

Beacon - A Rapidly Deployable Cellphone Network

Paul Crane

a thesis submitted for the degree of
Master of Science
at the University of Otago, Dunedin,
New Zealand.

May 14, 2012

Abstract

In this thesis we examine the use of rapidly deployable cellphone networks for disaster relief efforts. Disasters are require a coordinated response from many different groups and clear communication to civilians affected by the disaster. This needs a good communication system that does not rely on existing infrastructure that is set up in a small amount of time. Our proposed system is targeted towards use by civilians without a connection to existing telecommunications systems. We have developed a method that allows users of the system to claim telephone numbers along with an automated reputation based moderation system. We undertook experiments and recorded results in the form of logs from consoles and files. These experiments show that the proposed system can be used to provide a decentralised telephony service. Proposed future work includes research into how best to provide information to civilians affected by disasters.

Acknowledgements

I would like to thank my supervisor, Dr. Mariusz Nowostawski, for his guidance and support throughout the duration of this thesis.

Many thanks go to my family, friends, and fellow students who have provided support and encouragement. Especially those who helped with the proof-reading, Alan, Heidi, Lisa, Matt, Rosi, Sally, and Tony.

Contents

1	Introduction	1
1.1	Problem Overview	1
1.2	Dimension of Interest	2
1.3	Thesis Outline	2
2	Disaster Response and Recovery	4
2.1	Introduction	4
2.2	Response to Disasters	5
2.2.1	Case Studies	7
2.2.2	Summary	10
2.3	Information Sharing	10
2.4	Conclusion	14
3	Rapidly Deployable Networks	15
3.1	Introduction	15
3.2	Review of the State of the Art	16
3.2.1	Existing Solutions	16
3.2.2	Wireless Mesh Networks	20
3.2.3	Number Discovery	25
3.3	Research Question	26
4	Beacon Infrastructure	27
4.1	Components	29
4.1.1	Hardware	29
4.1.2	Software	31
4.1.3	OpenBSC	34
4.2	Experimental Setup	35
4.2.1	GSM	35
4.2.2	Telephony and Routing	37
4.3	Call Routing	39
4.4	Evaluation	41
4.4.1	GSM Network connectivity	41
4.4.2	Number Lookups	46
4.5	Conclusion	49
4.5.1	Unique Contribution	49
4.5.2	Discussion	49

4.5.3	Future Work	49
5	User Management	51
5.1	Overview	51
5.2	User Verification in Literature	51
5.3	Research Question	53
6	Beacon User Management	55
6.1	Overview	56
6.2	Distributed Databases	57
6.3	Web-based Interface	59
6.4	Phone Numbers	60
6.4.1	Database Schema and Programmer Interface	60
6.4.2	Call Flow for Setup	62
6.4.3	Setup Evaluation	62
6.4.4	Number Discovery Call Flow	64
6.4.5	Number Discovery Evaluation	65
6.5	User Verification	67
6.5.1	Database Schema and Programmer Interface	68
6.5.2	Evaluation	69
6.6	Discussion	72
6.6.1	Abuse	72
6.6.2	Unique Contribution	73
6.6.3	Future Work	73
7	Conclusions and Future Work	76
7.1	Conclusion	76
7.2	Unique Contribution	78
7.3	Future Work	79
7.3.1	Overview	79
7.3.2	Rapidly Deployable Network	79
7.3.3	User Management	81
	References	83
A	Setup	91
A.1	MySQL	91
A.1.1	Installation	91
A.1.2	Configuration	91
A.2	Asterisk	92
A.2.1	Installation	92
A.2.2	Configuration	93
A.3	Boost	93
A.4	GNU Radio	93
A.5	OpenBTS	94
A.5.1	libosip	94

A.5.2	Installation	94
A.6	Configuration	95
A.7	Provisioning	95
B	Configuration	97
B.1	Asterisk	97
B.1.1	extensions.conf	97
B.1.2	sip.conf	100
B.1.3	dundi.conf	100
B.1.4	res_mysql.conf	101
B.1.5	extconfig.conf	101
B.2	OpenBTS	102
B.3	Django	105
B.3.1	API — api/handlers.py	105
B.3.2	Cassandra Interface — api/score.py	107
B.3.3	Models — asterisk/models.py	111
B.4	AGI Scripts	114
B.4.1	JSON Parser — parse_json.py	114
B.4.2	Rating — rate.py	115
B.5	MySQL	116

List of Tables

2.1	The different agencies involved in disaster operations in Otago	6
3.1	Acronyms used in GSM network diagrams (Mehrotra, 1997)	18
4.1	Features not supported in the public version of OpenBTS	33
4.2	A brief example of the data stored in the database	37
5.1	Three ways to prove someone is who they claim to be	51
6.1	The relationship between Column Names, and Values in Cassandra. . .	58
6.2	The relationship between Row Keys, Column Names, and Values in Cassandra.	58
7.1	Key differences between our work and the Serval Project	78
7.2	Enhancements of our work over the Serval Project	79
7.3	Proposed additional features.	80

List of Figures

2.1	Ushahidi Disaster Information System	12
3.1	GSM Architecture Design	17
3.2	A COLT	19
3.3	Different mesh network architectures	21
3.4	The Mesh Potato	23
4.1	Network Architecture Design	27
4.2	The network topology in Kretschmer, Hasse, Niephaus, Horstmann, and Jonas (2011), the Global Standard for Mobile Communications, or Groupe Spécial Mobile (GSM) access node with remote telephony service.	28
4.3	Beacon Logical Diagram	29
4.4	USRP	30
4.5	The difference between the OpenBTS and OpenBSC projects.	34
4.6	Experimental Architecture Design	35
4.7	Uplink	36
4.8	Screen capture showing the OpenBTS network	37
4.9	The mesh network architecture	40
4.10	Mobile originated call flow diagram	44
4.11	Call Routing Experimental Architecture Design.	46
5.1	GSM Architecture Design	52
6.1	Information stored in the network.	56
6.2	High-level overview of the different components of the user verification and mapping system.	57
6.3	Reputation Architecture.	61
6.4	Reputation and Call Flow Experiment setup.	62
6.5	Reputation and Call Flow Experiment setup.	69

List of Abbreviations

(A)GPL	GNU Affero General Public License
AGI	Asterisk Gateway Interface
API	Application Programmer Interface
ARFCN	Absolute Radio Frequency Channel Number
ATM	Asynchronous Transfer Mode
AuC	Authentication Center
BATMAN	Better Approach to Mobile Ad-hoc Networking
BSC	Base Station Controller
BTS	Base Transceiver Station
CAP	Common Alerting Protocol
CIMS	Coordinated Incident Management System
COLT	Cell on a Light Truck
COW	Cell on Wheels
DISA	Direct Inward System Access
DNS	Domain Name System
DUNDi	Distributed Universal Number Discovery
EDGE	Enhanced Data Rates for GSM Evolution
ENUM	E.164 Number Mapping
FM	Frequency Modulation
GIS	Geographic Information System
GPL	GNU General Public License
GPRS	General Packet Radio Service

GPS Global Positioning System

GSM Global Standard for Mobile Communications, or Groupe Spécial Mobile

HLR Home Location Register

IMS IP Multimedia Subsystem

IMSI International Mobile Subscriber Identity

IP Internet Protocol

JSON Javascript Object Notation

LAC Location Area Code

LAN Local Area Network

LMDS Local Multipoint Distributed Service

MANET Mobile Ad-hoc Network

MAP Mobile Application Part

MCC Mobile Country Code

MCC Mobile Country Code

ME Mobile Equipment

MMS Multimedia Message Service

MNC Mobile Network Code

MOC Mobile Originated Call

MSC Mobile Switching Center

MSISDN Mobile Station Integrated Services Digital Network

OCDEMG Otago Civil Defence Emergency Management Group

PDA Personal Digital Assistant

PSTN Public Switched Telephone Network

RDN Rapidly Deployable Network

RSS Really Simple Syndication

SIM Subscriber Identity Module

SIP Session Initiation Protocol

SMS Short Message Service

SON Self-organising Network
SS7 Signalling System No. 7
SVN Subversion
TMSI Temporary Mobile Subscriber Identity
UHF Ultra High Frequency
USRP Universal Software Radio Peripheral
VHF Very High Frequency
VLR Visitor Location Register
VoIP Voice over Internet Protocol
XML Extensible Markup Language

Chapter 1

Introduction

1.1 Problem Overview

In 2011, damage from weather-related disasters in the United States exceeded US\$50 billion (Angle, 2011). World-wide the major disasters include the earthquake and tsunami in Japan, causing 15,842 deaths and 125,000 buildings damaged or destroyed (Agency, 2011); flooding in Thailand, an estimated 12.8 million people are affected with the cost of damages approaching US\$45 billion (Tang, 2011). These are examples of major disasters that affect a large number of people. The Christchurch earthquakes, which killed 181 people (Police, 2011) and the rebuilding cost is estimated at NZ\$15 billion (Rotherham, 2011), are small in comparison they nevertheless have a large affect.

Disasters are difficult to define, some have used physical effects whilst others have used the effects upon society (Quarantelli, 1985). For the purposes of this thesis we follow the definition given by Norris, Galea, Friedman, and Watson where he stated: A disaster is a potentially traumatic event that is collectively experienced, has an acute onset, and is time delimited; disasters may be attributed to natural, technological, or human causes (Norris *et al.*, 2006).

Regardless of how many people are affected by the disaster, the resilience shown is linked to the strength of their sense of community. Of vital importance in the immediate and ongoing recovery process is good communication (Norris, Stevens, Pfefferbaum, Wyche, and Pfefferbaum, 2008). Hence, in this thesis, we focus almost exclusively on the communication aspects related to natural disasters.

1.2 Dimension of Interest

We are interested in the telecommunications systems and how they are affected by disasters and how these systems can be used to help civilians in a disaster.

More specifically, we wish to maintain a basic level of service (voice communications) in the aftermath of a disaster without having to rely on existing network infrastructure while still using existing consumer devices.

The key design element is centered around a mesh network with distributed telephony services operating on top. We focus on routing between different nodes in the network and providing support for civilians to use the network like they use existing telephone networks. Users can access this network using consumer cell-phones.

We also propose that this network be used to augment existing public broadcast systems by facilitating public message broadcast messages. A system such as this has the capability to provide both a data collection and dissemination point.

Existing research on mesh networking and disaster information systems (as well as the security of the information stored in such systems) means we advance this research by prototyping the telephony aspect using fixed network hardware. In addition, we also investigate the complete separation from the existing network infrastructure.

1.3 Thesis Outline

The development of a solution to the problem outlined above requires several components:

- a network infrastructure
- call routing between users
- user account information management

Disaster response and relief is discussed in Chapter 2. We conclude that there is work to be done to improve the communication situation amongst civilians caught in a disaster scenario. In Chapter 3 we discuss the networking infrastructure and routing of calls. We develop a decentralised system that allows users of mobile phones to make and receive calls without relying on the existing infrastructure. The design and experimentation of this is described in Chapter 4. We then describe how users are verified in existing networking scenarios in Chapter 5. The design and experimentation

of our implementation is described in Chapter 6. Finally, we provide conclusions in Chapter 7.

Chapter 2

Disaster Response and Recovery

Information in a crisis is a patchwork of sources. You can only hope to build up a full picture by having as many sources as possible.

Ory Okolloh, developer of Ushahidi (Okolloh, 2009)

2.1 Introduction

Disasters cover a wide range of events — these include biological, for example an epidemic; geological, for example an earthquake, tsunami, landslide, or volcanic eruption; meteorological, for example a drought, flood, or wild fire; or technological, for example industrial, hazardous chemical accident or transportation accident (Eshghi and Larson, 2008). Some of these disasters affect the civilian population more than others. According to Eshghi and Larson (2008), from 1900-2005 earthquakes have killed 107 million people, whereas industrial and hazardous chemicals accidents have accounted for 7.5 million deaths.

Sudden and catastrophic natural disasters cause the most havoc. Because of the scale of such potential problems we shall make these the focus of this study, rather than those disasters, like famines, which unfold more slowly. During and after a disaster telecommunications systems, whether they be voice, video, or text based play a vital role in coordination and cooperation of the recovery process. As the communications infrastructure becomes damaged or congested then it is more and more difficult to provide effective relief. Without the ability to provide effective and timely relief civilian populations suffer.

This Chapter looks at the response to disasters with examples of operating procedure and the practical results of telecommunications infrastructure. In Section 2.2 we

examine what procedures are in place for different disasters and see what effect these disasters have on the telecommunications infrastructure. Section 2.3 describes the need for effective communication strategies, including alerts and warnings to directions and even basic first aid instruction, in a disaster with particular emphasis on the general population.

2.2 Response to Disasters

Responding to disasters is a complex and dangerous task. Not only do the responders have to maintain an awareness of their environment and personal safety, but they have to inter-operate with a number of different agencies with different emphasis.

New Zealand uses the Coordinated Incident Management System (CIMS) methodology for planning emergency response (Flanagan, 2011). The following discussion uses Dunedin's plan (OCDEMG, 2004) as an example of a publicly available planning strategy. Disaster response involves groups and organisations with different specialties. Table 2.1 (OCDEMG, 2004) outlines the groups along with their tasks and their primary communications method.

Group	Task	Primary Communication
Welfare*	provides shelter, food, clothing	PSTN
Media Liaison*	provide public with information	PSTN
Transportation*	movement of people and materials for emergency purposes	PSTN
Supply*	sources and distributes supplies for civil defence operations	PSTN
Insurance*	provide advice and coordinating of resources	PSTN
Control	establishes priorities for emergency actions and ensuring a timely response	PSTN / VHF Radio
Public Health	identify, mitigate or isolate health hazards	PSTN & pagers and/or VHF Radio
Rescue	saving lives, extracting trapped people	VHF Radio
Medical	provides treatment to people	Ambulance (VHF) Radio
Fire Fighting Services	provide for effective coordination and use of fire fighting services	VHF/UHF Radio
Law and Order	maintain law and order	VHF/UHF Radio
Engineering	salvage resources, reinstatement of services, support rescue efforts	Radio ^a
Rural Support	represent and support the special interests of the rural sector	not specified
Communications	maintain the communications between the different groups	any and all networks available

^a or what ever systems the contractors use already

Table 2.1: The different agencies involved in disaster operations in Otago

Most of the non-emergency organisations (indicated with an asterisk) are using the existing telephone network. The Media Liaison group is responsible for keeping both the public in the affected areas and the news media up-to-date with information about

the disaster.

The Otago Civil Defence Emergency Management Group (OCDEMG) plan covers details about how best to make information available, including updating websites, making effective use of call centres, and news media updates. Each group must communicate within themselves as well as with the Control group. All relevant parties that need to communicate have inter-operable radio systems (OCDEMG, 2004). This is not always the case as the case studies described below will demonstrate.

The list below summarises different agencies that depend on effective communication in order to coordinate rescue operations, depending on the requirements of the rescue. Some rescues require the use of trained personnel and specialised equipment (classified as specialist and structural in OCDEMG (2004)), whereas others have no such requirements and can proceed with basic manpower and basic equipment (classified as light in OCDEMG (2004)).

- New Zealand Fire Service
- Red Cross Emergency Response Unit
- Search and Rescue
- The New Zealand Army
- The Royal New Zealand Navy
- Search Dogs Otago
- Rural Fire crews
- Helicopter Operators
- NZ Mines Rescue Team

2.2.1 Case Studies

We have been looking at the local plans for a disaster. We use (OCDEMG, 2004) as an example of a publicly available plan. We examine three case studies that illustrate the reality of a disaster situation with respect to telecommunications infrastructure.

Hurricane Georges, 1998

Hurricane Georges struck countries in the Caribbean and the Gulf Coast of the United States over two days in September 1998. Wind gusts reached 165 mph, with a maximum sustained speed of 102 mph. The storm caused approximately \$3.25 billion in property damage, left approximately 133,000 people homeless, damaging or destroying 162,000 homes (Sattler, Preston, Kaiser, Olivera, Valdez, and Schlueter, 2002).

One of the problems identified by McEntire (1999) was the level of coordination between different agencies. The author describes a situation in which the duplication of effort by agencies (public, private, and governmental) was caused by repeating assessments of the disaster. Another situation that they described is that the Civil Defence placed shelters and did not tell the Red Cross. After the disaster there were plenty of instances of co-ordination. Organisations in the Dominican Republic worked closely to meet the needs of the victims. Local social groups, and other humanitarian organisations were in contact with the Dominican Republic's emergency managers (McEntire, 1999). However, there were still issues. For example, the government's radio station continued with the scheduled programming even though the storm was raging outside. There were also reports that authorities weakened the weather reports so as not to alarm the population (McEntire, 1999).

Although a technological solution to infrastructure problems could not possibly hope to prevent bad decisions, it may mitigate the consequences by providing a platform by which effective communication is maintained.

Kobe Earthquake, January 1995

In January 1995, a 7.2 magnitude earthquake struck the port town of Kobe in Japan. The earthquake caused a horizontal movement of 1.5-2m, a vertical thrust of 1.2m and a twisting motion. In the city 100,000 buildings were destroyed and a further 233,000 damaged. The infrastructures of water, sewage, gas, power, rail, and highway systems were all severely damaged. After the earthquake fires burned uncontrollably as the water supply was insufficient to put them out. 300,000 people were made homeless by this disaster (Horwich, 2000).

Noam and Sato (1995) describe the problems with a centralised approach to information management. The main problem lay with the politicians who were slow to realise the extent of the damage and, due to the bureaucracy the information had to pass through, they were slow to react. It transpired that civilians from neighbouring

prefectures, whose infrastructure remained in tact, visited the severely damaged areas and posted message on internet forums.

In the aftermath it was revealed that the infrastructure was not damaged, but overloading of the systems rendered them useless. In general, telecommunications networks are designed for 10% of the population to talk simultaneously (Thompson, 2005). However, when a disaster strikes many people rush to the phones and often renders any communication impossible due to system overloading.

Hurricane Katrina, 2005

In 2005 Hurricane Katrina struck four states along the Gulf Coast. In the days leading up to the Hurricane's landfall, an estimated 1.2 million people were evacuated. There were between 100,000 and 120,000 residents that had remained in the city that subsequently need relocation after the hurricane made landfall (Nigg, Barnshaw, and Torres, 2006). The hurricane destroyed about 350,000 houses and killed 1,300 people (Rodríguez and Aguirre, 2006). Breaches in the levees in New Orleans made the flooding worse (Rodríguez and Aguirre, 2006).

Thompson (2005) describes the effects of Hurricane Katrina on the telecommunications networks. When he tried to call a friend he was met with a busy signal and a dead phone line. The only people able to communicate were the governmental officials and those who had satellite phones with charged batteries.

World Trade Centre Terrorist Attacks, September 2001

The terrorist attacks on the World Trade Centre in September 2001 claimed almost 3,000 lives and 30 million square feet of office space in Lower Manhattan. The estimated cost to repair the damaged infrastructure is expected to reach \$21.6 billion (Bram, Orr, and Rapaport, 2002).

The infrastructure damage included important switches used for routing local phone calls. These switches were close to the World Trade Center. The resultant damage left parts of New York without communications capacity (Thompson, 2005).

Civilians are just one group that can benefit from a reliable communication system, the responders need a robust system too. Manoj and Baker (2007) discusses the need for a common underlying network technology. This is due to fire, police, and other responders using different equipment that cannot inter-operate.

2.2.2 Summary

These case studies demonstrate information sharing and collaboration between different agencies, including civilians, are key problems in disaster relief and response and that it is not only a technological issue. Disasters often render existing communications infrastructure useless by congestion or damage. Each of the case studies above requires many large groups to interact, coordinate, and cooperate. These tasks are hindered by an ineffectual communications systems.

Gathering reliable information about the extent of disasters is also a key problem in management of disasters. Civilians can be used to report the extent of damage and what types of assistance are needed in which locations. Any disaster management system should be able to benefit from all sources of information and not just solely on a single official word. We discuss this aspect further below.

2.3 Information Sharing

It is vital to have good sources of information in a disaster. This information can range from physical dangers (falling buildings, aftershocks, tsunamis, etc.), to societal dangers (looting, rioting) as well as medical emergencies (disease outbreaks).

The disaster plans described in Section 2.2 specify how the different groups, agencies, and organisations communicate between themselves. The plan (OCDEMG, 2004) aims to provide the public with information about the environment and to provide training and education pre-disaster.

Civilians rely on mass media to get information about crises and often receive information from friends in an ad-hoc manner and the messages can change along the way like a game of Chinese Whispers (Addams-Moring, Kekkonen, and Zhao, 2005).

The official sources of information appear to be more concerned with alerting and warning during a disaster. There is a lack of information about basic first aid during a disaster. Having resources to search for necessary information can help save lives. For example an American film-maker, Dan Wooley, was buried under rubble in a hotel in Haiti after the earthquake in January 2010. He used his iPhone to find information about how to treat his injuries (Chen, 2010). This type of resource assumes that people have access to a smart phone and that they've downloaded the application before the disaster (or after if the communications channels are available). Worldwide smart phones represent 13% of the cell-phones in use (Cisco, 2011).

It is not only civilians that benefit from the just-in-time learning aspects, but the

authorities can also benefit from observations gathered from civilians. Civilians can report any number of events from looting to requests for medical aid.

Incident management systems are key to a timely response in emergency situations. These systems are concerned with getting the right information to the right people in the right format at the right place at the right time (Iannella and Henricksen, 2007).

Iannella and Henricksen (2007) has broken the functionality of these systems into five key top-level functional services, and focus on the resource and notification management aspects.

- Incident Management
- People Management
- Situational Awareness Management
- Resource Management
- Notification Management

An example of an incident management system is Ushahidi. It is a free and open-source platform that allows civilians to make anonymous reports by Short Message Service (SMS) messages of events which are then processed on to live maps (Okolloh, 2009; Gao, Barbier, and Goolsby, 2011). Initially Ushahidi was developed to combat post-election violence in Kenya in December 2007–January 2008. There was a government ban on live media as well as self-censorship within the mainstream media. The government argued that false or biased reporting would incite violence and wanted to be able to review the reports before they went live (Okolloh, 2009).

This system has also been setup in numerous locations, of note is its use in the Haitian Earthquake in January 2010 (Bilham, 2010). The Ushahidi system was setup within two hours after news of the disaster (Gao *et al.*, 2011). After two weeks of use there were more than 2,500 incident reports in the system. It was also set up after the Christchurch Earthquake in February, see Figure 2.1, aggregating Twitter, SMS, and email messages.

We note that there is a question of how much trust to place in the crowd sourced information. This information comes from unauthenticated sources and through corroboration with other reports about the same events, the information becomes more trusted. The authors of (Goodchild and Glennon, 2010) mention that the benefits of crowd-sourced information can outweigh the risks. One only needs to look at Wikipedia

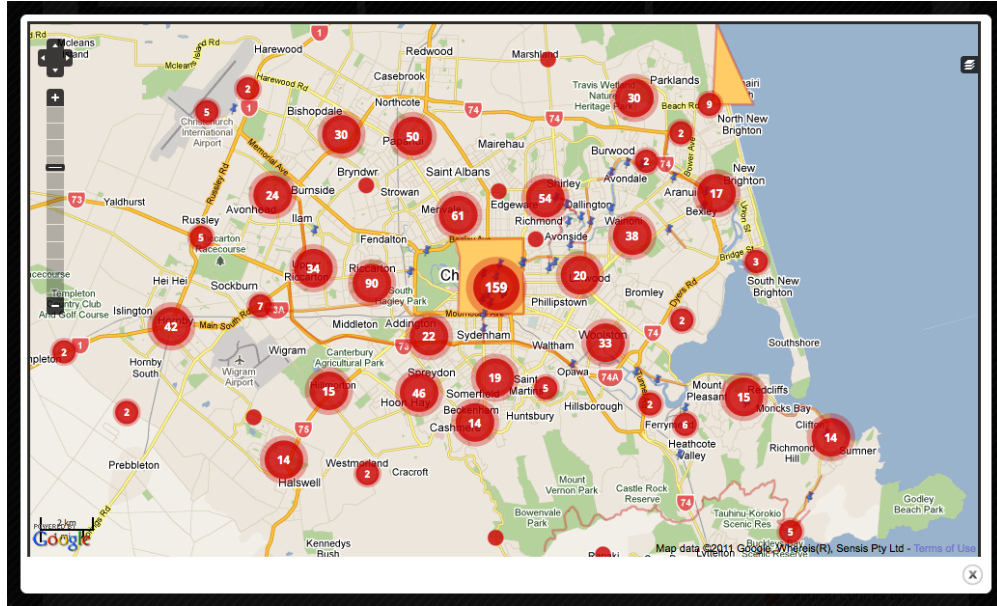


Figure 2.1: The Ushahidi Disaster Information Management system, showing the deployment for the Christchurch Earthquake, 28 February 2011. Source: <http://community.ushahidi.com/index.php/deployments/deployment/christchurch-recovery-map>

as an example of a system that self-governs, corroborates and collaborates between a disparate set of users world wide (Bryant, Forte, and Bruckman, 2005). Wikipedia is an online encyclopedia that relies on consensus to provide the content of articles instead of the traditional model which uses experts. There are different levels of access given to volunteers who have a good reputation within the community.

It should be possible to develop a level of trust in people over time. The more that a person reports correct events, the more trusted those reports should become. The authors of (Okolloh, 2009) describe their approach to verifying field reports. They provide a set of volunteers to collate and collect the reports from different sources, including the official sources. If they cannot verify the report, then it is marked as such and placed in the system. Once these reports are verified there seems to be little reason not to use them to help coordinate relief efforts.

Previous events have proven that official information from governments can lag behind with real on the ground sources of information, this occurred in Japan with the Kobe earthquake in 1995 (Noam and Sato, 1995). The technology is there for use but the governments have to make the move to use them. Some are uncertain as to how best to utilise these technologies. The Dubai government has a communications system

that utilises mobile phones and text messaging. The Dubai Civil Defence appears as one of the providers of a push-to-talk service. This is where users can press a button on their devices and transmit voice to any nearby devices without the need to be in a call, in effect the devices become walkie-talkies. This push-to-talk service is only for registered users and not for anyone with a mobile phone (Addams-Moring *et al.*, 2005).

Contrary to this example, after the South-East Asia earthquake and subsequent tsunami in 2004, the telecommunications network operators used group messaging and location services in order to locate and alert their customers about evacuation flights for tourists and other relevant information (Addams-Moring *et al.*, 2005). The authors make the point the mobile position feature of the GSM network allowed rescuers to find 71 people.

If the mobile phone network were to be used to alert civilians then the message contents needs careful consideration; for example, from Addams-Moring *et al.* (2005)

You are in an emergency area.

Stay calm, help yourself and those around you.

Use only text messages.

Do not make voice calls.

Help is coming in two hours.

The Common Alerting Protocol (CAP) is a simple but general format for exchanging all-hazard emergency alerts and public warnings over different networks. CAP allows a consistent dissemination of warning messages simultaneously over different warning systems. This increases warning effectiveness, by reaching more people, while simplifying the warning task, by decreasing the administrative overhead. CAP also facilitates the detection of emerging patterns in local warnings such as might indicate an undetected hazard or hostile act. Additionally CAP provides a template for effective warning messages based on best practices identified in academic research and real-world experience (OASIS Emergency Management Technical Committee, 2010). Because CAP is an extension of Extensible Markup Language (XML) it is possible to inter-operate with other technologies like Really Simple Syndication (RSS) feeds (Raivio and Addams-Moring, 2006).

