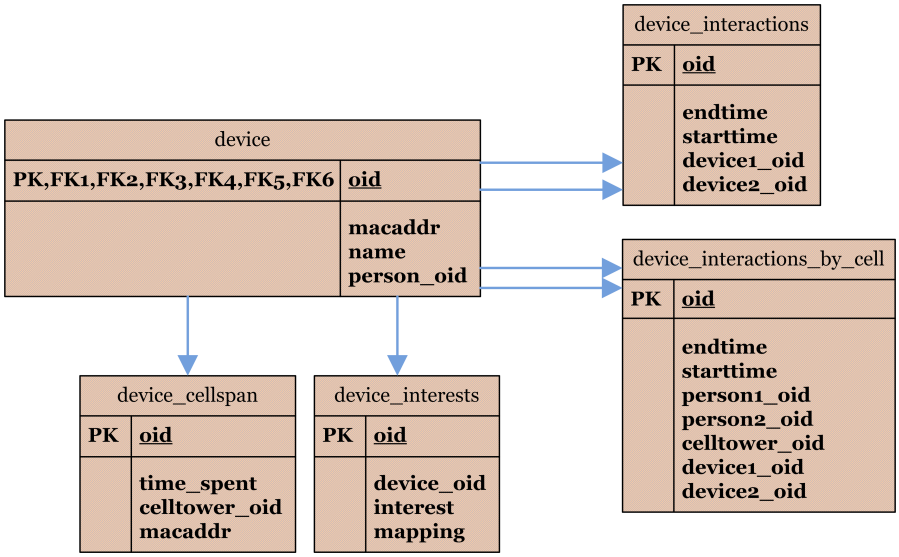


Figure A.1: Database schema diagram.

A.2 Simulation

Class diagram of the main classes used in the simulation application as described in section 3.4.

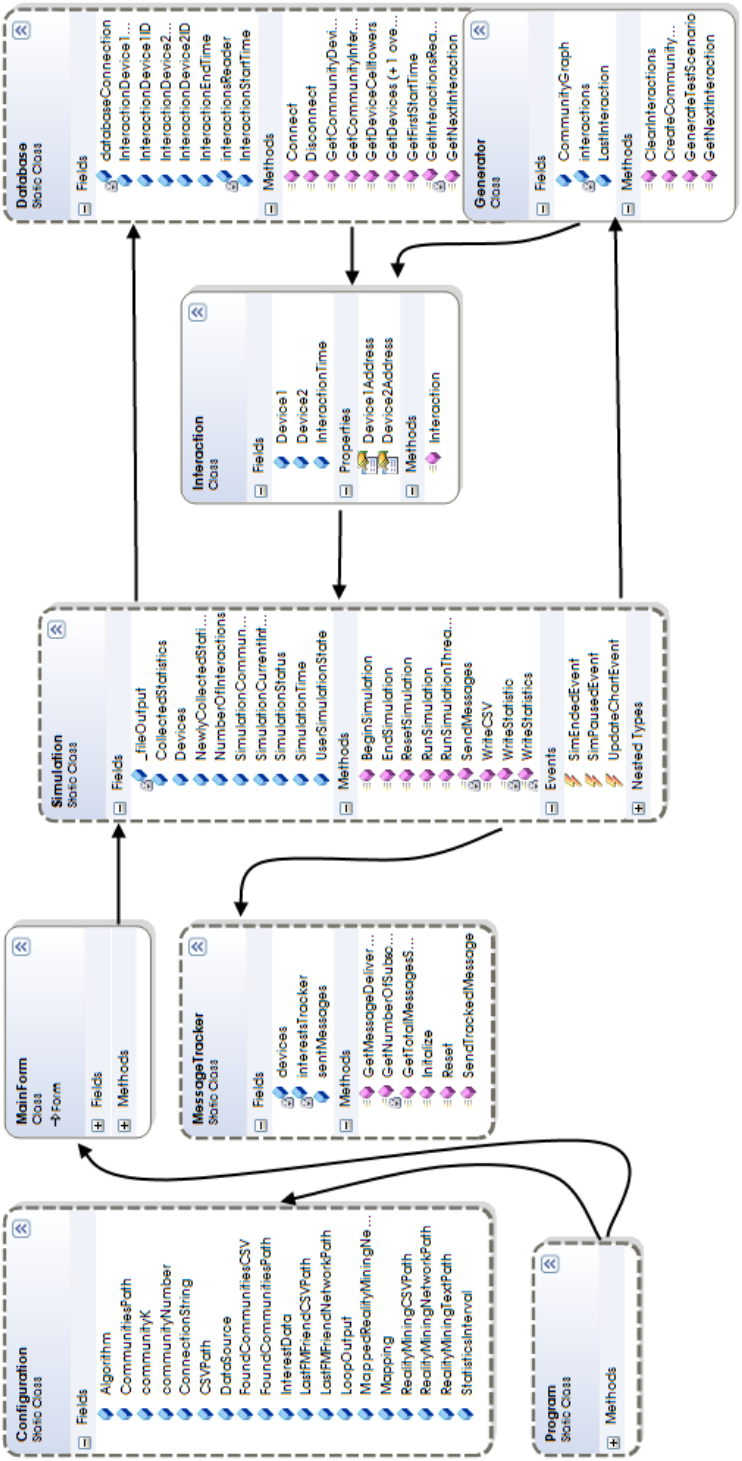


Figure A.2: Class diagram of simulation program.

A screenshot of the simulation application. A treeview on the left shows the state of each node in the interaction, including the state of each folder and the node's interest history. The chart on the bottom right shows the message delivery ratio (in blue) and the number of sent messages (in red) as the simulation is running and is updated every 100 interactions. Finally, top right is the Microsoft GLEE control which automatically draws graphs - useful for visualising the simpler test cases.

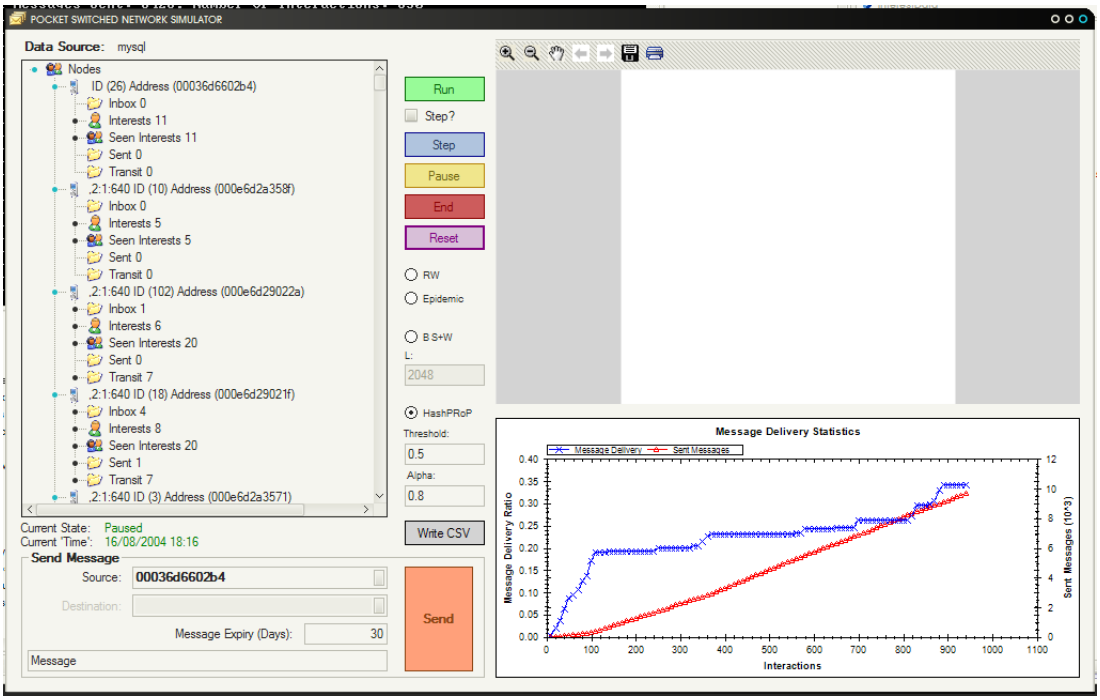


Figure A.3: Screenshot of simulation GUI.

A.3 Mapping

Mapping of many disparate music genres to a simplified set was necessary to reduce the complexity when creating a test workload for evaluation. There were several hundred different genres which were (semi-manually) mapped onto 22 basic genres.

Original Interest	Mapped Interest
electroclash	electronic
electropop	electronic
hardrock	rock
industrialrock	rock
gothicmetal	metal
blackmetal	metal

Table A.1: List of example mapped interests.