McGinley, Turk, and Bennett (2006) describe a design for a system that utilises CAP to support multiple recipients, channels, hazards, stakeholders, senders, platforms, and write once message composition.

This work assumes that the infrastructure is fully functioning. This may not be the case. They have built a prototype system that incorporates voice, fax, email, SMS,

website messaging channels; as well as list-base delivery, location-base delivery, as well as standards-based inter-operability.

Suggestions for improvement include sending images (such as fire and cyclone maps) to email recipients and Multimedia Message Service (MMS) capable phones; as well as automated polling via telephone keys. Areas for further research include the design of message composition interfaces and inter-agency collaboration strategies for messages. They also suggest a useful feature of such a system would be to include the ability to send messages based on the location of the recipient. The authors of (Valtonen, Addams-Moring, Virtanen, Järvinen, and Moring, 2004; Fernandes, 2008) have proposed using SMS messages with the location feature of GSM base stations (the latter have proposed integrating the CAP messages with the SMS and location features).

2.4 Conclusion

In this chapter we have reviewed current approaches to disaster response and recovery as it relates to telecommunications infrastructure. There is a large body of research about sharing information in a disaster scenario but it is apparent that there are assumptions about the availability of the networking and/or communication infrastructure.

We acknowledge that providing a technological solution to the infrastructure problem will not solve any of the political issues that surround how the gathered information is used and disseminated.

Regardless of the political issues there is an under utilised source of information that could be of use to disaster relief organisations. Civilians, who have access to mobile phones, are generally capable of providing information to the disaster managers, this is a two-way street. At the same time that the civilians provide the information they can also be consuming information. This is where using a standards based approach to phrasing and updating alert messages is necessary.

Chapter 3

Rapidly Deployable Networks

[The public] phone system is so robust, our mobile phones are so ubiquitous and voice mail and e-mail are such reliable backups that... [it] seem[s] a right and not a privilege... Except when disaster hits.

Clive Thompson (Thompson, 2005)

3.1 Introduction

Chapter 2 provided a general discussion of why the telecommunications infrastructure is vital in a disaster. In this chapter we shift our focus specifically to rapidly deployable networks. A rapidly deployable network is a network that is designed to be set up quickly to ease congestion or to replace damaged portions of existing infrastructure.

One way to combat the problems of a damaged or congested communications system service is to over-provision before a disaster, that is provide more available capacity than is used. The problem with this approach is that it is expensive — the telecommunications companies are, in effect, wasting the excess capacity. Thus it is not economically viable for telecommunications providers to over-provision. It would only be possible through governmental regulation. Even if a service is over-provisioned, that would only affect the congestion problem. The problem of potential damage to the infrastructure remains.

Rapidly deployable networks are designed to be setup in a short period of time to compensate for increased load or to replace portions of the communications system where there is inadequate coverage (Perkins, 2001) whether due to congestion or damage.

This study differs from existing solutions due to the underlying assumptions made about the situation. Commonly, other work assumes that people have access to smart phones and/or have already downloaded a special application on to their phones. Alternatively, previous work focuses on the management and recovery teams perspective. Our work is targeting the GSM network. This allows anyone with a GSM-compatible mobile phone to use the network. There is no requirement for special devices, nor download special applications to their phones, meaning that unprepared people are able to seek assistance when they need it.

This chapter reviews existing solutions and expands toward peer-to-peer mesh networking in Section 3.2. Section 3.3 addresses the question that we will be answering, namely how to provide access to a telecommunications network without the need for the user to have special hardware. Chapter 4 describes a possible solution, evaluates our solution, and discusses the results.

3.2 Review of the State of the Art

In a disaster the communications infrastructure is vital to the response efforts. The infrastructure can become useless due to damage or congestion. In this section we discuss Rapidly Deployable Networks (RDNs). Often these networks are setup in an ad-hoc manner as needed to cover the disaster areas (Perkins, 2001). As mentioned in Chapter 2, a working communications system is vital to coordinate disaster relief and response efforts.

Due to technical limitations, we will be emphasising voice communications, but we examine other forms for completeness. Given the modular nature of the implementation's design it will be straightforward to add in the additional functionality as the technical limitations are eliminated (this is covered in more detail in Chapter 4).

3.2.1 Existing Solutions

Femtocells are often classified as a rapidly deployable network. These devices are short range, low cost and low power base-stations installed by home users to boost indoor coverage of the commercial networks (Chandrasekhar, Andrews, and Gatherer, 2008). These devices are intended to provide a small home-sized GSM coverage area to boost performance for consumers in poor coverage areas. These technologies differ in their capabilities and their ability to provide support in unstructured, ad-hoc circumstances (Chandrasekhar *et al.*, 2008). Our work focuses on a specific and narrow meaning of the

term “rapidly deployable networks” with the emphasis on decentralised, ad-hoc, cheap and truly rapidly deployable solutions. There has been work on providing access to the IP Multimedia Subsystem (IMS) core network from femtocells whereby the signalling is converted to Session Initiation Protocol (SIP) (3rd Generation Partnership Project (3GPP), 2010b). We will focus on networks without the IMS component.

We focus on GSM as it is a popular protocol for mobile telephony. According to (World, 2010), there were more than three billion GSM connections in 2008. We will first cover the GSM architecture’s main components.

Figure 3.1 covers the main components of the GSM architecture. Table 3.1 lists the abbreviations. For more information about the GSM architecture see (Mehrotra, 1997).

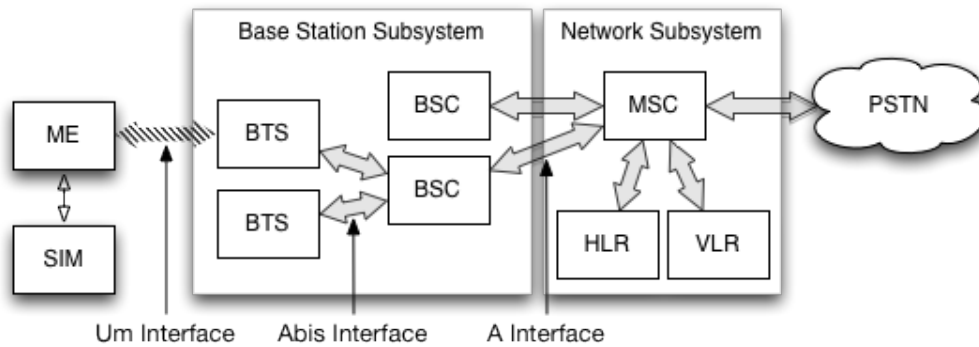


Figure 3.1: GSM Network Architecture Design, redrawn from (Mehrotra, 1997)

Commercial systems exist that provide rapidly deployable network services, an example of this system is a Mobile Base Transceiver Station (BTS), or a Cell on Wheels (COW) sometimes referred to as a Cell on a Light Truck (COLT) (see Figure 3.2). These devices are a BTS on a trailer and replace the last BTS unit in Figure 3.1 (Kwasinski, Weaver, Chapman, and Krein, 2009).

Acronym	Expanded	Purpose
SIM	Subscriber Identity Module	Identify the user
ME	Mobile Equipment	Device that accesses the network
BTS	Base Transceiver Station	Receives radio signals from ME
BSC	Base Station Controller	Controls the BTSs
MSC	Mobile Switching Center	Controls the BSC
HLR	Home Location Register	Database for provider's users
VLR	Visitor Location Register	Database for non-provider's users
PSTN	Public Switched Telephone Network	Traditional telephone network

Table 3.1: Acronyms used in GSM network diagrams (Mehrotra, 1997)

This image has been removed to avoid copyright issues.

Figure 3.2: A satellite COLT providing service for a Texas Department of Public Safety checkpoint on I-45. September 20, 2008. Source: http://www.corp.att.com/ndr/deployment_2008_09_galveston.html

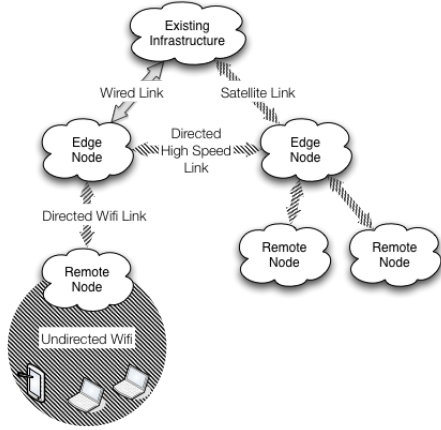
COWs, femtocells, and microcells suffer from their continued reliance on a fixed radio access network infrastructure and back-haul connectivity to the fixed infrastructure. Therefore, this approach cannot provide reliable communication capabilities (even locally around the deployment area) in certain emergency situations and disasters, where the fixed network infrastructure is destroyed or the back-haul connectivity is not available (Abusch-Magder, Bosch, Klein, Polakos, Samuel, and Viswanathan, 2007; Kwasinski *et al.*, 2009). Another problem faced by these COWs is that they are not practical if roads have been damaged. In the earthquake and subsequent tsunami that struck South-East Asia in 2004, it took approximately two months before road access was restored in some parts (Ghobarah, Saatcioglu, and Nistor, 2006).

There have been experiments in using a Self-organising Network (SON) to configure the latest telecommunications provider networks (Ericsson, 2009). These SONs are used to configure adjacent nodes in the network to improve coverage and service performance while reducing the amount of work that the operators need to perform.

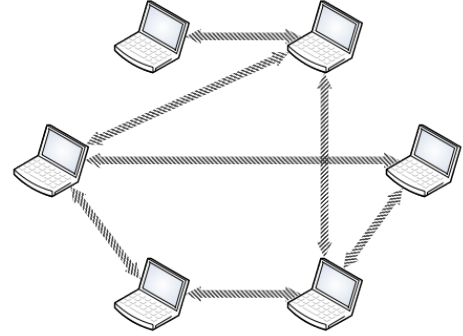
3.2.2 Wireless Mesh Networks

A common approach to the problem of reliance on the fixed network architecture is to form a wireless mesh network. A wireless mesh network is where nodes in the network are both clients and routers. Each node is capable of forwarding packets for other nodes that may not be in range of the destination node. Mesh networks form dynamically and maintain connectivity automatically (Akyildiz, Wang, and Wang, 2005), they are in effect a SON (the major difference being that there is an advertisement component to SONs) Razzaque, Dobson, and Nixon (2010). Note that the routers usually have minimal mobility, while the clients can be stationary or mobile.

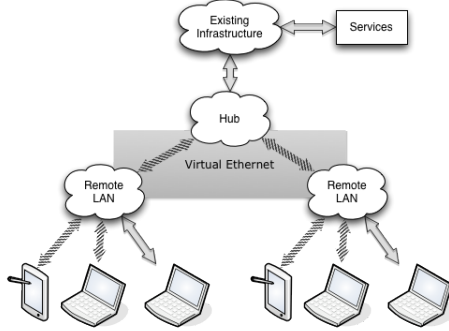
The development of mesh networks is dependant on the goals of the network. Different goals produce different network architectures. Usually, connectivity to these networks is via commodity wireless network interface cards. If a device does not have such hardware then there are other methods such as using Ethernet (Akyildiz *et al.*, 2005). However, this has not always been the case. The mesh network described in Sanchez *et al.* (1998) uses two wireless technologies to provide a point-to-point link as well as a local wireless network with the ability to route traffic over the existing point-to-point links. The solution presented in their paper uses the Asynchronous Transfer Mode (ATM) protocol with the ability to reconfigure the network as needed. Their results indicate that they can achieve a 1Mb/s connection to a node 10km away. Evans *et al.* (1999) describe the physical setup. This paper describes the radio controllers and



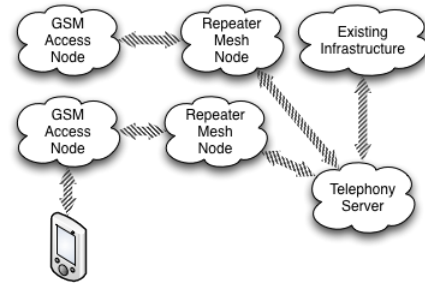
(a) Sanchez *et al.* (1998); Evans *et al.* (1999) uses point to point links to services WiFi access points.



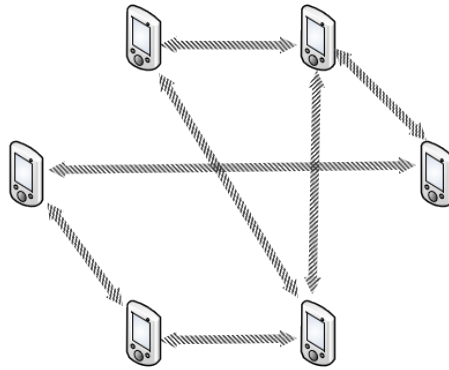
(b) Jang *et al.* (2009) using laptops to form a mesh network with applications on top.



(c) Midkiff and Bostian (2002) star-shaped topology with GIS on top.



(d) Kretschmer *et al.* (2011) GSM access node with remote telephony service.



(e) Sicard *et al.* (2010).

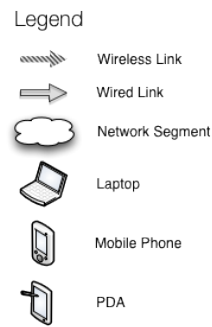


Figure 3.3: Network architectures as described in the literature.

the ability to direct the radio paths programmatically. Figure 3.3a shows this network topology.

Jang *et al.* (2009) use first responders' laptop and notebooks to provide a mesh network which is the basis for a rescue information system (Figure 3.3b). The main focus of their paper is this rescue information system. The system is designed to operate without a connection to the internet, as the disaster may have damaged key infrastructure.

Midkiff and Bostian (2002) choose a different approach. Instead of using a mesh network where each node can route for others, their architecture consists of a central point which then hooks into existing infrastructure (Figure 3.3c). This topology is described as a star shape. Midkiff and Bostian are also interested in a geographic information system that works on top of this network. The key difference between this and other systems which use the standard WiFi (802.11) is that they are using the Local Multipoint Distributed Service (LMDS) (between 26GHz and 29GHz (Tipparaaju, 2000)), whereas WiFi uses 2.4GHz or 5GHz, depending on the standard (Gast, 2005).

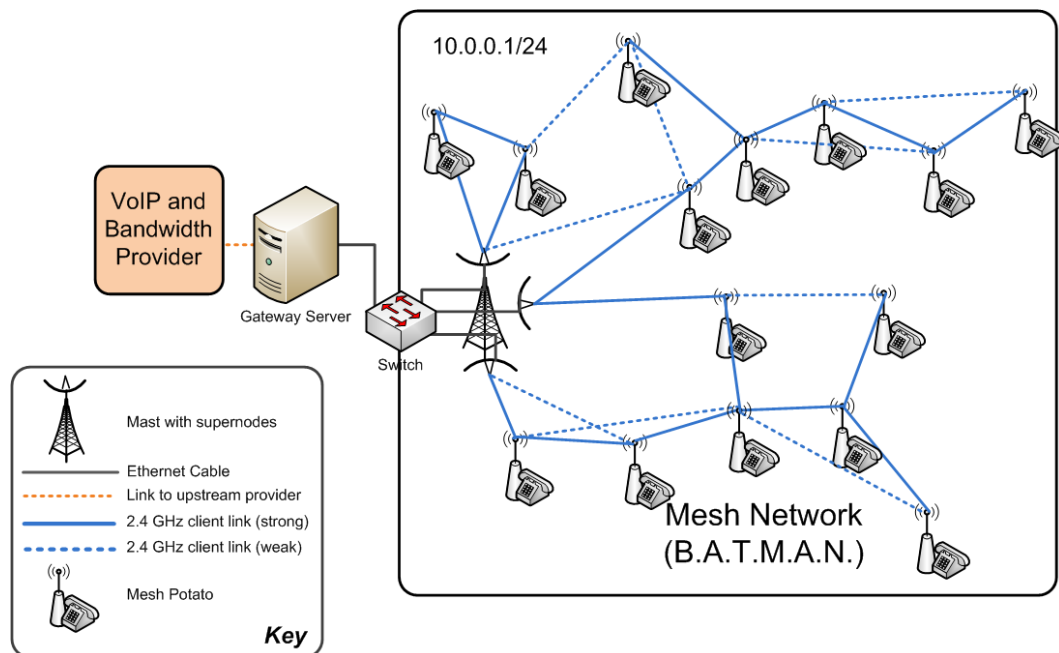
The CalMesh, as described in Braunstein, Trimble, Mishra, Manoj, Lenert, and Rao (2006) is a hybrid system that uses multiple different back-haul technologies (wired, wireless Local Area Network (LAN), cellular network, or satellite) to provide a disaster area with internet-accessible WiFi coverage. This network architecture is capable of delivering video and voice with approximately 50ms of jitter (the variation in delay between packets). Kretschmer *et al.* (2011) integrated a GSM nanocell into their wireless mesh network (itself an extension of the work in Banchs, Bayer, Chieng, de la Oliva, Gloss, Kretschme, Murphyk, Natkaniec, and Zdarsky (2008)). These papers show that it is possible to route Voice over Internet Protocol (VoIP) traffic across a mesh network without severe problems.

There are two projects that are in development to support telephony over a wireless mesh network. The first is The Village Telco project (Adeyeye and Gardner-Stephen, 2011). It bridges analogue telephony and a WiFi mesh network using a device called the mesh potato (see Figure 3.4). The mesh potato has a WiFi Ethernet card as well as an analogue telephony adaptor that allows a user to connect a normal telephone to a wireless mesh network. The second is the Serval Project (Gardner-Stephen, 2011) described by the authors as a derivative of the Village Telco. It uses slightly modified Android operating system (OS) to form an ad-hoc wireless network instead of a special device to do the bridging. Stock Android OS does not support ad-hoc WiFi and so the Android OS needs to be modified by users obtaining 'super-user' (or administrator/root control) of the phone in order to enable the ad-hoc networking.

Both projects use a routing protocol called Better Approach to Mobile Ad-hoc



(a) The Mesh Potato. Source: <http://villagetelco.org/mesh-potato/>



(b) The Mesh Potato network architecture. Source: http://wiki.villagetelco.org/index.php?title=Network_Design

Figure 3.4: The Mesh Potato

Networking (BATMAN), targeted for large meshes for static nodes. The basic idea is that each node periodically broadcasts ‘hello’ packets, which the network uses to

discover the next best node to send the traffic to. These ‘hello’ packets (officially called originator messages) consist of an originator address, the sending node address, and a unique sequence number. Upon receipt the receiving node changes the sending address and rebroadcasts the packet. Once a node receives its own packet (by checking the originator field) it then tests the reverse path (Johnson, Ntlatlapa, and Aichele, 2008).

In addition to using the BATMAN protocol to perform the routing of packets, both projects use the same mechanism for addressing the users in the system. Subsection 3.2.3 discusses this further.

We propose using Mobile Ad-hoc Network (MANET) technology to form an ad-hoc backbone for our network, the exact details are unimportant as long as there is enough capacity for the telephony component (and a little other data traffic). Jang *et al.* (2009); Braunstein *et al.* (2006) show that it is feasible to route application and telephony data over a MANET. Due to this existing work we focus on higher level number discovery and user management.

The previous work concerns stationary nodes. If we take mobility (in terms of physical movement) into account, we need different routing protocols. These networks also have different performance characteristics when compared to the above ad-hoc networks. Often there are more constraints placed on the devices in these types of network; for example, in a wireless sensor network, battery life is a concern (Li, 2008).

There has been a large amount of research on routing protocols and medium access in these MANETs (Toh, 2002; Li, 2008; Murthy and Manoj, 2004). These protocols all require the use of nodes with custom software installed. The authors of (Manoj and Baker, 2007) propose the use of mobile devices (such as smart phones or laptops with wireless networking capabilities) to form a wireless mesh network.

Kretschmer *et al.* (2011) use GSM as an access technology to interface to a remote telephony service. A disadvantage with this design is that the telephony service remains centralised. If the disaster were to damage either that service or any links to that service then it would render it useless.

Given that disasters are not predictable we question the practicality of these solutions. Since disasters catch civilians off-guard, people are not likely to have the software installed on their phones before the disaster occurs (especially if they need to perform some complex steps in order to install it) and will not be able to get the software afterwards if infrastructure is compromised.

People living in sparsely populated areas will have trouble using the MANETs

because there are limits to the propagation of radio signals. If the nodes are battery powered this becomes an additional limitation.

So far, the discussion has focused around civilians. However, rescuers could also benefit from having a common network to use in a disaster. During the terrorist attacks in New York on 11 September 2001, numerous agencies were involved (police, fire, medical, federal) and each had their own radio equipment that was not interoperable between agencies (Manoj and Baker, 2007). This type of situation can be avoided if there is a common system that everyone has access to.

Mesh networks are good until they need to interconnect with the rest of the world, particularly if there is no available infrastructure in the disaster area to connect to. If there is no infrastructure the MANETs form isolated islands where it is impossible to share information or services between these islands.

3.2.3 Number Discovery

The International Public Telecommunication Numbering Plan (ITU, 2010) describes the formatting of telephone numbers. These numbers are familiar to everyone. If one were to dial a phone number in New Zealand from an international location, the number dialled would be +64 21 123 4567. In this example the caller needs to know their country's international prefix (abbreviated to a +), then the number begins with the country code, followed by area code, and the user's number (in the telecommunications literature, the user is referred to as the subscriber, in this work we will say user to simplify things).

The E.164 Number Mapping (ENUM) group has proposed a mechanism to map telephone numbers on to the Domain Name System (DNS) (Van Meggelen, Madsen, and Smith, 2007). If we treat users as domain names, we want to find out which address (or phone number) points to a particular domain (or user). Normally, DNS provides a mapping from domain to Internet Protocol (IP) address, or forward DNS. The mapping from IP address to domain is called reverse DNS (Albitz and Liu, 2001).

If we have a host with the domain name of `www.example.com` that points to IP address `192.168.0.110`, this is called the forward look up, conversely the reverse query looks up `110.0.168.192.in-addr.arpa` and finds the name, in this case `www.example.com`. Note that we use the domain `in-addr` for the reverse look up. The look-up that we perform to find the termination point for a voice service is performed in a similar manner; the query looks like `7.6.5.4.3.2.1.2.1.6.4.e164.arpa`. (Cannon, 2001; Huston, 2002). This method requires a central authority to provide

the database of phone numbers and termination points (Cannon, 2001; Huston, 2002). Another method for resolving numbers is the Distributed Universal Number Discovery (DUNDi) protocol, a fully distributed peer-to-peer system for locating telephony services (Spencer, 2004b). Chapter 4 discusses this in more detail.

3.3 Research Question

We have seen that there exists a large body of research on routing protocols and have discussed the technology behind MANETs. There is also research in the area of real-time traffic and carrier-grade MANETs (Perkins and Hughes, 2002; Kretschmer *et al.*, 2011; Banchs *et al.*, 2008) (VoIP is an example of real time traffic that has been mentioned in Perkins and Hughes (2002)).

There has been some research into how to route calls to users in various ad-hoc networks. In all those projects however, there are two essential components required. The first is a sufficiently capable phone, usually requiring WiFi hardware. The second is that the phones are modified to support the ad-hoc routing protocols. These two limitations restrict the available audience for these services.

The question that we wish to answer is: is it possible to develop a network that provides the benefits of MANETs while offering access to more users without the need for special hardware from the client?

This allows any GSM mobile phone to access the network regardless of the additional capabilities of the handsets. Thus allowing more people access to any information about the disaster that is shared on this network.

In Chapter 4 we discuss the design and implementation of the solution to this question.

Chapter 4

Beacon Infrastructure

In Chapter 3 we discussed the state of the art with respect to rapidly deployable networks and stated the problem that we wish to solve.

We aim to develop a network that provides the benefits of MANETs while offering access to more users without the need for special hardware from the client. In particular we are concerned with voice communications and using GSM mobile phones to access the network. Our work assumes that there is a MANET capable of handling the telephony and data traffic. In this chapter we discuss our proposed solution.

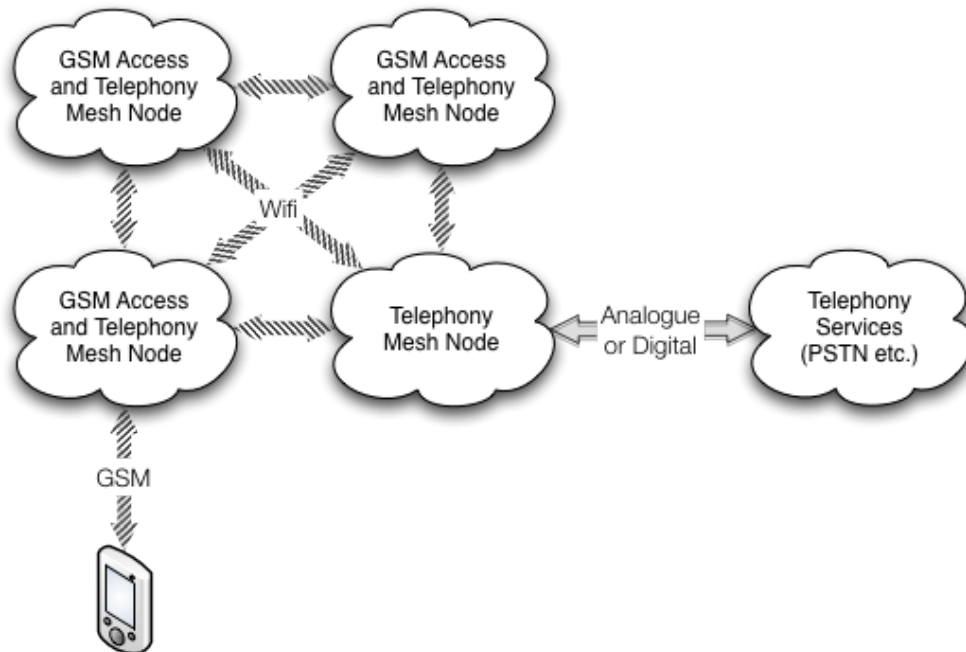


Figure 4.1: Our Network Architecture Design

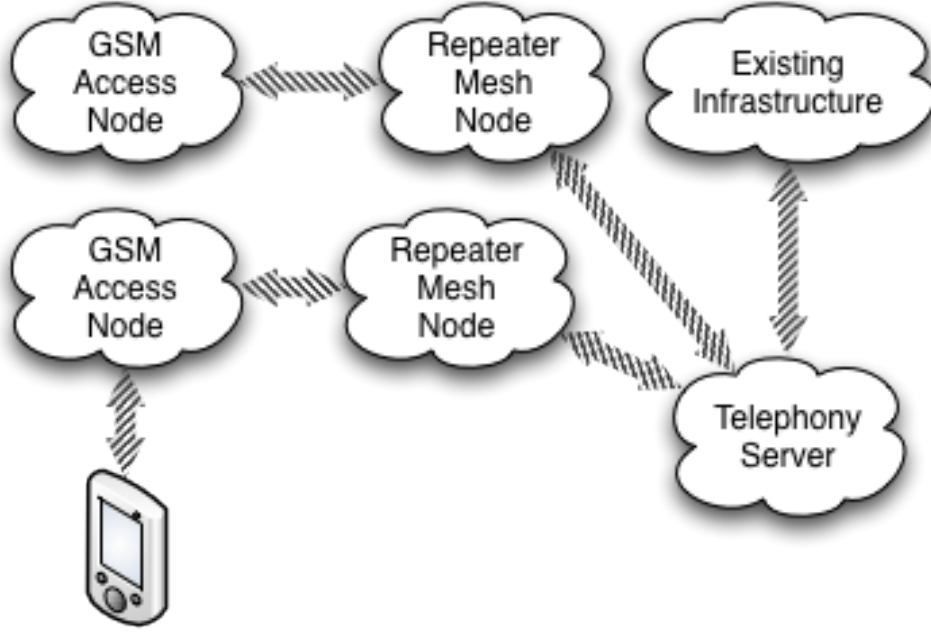


Figure 4.2: The network topology in Kretschmer *et al.* (2011), the GSM access node with remote telephony service.