A.4 Algorithm

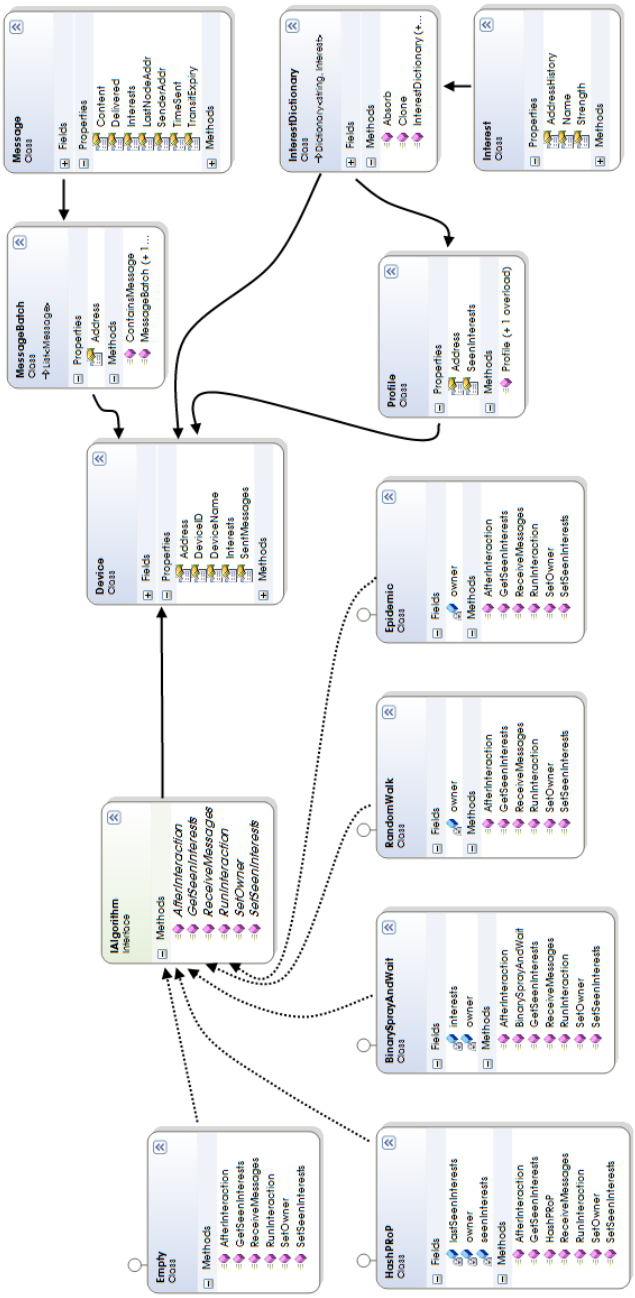


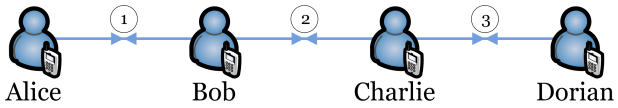
Figure A.4: Class diagram of algorithm project.

Appendix B

Sample Test Cases

The example calculations made in Excel and verified in the simulation application are shown here for a few selected tests mentioned in section 3.3.3.

B.1 Basic (Unidirectional)



Test to check basic routing. Nodes were all given the same subscribed interests and a single message was sent from the first node. All communicating nodes should have been delivered a copy of the message.

Node	Interest	Probability
Start State		
a	physics	1
b	physics	1
c	physics	1
d	physics	1
End State		
a	physics	1
b	physics	1
c	physics	1
d	physics	1

Table B.1: Basic (Unidirectional): Node interests at start and end.

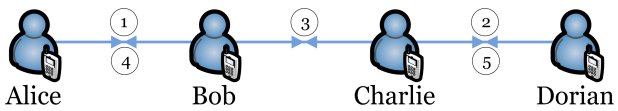
Node	Sent	Inbox
Start State		
a	test #physics	
b		
c		
d		
End State		
a	test #physics	
b		test #physics
c		test #physics
d		test #physics

Table B.2: Basic (Unidirectional): Node message folders at start and end.

Number	Node 1	Node 2
1	a	b
2	b	c
3	c	d

Table B.3: Basic (Unidirectional): Node interactions.

B.2 Probability (Bidirectional)



Test to check that probabilities were being calculated correctly after interactions. Two nodes at the edge of the network were given interests and the probabilities calculated up until the finish state. (No messages were sent.)

Node	Interest	Probability
Start State		
a	physics	1
d	compsci	1
End State		
a	physics	1
b	physics	1
c	physics	0.512
d	physics	0.512
a	compsci	0.512
b	compsci	0.512
c	compsci	1
d	compsci	1

Table B.4: Probability (Bidirectional): Node interests at start and end.

Number	Node 1	Node 2
1	a	b
2	c	d
3	b	c
4	a	b
5	c	d

Table B.5: Probability (Bidirectional): Node interactions.

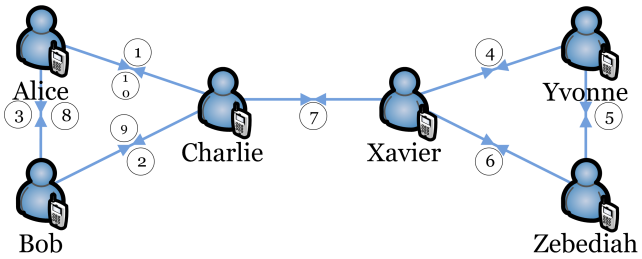
physics					
Interaction	1	2	3	4	5
a	1	1	1	1	1
b	0.8	0.8	0.64	0.8	0.8
c	0	0	0.64	0.64	0.512
d	0	0	0	0	0.512

Table B.6: Probability (Bidirectional): Calculated node interest probabilities for 'physics' after each interaction.

compsci					
Interaction	1	2	3	4	5
a	0	0	0	0.512	0.512
b	0	0	0.64	0.512	0.512
c	0	0.8	0.64	0.8	0.8
d	1	1	1	1	1

Table B.7: Probability (Bidirectional): Calculated node interest probabilities for 'compsci' after each interaction.