The architecture that we propose is similar to Kretschmer *et al.* (2011), see Figure 4.1 and Figure 4.2, with the key difference being that we place the telephony routing and processing as close to the mobile phone as possible. This is in contrast with Kretschmer *et al.* (2011), where they use the mesh network as an access network for a remote telephony service. Keeping the telephony service close to the clients allows us to adjust the network to suit the needs of the situation. In our design, the mesh network is used to link the separate telephony services to provide coverage over a wide area.

Our nodes can function as either a standalone GSM access point to provide access to the Public Switched Telephone Network (PSTN), or as a peer-to-peer node in the mesh network, or as a repeater node (to extend the range of the mesh network. In this work we focus on the ability to act as a peer-to-peer node.

It should be noted that there is no modification needed to the client's devices, and that any GSM compatible phone can be used with this network.

In Figure 4.3 we present the design of an individual node. We introduce each of the different components in the following section.

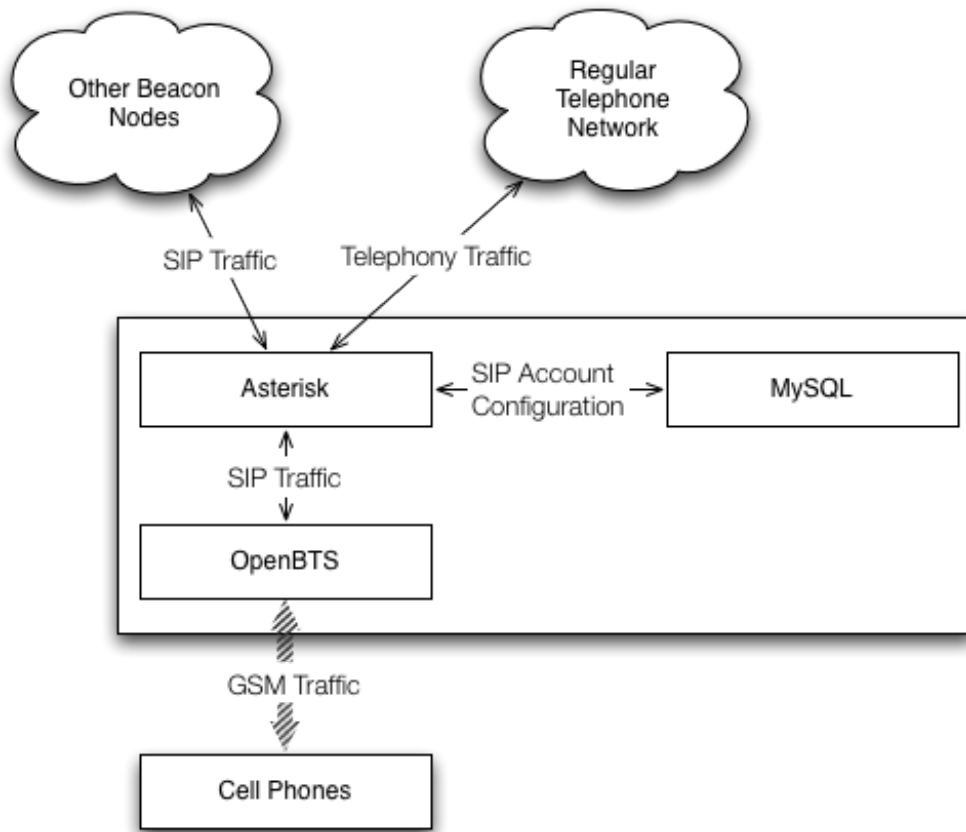


Figure 4.3: An individual Beacon node's logical diagram

4.1 Components

In this section we discuss the technical aspects of the hardware and software that we have used.

4.1.1 Hardware

Our solution uses a Universal Software Radio Peripheral (USRP) and off-the-shelf computer components. We describe the specifics of these components below.

Universal Software Radio Peripheral (USRP)

The USRP is a piece of hardware that allows the developer to program their own radio protocols for development. It allows for expansion into different frequencies by the use of daughter boards (Valerio, 2008), for example, as Frequency Modulation (FM)

radio transmitter and receiver, Global Positioning System (GPS) receiver, and a digital television decoder (Blossom, 2004).

This image has
been removed to
avoid copyright
issues.

Figure 4.4: The Universal Software Radio Peripheral (USRP). Source: <http://www.ettus.com/products>

For our setup, we use the USRP with two RFX1800 daughter boards. The daughter boards are capable of transmitting and receiving in the range of 1.5–2.1GHz which coincides with the DCS-1800 (uplink: 1710–1785MHz, downlink: 1805–1880MHz) and PCS-1900 (uplink: 1850–1910MHz, downlink: 1930–1990MHz) bands (3rd Generation Partnership Project (3GPP), 2010a). These are frequency bands that GSM uses.

Computer

We use standard off-the-shelf components for the computer. The computer used in these tests comprises an Intel i5 CPU, 2GB RAM, USB 2.0, 80GB Hard drive. The USRP requires a high-speed USB connection (at least USB 2.0). While Asterisk can run on old hardware, best audio quality is obtained with a modern CPU.

Mobile Phones

We used the following mobile phones during the testing of the network. These were a range of mobile phones available during testing. They range in age from 1 year to 7 years old, with a mix of open and closed hardware and software.

- Samsung Nexus S (running Android 2.3)
- Samsung Galaxy (running Android 1.6)
- Neo FreeRunner (running OpenMoko Om2008.12)
- Nokia N95
- Motorola V3x

In order to get the phones to attach to the network, we had to manually search for our test network. The phones were setup to connect to only 2G networks. Once the network had been found (and an appropriate entry had been made in the database—this is discussed below), the phones were able to find and attach to the network without manual intervention¹. This step could be skipped if the network were setup to use the same parameters as those operated by the existing telecommunications providers, and in a disaster this would be a necessary step.

4.1.2 Software

The test network is composed of the software listed below. Each component is discussed in turn below.

- GNURadio
- OpenBTS
- MySQL
- Asterisk

OpenBTS provides the Base Transceiver Station component of a traditional GSM network (see Chapter 3 for an overview of GSM Architecture). This software converts from GSM to SIP. OpenBTS uses GNURadio to interface with the hardware. Asterisk handles the call routing decisions, while MySQL provides dynamic SIP account handling. We provide a brief overview of each of these components below.

¹Some of the mobiles had more trouble with this than others.

GNURadio

GNURadio is a free and open-source toolkit that provides signal processing blocks for software-defined radio. Combining the signal-processing blocks into different graphs produces different radios, FM, GPS, and so on (Valerio, 2008).

OpenBTS

OpenBTS is a project that uses GNURadio to provide basic GSM base station functionality. There are numerous projects that use OpenBTS. The developer's blog (Burgess, 2009) has the details of experimental deployments at the Burning Man Festivals in 2008 and 2009 as well as a chronicle of a commercial deployment in Niue in 2010.

There are two distributions of the software: the commercial release and the public release. The commercial release has a mix of GNU General Public License (GPL) and non-GPL licences. It is intended to provide cellular services in industrial, governmental and commercial applications. The public release is a subset of the commercial release and is distributed under the GNU Affero General Public License ((A)GPL)v3 with copyright assigned to the Free Software Foundation, and is intended for experimentation, education, evaluation and proof-of-concept work (Burgess, 2011). OpenBTS is a GSM to SIP gateway. The 2.6 public release of OpenBTS supports mobile calls and SMS messaging. Both originated from the handset as well as from the network. The public version does not support the features outlined in Table 4.1 (Burgess and Samra, 2008).

We note here that in traditional GSM networks the link between the phone and the BTS is encrypted (to setup the encryption the Authentication Center (AuC) is needed). In the version of OpenBTS that we use there is no support for encryption. When a phone sends a location update request, OpenBTS skips the normal authentication request/response and cyphering mode steps and proceeds directly to accepting the location update. There are patches available (as described in (Kostrzewa, 2011)) to perform the encryption with the version of OpenBTS that we are using, however, the 2.8 commercial release has support for this encryption (and it is also being added to the public release) (Burgess, 2012).

A further restriction placed on us is the number of channels available. A channel is one direction of a call, a typical call comprises of two channels, one for the caller's audio and the other for the callee's. We are only able to use six channels due to the implementation details. This means that we are only able to make three mobile to

Feature	Description
Handovers	moving from one BTS unit to another
Half-rate Services	trade-off between voice quality and capacity
Multiple ARFCNs	allows for more capacity
Frequency Hopping	moves traffic to under used frequencies
GPRS or EDGE	mobile packet data services
Encryption	normally the link between the handset and BTS is encrypted

Table 4.1: Features not supported in the public version of OpenBTS

mobile calls.

Asterisk

Asterisk is an open-source telephony toolkit, used world-wide as a telephone exchange for hobbyists, carriers, and businesses of all sizes (Digium, 2010).

We are using version 1.8 of Asterisk. One of the technical limitations of Asterisk is that the main-line branches do not support SIP messages outside of a dialogue. A dialogue is a sequences of requests and responses that set up a session (voice, video, text) between two (or more) users. In our case we use it to set up a voice channel (Rosenberg, Schulzrinne, Camarillo, Johnston, Peterson, Sparks, Handley, and Schooler, 2002).

Fortunately there is a branch (developed by Russell Bryant, available at <https://code.asterisk.org/code/browse/asterisk/team/russell/messaging>) that can transmit the SIP messages outside of a dialogue. The problem becomes one of how OpenBTS handles the SMS messages. The SMS message implementation does not use any routing in the Asterisk layer. It assumes that the receiver is on the same machine as the source of the message.

MySQL

MySQL is a dual licensed open-source relational database. There are no specific requirements for the version of the software provided that it is compatible with Asterisk.

PostgreSQL (another open-source relational database) is an alternative database which is supported by Asterisk. We chose MySQL due to the level of support in the Asterisk community.

4.1.3 OpenBSC

OpenBSC (laforge, 2011) is another project that is producing an (A)GPL licensed software implementation of the GSM protocol stack. This project relies on using one of the specific devices listed below (the developers of OpenBSC are adding support for the items in italics).

- Siemens BS-11
- ip.access nanoBTS
- *HSL 2.75G Femtocell*
- *Ericsson RBS2308 and RBS2401*
- *Motorola Horizon macro BTS*

The key difference between the OpenBSC project and the OpenBTS project is the interface that they use to communicate with the Mobile Equipment (ME). OpenBTS implements the Um interface (or Air interface between the BTS and the ME) whereas the OpenBSC project implements the Abis interface (one step further away from the ME than OpenBTS). Figure 4.5 shows the difference between the two projects.

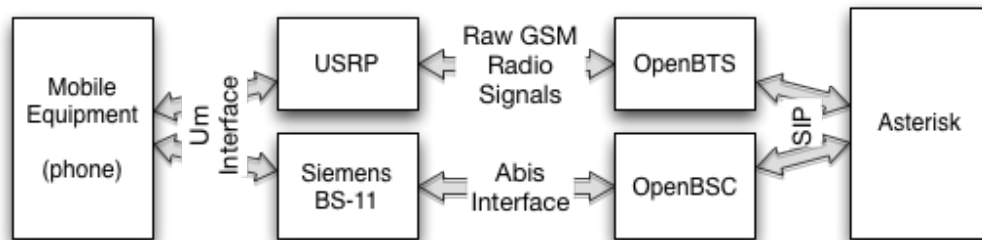


Figure 4.5: The difference between the OpenBTS and OpenBSC projects.

Due to the difficulty in acquiring hardware for use with OpenBSC, we decided to use OpenBTS for our work. It is possible for OpenBSC to directly interact with Asterisk (Laforge, 2011) so we can swap OpenBTS and OpenBSC without changing the underlying Asterisk functionality.

4.2 Experimental Setup

In this section we describe the experimental setup of our test network. Figure 4.6 shows the network setup for the experiments.

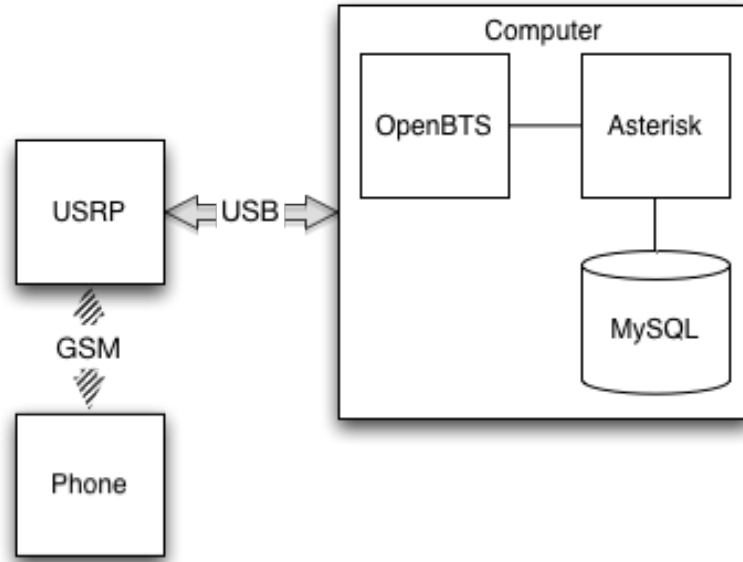


Figure 4.6: Experimental Architecture Design

4.2.1 GSM

We set up our network so as not to interfere with the commercial services of Vodafone, Telecom, or 2degrees (the three major telecommunications companies in New Zealand). This meant that we operated at low power (with a range the size of a small room), on an unoccupied frequency (uplink: 1900MHz, downlink: 1980MHz, Figure 4.7 shows a spectrum analysis of the uplink frequency), and with a closed network (only authorised Subscriber Identity Module (SIM) cards were allowed to connect). The configuration file in Appendix B.2 has the configuration options that we used for our network. Appendix A contains a guide that explains the steps needed to set up OpenBTS for our purposes. Appendix B shows the relevant configurations for Asterisk (Appendix B.1) and OpenBTS (Appendix B.2).

We wanted to emulate the conditions that a civilian would find themselves in if a disaster were to strike a region. Thus we used a selection of SIM cards from local (and one foreign) telecommunications providers. We had access to a total of two SIM cards

from each of Vodafone New Zealand and 2 Degrees Mobile, and one from Telecom New Zealand, and one from Vodafone UK. There was no effect due to the different SIM cards that registered to the network.

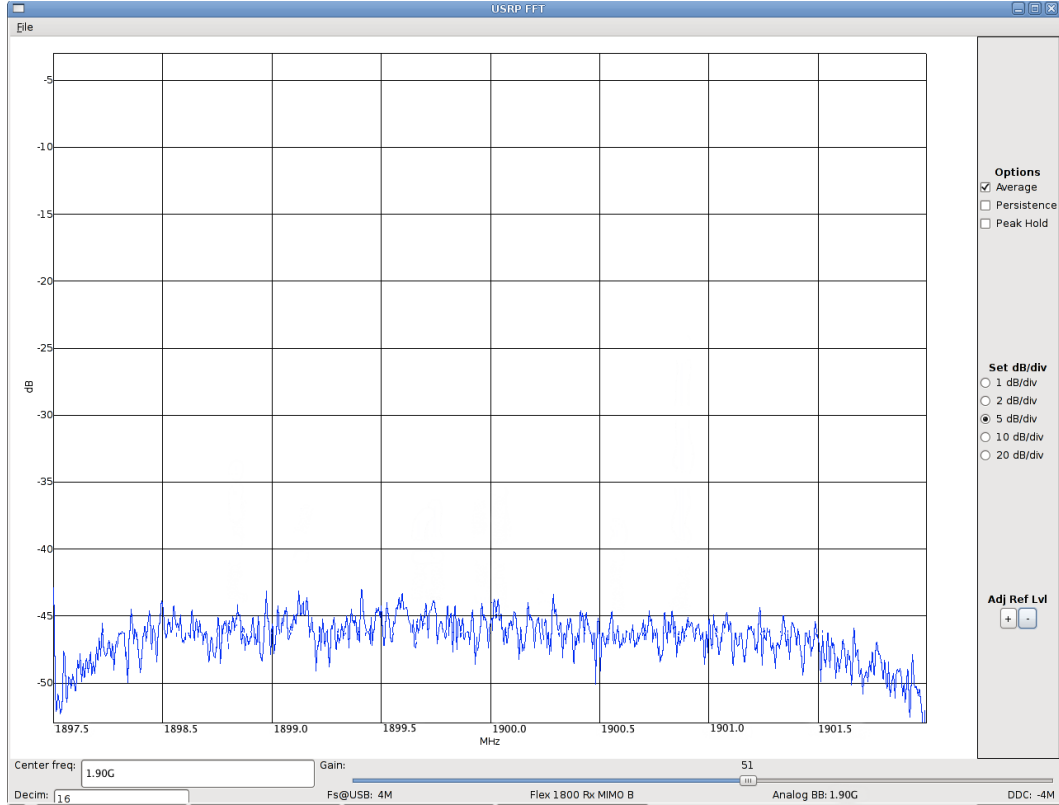


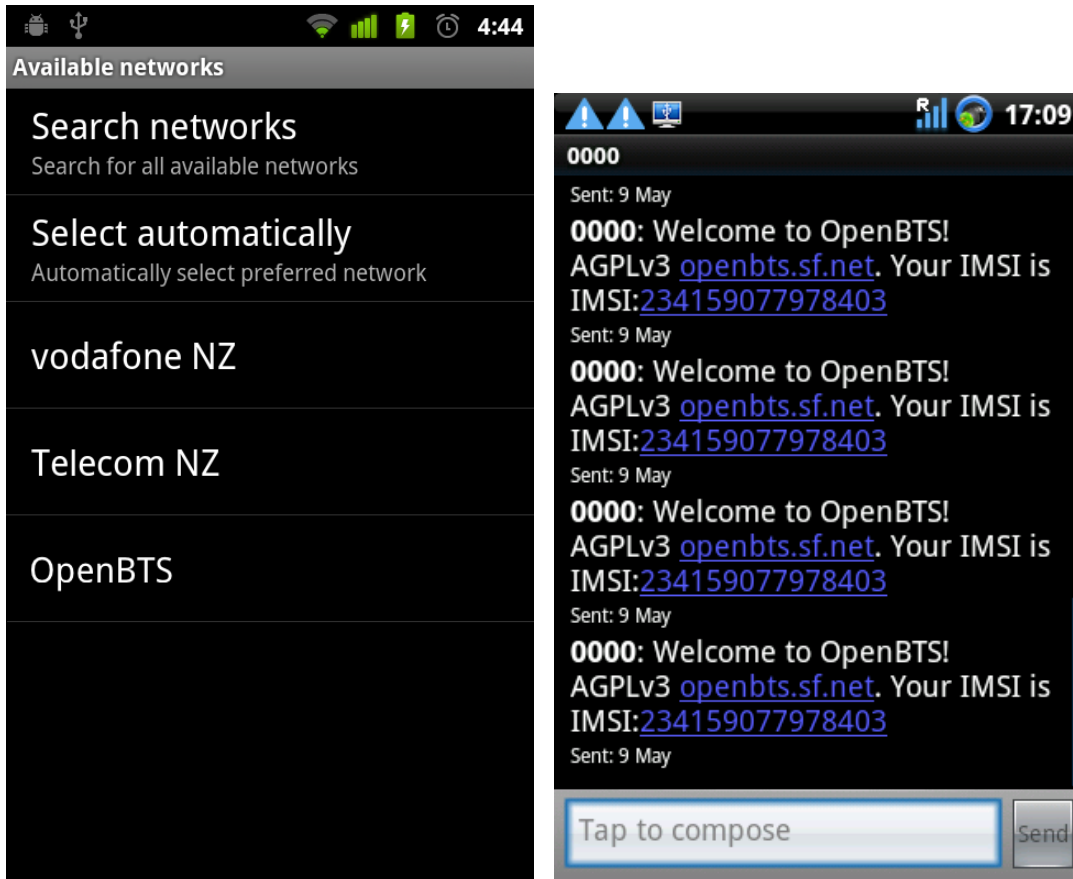
Figure 4.7: Uplink Spectrum Analysis

Each SIM card used in the phones had an entry in the MySQL database that Asterisk was able to query for the account information (the schema is in Appendix B.5, with an example in Table 4.2). The index is provided by the International Mobile Subscriber Identity (IMSI) number. Figure 4.8a shows the mobile phone finding our test network (OpenBTS) along with real providers (Vodafone and Telecom). Figure 4.8b shows the welcome message sent to mobile phones as they register on the network. We have provided the screenshots from the Android phones but the results are exactly the same regardless of make or model of phone.

The success shown in Figure 4.8 indicates that the network has been set up correctly and that the hardware and software functions as intended.

name	context	qualify	...
IMSI530240100139366	internal	yes	...
IMSI530240100555089	internal	yes	...
IMSI234159077978403	internal	yes	...
...	...	...	...

Table 4.2: A brief example of the data stored in the database



(a) OpenBTS network on the phone

(b) Welcome SMS messages from OpenBTS

Figure 4.8: Screen capture showing the OpenBTS network

4.2.2 Telephony and Routing

In this subsection we discuss the steps that occur when a phone successfully registers to the network and the subsequent call routing.

RealTime SIP peers are defined in a database and not a static configuration file. The advantage of using RealTime in Asterisk is that modifications to peers occur as soon as the users disconnect and reconnect to the server. Because we use RealTime

SIP peers, Asterisk provides us with a single **Noop** (No operation) entry that relates to the name of the peer. This is controlled by the peer option **regex** in each peer. Listing 4.1 shows our **internal** context.

```
1 [internal]
2 exten => _IMSI.,2,Answer()
3 exten => _IMSI.,n,Dial(SIP/${EXTEN})
```

Listing 4.1: Basic dialplan snippet.

Then, after the phone with IMSI530240100555089 has registered, it becomes equivalent to Listing 4.2.

```
1 [internal]
2 exten => IMSI530240100555089,1,Noop()
3 exten => _IMSI.,2,Answer()
4 exten => _IMSI.,n,Dial(SIP/${EXTEN})
```

Listing 4.2: Basic dialplan snippet after a RealTime peer has registered.

Asterisk processes the dialplan in two steps. The first step is to process the priorities from 1 (if it is missing, then it does not process any of the instructions). The second is that Asterisk will order the execution by the most specific pattern to the least specific.

If we dial IMSI530240100555089, then Asterisk will first execute the **Noop** line. Then, because the number dialed matches the **IMSI** pattern, it answers the incoming call, then places the second leg of the call to the device. The following listing shows the results of a call from IMSI530240100139367 to IMSI530240100555089.

```
1 this*CLI>
2 -- Executing [IMSI530240100555089@internal:1] NoOp("SIP/
   IMSI530240100139367-000009b1", "IMSI530240100555089") in new
   stack
3 -- Executing [IMSI530240100555089@internal:2] Answer("SIP/
   IMSI530240100139367-000009b1", "") in new stack
4 -- Executing [IMSI530240100555089@internal:3] Dial("SIP/
   IMSI530240100139367-000009b1", "SIP/IMSI530240100555089,,")
   in new stack
5 -- Called SIP/IMSI530240100555089
6 -- SIP/IMSI530240100555089-000009b2 is ringing
```

```
7      -- SIP/IMSI530240100555089-000009b2 answered SIP/  
      IMSI530240100139367-000009b1  
8      -- Locally bridging SIP/IMSI530240100139367-000009b1 and SIP/  
      IMSI530240100555089-000009b2
```

Listing 4.3: Asterisk console log of the basic dialplan snippet.

4.3 Call Routing

Once we have the phones registering successfully and able to make calls on the same node, we can extend this to provide access to more than one node. We expand on the previous work by including automatic number discovery. One of Asterisk's features is a protocol that will allow us to query the existence of dialplan patterns or numbers on another machine using the DUNDi protocol. DUNDi is a peer-to-peer system for location of telephony services. Another alternative is the e.164 standard (Faltstrom, 2000). e.164 uses DNS to provide the location to terminate a particular number. The DUNDi protocol works by querying the rest of the peers in the network and presenting a way to access that extension. Richardson (2007) give the following plain text example:

```
PBX A: Hello peer PBX B. Do you know how to reach extension  
201?  
PBX B: Hello peer PBX A. Yes I do have extension 201, here's how  
you will reach it: IAX2/username:password@10.10.14.221/201
```

The queries may pass through more than one peer. Each peer keeps track of the response so when it comes for them to respond to another query for the same number they can lookup the information in the cache (if it has not expired).

The internet draft, Spencer (2004a), describes an architecture that is necessary for carrier grade operation. For our test network we decided to go with a straightforward peer to peer setup, for the sake of simplicity. Once this peer to peer setup works then it should be possible to add the extra infrastructure without issue.

The current design of the network is shown in Figure 4.9.

Our system requires that each DUNDi peer has to connect to at least one other on the network (the more connections to other peers the more robust the network is going to be). We configure the nodes to peer with each other thus we can perform the lookups in both directions.

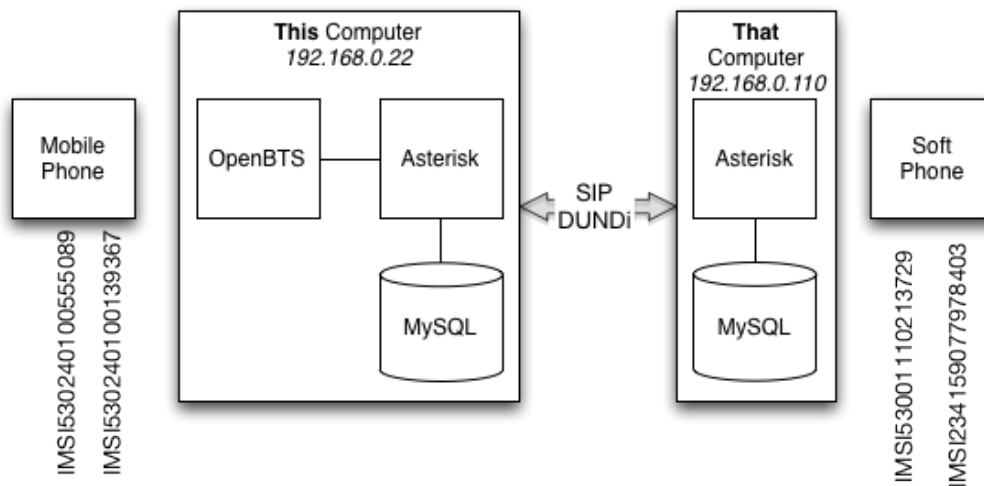


Figure 4.9: The mesh network architecture

Each node has a local MySQL database that keeps track of the devices registered to that node, so we change the `regcontext` to be `dundiextens`, and include that context in the previously described `internal` context. The dialplan is now as shown in Listing 4.4.

```

1 [dundiextens]
2 exten => IMSI234159077978403,1,Noop()
3
4 [internal]
5 include => dundiextens
6 exten => _IMSI.,2,Answer()
7 exten => _IMSI.,n,Dial(SIP/${EXTEN})

```

Listing 4.4: Dialplan showing the dynamic registration of a SIP peer

The reason we split the `regcontext` from the `internal` context is so that none of the fixed extensions get leaked to the network accidentally. We will use the `internal` context to provide some extra functionality as discussed in Chapter 6. The call routing still behaves the same as previously described.

The other task is to set up a SIP account on each of the peers that are used to terminate calls from other nodes in the network. Our setup we have called this account `trunk`. We need to have one of these accounts per node in the network. See Appendix B.1 for the setup details.

This allows us to advertise the local dynamically registered devices across the network without manual intervention.

4.4 Evaluation

4.4.1 GSM Network connectivity

Listing 4.5 shows the logs produced by OpenBTS of a mobile phone registering to the network, an explanation of this log follows.

```
1  FORCE  Logger.cpp:120:gSetLogFile: new log path test.out
2  ALARM  OpenBTS.cpp:129:main: OpenBTS starting, ver 2.6PUBLIC build
   date May  3 2011
3  TRXManager.cpp:248:sendCommandPacket: command CMD POWEROFF
4  TRXManager.cpp:248:sendCommandPacket: command CMD SETTSC 2
5  TRXManager.cpp:248:sendCommandPacket: command CMD RXTUNE 1900000
6  TRXManager.cpp:248:sendCommandPacket: command CMD TXTUNE 1980000
7  TRXManager.cpp:248:sendCommandPacket: command CMD POWERON
8  TRXManager.cpp:248:sendCommandPacket: command CMD SETPOWER 7
9  TRXManager.cpp:248:sendCommandPacket: command CMD SETMAXDLY 1
10 TRXManager.cpp:248:sendCommandPacket: command CMD SETRXGAIN 47
11 TRXManager.cpp:248:sendCommandPacket: command CMD SETSLOT 0 5
12 OpenBTS.cpp:290:main: system ready
13 GSML3Message.cpp:162:parseL3: L3 recv MM Location Updating Request
   LAI=(MCC=001 MNC=01 LAC=0x3e8) MobileIdentity=(TMSI=0x4dbf74d9)
   classmark=(revision=1 ES-IND=1 A5/1=0 powerCap=0)
14 GSMLogicalChannel.cpp:76:send: L3 SAP0 sending MM Identity Request
   type=IMSI
15 GSML3Message.cpp:162:parseL3: L3 recv MM Identity Response mobile
   id=IMSI=530240100555089
16 ControlCommon.cpp:693:resolveIMSI: 0xb66004c0
17 SIPEngine.cpp:148:Register: user IMSI530240100555089 state Null 0
   callID 1212804949
18 SIPInterface.cpp:107:addCall: creating SIP message FIFO callID
   1212804949
19 SIPInterface.cpp:170:write: write REGISTER sip:127.0.0.1 SIP/2.0
20 SIPInterface.cpp:195:drive: read SIP/2.0 200 OK
```

Listing 4.5: Excerpt from the OpenBTS log of a phone registering to the network.

The first two lines of Listing 4.5 are produced by OpenBTS setting up the logging file and starting up. The next nine lines are where OpenBTS tells the USRP what power options to use as well as the frequencies for the transmit and receive. We then get told that the system is ready (line 12) to receive connections from phones. Between lines 11 and 12 is GSM channel setup information which we have omitted for the sake of brevity and clarity. The phone then asks OpenBTS which network it's trying to connect to (asking for the Mobile Country Code (MCC), Mobile Network Code (MNC), and Location Area Code (LAC)). We then ask the phone for its IMSI number, and once we have a response we create a SIP registration message (which is then passed off to Asterisk). The next thing that we see is that the Asterisk server has responded with OK, meaning that the phone has successfully registered.

After we have registered the phones to the network, we test the ability to make and receive calls to and from the mobile phones. Listing 4.6 shows the OpenBTS logs for a mobile originated call, that is one that was initiated on the mobile phone and not from the network.

```

1  GSML1FEC.cpp:502:writeLowSide: obj: 0x860b440 RACHL1Decoder
   received RA=64 at time 0:1124319 with RSSI=-3.0000 timingError
   =1.2578
2  RadioResource.cpp:144:AccessGrantResponder: RA=0x40 when=0:1124319
   age=4 delay=1.2578 RSSI=-3.0000
3  RadioResource.cpp:221:AccessGrantResponder: sending PageMode=(0)
   DedicatedModeOrTBF=(TMA=0 Downlink=0 DMOrTBF=0)
   ChannelDescription=(typeAndOffset=SDCCH/4-0 TN=0 TSC=2 ARFCN
   =761) RequestReference=(RA=64 T1'=15 T2=1 T3=24) TimingAdvance=1
4  CallControl.cpp:570:MOCStarter: MM CM Service Request serviceType=
   MOC mobileIdentity=(TMSI=0x4dc25db9) classmark=(revision=1 ES-
   IND=1 A5/1=0 A5/2=0 A5/3=1 powerCap=0 PS=1 SSScreenInd=1 SM=1 VBS
   =0 VGCS=0 FC=0 CM3=1 LCSVA=1 SoLSA=0 CMSF=1)
5  GSMLogicalChannel.cpp:76:send: L3 SAP0 sending MM CM Service Accept
6  CallControl.cpp:613:MOCStarter: CC Setup TI=(0,0)
   CalledPartyBCDNumber=(type=unknown plan=E.164/ISDN digits=600)
7  ControlCommon.cpp:238:add: new transaction 1804289383 TI=(0,0) IMSI
   =530240100555089 MOC to=600 Q.931State=MOC initiated SIPState=
   Null (0 sec)
8  SIPEngine.cpp:231:MOCSendINVITE: user IMSI530240100555089 state
   Null
9  SIPInterface.cpp:107:addCall: creating SIP message FIFO callID
   1085953462

```

```

10 SIPInterface.cpp:170:write: write INVITE sip:600@127.0.0.1 SIP/2.0
11 GSMLogicalChannel.cpp:76:send: L3 SAP0 sending CC Call Proceeding
    TI=(1,0)
12 SIPInterface.cpp:195:drive: read SIP/2.0 100 Trying
13 SIPInterface.cpp:195:drive: read SIP/2.0 200 OK
14 GSML3Message.cpp:162:parseL3: L3 recv RR Assignment Complete cause
    =0x0
15 RadioResource.cpp:312:AssignmentCompleteHandler: service=MOC
16 CallControl.cpp:711:MOCController: transaction: 1804289383 TI=(0,0)
    IMSI=530240100555089 MOC to=600 Q.931State=MOC proceeding
    SIPState=Starting (1 sec)
17 CallControl.cpp:725:MOCController: MOC A: wait for Ringing or OK
18 SIPEngine.cpp:277:MOCWaitForOK: user IMSI530240100555089 state
    Starting
19 CallControl.cpp:725:MOCController: MOC A: wait for Ringing or OK
20 SIPEngine.cpp:277:MOCWaitForOK: user IMSI530240100555089 state
    Proceeding
21 CallControl.cpp:766:MOCController: wait for SIP OKAY
22 CallControl.cpp:803:MOCController: sending Connect to handset
23 SIPEngine.cpp:337:MOCSendACK: user IMSI530240100555089 state Active
24 SIPInterface.cpp:170:write: write ACK sip:600@127.0.0.1 SIP/2.0
25 CallControl.cpp:232:callManagementDispatchGSM: GSM Connect
    Acknowledge IMSI=530240100555089
26 CallControl.cpp:551:callManagementLoop: IMSI=530240100555089 call
    connected
27 SIPInterface.cpp:195:drive: read SIP/2.0 200 OK

```

Listing 4.6: Excerpt from the OpenBTS log of a phone dialing 600.

The first step is the allocation of a radio channel for the call control (lines 1–3). Then the phone generates a service request (of type Mobile Originated Call (MOC)). This request contains the identifying information of the phone, either the IMSI or the Temporary Mobile Subscriber Identity (TMSI) (line 4). If the network is able to fulfil the request it responds with a **Service Accept** message (line 5). Line 6 is the message that the phone sends with the number that the users wishes to dial.

We then start a new SIP transaction to **INVITE** the called party to the call (lines 7–10). We update the phone to let it know that we are processing the call (line 11). We receive two SIP messages from Asterisk, the first is to let us know that it's **Trying** to place the call (line 12), then once it places the call it sends **OK** (line 13). The next nine lines are updating the internal state of the transaction. We now have almost finished the SIP portion of the call setup. We tell the phone that we have connected

via a `connect` message (line 22). Once we have sent the `connect` message we send a SIP ACK (line 24) and connect the audio streams. We can read both the GSM Connect Acknowledge (line 25) and the SIP OK (line 27). Now we have completed the call setup. Figure 4.10 shows this process as a call flow diagram.

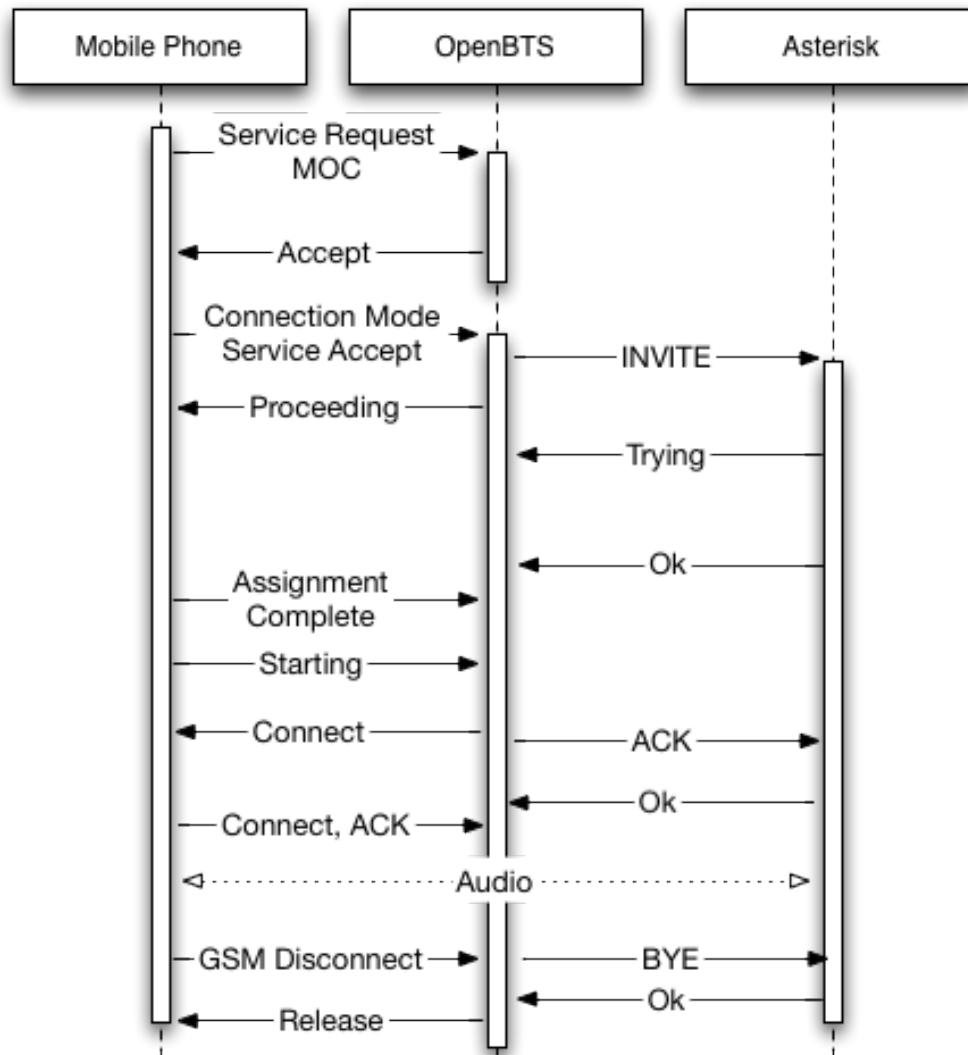


Figure 4.10: Mobile originated call flow diagram

```

1 GSML3Message.cpp:162:parseL3: L3 recv CC Disconnect TI=(0,0) cause
  =(location=0 cause=0x10)
2 CallControl.cpp:280:callManagementDispatchGSM: GSM Disconnect IMSI
  =530240100555089
3 GSMLogicalChannel.cpp:76:send: L3 SAP0 sending CC Release TI=(1,0)
  
```

```
4 SIPEngine.cpp:378:MODSendBYE: user IMSI530240100555089 state Active
5 SIPInterface.cpp:170:write: write BYE sip:600@127.0.0.1 SIP/2.0
6 SIPInterface.cpp:195:drive: read SIP/2.0 200 OK
7 CallControl.cpp:305:callManagementDispatchGSM: GSM Release Complete
  IMSI=530240100555089
8 GSMLogicalChannel.cpp:76:send: L3 SAP0 sending RR Channel Release
  cause=0x0
9 SIPInterface.cpp:114:removeCall: removing SIP message FIFO callID
  1085953462
```

Listing 4.7: Excerpt from the OpenBTS log of a phone hanging up.

Following the progress of a completed call is more straight forward. The mobile phone sends a **Disconnect** message. OpenBTS then send a **Release** message to indicate that we are shutting the channels down. Once the handset has received that command, it responds with a **Release Complete** message. OpenBTS then indicates to Asterisk via a SIP BYE message that the caller has terminated the call, OpenBTS receives an **OK** message back. At this point, we let the handset know that the call completed and send a **Channel Release** message.

In this case the phone is terminating the connection. If the network were to terminate the connection the roles would be swapped apart from the final **Channel Release** message. This **Channel Release** is always sent by the network (Eberspächer and Vögel, 1998).

4.4.2 Number Lookups

In order to illustrate the number lookup functionality, we have modified the setup slightly from the previous experiment. We now have a second server and two phones. This is to test the distributed number look up ability. In this experiment we register the IMSI530240100139367 account to the node on the left (or **this**) and the IMSI234159077978403 account to the node on the right (or **that**).

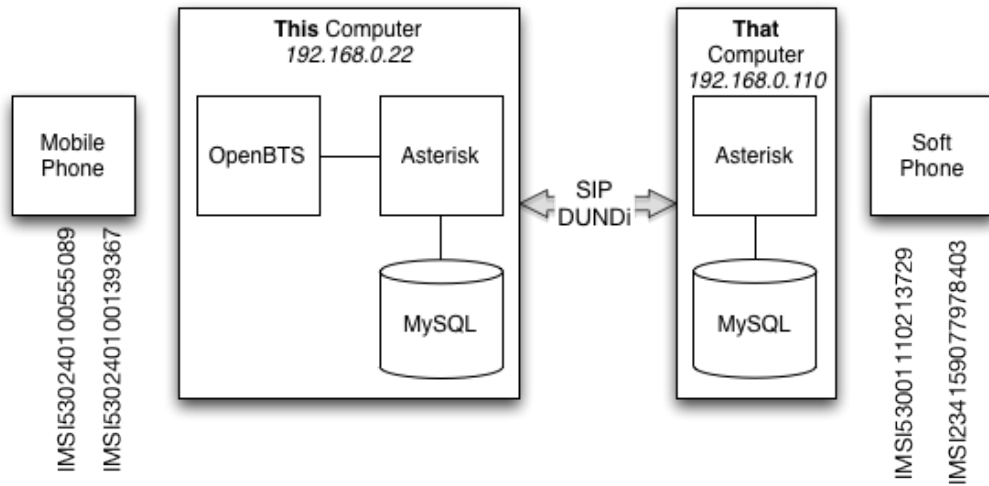


Figure 4.11: Call Routing Experimental Architecture Design.

We add a SIP peer to the database and perform the following commands from each of the Asterisk servers. We query phones registered to the remote machine — DUNDi is unable to perform lookups on the same machine that the phone is registered to. Listing 4.8 shows a successful DUNDi query, and Listing 4.9 shows an unsuccessful query.

```
1 this*CLI> dundi lookup IMSI234159077978403@priv
2 1. 0 SIP/trunk/IMSI234159077978403 (EXISTS)
3 from 00:26:18:ee:49:19, expires in 5 s
4 DUNDi lookup completed in 25 ms
```

Listing 4.8: DUNDi lookup performed querying a phone registered to a remote machine.

```
1 this*CLI> dundi lookup IMSI530240100139367@priv
2 DUNDi lookup returned no results.
```

```
3 DUNDi lookup completed in 0 ms
```

Listing 4.9: DUNDi lookup that fails.

Now that DUNDi has been shown to be working, we have the following call log (Listing 4.10) showing the progression of a call from one device on one node to another device on a remote node.

```
1 this*CLI> sip show peers
2 Name/username      Host      Port      Status      Realtime
3 IMSI530240100139366 192.168.0.15 5060      OK (1 ms)   Cached RT
4 trunk              192.168.0.22 5060      OK (1 ms)
5 2 sip peers [Monitored: 1 online, 1 offline Unmonitored: 0 online,
   0 offline]
6 this*CLI> dundi lookup IMSI234159077978403@priv
7 1.      0 SIP/trunk/IMSI234159077978403 (EXISTS)
8      from 00:26:18:ee:49:19, expires in 5 s
9 DUNDi lookup completed in 25 ms
10 this*CLI>
11 == Using SIP RTP CoS mark 5
12 == Using SIP RTP CoS mark 5
13 -- Called SIP/trunk/IMSI234159077978403
14 -- SIP/trunk-000009b4 answered SIP/IMSI530240100139366-000009b3
15 -- Locally bridging SIP/IMSI530240100139366-000009b3 and SIP/
   trunk-000009b4
16 == Spawn extension (internal, IMSI234159077978403, 1) exited non-
   zero on 'SIP/IMSI530240100139366-000009b3'
```

Listing 4.10: Asterisk console log for a call from a remote machine to a peer on this machine.

Lines 2–5 of Listing 4.10 show the result of `sip show peers` (executed on line one). We also perform a DUNDi lookup (line 60) and show the result (lines 7–9). Line 11 is where the call is being placed, we can see that this node places a call via the SIP trunk (line 13), and then starts sending the audio (line 15). Moments later, the phone hangs up the call (line 16).

```
1 remote*CLI>
2 -- Registered SIP 'IMSI234159077978403' at 192.168.0.15:5060
3 -- Added extension 'IMSI234159077978403' priority 1 to
   dundiextens
```

```

4 remote*CLI> sip show peers
5 Name/username      Host      Port      Status      Realtime
6 IMSI234159077978403 192.168.0.15 5060      OK (1 ms)   Cached RT
7 trunk              192.168.0.110 5060      OK (1 ms)
8 2 sip peers [Monitored: 1 online, 0 offline Unmonitored: 1 online,
   0 offline]
9 localhost*CLI> dundi lookup IMSI530240100139366@priv
10 1.      0 SIP/trunk/IMSI530240100139366 (EXISTS)
11      from 00:50:8d:f3:6b:32, expires in 5 s
12 DUNDi lookup completed in 1 ms
13 remote*CLI>
14 == Using SIP RTP CoS mark 5
15 -- Executing [IMSI234159077978403@internal:1] NoOp("SIP/trunk
   -00000000", "IMSI234159077978403") in new stack
16 -- Executing [IMSI234159077978403@internal:2] Answer("SIP/trunk
   -00000000", "") in new stack
17 -- Executing [IMSI234159077978403@internal:3] Dial("SIP/trunk
   -00000000", "SIP/IMSI234159077978403,,") in new stack
18 == Using SIP RTP CoS mark 5
19 -- Called SIP/IMSI234159077978403
20 -- SIP/IMSI234159077978403-00000001 is ringing
21 -- SIP/IMSI234159077978403-00000001 answered SIP/trunk-00000000
22 -- Locally bridging SIP/trunk-00000000 and SIP/
   IMSI234159077978403-00000001

```

Listing 4.11: Asterisk console log for the same call from the remote machine.

Listing 4.11 shows the same call progression from the remote machine. Lines 2–3 show the phone registering to the network. We then check that the SIP clients are reporting properly (lines 5–8). As a test, we perform the DUNDi query (line 9) and get a result (lines 10–12). We receive a request for the extension IMSI234159077978403 in the `internal` context (line 15). The first instruction is a `Noop`, this is the `Noop` that the dynamic registration gives us. Then we proceed by answering the incoming leg of the call (line 16) and place the outgoing leg of the call (line 17). We are notified that IMSI234159077978403 has been called (line 19) and is ringing (line 20). Then we bridge the legs of the calls together (line 22). Now both parties are able to talk.

4.5 Conclusion

The problem stated at the beginning of this chapter has been solved: as shown in Section 4.4. Unmodified GSM phones can associate with the network (Subsection 4.4.1). Those same devices can place calls to other devices on the same node and on a remote node (Subsection 4.4.2).

We have shown the basic call flow between devices on the same node and between devices on different nodes using DUNDi as the number discovery mechanism (Subsection 4.2.2). Using DUNDi protocol to route calls to devices connected via OpenBTS is a legitimate way to provide the call routing functionality.

4.5.1 Unique Contribution

We have described a peer-to-peer network capable of routing voice traffic to users of the network dynamically. This implementation requires no interaction from an administrator apart from the initial setup of the nodes, and in our case the addition of SIP accounts for each of the SIM cards on the network. The manual administration of accounts is due to the experimental nature of the network and not wanting to interfere with commercial services, while in a real world system this would be automated.

There has been no prior published work on the ability to use dynamic SIP peers and dynamic call routing using DUNDi with OpenBTS.

4.5.2 Discussion

The network as implemented in this chapter relies on knowing the callee's IMSI number. This is not normally information that users remember. We would like to enable the user's phone books to be useful when using this network. This means that there needs to be a mechanism to map between the phone number and the IMSI. Given that this network is separate from the existing networks this network needs to provide support for this function. This work is presented in Chapter 5.

4.5.3 Future Work

Investigate the different topologies for the nodes in the network, in other words, as a standalone node (where calls are routed to the PSTN), or as a repeater node (that is there to extend the coverage of the mesh network).

There has been no testing as to the capacity of the network nor the amount of bandwidth needed for calls/other traffic (and thus it is unknown how feasible it is to put this work on top of existing RDN technologies). There has been work on pure VoIP systems on top of rapidly deployable networks, but no work on using DUNDi as the number discovery protocol.

A key element in solutions of this nature is to prove the scalability and resiliency. For our proposed solution we have not tested this aspect. Use of a software simulator (such as NS2) would provide this data and ensure that any problems are found and solved before beginning field trials.

Coincidentally there has been a new version of the OpenBTS software released. This release changes the way that the USRP is set up. They now use a database to store settings and so most of the options are now dynamic, that is they can be adjusted on a running system.

We use DUNDi to perform the automatic routing. The recommended setup requires a core network with fail-over and so on. In our network if a node goes down (due to lack of power, physical damage, or network outage, etc.), then there is no backup for that node's state. This means that all the information about how to route to mobiles connected to (and via) that node would not be available. However, if a node goes down then those people attached to that node will not be able to use their phones anyway.

Another improvement would be to automate the addition and removal of DUNDi peers, as well as the SIP trunking accounts. In order to add a DUNDi peer a set of cryptographic keys needs to be distributed to each of the nodes that wish to become part of the cluster, as well as configuration of VoIP trunks, and DUNDi peers themselves. We envisage an Application Programmer Interface (API) that allows remote nodes to query this information from other nodes in the network — much like zero-conf networking systems. The one problem with this approach is that Asterisk's DUNDi configuration is set in static files which would require reloading that node's configuration every time a peer was added or removed from the network. This increases the load unnecessarily.

We would also like to automate the addition of SIP accounts via OpenBTS for the open network setup.

Chapter 5

User Management

5.1 Overview

The civilian users of the phone system that we presented in Chapter 3 are going to expect to be able to dial normal phone numbers, that is those of friends and family as well as emergency services. They will also be expecting their phone's contact list to still be operational. In this chapter we describe the method that we use to map between a user's phone and their phone number. There are several ways to verify a claim to ownership.

The GSM architecture defines a centralised approach. The components of note are the Home Location Register (HLR) and Visitor Location Register (VLR). This and other mechanisms are discussed in Section 5.2. The research question is discussed in Section 5.3.

5.2 User Verification in Literature

Abadi, Burrows, and Needham (1990) list three ways to prove that someone has legitimate access to resources, these are replicated in Table 5.1.

Property	Definition	Example
Proof by Knowledge	knowing a shared token	a shared secret
Proof by Possession	possessing an authenticated token	a smart card
Proof by Property	ability to measure something unique	user's voice

Table 5.1: Three ways to prove someone is who they claim to be

There is a common approach to verifying account information on the Internet by associating an email address with an account. For example, if a user creates an account and registers with a mailing list, then the mailing list should verify that the user owns the supplied email address. The service sends a unique token to the email address and the user then supplies that token to the service, thus proving ownership of the email account (National and Libes, 1996). This mechanism will not work in our system. The key difference between creating the mailing list account and verifying the phone numbers of our users is that the phone numbers are already associated with a person, whereas the created account has no prior association.

Traditional GSM networks have two components that handle the user account information. These are the HLR and the VLR. Both of these components essentially perform the same task—keeping a record of the IMSIs on the network. The HLR contains a list of all the IMSIs issued by the provider. The VLR contains a list of roaming IMSIs (viz. those that belong to a different provider). Both the HLR and the VLR map the phone number, in the telecommunications literature as Mobile Station Integrated Services Digital Network (MSISDN), to the IMSI (Mehrotra, 1997). The IMSI, and often MSISDN, numbers are stored on the SIM card contained in GSM mobile phones.

In these networks, when a user roams from the home network to another, the visited network updates the home network’s HLR enabling the home network to find the roaming user. This is done by using the Mobile Application Part (MAP) protocol (one of the protocols in the Signalling System No. 7 (SS7) protocol suite).