B.3 Probability (Larger Test Case)



A larger test case to see how messages are routed across two groups and how probabilities filter through. One message was sent from Z.

Node	Interest	Probability
Start State		
a	biology	1
a	physics	1
b	physics	1
c	physics	1
x	biology	1
y	biology	1
z	biology	1
End State		
a	biology	1
b	biology	1
c	biology	1
x	biology	1
y	biology	1
z	biology	1
a	physics	1
b	physics	1
c	physics	1
x	physics	0.8

Table B.8: Probability (Larger Test Case): Node interests at start and end.

Node	Sent	Inbox
Start State		
a		
b		
c		
x		
y		
z	test #biology	
End State		
a		test #biology
b		
c		
x		test #biology
y		test #biology
z	test #biology	

Table B.9: Probability (Larger Test Case): Node message folders at start and end.

Number	Node 1	Node 2
1	a	b
2	b	c
3	c	a
4	x	y
5	y	z
6	z	x
7	c	x
8	a	b
9	b	c
10	c	a

Table B.10: Probability (Larger Test Case): Node interactions.

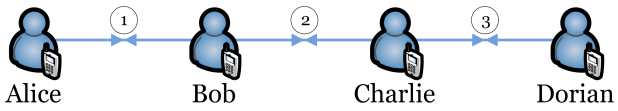
physics										
Interaction	1	2	3	4	5	6	7	8	9	10
a	1	1	1	1	1	1	1	1	1	1
b	1	1	1	1	1	1	1	1	1	1
c	1	1	1	1	1	1	1	1	1	1
x	0	0	0	0	0	0	0.8	0.8	0.8	0.8
y	0	0	0	0	0	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0	0

Table B.11: Probability (Larger Test Case): Calculated node interest probabilities for 'physics' after each interaction.

biology										
Interaction	1	2	3	4	5	6	7	8	9	10
a	1	1	1	1	1	1	1	1	1	1
b	0.8	0.64	0.64	0.64	0.64	0.64	0.64	1	1	1
c		0.64	1	1	1	1	1	1	1	1
x	1	1	1	1	1	1	1	1	1	1
y	1	1	1	1	1	1	1	1	1	1
z	1	1	1	1	1	1	1	1	1	1

Table B.12: Probability (Larger Test Case): Calculated node interest probabilities for 'biology' after each interaction.

B.4 Message Expiry



Simple test case to check that messages are being removed once they have expired.

Node	Interest	Probability
Start State		
a	physics	1
b	physics	1
c	physics	1
d	physics	1
End State		
a	physics	1
b	physics	1
c	physics	1
d	physics	1

Table B.13: Message Expiry: Node interests at start and end.

Node	Sent	Inbox
Start State		
a	test #physics, expiry 3 hours	
b		
c		
d		
End State		
a	test #physics	
b		test #physics
c		test #physics
d		

Table B.14: Message Expiry: Node message folders at start and end.

Number	Node 1	Node 2	Time
1	a	b	0
2	b	c	2
3	c	d	4

Table B.15: Message Expiry: Node interactions.

Appendix C

Supplementary Figures from Evaluation

k	Community Number	Number of Nodes	Number of Cell Towers
3	0	19	55
3	1	4	15
3	2	7	18
3	3	6	14
3	4	5	20
3	5	15	35
3	6	3	7
4	0	4	15
4	1	7	18
4	2	6	14
4	3	5	20
4	4	15	35
4	5	15	44
5	0	5	14
5	1	10	23
5	2	15	44
5	3	6	15
5	4	9	27
6	0	6	15
6	1	8	24
6	2	14	43
6	3	7	17
7	0	7	17
7	1	13	39

Table C.1: Summary of extracted communities.