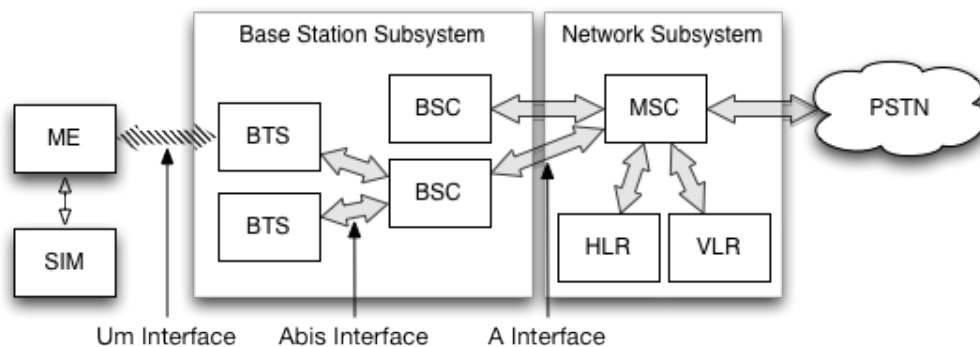


Figure 5.1: GSM Network Architecture Design, duplicated from Figure 4.1.

In Brainard, Juels, Rivest, Szydlo, and Yung (2006), the authors propose using a social network approach to the verification problem. In their paper the authors describe a situation where a user in an office has forgotten their password. When they phone

the help desk to get it reset, the help desk staff have to perform authentication via some other method, for example, identity cards. If the staff member recognises the user's voice they can potentially skip other forms of authentication.

Xiong and Liu (2004a) outlines several existing schemes that allow us to validate users based on reputation. That is what other users of the system think. Sites like eBay use a rating system (1,0,-1) and allows users to leave comments (Resnick, Kuwabara, Zeckhauser, and Friedman, 2000), Amazon uses a 1-5 scale for different attributes (friendliness, prompt response, quality product) (Resnick *et al.*, 2000). These ratings are aggregated and presented to potential traders who are then able to make a more informed decision about who to trade with.

The Serval Project (Gardner-Stephen, 2011) uses Android-powered smart-phones to form an ad-hoc WiFi network. In this network, the researchers want to make the user's phone numbers available to others on the network. They propose two methods for making the phone numbers available to others. First they suggest that each of the users submits their own number and perhaps a voice 'fingerprint', for example the user saying their name and playing this to potential callers before or after connecting the call. From here the number is either verified by a centralised authority or, if the user recorded a 'fingerprint', callers can verify before a call is connected. This allows the caller a chance to verify the recipient before making the call. The second method to allocate the phone numbers to users is to generate them from the IP address of each node. They also propose that the called party is identified after a call has been completed. Each user in the Serval system has a Subscriber ID (which is a public key) which is also used to sign digital certificates to indicate that a user verifies another. This prevents spoofing of phone numbers. It also allows for a formation of a web of trust to form between users (Gardner-Stephen, 2011). This differs from our work in the web of trust component, we only keep a record of the number of times someone has been verified and not who has verified whom.

5.3 Research Question

The major source of the problem here is the ability to link IMSIs and phone numbers. We are unable to query a trusted party because it may have been damaged or is otherwise unavailable—we are assuming that the network is operating in a disaster scenario, in other words, our network is physically separated. We are also unable to trust the users to provide complete and accurate information every time, people can

make mistakes or they can lie.

Can we devise a system that is able to map between IMSI and phone numbers in a way that allows the users to provide that mapping, while providing a way for verification of those users in a fully distributed manner without the need for smart phones?

Chapter 6

Beacon User Management

In Chapter 5 we explored methods to manage users in a decentralised environment. In this chapter we propose a simple user management scheme.

Before we start describing the implementation we first explain our assumptions. The major assumption is that we are separate from the existing network infrastructure. There is no method to query which phone number belongs to which IMSI. To pre-load our system with that information would require interfacing with existing providers' systems using the MAP protocol to query and update their VLRs and HLRs (Mehrotra, 1997). While Asterisk does have support for SS7 via third-party libraries (both open source and commercial). Integrating with the existing provider's network remains outside the scope of this thesis.

As a natural consequence of this there is no way to route calls from outside the network to devices on the network by their phone number. In other words, if someone outside the network calls a phone number when that device is on our network there is no way the person will be able to receive that call.

In our solution, we move the HLR/VLR functionality to a distributed database and provide a mechanism to manipulate the data, described in Section 6.1. The method employed here is used without any interaction on the part of system administrators or existing telecommunications companies.

Figure 6.1 shows where we store the different information in the network. Each user of the system creates an entry in a MySQL database for the SIP account settings. These settings are local to that node and are never shared across the network. The IMSI, phone number, and reputation elements are stored in a distributed database that every node in the network has access to. In Figure 6.1 there is only one phone on the network, it has data on its local database (the MySQL database) and in the

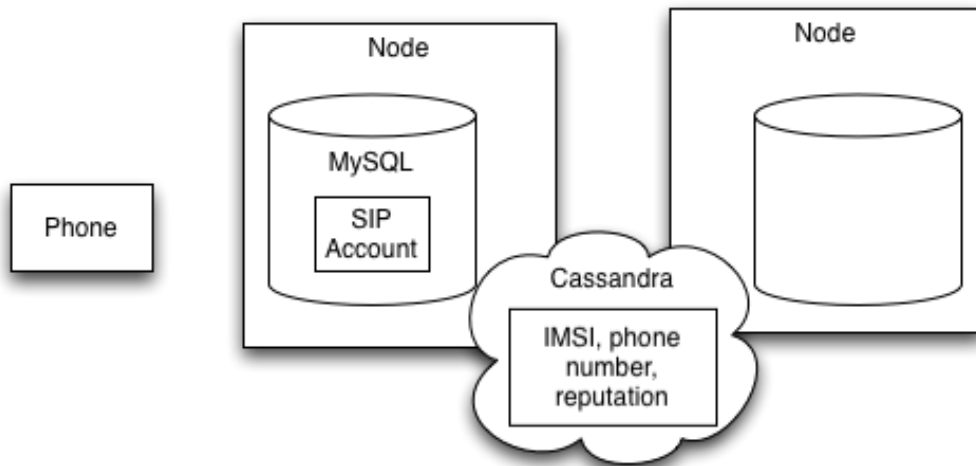


Figure 6.1: The different places that information can be stored in the network.

distributed database.

Section 6.2 covers the background information for the distributed database component used by both the mappings and verification components. Section 6.3 describes the web interface we use to abstract the reputation logic away from the call routing. In Section 6.4 discusses how to map between the IMSI and phone number. This section covers everything from the design to the evaluation of the mappings between phone number and IMSI. Section 6.5 follows the same pattern as Section 6.4 and discusses the reputation management system.

6.1 Overview

In this section we discuss the design of the different components in our system to support user verification as well as the mappings between the phone numbers and the IMSI.

An overview of the design appears in Figure 6.2. The Cassandra Database is the only component that is distributed across the network; all the other components are local to each node.

The administration functions are a web-interface to the Asterisk Realtime database. They grant a user the ability to create and remove SIP accounts from the system because at this point in time, the auto-creation of SIP accounts is not functional. In future, we envisage that these functions will be expanded as discussed in Subsection 6.6.3.

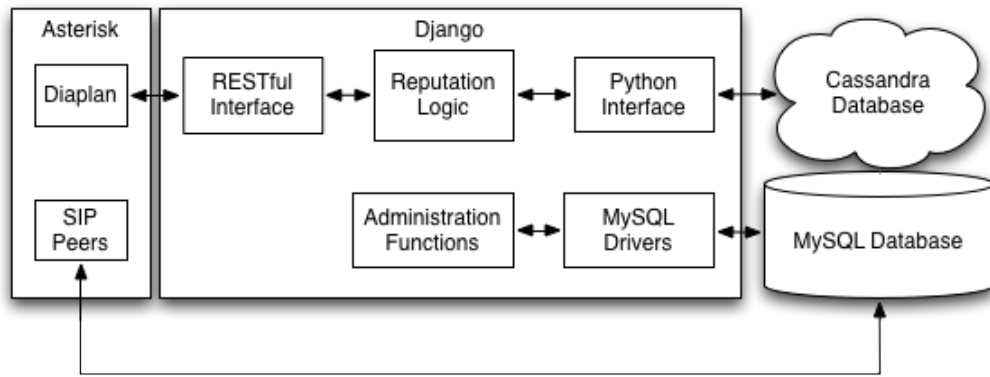


Figure 6.2: High-level overview of the different components of the user verification and mapping system.

6.2 Distributed Databases

We want information to be global to the network without having to rely on a central server to manage this for us. Fortunately there is a solution to this type of problem. Distributed databases (Ozsu and Valduriez, 2011) provide a mechanism that allows for dynamically adding and removing nodes from a cluster, as well as handling data replication. We have chosen to use Apache’s Cassandra, but in reality any distributed database implementation would suffice. Cassandra was developed at Facebook and is currently being used by some of the web’s most popular websites for example, Reddit, Digg, and some components of Twitter (Hewitt, 2010).

Node Maintenance

In order to support the distributed nature of the design, the Cassandra nodes need to join a cluster on the fly. When two or more Cassandra instances are acting together we have a cluster which allows Cassandra to scale. To add a node we tell the node which cluster to join. Once the node is available it discovers the topology of the cluster and will accept data for which it is responsible. Once this has been completed the node is now a fully fledged member of the cluster and can start accepting operations (Hewitt, 2010).

Cassandra monitors the status of nodes in the same cluster by the use of a ‘gossip’ protocol. Each node in the cluster runs a gossip every second. The gossip chooses another node at random and starts a gossip session. The initiator sends a **SYN** packet to which the receiver sends **ACK** back. The initiator then sends a final **ACK2** message.

Column Name	Value
phone_number	64211234567
score	0

Table 6.1: The relationship between Column Names, and Values in Cassandra.

Row Key	Column Name	Value
IMSI530240100139367	phone_number	64211234567
IMSI530240100139367	score	0
IMSI530011102137279	phone_number	64212742118
IMSI530011102137279	score	1

Table 6.2: The relationship between Row Keys, Column Names, and Values in Cassandra.

If this sequence is not completed then the node is marked as ‘dead’ (Hewitt, 2010; Lakshman and Malik, 2010).

Once a node is ‘dead’, any write operations are processed according to a ‘hinted hand off’. The ‘hinted hand off’ works with active nodes storing the write information on behalf of the ‘dead’ nodes until those nodes can recover (Hewitt, 2010; Lakshman and Malik, 2010).

Data Partitioning

The data is split across nodes in the cluster using consistent hashing functions. In these functions the output is treated as a fixed ring. Each node in the cluster is assigned a value within this ring. The data’s key is then hashed, and the ring traversed to find the node responsible for that data (Lakshman and Malik, 2010).

Data Model

Cassandra stores a map of key-value pairs (or **Column Name** and **Value** in the Cassandra lingo (Hewitt, 2010)). Table 6.1 shows an example mapping.

RowKeys group these key-value pairs. In Table 6.2 we are grouping by the **RowKey**, which is a unique identifier for each set of key-value pairs (Hewitt, 2010). A **Keyspace** is a group of these **Row Keys**.

Cassandra in Beacon

We use Cassandra to provide a robust, distributed global data store for information about the users in the network. We provide the details in Section 6.4 of how this global data store works.

6.3 Web-based Interface

We chose to abstract the reputation logic from the call routing logic. This allows for greater flexibility and extensibility in the implementation of the reputation logic. We discuss the extensibility in Section 6.6.3.

The interface is provided by a web API. We use Django to build the web API. Django is a web-framework written in Python that follows a model-view-controller design pattern. Our API uses Javascript Object Notation (JSON) as the language to pass data.

The three files that contain the majority of the work are presented in Appendix B.3. We discuss each in turn below.

The API implementation is in Appendix B.3.1, we define three methods that are allowed, these relate to the HTTP methods of `GET`, `PUT`, and `POST` and are mapped to the `read`, `update`, and `create` methods respectively.

The `read` method accepts either an `IMSI` or `phone number`, and returns the record associated with it. The `update` method accepts an `IMSI` and requires a list of fields and one of `["up", "+", "inc", "++"]` or `["down", "-", "dec", "--"]` associated with that field. The provided fields are either incremented or decremented. The `create` method accepts an `IMSI` number and creates a new reputation entry in the Cassandra database.

In order to interface with the Cassandra database we provide a series of utility functions. These are shown in Appendix B.3.2. We split the functionality in to two. The first part is intended as private, and are denoted by the underscores in the function names (`_get_imsi`, `_get_number`, and `_inc_field`). The remaining functions are intended as a public part and are convenient wrappers around the private functions. The most interesting is the `create_imsi_record` function, it creates a new association between an `IMSI` number and a `phone number`. If there's an existing association, then it resets the fields back to their initial values. The effect of this is it overwrites the existing information.

Finally, the model that provides the SIP account information appears in Ap-

pendix B.3.3. Django generates an administration interface for management of models. Asterisk is configured to read this database in the `extconfig.conf` file, this appears in Appendix B.1.5.

6.4 Phone Numbers

In this section we discuss how we have implemented the mappings between IMSI and phone number. The network that we extend is able to locate devices by IMSI across the network. We wish to associate those IMSIs with a phone number. This mapping should be global — that is, every node on the network should be able to read and update this information regardless of which node a device is associated with.

We propose storing the mapping in a distributed database indexed by the IMSI as well as by phone number. Xiong and Liu (2004a) have spawned a large body of research into trust in peer to peer systems, with different reputation models applied to different systems. We chose to implement a scheme similar to the basic Xiong and Liu (2004a) model because of the simplicity of the architecture. Figure 6.3 shows the architectures both from (Xiong and Liu, 2004a) and our own proposed architecture.

The key difference is how the data is located. In Xiong and Liu (2004a) they use a data locator to find where the data is stored in the peer-to-peer network. We do not need this explicit data locator as Cassandra handles the data distribution for us. In other respects, our architecture is similar to theirs.

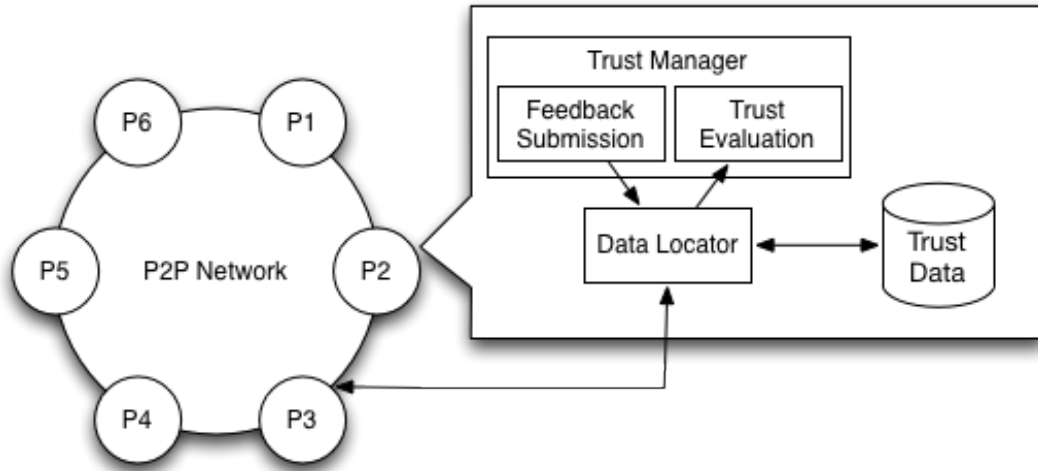
6.4.1 Database Schema and Programmer Interface

For this part of the work we need to store the phone number associated with the given IMSI. We do this by setting the `RowKey` to the IMSI, `Column Name` to `number` and the `Value` is the phone number.

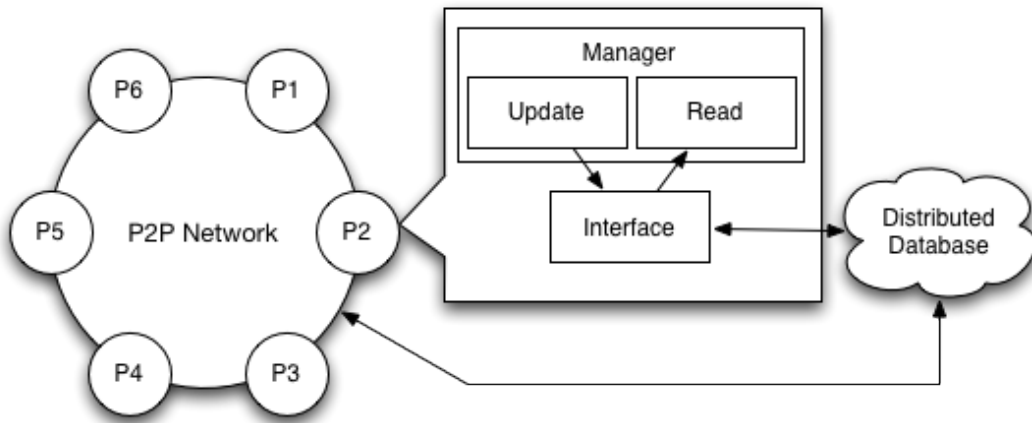
Each of the nodes in the network has a local Django server that provides a web interface between the dialplan and the Cassandra database. We've modified the architecture as described in Xiong and Liu (2004a). Because we utilise Cassandra's distributed database we do not need to have the data locator; Cassandra handles that for us. The TrustManager is the Django server component.

This interface needs to support the following operations:

- Read by IMSI
- Read by phone number



(a) The architecture in (Xiong and Liu, 2004a). The Trust database is local to each node in the network. Data is found through the data locator component.



(b) Our proposed architecture. The manager provides functions to update and read information stored in the distributed database.

Figure 6.3: A comparison between Xiong and Liu (2004a) and our trust architectures. P_n indicates the n^{th} peer. The database is distributed amongst the peers in the network.

- Create a new record

We have chosen a web interface to allow for flexible expansion in the future. We use JSON to pass data between the Asterisk dialplan and the web interface. This is discussed in Section 6.3.

6.4.2 Call Flow for Setup

Before the system can perform a lookup, the system needs to associate a phone number with an IMSI. When a user first connects to the network, a welcome message is sent. In that same message the system can ask them to call an extension to perform the setup. Once we receive a call to this extension, the system answers and requests the user to enter their phone number. This number is then repeated back to them. If the user then confirms the number, the data is passed through the web interface and stored in the Cassandra database.

6.4.3 Setup Evaluation

We manually dialled the phone numbers using a soft-phone (Zoiper, and X-lite—both are phones implemented in software as opposed to real hardware) to take the place of the mobile phone. Note that in Figure 6.4 that there are two IMSI numbers on each side of the diagram, these allow us to test calls between each of those four users. We collected the Asterisk call logs and present them in Listing 6.1. The dialplan and other source code appears in Appendix B. The experimental setup is depicted in Figure 6.4.

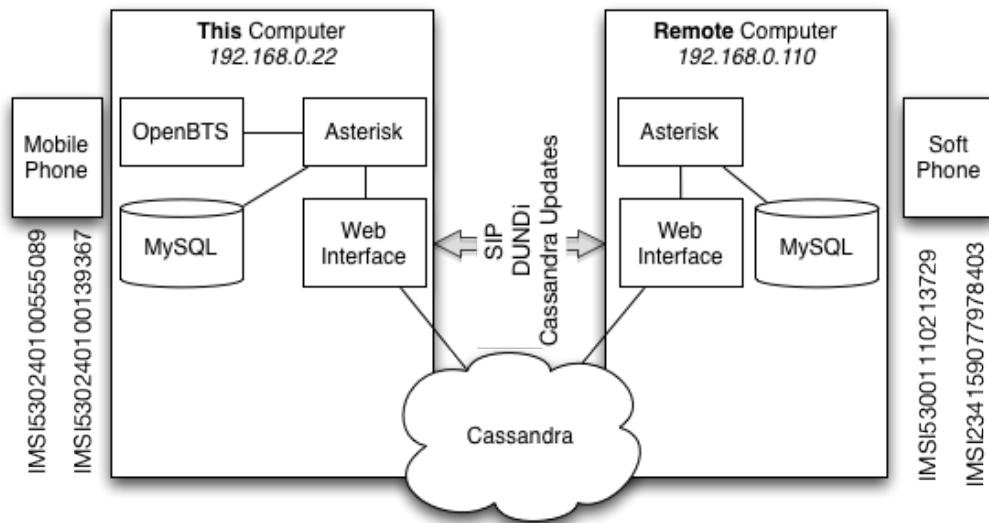


Figure 6.4: Reputation and Call Flow Experiment setup.

In order to associate a phone number and an IMSI, we need to perform a setup step where we insert the records in to the Cassandra database.

```
1 -- Executing [777@internal:1] Goto("setup,s,1") in new stack
```

```

2      -- Goto (setup,s,1)
3      -- Executing [s@setup:1] Answer("") in new stack
4      -- Executing [s@setup:2] NoOp("Beacon - The Rapidly Deployable
      GSM Network") in new stack
5      -- Executing [s@setup:3] Set("count=0") in new stack
6      -- Executing [s@setup:4] Festival("To get started, enter your
      phone number (with country code), when done press the # key
      ") in new stack
7      -- Executing [s@setup:5] GotoIf("0?error") in new stack
8      -- Executing [s@setup:6] Set("count=1") in new stack
9      -- Executing [s@setup:7] Read("num") in new stack
10     -- User entered '64212742118'
11     -- Executing [s@setup:8] Festival("You entered 64212742118") in
      new stack
12     -- Executing [s@setup:9] NoOp("Does the number exist?") in new
      stack
13     -- Executing [s@setup:10] AGI("parse_json.py,
      IMSI530011102137279") in new stack
14     -- Launched AGI Script /var/lib/asterisk/agi-bin/parse_json.py
15     -- AGI Script parse_json.py completed, returning 0
16     -- Executing [s@setup:11] GotoIf("0?collect") in new stack
17     -- Executing [s@setup:12] Festival("Creating your reputation
      ...") in new stack
18     -- Executing [s@setup:13] NoOp("64212742118") in new stack
19     -- Executing [s@setup:14] System("/usr/bin/curl -X POST http
      ://192.168.0.22:8080/api/account/IMSI530011102137279/ -d '{"
      data": [{"number": "64212742118"}]}') in new stack
20     -- Executing [s@setup:15] Festival("Created") in new stack
21     -- Executing [s@setup:16] Hangup("") in new stack

```

Listing 6.1: Asterisk console output of a call to the setup routine

Every phone has been set up to use the **internal** context as the default. In that context we have created an extension **777** that redirects to the **setup** context. We then answer the call and set up a count to keep track of the number of times we have asked the user to enter their number to prevent looping forever. Then we ask the user to enter their number and check to see if it exists via the mechanism described below.

The `parse_json.py` (presented in Appendix B.4.1) script performs the lookup. It is an Asterisk Gateway Interface (AGI) script. Asterisk has the ability to run external commands (or scripts) passing the dialplan variables as a series of key-value pairs. The script can write information back to the Asterisk dialplan by using the same key-value

pairs.

If the number does not exist then we create a new reputation record for that IMSI associating the entered number with that IMSI account, this step is performed via the web interface. This interface then updates the records in the Cassandra database. Listing 6.2 shows the Cassandra database before the changing the number.

```
1 [default@Beacon] list IMSI;
2 Using default limit of 100
3 -----
4 RowKey: IMSI530011102137279
5 => (column=initiated_calls, value=0, timestamp=1319486046761815)
6 => (column=number, value=64212742117, timestamp=1319486046761815)
7 => (column=received_calls, value=0, timestamp=1319486046761815)
8 => (column=score, value=0, timestamp=1319486046761815)
```

Listing 6.2: Cassandra console output the data stored previous to the update

We have now completed the setup above and have changed the number associated with this IMSI, see Listing 6.3.

```
1 [default@Beacon] list IMSI;
2 Using default limit of 100
3 -----
4 RowKey: IMSI530011102137279
5 => (column=initiated_calls, value=0, timestamp=1319486133812723)
6 => (column=number, value=64212742118, timestamp=1319486133812723)
7 => (column=received_calls, value=0, timestamp=1319486133812723)
8 => (column=score, value=0, timestamp=1319486133812723)
```

Listing 6.3: Cassandra console output the data stored after the update. Note that the timestamps (the digits on the right) and number value have changed.

6.4.4 Number Discovery Call Flow

We need to modify the steps from the previous network to allow calls to be placed via phone number. Previously, we just handled dialing by IMSI, that is, if the number began with IMSI with digits after it, look up how to dial that number using DUNDi and place a call to it.

```
1 if the number dialed begins with IMSI
2     find the device's address using DUNDi
3     dial that address
```

```

4  else
5      error message
6      exit

```

Listing 6.4: Algorithm for basic call routing.

Now, we need to perform an initial lookup to discover the IMSI for that number.

```

1  if the number dialed contains only digits
2      lookup the IMSI for the number dialed in the database
3      if there is a result
4          goto previous routing logic
5      else
6          error message
7          exit

```

Listing 6.5: Algorithm for number based call routing.

The lookup occurs in an AGI script. These scripts are a way for third party applications to interact with the Asterisk dialplan. The dialplan variables are passed as the standard input to the script and instructions are sent back via standard out. This script parses the JSON response from the API in to dialplan variables.

6.4.5 Number Discovery Evaluation

This work allows calls made to phone numbers both on the network (i.e. the number is registered) and non-existent (i.e. the number is made up). In order to determine if the system knows about a number, we perform a look up. The results from some example calls are presented in Listing 6.6 and Listing 6.7.

```

1      -- Executing [64212742118@internal:1] Answer() in new stack
2      -- Executing [64212742118@internal:2] AGI("parse_json.py
      ,64212742118") in new stack
3      -- Launched AGI Script /var/lib/asterisk/agi-bin/parse_json.py
4      -- AGI Script parse_json.py completed, returning 0
5      -- Executing [64212742118@internal:3] NoOp("SUCCESS") in new
      stack
6      -- Executing [64212742118@internal:4] NoOp("IMSI =
      IMSI530011102137279") in new stack
7      -- Executing [64212742118@internal:5] NoOp("PhoneNumber =
      64212742118") in new stack
8      -- Executing [64212742118@internal:6] NoOp("InitiatedCalls =
      0") in new stack

```

```

9      -- Executing [64212742118@internal:7] NoOp("ReceivedCalls = 0")
      in new stack
10     -- Executing [64212742118@internal:8] NoOp("Score = 0") in new
      stack
11     -- Executing [64212742118@internal:9] GotoIf("1?found") in new
      stack
12     -- Executing [64212742118@internal:12] Goto(internal,
      IMSI530011102137279,1) in new stack

```

Listing 6.6: Asterisk console log of a call to a mapped number

In this example, we are calling the number 64212742118, which has been associated with IMSI530011102137279. The `parse_json.py` AGI script is able to get the information from the Cassandra database, and write the variables into the dialplan. This is shown by the sequence of `Noop` lines in the middle, these lines are displaying the value of dialplan variables. Once the associated IMSI has been found, we redirect execution to the `internal` context to place the call as before.

```

1      -- Executing [64212742117@internal:1] Answer() in new stack
2      -- Executing [64212742117@internal:2] AGI("parse_json.py
      ,64212742117") in new stack
3      -- Launched AGI Script /var/lib/asterisk/agi-bin/parse_json.py
4      -- AGI Script parse_json.py completed, returning 0
5      -- Executing [64212742117@internal:3] NoOp("FAILURE") in new
      stack
6      -- Executing [64212742117@internal:4] NoOp("IMSI = ") in new
      stack
7      -- Executing [64212742117@internal:5] NoOp("PhoneNumber = ") in
      new stack
8      -- Executing [64212742117@internal:6] NoOp("InitiatedCalls = ")
      in new stack
9      -- Executing [64212742117@internal:7] NoOp("ReceivedCalls = ")
      in new stack
10     -- Executing [64212742117@internal:8] NoOp("Score = ") in new
      stack
11     -- Executing [64212742117@internal:9] GotoIf("0?found") in new
      stack
12     -- Executing [64212742117@internal:10] Festival("I'm sorry, but
      I cannot find the number 64212742117") in new stack
13     -- Executing [64212742117@internal:11] Hangup("") in new stack

```

Listing 6.7: Asterisk console log of a call to a non-mapped number.

6.5 User Verification

We propose that people verify each other's number after the call has completed (with the options of 'known', 'unknown', or 'not them'). Based on the evidence from systems such as eBay, as described in Chapter 5, we have decided to implement a simple count-based reputation system for the purpose of verification in our system.

The user verification information needs to be global to enable clients to move from one node to another without having to go through the setup process again (as would be the case if it were stored locally). We still store the SIP account information locally on each node (which only relies on the IMSI number and not the phone number). The SIP account information, except the IMSI number, is the same for every IMSI on the network.

For this part it is clearer if we describe the call flow and rating system using a concrete pair of users, Alice and Bob. In this scenario Alice calls Bob. After the call has finished, regardless of who finished the call the system calls Alice and asks her if Bob's number is pointing to the correct person.

The verification algorithm is shown in Listing 6.8, and has been extended from Listing 6.5.

```
1  if the number dialed contains only digits
2      lookup the IMSI for the number dialed in the database
3      if there is a result
4          dial the IMSI, on hangup continue executing this dialplan
5          increase received calls for recipient
6          increase initiated calls for caller
7      else
8          error message
9          exit
10
11     dial the caller
12     if the recipient was expected
13         increase the recipient's score
14     else
15         decrease the recipient's score
16     exit
17
18 if the number dialed begins with IMSI
19     find the device's address using DUNDi
20     dial that address
21 else
```

```
22     error message
23     exit
```

Listing 6.8: Algorithm for handling the users reputations.

Asterisk’s Dial application has option ‘flags’ that control what happens when a call has finished. There are two possibilities in this situation: Alice hangs up first or Bob does. If Bob hangs up first then we can continue execution of the dialplan by using the **F** option (capitalisation is important as the **f** option modifies the caller-ID in certain situations).

F: Proceed with dialplan execution at the next priority in the current extension if the source channel hangs up.

If Alice hangs up first then we perform the same step, but the option for this is **g** (again the capitalisation matters the **G** option does something different).

g: Proceed with dialplan execution at the next priority in the current extension if the destination channel hangs up.

Both these options accept parameters describing the priority, context, and extension to continue execution.

Once the call has progressed passed the **Dial** application we run an AGI script. This script generates a **call file** which is a plain text file that Asterisk reads to initiate the call to Alice asking if she was talking to Bob.

Having the verification take place in a separate call allows us to control if or when we ask Alice. If many people confirm that Bob is who he says he is, then there is no need to contact Alice. The AGI script controls the threshold value, in its present state it is set to three calls and it will accept multiple verifications by the same person for the same person. This threshold value will need to be configured and experimented with to find the optimal setting to prevent abuse.

6.5.1 Database Schema and Programmer Interface

- IMSI
- phone number
- score

- number of received calls
- number of initiated calls

We extend the schema from the work presented in Section 6.4 to include the score (or reputation), number of received calls by this IMSI, and number of initiated calls made by this IMSI. The programmer interface remains practically unchanged, with added calls to update the score, received calls, and initiated calls.

6.5.2 Evaluation

The `rate.py` AGI script generates a call file using the parameters passed to it. A call file allows us to specify parameters of a call in a plain text file that Asterisk can then process.

There are two legs for every call. The first leg in this case is back to the caller or the original call, we use the existing `[internal]` context for this (so that we get all the lookups and so on). The second leg is to a context, `rate`, that handles the logic for the reputation of the original callee.

Listing 6.9 shows a call from IMSI530240100139367 to IMSI530011102137279. This was recorded from IMSI530011102137279's 'home' node. Referring back to Figure 6.4, we are on the right hand node.

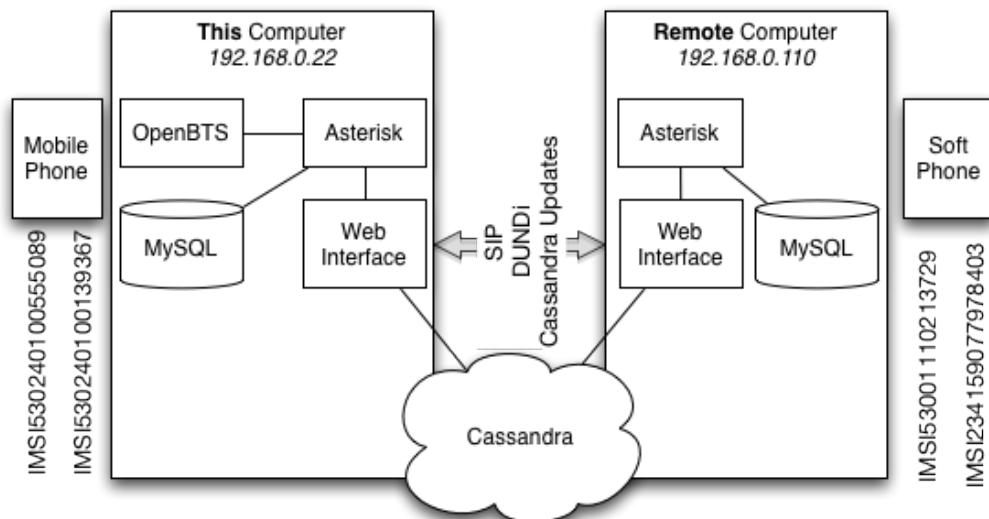


Figure 6.5: Reputation and Call Flow Experiment setup.

```

1      -- Executing [IMSI530011102137279@internal:1] NoOp("
      IMSI530011102137279") in new stack
2      -- Executing [IMSI530011102137279@internal:2] Answer("") in new
      stack
3      -- Executing [IMSI530011102137279@internal:3] Set("_from=
      IMSI530240100139367") in new stack
4      -- Executing [IMSI530011102137279@internal:4] Set("_to=
      IMSI530011102137279") in new stack
5      -- Executing [IMSI530011102137279@internal:5] Dial("SIP/
      IMSI530011102137279,,gF(internal~IMSI530011102137279^
      finished)") in new stack
6      -- Called SIP/IMSI530011102137279
7      -- SIP/IMSI530011102137279-0000004c is ringing
8      -- SIP/IMSI530011102137279-0000004c answered
9      -- Executing [IMSI530011102137279@internal:6] NoOp("Call has
      finished, wait a second before updating the score") in new
      stack
10     -- Executing [IMSI530011102137279@internal:7] Wait("1") in new
      stack
11     -- Executing [IMSI530011102137279@internal:8] GotoIf("0?end")
      in new stack
12     -- Executing [IMSI530011102137279@internal:9] AGI("rate.py,
      IMSI530240100139367,IMSI530011102137279") in new stack
13     -- Launched AGI Script /var/lib/asterisk/agi-bin/rate.py
14     -- Attempting call on Local/IMSI530240100139367@rate/n for
      IMSI530011102137279@rate:1
15     -- AGI Script rate.py completed, returning 0
16     -- Executing [IMSI530011102137279@internal:10] NoOp("Increasing
      received calls for IMSI530011102137279") in new stack
17     -- Executing [IMSI530011102137279@internal:11] System("/usr/bin
      /curl -X PUT http://192.168.0.22:8080/api/account/
      IMSI530011102137279/ -d '{"data": [{"received_calls": "up
      "}]}'") in new stack
18     -- Executing [IMSI530011102137279@internal:12] NoOp("Increasing
      initiated calls for IMSI530240100139367") in new stack
19     -- Executing [IMSI530011102137279@internal:13] System("/usr/bin
      /curl -X PUT http://192.168.0.22:8080/api/account/
      IMSI530240100139367/ -d '{"data": [{"initiated_calls": "up
      "}]}'") in new stack
20     -- Executing [IMSI530011102137279@internal:14] Hangup("") in
      new stack
21     == Spawn extension (internal, IMSI530011102137279, 14) exited non

```

Listing 6.9: Asterisk console log of a call with reputation

The first four lines are answering the call coming from the other server and setting some channel variables (`_from` and `_to`—in this case they are global variables). We then call the device. The options after (that is the `gF(...)` options mentioned above) describe what to do when either end of the call hangs up. In this case, we want the dialplan to continue executing. In the next step we log the call as it finishes and wait one second before asking the caller to rate the callee. This spawns a new call and we complete this call by updating statistics through the web API. The spawned call proceeds as in Listing 6.10.

```
1      -- Called SIP/trunk/IMSI530240100139367
2      -- Executing [IMSI530011102137279@rate:1] NoOp("
      IMSI530011102137279") in new stack
3      -- Executing [IMSI530011102137279@rate:2] Answer("") in new
      stack
4      -- Executing [IMSI530011102137279@rate:3] Wait("1") in new
      stack
5      -- Executing [IMSI530011102137279@rate:4] NoOp("
      IMSI530240100139367 rates IMSI530011102137279") in new stack
6      -- Executing [IMSI530011102137279@rate:5] Set("count=0") in new
      stack
7      -- Executing [IMSI530011102137279@rate:6] Festival("Was the
      person you called the person you expected? 1 for yes, 2 for
      no and 3 is unsure") in new stack
8      -- Executing [IMSI530011102137279@rate:7] Set("count=1") in new
      stack
9      -- Executing [IMSI530011102137279@rate:8] Read("rating,,1") in
      new stack
10     -- Accepting a maximum of 1 digits.
11     -- User entered '1'
12     -- Executing [IMSI530011102137279@rate:9] GotoIf("1?yes") in
      new stack
13     -- Goto (rate,IMSI530011102137279,14)
14     -- Executing [IMSI530011102137279@rate:14] NoOp("Increasing
      score for IMSI530011102137279") in new stack
15     -- Executing [IMSI530011102137279@rate:15] System("/usr/bin/
      curl -X PUT http://192.168.0.22:8080/api/account/
      IMSI530011102137279/ -d '{"data": [{"score": "up"}]}') in
      new stack
```

```

16      -- Executing [IMSI530011102137279@rate:16] Goto("end") in new
      stack
17      -- Goto (rate,IMSI530011102137279,21)
18      -- Executing [IMSI530011102137279@rate:21] NoOp("") in new
      stack
19      -- Executing [IMSI530011102137279@rate:22] Festival("Thanks")
      in new stack
20      -- Executing [IMSI530011102137279@rate:23] Hangup("") in new
      stack
21  == Spawn extension (rate, IMSI530011102137279, 23) exited non-
      zero

```

Listing 6.10: Asterisk console log of a call with rating

We **Answer** the call and wait a second (to ensure that the audio is ready) before asking the original caller to rate the callee. We will only perform this three times (hence the need for the count variable). The user's rating is read in to a variable and using this variable we determine which action to take. In this example, the original caller has confirmed that the callee is who they were expecting. We update the score (once again, using the web API), thank the user, and hangup. The full code listing is available in the Appendix B.1.1.

The ratings are handled by the `rate.py` AGI script (see Appendix B.4.2). In its present state as soon as a callee has a rating of more than three we no longer generate the call file. We can also use this same script to provide the removal of association to allow the system to reclaim wrongly assigned phone numbers.

6.6 Discussion

6.6.1 Abuse

The proposed system has some potential flaws as described below.

Firstly, if we sum the feedback then we can have the situation that a person with dozens of transactions who has lied one out of every four cases will have a higher reputation than someone who has performed only a limited number of transactions (Xiong and Liu, 2004b).

Secondly, we do not track who has rated whom. This means that a user could attack another by constantly voting in the negative, or vice versa, a group of friends could all continually vote positively for someone who does not own the claimed number. This is

where the Serval Project differs. They form a web of trust by using digital certificates (Gardner-Stephen, 2011). Given that they distribute their database around their clients it is possible that those databases become corrupted and thus open to abuse. Our system is simple, and while there are potential flaws, the exact method that is used can be changed and adjusted at will.

Thirdly this method will not work for emergency services as they may be using multiple staff members to answer calls. For example, when calling the police, you do not personally know the recipient of the call. Setting static routes for the number (e.g. send the police's number to the call center via a trunk) or restricting the pattern of numbers that the system accepts would counter this type of abuse.

6.6.2 Unique Contribution

The novelty in this component is that from the user's point of view the new network behaves as if it were a traditional GSM network. Due to the ubiquity of cell phones users expect certain behaviour from them as soon as the behaviour changes it causes confusion.

Our system requires no need for a centralised database mapping between phone numbers and IMSI. This allows us to experiment with a fault tolerant network. In a disaster this is vital as the effects are unpredictable, and vital infrastructure may be affected even after it has been repaired or replaced.

We use a simple model of trust to verify that the users of the system are who they claim to be. This trust model can be modified and adjusted as needed to prevent abuse.

6.6.3 Future Work

1. Calls cannot be received from outside the network. There are two ways to solve this, firstly, integrate with existing telecommunications providers, and update their records from our network. Secondly, provide a public number that people outside the network can dial. Once the call is answered the system then asks them to dial the number they wish to reach. This is similar to having a manual operator connecting calls. This mechanism is supported in Asterisk by using the Direct Inward System Access (DISA) command (Van Meggelen *et al.*, 2007).
2. Xiong and Liu (2004a) describe how the verification system used by eBay (positive, neutral, negative votes, and a simple summation of those votes) is open

to abuse. We have implemented that system here and thus it will be open to the same abuses as the eBay system. One potential abuse is where a group of friends constantly verify each other. Disasters are a time when people are at their most vulnerable and where people take advantage of others. Future work could include recording other aspects and finding ways around the abuses described above. Alternatively migrate to a scheme similar as used by the Serval Project (using digital certificates).

3. Add in support for calling of numbers outside the network via advertising the route via the distributed number lookup, as well as provide support for dynamic dialplan configuration. Also with the ability to make those changes either network wide, or on a per node basis.
4. Add support for keeping track of who rates whom and only allowing one person to rate another once per phone number.
5. Add support for monitoring/reporting mechanisms (like top callers, utilisation, etc.).
6. Ability to make the rating scheme customisable (like change when the verification call takes place, i.e. the thresholds/functions that decide when to place the call), and should be network wide, not just per-node.
7. Add security to the network, especially this distributed database we simply assume that nodes in the database cluster are 100% trusted. This may not be the case however. An attacker could join the cluster and change any of the data in the distributed database. This would enable them to claim any number that they desired, or to spoof the reputation data so that none of the legitimate users can use the system.
8. Capacity testing, ensuring that the distributed database can keep up with the adding/removing of extra nodes as well as all the read/write traffic. Given that Cassandra was developed and is used by large websites we can be safe in the knowledge that it is possible to get Cassandra performing well.
9. Add in support for more complex operations — like synchronising the dialplan across multiple nodes or distributing resources (like sound files) across the network. One possible method is to develop a web-based interface to Asterisk's RealTime database, and use the same API that provides IMSI, phone numbers,

and reputation access. This would allow the people managing the disaster to provide customised messages, routing as needed.

Chapter 7

Conclusions and Future Work

7.1 Conclusion

In Chapter 2 we have shown that disasters have a large effect on populations regardless of the type of the disaster. There are only a small number of major disasters in a given year, but there can be a larger number of smaller ones. Even those small disasters have a large affect on the populations in the effected area.

Communications are vital not only to the management of disasters but also to civilians in the effected area. If the telecommunications infrastructure is unavailable, either through damage or congestion, then communications between different response groups and civilians is hindered.

One possible solution to re-establish communications in the event of a disaster is to use Rapidly Deployable Networks. An RDN is a network that can be set up in a short amount of time to alleviate congestion or to replace damaged portions of the network.

Rapidly deployable networks exist in a commercial setting but require road infrastructure to be deployed. The road systems can also be affected by a disaster, thus rendering the commercial rapidly deployable networks unavailable.

We investigated different mesh networks, each of which had different characteristics. We found that it is possible to route voice traffic and provide services (like Geographic Information System (GIS) disaster management systems) over mesh networks. Existing telephony services are available using the mesh network as an access medium. There are two projects, one uses analogue telephones and the other smart phones. Because smart phones are physically mobile they require custom protocols to deal with the complexities introduced by the node's mobility. Both of the analogue and smart phone projects require either special hardware, or, modifications to the phones in order to

provide the telephony over mesh networking. In a disaster it is unlikely that civilians will be able to acquire either the special hardware or, for those with smart phones, to modify their phones to join the network.

In this thesis we used basic GSM mobile phones to access the telephony services provided by the mesh network. These devices are common and more easily accessible, from a civilian's point of view, than smart phones or the hardware needed to modify the analogue telephones.

The phones are used to access a distributed telephony network operating on top of a rapidly deployable mesh network. This network can replace, or augment, the existing network infrastructure, in the case of severe damage or to ease congestion.

In order to be useful to civilians a system providing replacement telephony services needs to support dialing of telephone numbers. Our system asks the users to claim their own telephone numbers. In doing this, the users of the system need to be sure that they are talking to the right person. We use a reputation system, following in the footsteps of eBay, to provide feedback about the recipients of the calls.

This feedback feeds into a simple trust model and from here it is possible to build trust over time. The feedback takes the form of positive, negative, or unknown ratings made by the caller about the person they called. This scheme works because there is a prior association between phone number and a person. The feedback is aggregated to form a trust rating. After a certain threshold value the callers are not asked to rate the recipient as it is deemed that the recipient is who they claim to be.

Through experimentation, we have demonstrated the proposed distributed telephony service works with GSM access method (which differs from previous work).

The experiments used a full custom-built prototype system with the following components at each node:

- Asterisk, for call routing
- Cassandra, for distributed reputation storage
- MySQL, for dynamic creation of users in the system
- Django, for basic administration and API functions
- OpenBTS, for GSM access

During the experiments, the nodes were then connected via a wired network to emulate the mesh network. The list below covers a summary of our experimental results:

Aspect	Serval Project	Beacon
Device	Smart Phone	GSM Phone
Access Technology	WiFi	GSM and WiFi
Traffic Routing	BATMAN	-
Number Discovery	Derived from IP address	DUNDi
Backbone Protocol	SIP	SIP
User Identifier	Subscriber ID	IMSI
User Verification	Digital Certificates	Counter

Table 7.1: Key differences between our work and the Serval Project

- unmodified, consumer-grade mobile phones can see and connect to test network
- those phones can make/receive calls on same node
- make/receive calls on remote node
- distributed method to locate devices on the network
- distributed database linking users to phone numbers
- reputation system like eBay's for user management

Using a system such as that prototyped in this thesis should enable the managers of the disaster to more effectively aid the civilian populations. Even if they were to use this type of technology to provide a common system for their own communications it could eliminate some of the mis-communications associated with relaying information.

The existing commercial systems are highly dependant on the availability of transportation infrastructure in order to be useful. Once the system described in this work becomes more mature, we envisage a situation where it could fit in someone's backpack and be taken by foot and be deployed in to a disaster area.

7.2 Unique Contribution

The closest existing work is the Serval Project (Gardner-Stephen, 2011) which is a derivative of the Village Telco. Table 7.1 highlights the differences between our work and the Serval Project's. While Table 7.2 describes the enhancements.

We have also presented a new environment for an existing trust model. Following eBay's rating scheme we supplied a reputation-based mapping between user-provided

Aspect	Description
Device	Requires no modification of consumer grade equipment.
Access Technology	Any GSM compatible phone works with the system.
Number Discovery	Unlimited pool of numbers (bounded only by the memory of the network).
User Verification	Simple, and able to be extended or adjusted as needed.

Table 7.2: Enhancements of our work over the Serval Project

phone numbers and the users of the system. Because the users supply their own phone numbers, in contrast to system-allocated numbers, they need to be sure that they are talking to the correct person. Users rate others at the end of a phone call confirming, or not, that the person is who the caller expected them to be. Overtime these ratings give an indication of how trustworthy each user is in the network. Once a user has reached a certain level of trust they no longer need to be rated.

7.3 Future Work

7.3.1 Overview

1. Place the distributed telephony services on top of a proper ad-hoc WiFi network. We have only tested on a wired network. We anticipate that there will not be any significant problems due to the use of standard network protocols.
2. Once this technology becomes more mature the network would need to be field tested, miniaturised, and optimised for battery life (and possible other targets).
3. Additional services would need to be implemented in order to be useful in a disaster, Table 7.3 lists a selected set of useful features summarised from the literature.

7.3.2 Rapidly Deployable Network

1. Use a network simulator to gather data on the resiliency and scalability of the network.

Feature	Description
Grouping	Group known types of responders (fire service, police, etc.) and provide group messaging
Information	Automated systems to help with providing assistance (e.g. first aid self-help directions)
Reporting	Provide a mechanism to allow civilians to report incidents, and integrate with other projects
Monitoring	Monitoring the status of the hardware, radio signals etc.

Table 7.3: Proposed additional features.

2. Investigate the different topologies for the nodes in the network, in other words, as a standalone node (where calls are routed to the existing telephone network), or as a repeater node (to extend the coverage of the mesh network).
3. We have not tested the amount of bandwidth needed for calls or other traffic. This means that while others have shown that telephony on mesh networks is possible, we are unsure quite how our custom-built prototype network will perform on a real mesh network.
4. In late October 2011, a new version of the OpenBTS software was released. This changes the way that the USRP hardware is configured. A database is now used to store settings and so most of the options are now dynamic and can be adjusted on a running system without having to restart the GSM network. This allows for system administrators to tune the performance of the network without the users losing connectivity, which is a bonus when compared to the version that we used.
5. We used a distributed number lookup to perform the automatic routing of calls. This distributed system's recommended setup requires a core network with fail-over and duplication. In our network if a node goes down (due to lack of power, physical damage, or network outage, etc.), then there is no backup for that node's state. This means that all the information about how to route calls to mobiles connected to (and via) that node would not be available. However, if a node goes down then those people attached to that node will not be able to use their phones anyway.
6. Automating the addition and removal of the node to the mesh network would be another improvement. This requires generation and distribution of a set of

cryptographic keys, as well as some additional configuration for the distributed number lookups. We envisage a mechanism that allows remote nodes to query this information from other nodes in the network—much like zero-conf networking systems.

7. We would also like to automate the addition of users accounts via OpenBTS for the open network setup. Instead of relying on creation of accounts on the fly by administrators.

7.3.3 User Management

1. Xiong and Liu (2004a) describe how the verification system used by eBay (positive, neutral, negative votes, and a simple summation of those votes) is open to abuse. We have implemented that system here and thus it will be open to the same abuses as the eBay system. One potential abuse is where a group of friends constantly verify each other. Disasters are a time when people are at their most vulnerable and where people take advantage of others. Future work could include recording other aspects and finding ways around the abuses. Alternatively migrate to a scheme similar as used by the Serval Project (using digital certificates).
2. Because our system replaces the existing telecommunications network, and the phone numbers are only advertised within our network. Calls from outside our network will be routed via the existing telecommunications systems, and not into our network. This means that calls from outside the network will not terminate into our network. This can be solved by providing public number that people outside the network can dial. Once the call is answered the system then asks them to dial the number they wish to reach. This is similar to having a manual operator connecting calls. There is a mechanism in Asterisk to support this functionality.
3. Add in support for calling of numbers outside the network via advertising the route via the distributed number lookup, as well as provide support for dynamic dialplans. Also with the ability to make those changes either network wide, or on a per node basis.
4. Add support for keeping track of who rates whom and only allowing one person to rate another once per phone number.

5. Add support for monitoring/reporting mechanisms (like top callers, utilisation, etc.).
6. Ability to make the rating scheme customisable (like change when the verification call takes place, i.e. the thresholds/functions that decide when to place the call), and should be network wide, not just per-node.
7. Add security to the network, especially this distributed database we simply assume that nodes in the database cluster are 100% trusted. This may not be the case however. An attacker could join the cluster and change any of the data in the distributed database. This would enable them to claim any number that they desired, or to spoof the reputation data so that none of the legitimate users can use the system.
8. Capacity testing, ensuring that the distributed database can keep up with the adding/removing of extra nodes as well as all the read/write traffic. Given that Cassandra was developed and is used by large websites we can be safe in the knowledge that it is possible to get Cassandra performing well.
9. Add in support for more complex operations — like synchronising the dialplan across multiple nodes or distributing resources (like sound files) across the network. One possible method is to develop a web-based interface to Asterisk's RealTime database, and use the same API that provides IMSI, phone numbers, and reputation access. This would allow the people managing the disaster to provide customised messages, routing as needed.

References

- 3rd Generation Partnership Project (3GPP) (2010a). Technical Specification Group GSM/EDGE Radio Access Network; Radio transmission and reception. Technical report, 3rd Generation Partnership Project (3GPP).
- 3rd Generation Partnership Project (3GPP) (2010b). Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS) aspects of architecture for Home Node B (HNB). Technical report, 3rd Generation Partnership Project (3GPP).
- Abadi, M., Burrows, M., and Needham, R. (1990). A logic of authentication. *Proceedings of the Royal Society*, 426(1871), 233–271.
- Abusch-Magder, D., Bosch, P., Klein, T., Polakos, P., Samuel, L., and Viswanathan, H. (2007). 911-NOW: A network on wheels for emergency response and disaster recovery operations. *Bell Labs Technical Journal*, 11(4), 113–133.
- Addams-Moring, R., Kekkonen, M., and Zhao, S. (2005). A Simple Taxonomy for Mobile Emergency Announcement Systems. In *Proceedings of the 2nd International ISCRAM Conference*.
- Adeyeye, M. and Gardner-Stephen, P. (2011). The Village Telco project: a reliable and practical wireless mesh telephony infrastructure. *EURASIP Journal on Wireless Communications and Networking*, 2011(1), 78.
- Agency, J. N. P. (2011). Damage Situation and Police Countermeasures associated with 2011 Tohoku district— off the Pacific Ocean Earthquake.
- Akyildiz, I., Wang, X., and Wang, W. (2005). Wireless mesh networks: a survey. *Computer Networks*, 47(4), 445–487.
- Albitz, P. and Liu, C. (2001). *DNS and Bind*. O’Reilly Media.