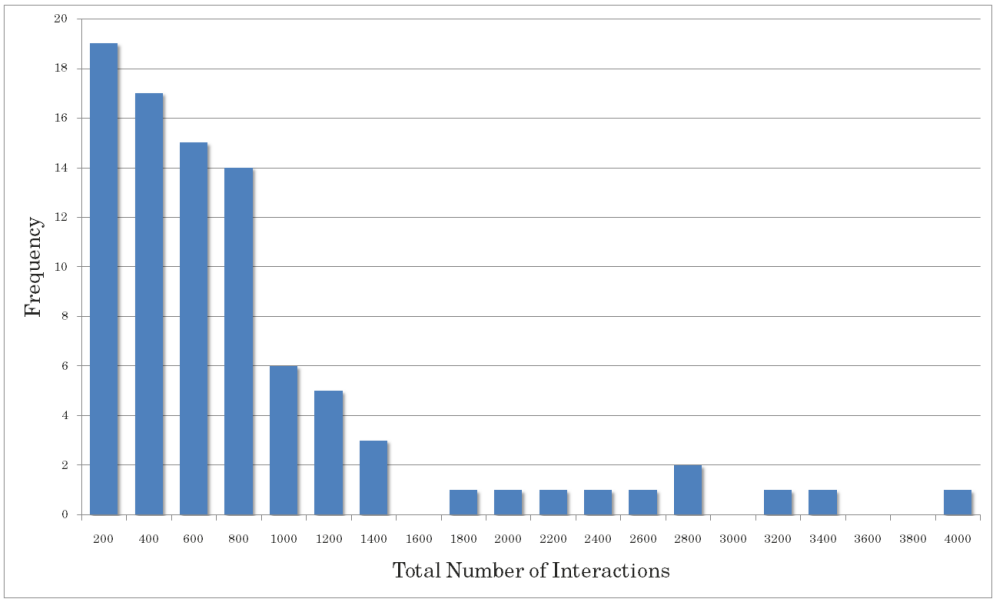


Figure C.1: Histogram showing distribution of node interactions.

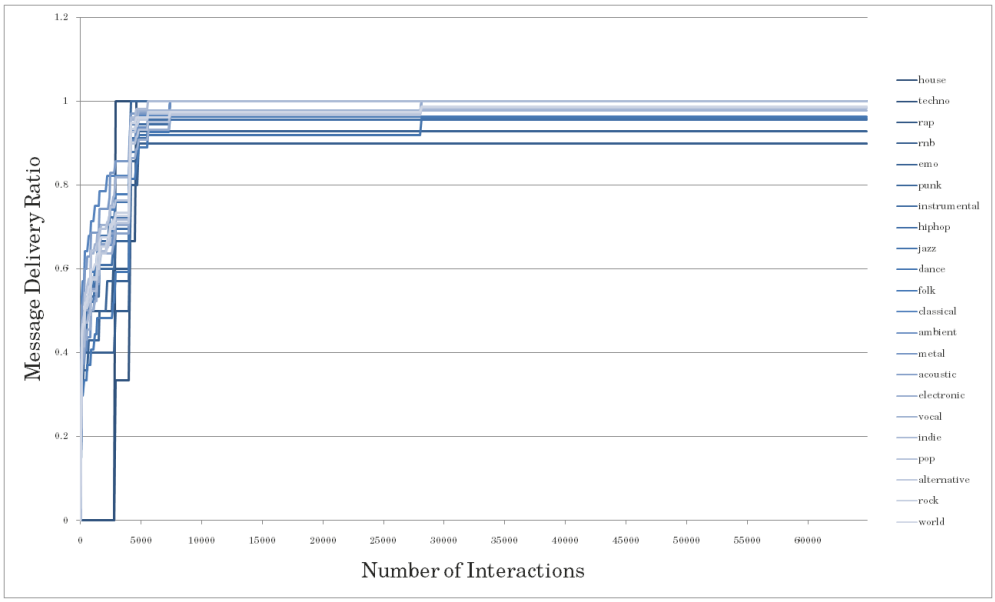


Figure C.2: MDR of different interests over all interactions.

Node Address	# of Interactions	Tagged Interest	Interest Ratio
k=3, 5 nodes (local)			
006057e14286	183	punk	0.2
k=3, 6 nodes (dispersed)			
000e6d2b0743	173	acoustic	0.1667
k=4, 15 nodes (local)			
000e6d6602b5	332	emo	0.0667
000e6d6c5b4c	372	jazz	0.0667
k=4, 15 nodes (dispersed)			
000e6d29021f	138	rnb	0.1333
000e6d668bd6	304	emo	0.0667

Table C.3: Test messages used to test performance of algorithms on different communities.

These figures show the effect on Number of Interactions to Convergence and the Number of Messages Sent for different parameter combinations. The parameters have a clearly noticeable effect; as both figures show, there is a difference in its effect between smaller communities (on the left) and larger communities (on the right).