- Angle, N. (2011). Help still needed after record-breaking year for disasters.
- Banchs, A., Bayer, N., Chieng, D., de la Oliva, A., Gloss, B., Kretschme, M., Murphyk, S., Natkaniec, M., and Zdarsky, F. (2008). Carmen: Delivering carrier grade services over wireless mesh networks. In *IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications*, 1–6. IEEE.
- Bilham, R. (2010). Lessons from the Haiti earthquake. *Nature*, 463(7283), 878–879.
- Blossom, E. (2004). GNU radio: tools for exploring the radio frequency spectrum. *Linux Journal*, 2004(122), 4.
- Brainard, J., Juels, A., Rivest, R., Szydlo, M., and Yung, M. (2006). Fourth-factor authentication: somebody you know. In *Proceedings of the 13th ACM conference on Computer and communications security*, 168–178. ACM.
- Bram, J., Orr, J., and Rapaport, C. (2002). *Measuring the effects of the September 11 attack on New York City*. National Emergency Training Center.
- Braunstein, B., Trimble, T., Mishra, R., Manoj, B., Lenert, L., and Rao, R. (2006). Challenges in using distributed wireless mesh networks in emergency response. In *Proceedings of the 3rd International ISCRAM Conference*, 30–38.
- Bryant, S., Forte, A., and Bruckman, A. (2005). Becoming Wikipedian: transformation of participation in a collaborative online encyclopedia. In *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, 1–10. ACM.
- Burgess, D. (2011). Range Networks OpenBTS™Public Release. <https://wush.net/trac/rangepublic>. [Online; accessed 01-December-2011].
- Burgess, D. (2012). Adding Authentication and Encryption to the Public Release. <http://wush.net/trac/rangepublic/wiki/A3A8A5>. [Online; accessed 10-April-2012].
- Burgess, D. and Samra, H. (2008). The OpenBTS Project. www.ahzf.de/itstuff/papers/OpenBTSProject.pdf. [Online; accessed 14-December-2011].
- Burgess, D. A. (2009). The OpenBTS Chronicles. <http://openbts.blogspot.com/>. [Online; accessed 06-December-2011].
- Cannon, R. (2001). ENUM: The Collision Of Telephony And DNS Policy. Working paper series.

- Chandrasekhar, V., Andrews, J., and Gatherer, A. (2008). Femtocell networks: a survey. *Communications Magazine, IEEE*, 46(9), 59–67.
- Chen, B. X. (2010). Man Buried in Haiti Rubble Uses iPhone to Treat Wounds, Survive. <http://www.wired.com/gadgetlab/2010/01/haiti-survivor-iphone/>. [Online; accessed 04-November-2011].
- Cisco (2011). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010-2015. Technical report, Cisco. [Online; accessed 04-November-2011].
- Digium (2010). Asterisk - The Open Source Telephony Projects — Asterisk. <http://www.asterisk.org/>. [Online; accessed 01-December-2011].
- Eberspächer, I. and Vögel, H. (1998). *GSM—Global System for Mobile Communication. Switching, Services and Protocols*. Chichester: John Wiley & Sons.
- Ericsson (2009). Self-Organizing Network solution tested in live LTE Network. http://www.ericsson.com/news/101119_lte_network_244218599_c. [Online; accessed 10-April-2012].
- Eshghi, K. and Larson, R. (2008). Disasters: lessons from the past 105 years. *Disaster Prevention and Management*, 17(1), 62–82.
- Evans, J., Minden, G., Shanmugan, K., Prescott, G., Frost, V., Ewy, B., Sanchez, R., Sparks, C., Malinimohan, K., Roberts, J., *et al.* (1999). The rapidly deployable radio network. *IEEE Journal on selected areas in communications*, 17(4), 689–703.
- Faltstrom, P. (2000). E.164 number and DNS. RFC 2916 (Proposed Standard). Obsoleted by RFC 3761.
- Fernandes, J. P. (2008). Emergency Warnings With Short Message Service. In H. G. Coskun, H. K. Cigizoglu, and M. D. Maktav (Eds.), *Integration of Information for Environmental Security*, NATO Science for Peace and Security Series C: Environmental Security, 191–196. Springer Netherlands.
- Flanagan, P. (2011). Reflections on the Christchurch earthquake. *ISBT Science Series*, 6(2), 350–353.
- Gao, H., Barbier, G., and Goolsby, R. (2011). Harnessing the Crowdsourcing Power of Social Media for Disaster Relief. *Intelligent Systems, IEEE*, 26(3), 10–14.

- Gardner-Stephen, P. (2011). The serval project: practical wireless ad-hoc mobile telecommunications. http://developer.servalproject.org/files/CWN_Chapter_Serval.pdf. [Online; accessed 14-December-2011].
- Gast, M. (2005). *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media.
- Ghobarah, A., Saatcioglu, M., and Nistor, I. (2006). The impact of the 26 December 2004 earthquake and tsunami on structures and infrastructure. *Engineering Structures*, 28(2), 312–326.
- Goodchild, M. and Glennon, J. (2010). Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3(3), 231–241.
- Hewitt, E. (2010). *Cassandra: The Definitive Guide*. O'Reilly Media, Inc.
- Horwich, G. (2000). Economic lessons of the Kobe earthquake. *Economic Development and Cultural Change*, 48(3), 521–542.
- Huston, G. (2002). ENUM–Mapping the E. 164 Number Space into the DNS. *The Internet Protocol Journal*, 5(2), 13–23.
- Iannella, R. and Henriksen, K. (2007). Managing Information in the Disaster Coordination Centre: Lessons and Opportunities. In P. B. B. Van de Walle and C. Nieuwenhuis (Eds.), *Proceedings of the 4th International ISCRAM Conference*.
- ITU (2010). The international public telecommunication numbering plan. Technical report, International Telecommunication Union. Recommendation ITU-T E.164.
- Jang, H., Lien, Y., and Tsai, T. (2009). Rescue information system for earthquake disasters based on MANET emergency communication platform. In *Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing: Connecting the World Wirelessly*, 623–627. ACM.
- Johnson, D., Ntlatlapa, N., and Aichele, C. (2008). Simple pragmatic approach to mesh routing using BATMAN. In *IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries*.
- Kostrzewa, A. (2011). Development of a man in the middle attack on the GSM Um-Interface.

- Kretschmer, M., Hasse, P., Niephaus, C., Horstmann, T., and Jonas, K. (2011). Connecting mobile phones via carrier-grade meshed wireless back-haul networks. In R. Popescu-Zeletin, I. A. Rai, K. Jonas, A. Villafiorita, O. Akan, P. Bellavista, J. Cao, F. Dressler, D. Ferrari, M. Gerla, H. Kobayashi, S. Palazzo, S. Sahni, X. S. Shen, M. Stan, J. Xiaohua, A. Zomaya, and G. Coulson (Eds.), *E-Infrastructures and E-Services for Developing Countries*, Volume 64 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 1–10. Springer Berlin Heidelberg.
- Kwasinski, A., Weaver, W., Chapman, P., and Krein, P. (2009). Telecommunications Power Plant Damage Assessment for Hurricane Katrina—Site Survey and Follow-Up Results. *Systems Journal, IEEE*, 3(3), 277–287.
- Laforge (2011). Howto OpenBSC with Asterisk and LCR. http://openbsc.osmocom.org/trac/wiki/OpenBSC_LCR. [Online; accessed 01-December-2011].
- laforge (2011). Welcome to OpenBSC. <http://openbsc.osmocom.org/trac/>. [Online; accessed 01-December-2011].
- Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2), 35–40.
- Li, X. (2008). *Wireless Ad-hoc and Sensor Networks: Theory and Applications*. Cambridge: Cambridge University Press.
- Manoj, B. and Baker, A. (2007). Communication challenges in emergency response. *Communications of the ACM*, 50(3), 51–53.
- McEntire, D. (1999). Issues in disaster relief: progress, perpetual problems and prospective solutions. *Disaster Prevention and Management*, 8(5), 351–361.
- McGinley, M., Turk, A., and Bennett, D. (2006). Design criteria for public emergency warning systems. In *Proceedings of the 3rd International ISCRAM Conference*, 154–163. Newark, NJ (USA).
- Mehrotra, A. (1997). *GSM System Engineering*. Boston: Artech House.
- Midkiff, S. and Bostian, C. (2002). Rapidly-deployable broadband wireless networks for disaster and emergency response. In *Presented at The First IEEE Workshop on Disaster Recovery Networks (DIREN 02)*. Citeseer.

- Murthy, C. and Manoj, B. (2004). *Ad hoc wireless networks: architectures and protocols*. Prentice Hall PTR.
- National, D. L. and Libes, D. (1996). Authentication by email reception. In *Proceedings of the Fifth System Administration, Networking, and Security Conference (SANS 96)*, 12–18.
- Nigg, J., Barnshaw, J., and Torres, M. (2006). Hurricane Katrina and the flooding of New Orleans: Emergent issues in sheltering and temporary housing. *The Annals of the American Academy of Political and Social Science*, 604(1), 113–128.
- Noam, E. M. and Sato, H. (1995). Kobe’s lesson: Dial 711 for ‘open’ emergency communications. *Telecommunications Policy*, 19(8), 595 – 598.
- Norris, F., Galea, S., Friedman, M. J., and Watson, P. J. (Eds.) (2006). *Methods for disaster mental health research*. New York: Guilford.
- Norris, F., Stevens, S., Pfefferbaum, B., Wyche, K., and Pfefferbaum, R. (2008). Community resilience as a metaphor, theory, set of capacities, and strategy for disaster readiness. *American Journal of Community Psychology*, 41(1), 127–150.
- OASIS Emergency Management Technical Committee (2010). Common Alerting Protocol Version 1.2. <http://docs.oasis-open.org/emergency/cap/v1.2/CAP-v1.2-os.pdf>. [Online; accessed 04-November-2011].
- OCDEMG (2004). Otago Civil Defence Emergency Management Group Plan. <http://www.orc.govt.nz/Publications-and-Reports/Civil-Defence-/Otago-Civil-Defence-Emergency-Management-group-Plan/>. [Online; accessed 04-November-2011].
- Okolloh, O. (2009). Ushahidi, or ‘testimony’: Web 2.0 tools for crowdsourcing crisis information. *Participatory Learning and Action*, 59(1), 65–70.
- Ozsu, M. and Valduriez, P. (2011). *Principles of Distributed Database Systems*. Upper Saddle River, NJ, USA: Prentice Hall.
- Perkins, C. (Ed.) (2001). *Ad-hoc Networking*. Addison-Wesley.
- Perkins, D. and Hughes, H. (2002). A survey on quality-of-service support for mobile ad hoc networks. *Wireless Communications and Mobile Computing*, 2(5), 503–513.

- Police, N. Z. (2011). List of Deceased.
- Quarantelli, E. (1985). What is disaster? The need for clarification in definition and conceptualization in research. In B. J. Sowder (Ed.), *Disasters and Mental Health: Selected Contemporary Perspectives*. Rockville, MD: National Institute of Mental Health.
- Raivio, Y. and Addams-Moring, R. (2006). Mobile Emergency Announcements with Really Simple Syndication (RSS 2.0). In B. Van de Walle and M. Turoff (Eds.), *Proceedings of the 3rd International ISCRAM Conference*.
- Razzaque, M., Dobson, S., and Nixon, P. (2010). Enhancement of Self-organisation in Wireless Networking through a Cross-layer Approach. *Ad Hoc Networks*, 144–159.
- Resnick, P., Kuwabara, K., Zeckhauser, R., and Friedman, E. (2000). Reputation systems. *Communications of the ACM*, 43(12), 45–48.
- Richardson, J. R. (2007). DUNDi, So Easy A Caveman Could Do It! http://www.voip-info.org/storage/users/813/47813/images/1654/DUNDi_So_Easy.pdf. [Online; accessed 01-December-2011].
- Rodríguez, H. and Aguirre, B. (2006). Hurricane Katrina and the healthcare infrastructure: A focus on disaster preparedness, response, and resiliency. *Frontiers of Health Services Management*, 23(1), 13.
- Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. (2002). SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard). Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.
- Rotherham, F. (2011). Quake rebuild will eat into GDP.
- Sanchez, R., Evans, J., Minden, G., Frost, V., and Shanmugan, K. (1998). RDRN: a prototype for a rapidly deployable radio network. *ACM SIGMOBILE Mobile Computing and Communications Review*, 2(2), 15–22.
- Sattler, D., Preston, A., Kaiser, C., Olivera, V., Valdez, J., and Schlueter, S. (2002). Hurricane Georges: A cross-national study examining preparedness, resource loss, and psychological distress in the US Virgin Islands, Puerto Rico, Dominican Republic, and the United States. *Journal of Traumatic Stress*, 15(5), 339–350.

- Sicard, L., Markovics, M., and Manthios, G. (2010). An Ad-hoc Network of Android Phones Using BATMAN. "[Online; accessed 17-December-2011]".
- Spencer, M. (2004a). Distributed Universal Number Discovery (DUNDi). <http://www.dundi.com/dundi.txt>. [Online; accessed 01-December-2011].
- Spencer, M. (2004b). Distributed Universal Number Discovery (DUNDiTM) and the General Peering Agreement (GPATM). Technical report, Digium.
- Tang, A. (2011). Thailand Cleans Up; Areas Remain Flooded.
- Thompson, C. (2005). Talking in the dark.
- Tipparaju, V. (2000). Local Multipoint Distribution Service (LMDS).
- Toh, C. (2002). *Ad-hoc Mobile Wireless Networks: Protocols and Systems*, Volume 1. New Jersey: Prentice Hall.
- Valerio, D. (2008). Open Source Software-Defined Radio: A survey on GNU-radio and its applications. Technical report, Forschungszentrum Telekommunikation Wien, Vienna.
- Valtonen, E., Addams-Moring, R., Virtanen, T., Järvinen, A., and Moring, M. (2004). Emergency announcements to mobile user devices in geographically defined areas. In *Proceedings ISCRAM2004*, Volume 151.
- Van Meggelen, J., Madsen, L., and Smith, J. (2007). *Asterisk: The Future of Telephony*. 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America: O'Reilly Media.
- World, G. (2010). History, Brief history of GSM & the GSMA. <http://www.gsmworld.com/about-us/history.htm>. [Online; accessed 14-December-2011].
- Xiong, L. and Liu, L. (2004a). Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. *Knowledge and Data Engineering, IEEE Transactions on*, 16(7), 843–857.
- Xiong, L. and Liu, L. (2004b). Reputation and trust. *IEEE Transactions on knowledge and data engineering*, 16, 19.

Appendix A

Setup

This chapter introduces the necessary components needed to get phones registering with OpenBTS and able to make calls (to the `echotest` application).

We assume that there are the necessary build tools already installed (viz. `gcc` and the like).

These instructions compliment the OpenBTS for Dummies, by Axelle Apvrille <http://gnuradio.org/redmine/attachments/219/fordummies.pdf>.

A.1 MySQL

We could use any database system that we liked, but MySQL is available on many platforms, and already has a large support base within the Asterisk Community.

A.1.1 Installation

Feel free to install this via your favourite package management system.

In Debian:

```
# apt-get install mysql-client mysql-server
```

A.1.2 Configuration

See Appendix B.5.

A.2 Asterisk

A.2.1 Installation

The first steps are to compile and setup Asterisk. Download the sources from <http://www.asterisk.org>. We use version 1.6.13, and as long as there is RealTime support the exact version is mostly irrelevant. You will also need to download the appropriate `asterisk-addons`. Because we use MySQL for storing the user accounts, you will need to install the client libraries too.

To summerise:

- Download asterisk
- Download asterisk-addons
- Install mysql client
- Compile and install asterisk
- Compile and install asterisk-addons

In Debian, we'd execute the following commands:

```
$ wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/
  asterisk-1.8.7.2.tar.gz
$ wget http://downloads.asterisk.org/pub/telephony/asterisk/releases/
  asterisk-addons-1.6.2.3.tar.gz
# apt-get install libmysqlclient15
$ tar -xzvf *.tar.gz
$ cd asterisk-1.6.2.13
$ ./configure && make
# make install
# make samples
$ cd ../asterisk-addons-1.6.2.2
$ ./configure && make
# make install
# make samples
```

NB, the urls are broken over the end of the line and should appear with no spaces in order for the above commands to work.

In the `asterisk-addons` directory run `make menuselect`. If the installation finds the MySQL client libraries they will appear under `Resource Modules`.

A.2.2 Configuration

For this we need to edit a few files. Firstly, we need to define where the database resides. Edit `/etc/asterisk/res_mysql.conf`, and add the following lines:

```
[general]
dbhost = 127.0.0.1
dbname = asterisk_realtime
dbuser = asterisk
dbpass = some_password
dbport = 3306
requirements=warn
```

Now, we need to tell Asterisk where to find these sip accounts. Add the following to `/etc/asterisk/extconfig.conf`.

```
sipusers => mysql,general,sip_devices
sippeers => mysql,general,sip_devices
```

At this point we need to *reload* the configuration. To check that it's working, run *realtime mysql status*, the result is shown below.

```
general connected to asterisk\_realtime@127.0.0.1, port 3306 with
username asterisk for 5 hours, 35 minutes.
```

A.3 Boost

The boost libraries are needed. *OpenBTS for Dummies* suggests using version 1.4.4.

A.4 GNU Radio

This took a long time to compile. Part of the reason for this is that we wanted to run extra tools (to see what channels/frequencies were available). For a pure OpenBTS installation, these tools are probably not required.

The build instructions on the GNURadio website are a good starting point, <http://gnuradio.org/redmine/wiki/gnuradio/BuildGuide>.

On Debian, we need a few extra packages:

```
# apt-get install libortp7-dev python-dev swig libfftw3-dev \  
    libccpunit-dev libboost1.35-dev duile-1.8 sdcc-nf
```

In the *OpenBTS for Dummies* has the following command:

```
$ ./configure --disable-all-components --enable-usrp \  
    --enable-omnithread --enable-mblock \  
    --enable-pmt --enable-gnuradio-examples \  
    --enable-docs --enable-doxxygen \  
    --enable-gnuradio-core --enable-gr-wxgui \  
    --enable-gruel --enable-gr-utils \  
    --enable-gr-usrp --enable-gr-qtgui
```

Once this command has completed (you may need to manually find the Boost libraries). You can complete the make and make install without difficulty.

A.5 OpenBTS

There are a couple of items that we need to discuss before we get our hands dirty this time.

A.5.1 libosip

For SMS support you will need to compile this from source. It's easy enough to get going if you follow the instructions included with the source. If you find that it complains about not being able to find some libraries, they are likely as not in `/usr/local/lib` as opposed to `/usr/lib`.

An important note is the version required. The SMS application server needs at least version 3.3 or later. This library is not available in a Debian package.

A.5.2 Installation

Note: These instructions cover version 2.6 of the public release. Because there is a new version, these instructions are likely out of date.

We use the Subversion (SVN) version of OpenBTS as there was a bug in the tar archive. The developers left a hack in from the Nieuwe deployment, this meant that we were not able to register cellphones to the network.

```
svn co http://gnuradio.org/svn/openbts/trunk/ openbts
```

Once you have the source code by performing an SVN check out, you should be able to follow the build instructions from the OpenBTS wiki page <http://gnuradio.org/redmine/wiki/gnuradio/OpenBTSBuildingAndRunning>.

We found that you need to run `ldconfig` and specify the `PKG\_CONFIG\_PATH` as below:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

We also discovered that the HLR does not compile properly. To fix this, we descended into the directory and ran the `make` command from there. This solves the problem.

Do not forget that these applications (OpenBTS and `smqueue`) need to run as root.

A.6 Configuration

See Appendix B.2.

A.7 Provisioning

So, by now we should have `asterisk`, `asterisk-realtime`, `OpenBTS` all setup and ready to go. But, we need to provision the phones. Here's a sample insert statement to get you started. Note: this is not a real IMSI, you need to get this from the phone, or from watching the `asterisk` console — it will tell you that registration failed.

```
INSERT INTO sip_devices (  
    name,  
    context,  
    host,  
    callerid,  
    defaulttip,  
    disallow,
```

```

        allow
    ) VALUES (
        'IMSI123456789012345',
        'demo',
        '127.0.0.1',
        '1000',
        '127.0.0.1',
        'all',
        'gsm'
    );

```

This should get you started. Start asterisk, OpenBTS, and smqueue (in separate terminals).

```

# /etc/init.d/asterisk start
# asterisk -r

```

```

cd apps/
./OpenBTS

```

```

cd smqueue
./smqueue

```

You should be able to call 600 (the echo-test application) as well as text 411 (this will give you the status of the sms queue — how many messages it has waiting etc.).

Appendix B

Configuration

B.1 Asterisk

B.1.1 extensions.conf

```
1  [general]
2  static=yes
3  writeprotect=no
4  autofallthrough=yes
5  clearglobalvars=no
6  priorityjumping=no
7
8  [globals]
9
10 [lookupdundi]
11 switch => DUNDi/priv
12
13 [incomingdundi]
14 exten => _IMSI.,1,Goto(internal,${EXTEN},2)
15
16 [setup]
17 exten => s,1,Answer()
18 exten => s,n,Set(result=${SHELL(/usr/bin/whoami)})
19 exten => s,n,Noop("Beacon - The Rapidly Deployable GSM Network")
20 exten => s,n,Set(count=0)
21 exten => s,n(collect),Festival("To get started, enter your phone
    number (with country code), when done press the # key")
22 exten => s,n,Gotoif($[${count} > 3]?error)
23 exten => s,n,Set(count=${count}+1)
```

```

24 exten => s,n,Read(num)
25 exten => s,n,Festival("You entered ${num}")
26 exten => s,n,Noop("Does the number exist?")
27 exten => s,n,agi(parse_json.py,${CALLERID(num)})
28 exten => s,n,GotoIf($["${phonenumber}" == "${num}"]?collect)
29 exten => s,n,Festival("Creating your reputation...")
30 exten => s,n,Noop(${num})
31 exten => s,n,System(/usr/bin/curl -X POST http://192.168.0.22:8080/
    api/account/${CALLERID(num)}/ -d '{"data": [{"number": "${num
    }"}]}'')
32 exten => s,n,Festival("Created")
33 exten => s,n,Hangup()
34 exten => s,n(error),Hangup()
35
36 exten => 0,1,Goto(collect)
37
38 [rate]
39
40 include => lookupdundi
41 include => dundiextens
42
43 exten => _IMSI.,2,Answer()
44 exten => _IMSI.,n,Wait(1)
45 exten => _IMSI.,n,Noop(${dst} rates ${EXTEN})
46 exten => _IMSI.,n,Set(count=0)
47 exten => _IMSI.,n(beginning),Festival("Was the person you called
    the person you expected? 1 for yes, 2 for no and 3 is unsure")
48 exten => _IMSI.,n,Set(count=${${count} + 1})
49 exten => _IMSI.,n,Read(rating,,1)
50 exten => _IMSI.,n,Gotoif($["${rating}" == "1"]?yes)
51 exten => _IMSI.,n,Gotoif($["${rating}" == "2"]?no)
52 exten => _IMSI.,n,Gotoif($["${rating}" == "3"]?maybe)
53 exten => _IMSI.,n,Gotoif($["${count}" != "3"]?beginning)
54 exten => _IMSI.,n,Hangup()
55 exten => _IMSI.,n(yes),Noop(Increasing score for ${EXTEN})
56 exten => _IMSI.,n,System(/usr/bin/curl -X PUT http
    ://192.168.0.22:8080/api/account/${EXTEN}/ -d '{"data": [{"score
    ": "up"}]}'')
57 exten => _IMSI.,n,Goto(end)
58 exten => _IMSI.,n(no),Noop(Decreasing score for ${EXTEN})
59 exten => _IMSI.,n,System(/usr/bin/curl -X PUT http
    ://192.168.0.22:8080/api/account/${EXTEN}/ -d '{"data": [{"score

```

```

        ": "down"]}]')
60 exten => _IMSI.,n,Goto(end)
61 exten => _IMSI.,n(maybe),Noop(Doing nothing)
62 exten => _IMSI.,n(end),Noop()
63 exten => _IMSI.,n,Festival("Thanks")
64 exten => _IMSI.,n,Hangup()
65
66 [internal]
67
68 include => lookupdundi
69 include => dundiextens
70
71 exten => 777,1,Goto(setup,s,1)
72
73 exten => _IMSI.,2,Answer()
74 exten => _IMSI.,n,Set(_from=${CALLERID(number)})
75 exten => _IMSI.,n,Set(_to=${EXTEN})
76 exten => _IMSI.,n,Dial(SIP/${EXTEN},,gF(internal~${EXTEN}~finished)
    )
77 exten => _IMSI.,n(finished),Noop(Call has finished, wait a second
    before updating the score)
78 exten => _IMSI.,n,Wait(1)
79 exten => _IMSI.,n,Gotoif("${x${from}" == "xasterisk"?end)
80 exten => _IMSI.,n,agi(rate.py,${from},${to})
81 exten => _IMSI.,n,Noop(Increasing received calls for ${to})
82 exten => _IMSI.,n,System(/usr/bin/curl -X PUT http
    ://192.168.0.22:8080/api/account/${to}/ -d '{"data": [{"
    received_calls": "up"}]}')
83 exten => _IMSI.,n,Noop(Increasing initiated calls for ${from})
84 exten => _IMSI.,n,System(/usr/bin/curl -X PUT http
    ://192.168.0.22:8080/api/account/${from}/ -d '{"data": [{"
    initiated_calls": "up"}]}')
85 exten => _IMSI.,n(end),Hangup()
86
87 exten => _X.,1,Answer()
88 exten => _X.,n,agi(parse_json.py,${EXTEN})
89 exten => _X.,n,Noop(${AGISTATUS})
90 exten => _X.,n,Noop(IMSI = ${imsi})
91 exten => _X.,n,Noop(PhoneNumber = ${number})
92 exten => _X.,n,Noop(InitiatedCalls = ${initiated_calls})
93 exten => _X.,n,Noop(ReceivedCalls = ${received_calls})
94 exten => _X.,n,Noop(Score = ${score})

```

```

95 exten => _X.,n,Gotoif("${AGISTATUS}" == "SUCCESS"?found)
96 exten => _X.,n,Festival("I'm sorry, but I cannot find the number ${
    EXTEN}")
97 exten => _X.,n,Hangup
98 exten => _X.,n(found),Goto(internal,${imsi},1)

```

Listing B.1: Extensions.conf

B.1.2 sip.conf

```

1  [general]
2  context=default
3  allowoverlap=no
4  canreinvite=no
5  udpbindaddr=0.0.0.0
6  tcpenable=no
7  srlookup=no
8  regcontext=dundiextens
9  nat=no
10 rtcachefriends=yes
11 accept_outofcall_message=yes
12 auth_message_requests=no
13
14 [authentication]
15
16 [trunk]
17 host=192.168.0.110
18 type=peer
19 context=internal
20 disallow=all
21 allow=ulaw
22 allow=alaw
23 allow=gsm
24 qualify=yes
25 secret=verysecret
26 insecure=port,invite
27 username=trunk

```

Listing B.2: Sip.conf

B.1.3 dundi.conf

```

1  [general]
2  department=Development
3  organization=OpenBTS Test Networks
4  locality=Dunedin
5  stateprov=Otago
6  country=New Zealand
7  email=paul.crane@otago.ac.nz
8  phone=+64212742117
9  entityid=00:26:18:EE:49:19
10 ttl=1 ; reduce the chance of circular routing ;)
11 bindaddr=192.168.0.22
12 cachetime=5
13 autokill=yes
14 [mappings]
15 priv => dundiextens,0,SIP,trunk/${NUMBER},nopartial
16 [00:50:8d:f3:6b:32]
17 model=symmetric
18 host=192.168.0.110
19 inkey=priv
20 outkey=priv
21 include=priv
22 permit=priv
23 qualify=yes
24 order=primary

```

Listing B.3: dundi.conf

B.1.4 res_mysql.conf

```

1  [general]
2  dbhost = 127.0.0.1
3  dbname = beacon
4  dbuser = asterisk
5  dbpass = asterisk357
6  dbport = 3306
7  requirements=warn ; or createclose or createchar