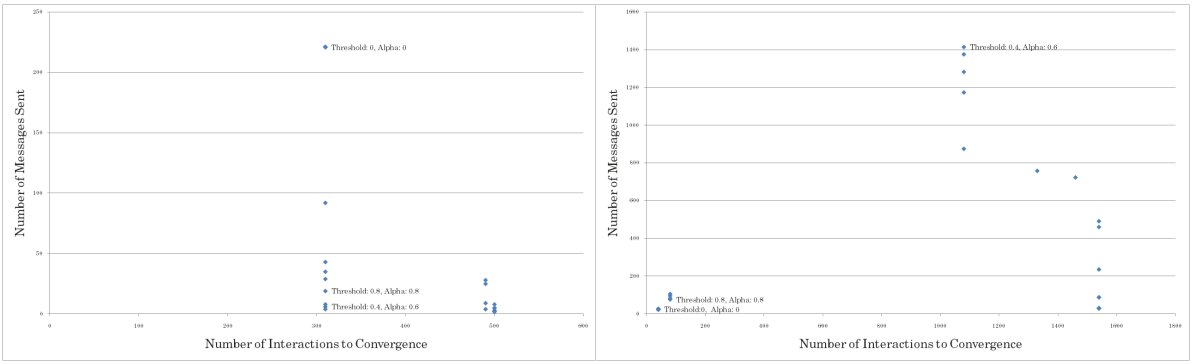


Figure C.3: Scatter graphs of Number of Messages Sent against Number of Interactions to Convergence for two local communities

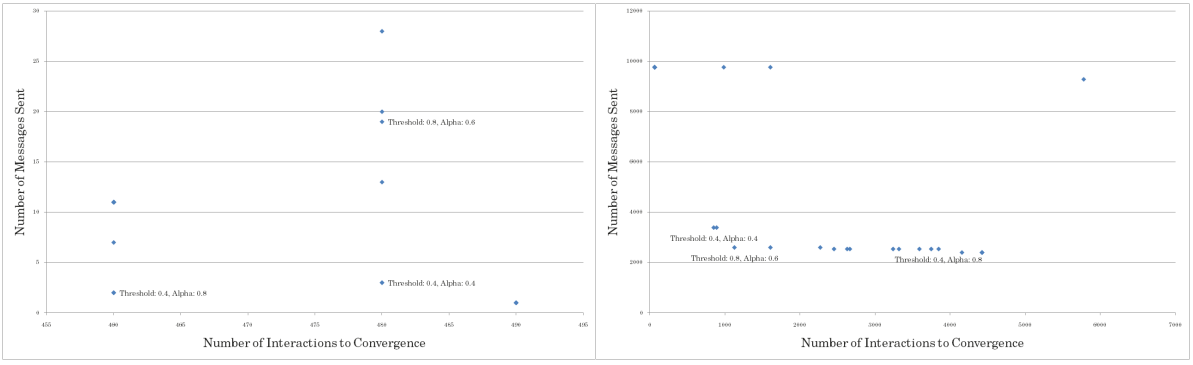


Figure C.4: Scatter graphs of Number of Messages Sent against Number of Interactions to Convergence for two dispersed communities

		Random Mapping	
Node Address	# of Interactions	Tagged Interest	Interest Subscription Ratio
000e6d2a35b6	100	techno	0.0313
000e6d29021f	199	rnb	0.1042
000e6d2a359f	304	punk	0.1563
000e6d6602b5	416	hiphop	0.2396
000e6d2a35c8	537	dance	0.2813
000e6d664887	661	classical	0.2917
000e6d6603d5	812	metal	0.3646
000e6d2a35af	913	electronic	0.4583
000e6d2a35c3	2291	indie	0.5521
000e6d2a3609	3259	alternative	0.7188
		Close Mapping	
Node Address	# of Interactions	Tagged Interest	Interest Subscription Ratio
000e6d2a35b6	100	rnb	0.0435
000e6d29021f	199	techno	0.0761
000e6d2a359f	304	punk	0.1848
000e6d6602b5	416	instrumental	0.2174
000e6d2a35c8	537	folk	0.2500
000e6d664887	661	acoustic	0.2935
000e6d6603d5	812	dance	0.3370
000e6d2a35af	913	vocal	0.4022
000e6d2a35c3	2291	electronic	0.4674
000e6d2a3609	3259	pop	0.5978
		Distant Mapping	
Node Address	# of Interactions	Tagged Interest	Interest Subscription Ratio
000e6d2a35b6	100	rap	0.0313
000e6d29021f	199	techno	0.0729
000e6d2a359f	304	hiphop	0.1458
000e6d6602b5	416	instrumental	0.2396
000e6d2a35c8	537	classical	0.2500
000e6d664887	661	dance	0.3125
000e6d6603d5	812	acoustic	0.3958
000e6d2a35af	913	vocal	0.4271
000e6d2a35c3	2291	indie	0.5104
000e6d2a3609	3259	alternative	0.6458

Table C.2: Test messages used to test performance of algorithms over whole dataset.