```

Listing B.4: res_mysql.conf

B.1.5 extconfig.conf

```
1 [settings]
2 sipusers => mysql,general,asterisk_sip
3 sippeers => mysql,general,asterisk_sip
```

Listing B.5: extconfig.conf

B.2 OpenBTS

```
1 Log.Level DEBUG
2 Log.FileName test.out
3 $static Log.FileName
4 Log.Alarms.TargetIP 127.0.0.1
5 $optional Log.Alarms.TargetIP
6 Log.Alarms.TargetPort 10101
7 Log.Alarms.Max 10
8 GSMTAP.TargetIP 127.0.0.1
9 $optional GSMTAP.TargetIP
10 Indications.TargetIP 127.0.0.1
11 Indications.TargetPort 10202
12 TestCall.Port 28670
13 TRX.IP 127.0.0.1
14 $static TRX.IP
15 TRX.Port 5700
16 $static TRX.Port
17 TRX.Path ../Transceiver52M/transceiver
18 $static TRX.Path
19 TRX.LogLevel DEBUG
20 $static TRX.LogLevel
21 TRX.LogFileName test.TRX.out
22 $static TRX.LogFileName
23 Asterisk.IP 127.0.0.1
24 Asterisk.Port 5060
25 $static Asterisk.IP
26 $static Asterisk.Port
27 Smqueue.IP 127.0.0.1
28 Smqueue.Port 5063
29 $static Smqueue.IP
30 $static Smqueue.Port
31 SIP.Port 5062
32 $static SIP.Port
```

```

33 RTP.Start 16484
34 RTP.Range 98
35 $static RTP.Start
36 $static RTP.Range
37 SIP.IP 127.0.0.1
38 SIP.SMSC smsc
39 PBX.Emergency 2101
40 SIP.RegistrationPeriod 7200
41 SIP.Timer.A 2000
42 SMS.FakeSrcSMSC 0000
43 SMS.DefaultDestSMSC 0000
44 Control.LUR.QueryIMEI
45 $optional Control.LUR.QueryIMEI
46 Control.LUR.QueryClassmark
47 $optional Control.LUR.QueryClassmark
48 Control.LUR.TMSIsAll
49 $optional Control.LUR.TMSIsAll
50 Control.TMSITable.MaxSize 10000
51 Control.TMSITable.MaxAge 72
52 $optional Control.TMSISavePath
53 $optional Control.OpenRegistration
54 Control.NormalRegistrationWelcomeMessage Welcome to OpenBTS! AGPLv3
    openbts.sf.net. Your IMSI is
55 Control.NormalRegistrationWelcomeShortCode 0000
56 Control.OpenRegistrationWelcomeMessage Welcome to OpenBTS! AGPLv3
    openbts.sf.net. Your IMSI is
57 Control.OpenRegistrationWelcomeShortCode 101
58 $optional Control.FailedRegistrationWelcomeMessage
59 Control.FailedRegistrationWelcomeShortCode 1000
60 GSM.NCC 0
61 GSM.BCC 2
62 GSM.MCC 001
63 GSM.MNC 01
64 GSM.LAC 1000
65 GSM.CI 10
66 GSM.ShortName OpenBTS
67 $optional GSM.ShortName
68 GSM.ShowCountry
69 $optional GSM.ShowCountry
70 $optional GSM.VEA
71 GSM.Band 1900
72 $static GSM.Band

```

```

73 GSM.ARFCN 761
74 $static GSM.ARFCN
75 GSM.Neighbors 39 41 43
76 GSM.MaxExpectedDelaySpread 1
77 GSM.RxGain 47
78 GSM.PowerManager.MaxAttenDB 30
79 GSM.PowerManager.MinAttenDB 7
80 GSM.PowerManager.TargetT3122 5000
81 GSM.PowerManager.SamplePeriod 2000
82 GSM.PowerManager.NumSamples 10
83 GSM.PowerManager.Period 6000
84 GSM.NumC7s 1
85 $static GSM.NumC7s
86 GSM.NumC1s 6
87 $static GSM.NumC1s
88 $optional GSM.HalfDuplex
89 GSM.RADIO_LINK_TIMEOUT 15
90 GSM.CCD.ATT 1
91 GSM.CCD.CCCH_CONF 1
92 GSM.RACH.MaxRetrans 1
93 GSM.RACH.TxInteger 14
94 GSM.RACH.AC 0
95 GSM.NCCsPermitted 1
96 GSM.CS.MS_TXPWR_MAX_CCH 0
97 GSM.CS.RXLEV_ACCESS_MIN 0
98 GSM.CS.CELL_RESELECT_HYSTERESIS 3
99 GSM.CS.NECI 1
100 GSM.LURejectCause 0x04
101 GSM.MS.TA.Max 5
102 GSM.MS.TA.Damping 50
103 GSM.MS.Power.Max 33
104 GSM.MS.Power.Min 5
105 GSM.MS.Power.Damping 50
106 GSM.T3212 12
107 GSM.T3122Min 2000
108 GSM.T3122Max 255000
109 GSM.T3113 10000
110 GSM.RRLP.Accuracy 60
111 GSM.RRLP.Retries 1
112 GSM.RRLP.Timeout 4000
113 GSM.AGCH.QMax 5
114 GSM.RSSITarget -15

```

```
115 GSM.PagingReservations 0
116 GSM.MaxSpeechLatency 2
117 CLI.Prompt OpenBTS>
```

Listing B.6: openbts.conf

B.3 Django

B.3.1 API — api/handlers.py

```
1  import re
2
3  from django.views.decorators.csrf import csrf_exempt
4  from django.utils import simplejson
5  from piston.handler import BaseHandler
6  from piston.utils import rc
7  from account.models import Account
8  from asterisk.models import Sip
9
10 import score
11
12 class AccountHandler(BaseHandler):
13     allowed_methods = ('GET', 'PUT', 'POST',)
14     model = None;
15
16     def read(self, request, imsi=None, phone_number=None):
17         sip = None
18         if imsi and not phone_number:
19             sip = score._get_imsi(imsi)
20         elif not imsi and phone_number:
21             sip = score._get_number(phone_number)
22         return sip
23
24     def update(self, request, imsi=None):
25         inc = ["up", "+", "inc", "++"]
26         dec = ["down", "-", "dec", "--"]
27         data = simplejson.loads(request.POST.items()[0][0])
28         if data is not None:
29             for json in data['data']:
30                 print json
31                 try:
```

```

32         if 'received_calls' in json:
33             if json['received_calls'] in inc:
34                 score.inc_received_calls(imsi)
35             else:
36                 return rc.BAD_REQUEST
37
38         if 'initiated_calls' in json:
39             if json['initiated_calls'] in inc:
40                 score.inc_initiated_calls(imsi)
41             else:
42                 return rc.BAD_REQUEST
43
44         if 'score' in json:
45             if json['score'] in inc:
46                 score.inc_score(imsi)
47             elif json['score'] in dec:
48                 score.dec_score(imsi)
49             else:
50                 return rc.BAD_REQUEST
51     except Exception:
52         return rc.NOT_FOUND
53     return self.read(request, imsi)
54 return rc.NOT_FOUND
55
56 def create(self, request, imsi):
57     data = simplejson.loads(request.POST.items()[0][0])
58     if data is not None:
59         for json in data['data']:
60             score.create_imsi_record(imsi, json['number'])
61             break
62     return self.read(request, imsi)
63
64 class SipHandler(BaseHandler):
65     allowed_methods = ('GET',)
66     model = Sip
67
68     # ignore the password fields
69     exclude = (re.compile(r'.*secret'),)
70
71     def read(self, request, imsi=None, phone_number=None):
72         base = Sip.objects
73         if imsi:

```

```

74         try:
75             return base.get(name=imsi)
76         except:
77             return rc.NOT_FOUND
78     elif phone_number:
79         sip = score._get_number(phone_number)
80         try:
81             return base.get(name=sip['imsi'])
82         except:
83             return rc.NOT_FOUND
84     else:
85         return base.all()

```

Listing B.7: API functions

B.3.2 Cassandra Interface — api/score.py

```

1  import pycassa
2  from pycassa.index import *
3  from pycassa.cassandra.ttypes import NotFoundException
4
5  #POOL = pycassa.ConnectionPool(keyspace='Beacon', server_list=[
6      localhost:9160', '192.168.0.22:9160', '192.168.0.110:9160'])
7  POOL = pycassa.ConnectionPool(keyspace='Beacon', server_list=[
8      192.168.0.22'])
9
10 #POOL = pycassa.connect(keyspace='Beacon', server_list
11     =['192.168.0.22'])
12 IMSI = pycassa.ColumnFamily(POOL, 'IMSI')
13
14
15 # From twissandra
16 #####
17 class DatabaseError(Exception):
18     pass
19
20 class NotFound(DatabaseError):
21     pass
22
23 #####
24
25 ### Note:
26 #

```

```

22 # The functions getting the fields (score, received_calls, and
    initiated_calls)
23 # treat them as numeric fields. This means that we can easily add/
    subtract from them
24 #
25 #
26
27
28 # base functions
29 #
30 # get the record associated with an IMSI number
31 def _get_imsi(i):
32     try:
33         imsi = IMSI.get(str(i))
34         imsi['imsi'] = str(i)
35     except NotFoundException:
36         raise NotFound('IMSI %s not found' % (i))
37     return imsi
38
39 def _get_number(n):
40     try:
41         number_expression = create_index_expression('number',n)
42         clause = create_index_clause([number_expression], count=1)
43         for key,value in IMSI.get_indexed_slices(clause):
44             value['imsi'] = key
45             return value
46         return None
47     except NotFoundException:
48         raise NotFound('Phone number %s not found' % (n))
49     return None
50
51 # increment the given field
52 def _inc_field(imsi,score=0,initiated_calls=0,received_calls=0):
53
54     s = get_score_by_imsi(imsi)
55     r = get_received_calls_by_imsi(imsi)
56     i = get_initiated_calls_by_imsi(imsi)
57     n = get_number_by_imsi(imsi)
58
59     if score > 0:
60         s = s + 1
61     elif score < 0:

```

```

62     s = s - 1;
63     if initiated_calls:
64         i = i + 1
65     if received_calls:
66         r = r + 1
67     create_imsi_record(imsi,number=n,score=s, initiated_calls=i,
        received_calls=r);
68
69 def get_score_by_imsi(imsi):
70     i = _get_imsi(imsi)
71     return int(i['score'])
72
73 def get_received_calls_by_imsi(imsi):
74     i = _get_imsi(imsi)
75     return int(i['received_calls'])
76
77 def get_initiated_calls_by_imsi(imsi):
78     i = _get_imsi(imsi)
79     return int(i['initiated_calls'])
80
81 def get_number_by_imsi(imsi):
82     i = _get_imsi(imsi)
83     return str(i['number'])
84
85 def get_imsi_by_number(number):
86     i = _get_number(number)
87     return str(i['imsi'])
88
89 def create_imsi_record(imsi,number=None,score=0,initiated_calls=0,
        received_calls=0):
90     if number == None:
91         raise Exception('Phone number cannot be None')
92
93     IMSI.insert(str(imsi),{ 'number':str(number),
94         'score':str(score),
95         'initiated_calls':str(initiated_calls),
96         'received_calls':str(received_calls)
97     })
98
99
100 def inc_score(imsi):
101     _inc_field(imsi,score=1)

```

```

102     return get_score_by_imsi(imsi)
103
104 def dec_score(imsi):
105     _inc_field(imsi,score=-1)
106     return get_score_by_imsi(imsi)
107
108 def inc_received_calls(imsi):
109     _inc_field(imsi,received_calls=1)
110     return get_received_calls_by_imsi(imsi)
111
112 def inc_initiated_calls(imsi):
113     _inc_field(imsi,initiated_calls=1)
114     return get_initiated_calls_by_imsi(imsi)
115
116 #### Testing
117 #
118 #create_imsi_record("IMSI530240100139366",score=0,initiated_calls
119     =0,received_calls=0)
120 #
121 #print get_score_by_imsi("IMSI530240100139366");
122 #print get_initiated_calls_by_imsi("IMSI530240100139366");
123 #print get_received_calls_by_imsi("IMSI530240100139366");
124 #
125 #create_imsi_record("IMSI530240100139366",score=1,initiated_calls
126     =1,received_calls=1)
127 #
128 #print get_score_by_imsi("IMSI530240100139366");
129 #print get_initiated_calls_by_imsi("IMSI530240100139366");
130 #print get_received_calls_by_imsi("IMSI530240100139366");
131 #
132 #inc_score("IMSI530240100139366")
133 #inc_initiated_calls("IMSI530240100139366")
134 #inc_received_calls("IMSI530240100139366")
135 #
136 #print get_score_by_imsi("IMSI530240100139366");
137 #print get_initiated_calls_by_imsi("IMSI530240100139366");
138 #print get_received_calls_by_imsi("IMSI530240100139366");
139 #
140 #print _get_imsi("IMSI530240100139367")
141 #print get_imsi_by_number("123")

```

Listing B.8: Cassandra Interface

B.3.3 Models — asterisk/models.py

```
1 from django.db import models
2 import time
3
4 class Sip(models.Model):
5     name = models.CharField(max_length=80,primary_key=True,default='')
6     type = models.CharField(max_length=10,default='friend')
7     username = models.CharField(max_length=80,default='')
8     fromuser = models.CharField(max_length=80,blank=True,null=True,
9         default=None)
10    fromdomain = models.CharField(max_length=80,blank=True,null=True,
11        default=None)
12    secret = models.CharField(max_length=80,blank=True,null=True,
13        default=None)
14    md5secret = models.CharField(max_length=80,blank=True,null=True,
15        default=None)
16    auth = models.CharField(max_length=10,blank=True,null=True,
17        default=None)
18    mailbox = models.CharField(max_length=20,blank=True,null=True,
19        default=None)
20    subscribewi = models.CharField(max_length=20,blank=True,null=
21        True,default='no')
22    vmexten = models.CharField(max_length=20,blank=True,null=True,
23        default=None)
24    callerid = models.CharField(max_length=40,blank=True,null=True,
25        default=None)
26    cid_number = models.CharField(max_length=40, blank=True,null=True
27        ,default=None)
28    callingpres = models.CharField(max_length=20, blank=True,null=
29        True,default=None)
30    usereqphone = models.CharField(max_length=10, blank=True,null=
31        True,default=None)
32    language = models.CharField(max_length=10,blank=True,null=True,
33        default=None)
34    incominglimit = models.CharField(max_length=10,blank=True,null=
35        True,default=None)
36    context = models.CharField(max_length=40,blank=True,null=True,
37        default=None)
38    subscribecontext = models.CharField(max_length=40,blank=True,null
39        =True,default=None)
```

```

24     amaflags = models.CharField(max_length=20, blank=True, null=True,
    default=None)
25     accountcode = models.CharField(max_length=20, blank=True, null=True
    ,default=None)
26     musicclass = models.CharField(max_length=20, blank=True, null=True,
    default=None)
27     mohsuggest = models.CharField(max_length=20, blank=True, null=True,
    default=None)
28     allowtransfer = models.CharField(max_length=20, blank=True, null=
    True, default=None)
29     callgroup = models.CharField(max_length=20, blank=True, null=True,
    default=None)
30     pickupgroup = models.CharField(max_length=20, blank=True, null=True
    ,default=None)
31     autoframing = models.CharField(max_length=10, blank=True, null=True
    ,default=None) # yes/no
32     disallow = models.CharField(max_length=20, blank=True, null=True,
    default='all')
33     allow = models.CharField(max_length=20, blank=True, null=True,
    default='gsm;alaw;ulaw')
34     maxcallbitrate = models.CharField(max_length=15, blank=True, null=
    True, default=None)
35     host = models.CharField(max_length=40, default='dynamic')
36     outboundproxy = models.CharField(max_length=40, blank=True, null=
    True, default=None)
37     ipaddr = models.CharField(max_length=40, default='')
38     defaultip = models.CharField(max_length=20, blank=True, null=True,
    default=None)
39     port = models.CharField(max_length=6, default='')
40     fullcontact = models.CharField(max_length=44, blank=True, null=True
    ,default='')
41     insecure = models.CharField(max_length=20, blank=True, null=True,
    default=None)
42     qualify = models.CharField(max_length=15, blank=True, null=True,
    default=None)
43     regseconds = models.IntegerField(default=0)
44     regexten = models.CharField(max_length=20, blank=True, null=True,
    default=None)
45     regserver = models.CharField(max_length=20, blank=True, null=True,
    default=None)
46     rtptimeout = models.CharField(max_length=15, blank=True, null=True,
    default=None)

```

```

47     rtpholdtimeout = models.CharField(max_length=15, blank=True, null=
        True, default=None)
48     rtpkeepalive = models.CharField(max_length=15, blank=True, null=
        True, default=None)
49     lastms = models.IntegerField(default=-1)
50     setvar = models.CharField(max_length=200, blank=True, null=True,
        default=None)
51     canreininvite = models.CharField(max_length=10, blank=True, null=True
        , default='no')
52     useragent = models.CharField(max_length=50, blank=True, null=True,
        default=None)
53
54     def __unicode__(self):
55         if self.regseconds > time.time():
56             return "%s@s (online)"%(self.name, self.context)
57         elif self.regseconds == 0:
58             return "%s@s (not seen)"%(self.name, self.context)
59         else:
60             return "%s@s"%(self.name, self.context)
61
62
63     class CDR(models.Model):
64         calldate = models.CharField(max_length=255, blank=False, default=""
        )
65         clid = models.CharField(max_length=80, blank=False, default="")
66         src = models.CharField(max_length=80, blank=False, default="")
67         dst = models.CharField(max_length=80, blank=False, default="")
68         dcontext = models.CharField(max_length=80, blank=False, default="")
69         channel = models.CharField(max_length=80, blank=False, default="")
70         dchannel = models.CharField(max_length=80, blank=False, default="")
71         lastapp = models.CharField(max_length=80, blank=False, default="")
72         duration = models.BigIntegerField()
73         billsec = models.BigIntegerField()
74         disposition = models.CharField(max_length=45, blank=False, default=
        "")
75         amaflags = models.BigIntegerField()
76         accountcode = models.CharField(max_length=20, blank=False, default=
        "")
77         uniqueid = models.CharField(max_length=32, blank=False, default="")
78         userfield = models.CharField(max_length=255, blank=False, default="
        ")
79

```

```

80     def __unicode__(self):
81         return "%s %s %s@%s (%s) - %s"(self.calldate,self.src,self.dst,
            self.dcontext,self.billsec,self.disposition)

```

Listing B.9: Asterisk's Models

B.4 AGI Scripts

B.4.1 JSON Parser — parse_json.py

```

1  #!/usr/bin/python
2
3  import sys
4  import re
5  import time
6  import random
7  import pycurl
8  import simplejson
9  from StringIO import StringIO
10 from pprint import pprint
11
12 def ReadAgiEnvironment():
13     env = {}
14     while 1:
15         line = sys.stdin.readline().strip()
16         if not line:
17             break
18         key,sep,data = line.partition(':')
19         env[key.strip()] = data.strip()
20     return env
21
22 def SendAgi(cmd):
23     sys.stdout.write("%s\n" % cmd)
24     sys.stdout.flush()
25     sys.stdin.readline()
26
27 env = ReadAgiEnvironment()
28
29 pprint(env,sys.stderr)
30
31 storage = StringIO()

```

```

32 c = pycurl.Curl()
33 c.setopt(c.URL, "http://192.168.0.22:8080/api/account/%s/"%(env['
    agi_arg_1']))
34 c.setopt(c.WRITEFUNCTION, storage.write)
35 c.perform()
36 c.close()
37
38 content = storage.getvalue()
39 data = simplejson.loads(content)
40
41 sys.stderr.write(content)
42 sys.stderr.flush()
43
44 for key,value in sorted(data.items()):
45     SendAgi("set variable %s \"%s\""%(key,value))

```

Listing B.10: Dialplan JSON Parser

B.4.2 Rating — rate.py

```

1  #!/usr/bin/python
2
3  import sys
4  import tempfile
5  import shutil
6  from pprint import pprint
7
8  def SendAgi(cmd):
9      sys.stdout.write("%s\n" % cmd)
10     sys.stdout.flush()
11     sys.stdin.readline()
12
13  def ReadAgiEnvironment():
14     env = {}
15     while 1:
16         line = sys.stdin.readline().strip()
17         if not line:
18             break
19         key,sep,data = line.partition(':')
20         env[key.strip()] = data.strip()
21     return env
22

```

```

23 env = ReadAgiEnvironment()
24
25 #if cmp(env['agi_arg_1'],'asterisk')== 0:
26     #return
27
28 # agi_arg_1 is 'from'
29 # agi_arg_2 is 'about'
30 call_file="Channel: Local/%s@rate/n\nMaxRetries: 1\nContext: rate\
nExtension: %s\nPriority: 1\nSetVar: dst=%s\n\n"%(env['agi_arg_1']
,env['agi_arg_2'],env['agi_arg_1'])
31
32 sys.stderr.write(call_file);
33
34 cf = tempfile.NamedTemporaryFile(delete=False)
35 cf.write(call_file)
36 cf.close()
37
38 shutil.move(cf.name,"/var/spool/asterisk/outgoing/")

```

Listing B.11: Dialplan User Rater

B.5 MySQL

Run through the setup as needed (you'll need to create a super-user and a password). We suggest that you create an account that will be used by Asterisk, something like the following:

```

CREATE DATABASE asterisk_realtime;
CREATE USER 'asterisk'@'localhost' IDENTIFIED BY 'some_password';
GRANT ALL PRIVILEGES ON 'asterisk' TO 'asterisk'@'localhost';
FLUSH PRIVILEGES;

```

Note: The following step is performed automatically when you use Django's `syncdb`. Now, as the asterisk user create the following table:

```

1 # Asterisk 1.6.x structure for sip 'device' table
2 CREATE TABLE 'sip_devices' (
3     'id' int(11) NOT NULL AUTO_INCREMENT,
4     'name' varchar(80) NOT NULL DEFAULT '',
5     'context' varchar(80) DEFAULT NULL,

```

```

6  'callingpres' enum('allowed_not_screened','allowed_passed_screen
   ', 'allowed_failed_screen',\
7    'allowed','prohib_not_screened','prohib_passed_screen','
   prohib_failed_screen','prohib',\
8    'unavailable') DEFAULT 'allowed_not_screened',
9  'permit' varchar(95) DEFAULT NULL,
10 'deny' varchar(95) DEFAULT NULL,
11 'secret' varchar(80) DEFAULT NULL,
12 'md5secret' varchar(80) DEFAULT NULL,
13 'remotesecret' varchar(250) DEFAULT NULL,
14 'transport' enum('tcp','udp','tcp,udp') DEFAULT NULL,
15 'host' varchar(31) NOT NULL DEFAULT '',
16 'nat' varchar(5) NOT NULL DEFAULT 'no',
17 'type' enum('user','peer','friend') NOT NULL DEFAULT 'friend',
18 'accountcode' varchar(20) DEFAULT NULL,
19 'amaflags' varchar(13) DEFAULT NULL,
20 'callgroup' varchar(10) DEFAULT NULL,
21 'callerid' varchar(80) DEFAULT NULL,
22 'defaulttip' varchar(15) DEFAULT NULL,
23 'dtmfmode' varchar(7) DEFAULT NULL,
24 'fromuser' varchar(80) DEFAULT NULL,
25 'fromdomain' varchar(80) DEFAULT NULL,
26 'insecure' varchar(4) DEFAULT NULL,
27 'language' char(2) DEFAULT NULL,
28 'mailbox' varchar(50) DEFAULT NULL,
29 'pickupgroup' varchar(10) DEFAULT NULL,
30 'qualify' char(3) DEFAULT NULL,
31 'regexten' varchar(80) DEFAULT NULL,
32 'rtptimeout' char(3) DEFAULT NULL,
33 'rtpholdtimeout' char(3) DEFAULT NULL,
34 'setvar' varchar(100) DEFAULT NULL,
35 'disallow' varchar(100) DEFAULT 'all',
36 'allow' varchar(100) DEFAULT 'g729;ilbc;gsm;ulaw;alaw',
37 'fullcontact' varchar(80) NOT NULL DEFAULT '',
38 'ipaddr' varchar(15) NOT NULL DEFAULT '',
39 'port' smallint(5) unsigned NOT NULL DEFAULT '0',
40 'username' varchar(80) NOT NULL DEFAULT '',
41 'defaultuser' varchar(80) NOT NULL DEFAULT '',
42 'subscribecontext' varchar(80) DEFAULT NULL,
43 'directmedia' enum('yes','no') DEFAULT NULL,
44 'trustrid' enum('yes','no') DEFAULT NULL,
45 'sendrid' enum('yes','no') DEFAULT NULL,

```

```

46  'progressinband' enum('never','yes','no') DEFAULT NULL,
47  'promiscredir' enum('yes','no') DEFAULT NULL,
48  'useclientcode' enum('yes','no') DEFAULT NULL,
49  'callcounter' enum('yes','no') DEFAULT NULL,
50  'busylevel' int(10) unsigned DEFAULT NULL,
51  'allowoverlap' enum('yes','no') DEFAULT 'yes',
52  'allowssubscribe' enum('yes','no') DEFAULT 'yes',
53  'allowtransfer' enum('yes','no') DEFAULT 'yes',
54  'ignorestdpversion' enum('yes','no') DEFAULT 'no',
55  'template' varchar(100) DEFAULT NULL,
56  'videosupport' enum('yes','no','always') DEFAULT 'no',
57  'maxcallbitrate' int(10) unsigned DEFAULT NULL,
58  'rfc2833compensate' enum('yes','no') DEFAULT 'yes',
59  'session-timers' enum('originate','accept','refuse') DEFAULT '
    accept',
60  'session-expires' int(5) unsigned DEFAULT '1800',
61  'session-minse' int(5) unsigned DEFAULT '90',
62  'session-refresher' enum('uac','uas') DEFAULT 'uas',
63  't38pt_usertpsource' enum('yes','no') DEFAULT NULL,
64  'outboundproxy' varchar(250) DEFAULT NULL,
65  'callbackextension' varchar(250) DEFAULT NULL,
66  'registertrying' enum('yes','no') DEFAULT 'yes',
67  'timert1' int(5) unsigned DEFAULT '500',
68  'timerb' int(8) unsigned DEFAULT NULL,
69  'qualifyfreq' int(5) unsigned DEFAULT '120',
70  'contactpermit' varchar(250) DEFAULT NULL,
71  'contactdeny' varchar(250) DEFAULT NULL,
72  'lastms' int(11) NOT NULL,
73  PRIMARY KEY ('id'),
74  UNIQUE KEY 'name' ('name'),
75  KEY 'name_2' ('name')
76 ) ENGINE=MyISAM DEFAULT CHARSET=latin1 ROW_FORMAT=DYNAMIC

```

Listing B.12: Asterisk realtime SIP devices table, taken from <http://www.voip-info.org/wiki/view/Asterisk+RealTime+Sip>.

This is all the setup needed in MySQL (unless you want to do some performance tuning but that's beyond the scope of this document).