Appendix D

Project Proposal

Distributed Twitter: Project Proposal

Sunil Shah(sds51), Fitzwilliam College

Originator: Nishanth Sastry

Special Resources Required

The use of my own Desktop PC (AMD Opteron 165 Dual Core, 2GB RAM and 1.8TB HDD)

Project Supervisor: Nishanth Sastry

Director of Studies: Dr. Pietro Lio'

Project Overseers: Dr. F. Stajano & Dr. D. Greaves

D.1 Overview

Twitter is a popular micro-blogging website which allows users to send short messages to acquaintances who have chosen to follow them. These short messages convey brief status updates and are about 140 characters in length. These messages are distributed by a variety of methods, including an online page which lists all status updates and the option to receive such updates by SMS or instant message. Status updates, despite their brevity, can sometimes convey quite meaningful information.

The basic structure of Twitter's architecture is client-server, the server handles all message processing and clients may request updates from the server or can send updated statuses for a particular user (of course, the original Twitter was purely web-based and there were no clients as such but this is less the case now). This project aims to implement a pocket switched network[1] which allows messages of a similar length to be passed from any particular person to another person by using their mobile device. Furthermore, this project will investigate the possibility of using pocket switched networks to transfer messages based on topics defined by hashtags. Evaluation will take place using actual human mobility traces which allow simulation of real life situations.

Pocket switched networks rely on the movement of a mobile device rather than any other medium to transfer messages. As such, they make it possible to send messages in areas where no infrastructure exists for message delivery. Pocket switched networks are only used for messages that are not time dependent since it is very difficult to guarantee the time period in which a message will reach its destination. When two mobile devices meet, the originating device will decide whether to pass on the message or not based on the likelihood of delivering it sooner.

Description

The project breaks down into several components, the base of which is to implement a simple pocket switched network. At its very basic level, this involves designing a suitable data format[2] for messages, one which will support any extensions which may be added and allow suitable storage of destination information and any other useful metadata. Once the data format for messages is agreed upon, the actual network has to be considered. For the initial implementation a basic routing algorithm is ample. This will all be implemented as a mobile phone application using the Windows Mobile SDK[3]. We wish the application to seemlessly connect to any surrounding phones and transfer messages as necessary.

Evaluation of this algorithm and its efficiency in a real life application will be tested using existing data which displays human traces (such as contact between humans) for about ninety people. Different strategies may be more efficient for performance (flooding

[1] See (Augustin Chaintrea, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, James Scott) Pocket Switched Networks: Real-world mobility and its consequences for opportunistic forwarding, February 2005

[2] XMPP, a standard protocol for online messaging systems, will be considered for the sake of adhering to standards and with a view perhaps to relatively easily extending the pocket switched network such that it could interoperate with traditional XMPP networks. http://xmpp.org/

[3] Windows Mobile will be used because Android in Version 1.0 of the SDK does not support Bluetooth. Windows Mobile offers access to both Bluetooth and Wi-Fi. J2ME has been considered but does not offer easy access to Bluetooth or Wi-Fi connectivity. Neither does any variant of the Symbian SDK.

of messages), whilst others might be less costly (a random walk). It all depends on the nature of situation; where it is unlikely that nodes encountered will pass on the message in a timely matter, it may be preferable to flood messages to increase the chance of it getting through. However, if nodes meet often then a random walk may be sufficient. In this project I will evaluate these strategies to balance cost and efficiency in various situations. Thus the second part of the project is to investigate and evaluate different approaches to message delivery. In order to do this, I will write a simulator which will allow me (with the use of adapter classes if necessary) to run the routing algorithm against the already existing data. The existing data will show contact information between test subjects - this can be used to simulate message routing and indicate how well the algorithm is working.

The final part and bulk of the project will consist of the addition of hashtags to messages. Hashtags are a Twitter feature (perhaps unofficial) that classify posts by topic, using a hash (#) followed by a keyword at the beginning of a message. These allow messages to be indexed by topic and recipients of messages to filter streams and read only the messages that interest them. We propose adding on top of this pocket switched network some mechanism to acquire the interests of the user and to use this when routing information. This would allow users to receive potentially useful information that may be passed to them by acquaintances who have similar interests. For example, people who are interested in the arrival of a particular bus outside the Computer Lab. could be notified by subscribing to this hashtag. Whenever the bus was near the Computer Lab., a message could be sent onto the network with the appropriate hashtag in order to notify people. This message would make its way towards all interested people[4].

There would be some method of correlating interests so that information that is not directly linked with one of the defined topics of interest but is instead linked with a loosely related interest could still be transmitted. The source for this list of interests shall be a site such as del.icio.us which offers a suitably substantial API. Some investigation will take place in order to determine how best to attain this connection of interests - it may be that fact that people who meet often in real life share interests, in which case possible interests can be taken from a friend of a friend (assuming both meet regularly). Alternatively this could be taken from del.icio.us - we will evaluate whether people who are friends on del.icio.us actually share similar interests. Whichever of these (real life or del.icio.us) shows a strong link may be used when recommending similar interests.

If XMPP is used, an extension could be the ability to route information over an existing XMPP network - this could apply to situations where it is preferable to send messages over a fixed network (perhaps sending messages between countries where it would be extremely rare for people to make contact on a regular basis).

It will also be useful to have some method of duplicate separation so that if the routing algorithm employs a multicast strategy (and sends many messages out to many other nodes) then a user is not bombarded with the same message.

[4]A lot depends on the efficiency of the routing algorithm and its ability to recognise the situation here. We wouldn't want this information reaching the people after the bus has departed.

Example Use Cases

1. Exchanging information about events based on mutual interests. This method of information passing could be used as a way of promoting events which may appeal to the interests of people met. For instance, messages may be sent out about seminars or talks on a particular topic which would be transferred amongst people with similar mutual interests (i.e., people who take an interest in physics would receive messages about physics seminars).

2. Passing status updates along. This could be used to broadcast messages about any area of interest. E.g., if someone who worked at the Computer Lab was to enter it as one of their interests, they would then receive any messages that alert them to new happenings concerning the Computer Lab. The benefit of passing messages this way is that it happens without requiring user intervention.

Success Criterion

There are two types of hashtags we can consider - those which are geographically centred (such as information about a particular bus stop or a building) and those that are interest based. The success of this project will rely on having routing algorithms that can tune between different hashtags as appropriate to ensure timely delivery. It may be the case that geographically centred hashtags are also heavily localised such that it makes sense to use the pocket-switched network to transfer these messages. For others, such as interests which may be geographically distributed, it may make sense to use non-distributed methods of delivery (Twitter itself).

The most essential item is to have a working application for mobile devices and be able to route messages. Further to this, a successful project will entail an application that offers advanced routing capability (somewhere in between a random walk and flooding) to take account of different mobility patterns, as well as advanced functionality to allow the exchange of messages between users based on the users' subscribed interests and the ability to recognise interests which are loosely connected to users. My algorithm dealing with message routing should ideally be better than a random walk algorithm in terms of time taken to reach its destination and it should cost less than flooding in terms of copies of data sent over the network.

Finally there will be some method of simulating any algorithms that are devised. Because it is not possible to actually test it out on many phones, this will be in the form of an application that can execute the same routing algorithm used on the mobile phone application using the existing human mobility trace data. This application will allow evaluation of the routing algorithm by outputting the route that messages have taken. The routing algorithm should be easily transferrable between the mobile application and simulator application.

Starting Point

- Windows Mobile development requires use of their own custom SDK with which I am not familiar (nor am I familiar with mobile device development). However, the

SDK is part of Visual Studio 2008 of which I have a summer's worth of experience writing in C# in.

- I am not familiar with the theory behind pocket switched networks so there will be some amount of reading necessary.

D.2 Resources

It is necessary to use a specialised SDK for mobile phone application development which will understandably not be available on university provided computing resources. As such, development of the project will proceed on my own desktop computer (AMD Opteron 165 Dual Core running Windows XP). Data back-up is faciliated by the presence of a RAID and in case of local filesystem corruption, back-up to a remotely located file server. In addition, code will be stored in an externally hosted SVN repository.

Testing will involve use of the emulator provided by the SDK. Eventually, if suitable trial mobile devices can be found then the application will be tested on a real world basis. If this is not possible then traces of human mobility will be used to emulate a real world situation and test success.

D.3 Plan of Work

Research into pocket-switched networks, familiarisation with SDK. Basic mobile application development. (2 weeks) *Milestone:* Basic application created. Pseudo-code algorithm for pocket-switching.

- 7th November 2008

Basic mobile application development and device communication implementation (2 weeks) *Milestone*: Working application and message transfer over wireless link.

- 28th November 2008

Design and implementation of message format with rudimentary routing algorithm (2 weeks) *Milestone*: Application works to send point to point messages, regardless of cost or efficiency of algorithm.

- 12th December 2008

Investigation into various routing methodologies (including creation of simulation program) and addition of hashtag interests (5 weeks - holidays will slow progress) *Milestone*: Ability to import interests from del.icio.us, send and route messages with certain hashtags. Simulation program working using test data and able to use routing algorithm from phone application.

- 16th January 2009

Investigating correlation of interests between users (2 week)

- 30th January 2009

Integration of interests into routing algorithms, algorithm for comparing interests between users (3 weeks) *Milestone:* Ability to 'suggest' similar interests when users meet and pass messages based on that.

- 20th February 2009

'Road-testing' of devices, efficiency of various algorithms to be tweaked (3 weeks) *Milestone:* Work on algorithm complete - meets success criteria. Limited real world mobility traces acquired.

- 13th March 2009

Write-up (7 weeks)

- 1st May 2009

Dissertation Submission: 15th May 2009