Cross-protocol re-routing of real-time communication sessions

Submitted in partial fulfilment
of the requirements of the degree
Bachelour of Science Honours in Computer Science
at Rhodes University

Justin R. Zondagh
Supervised by:
Professor Alfredo Terzoli
Professor Peter Clayton

6th November 2005

Acknowledgements

I would like to thank Alfredo and Peter for supervising my project and the constant inflow of ideas and Jason for all the time spent debugging Asterisk and my program, and for all the other assistance he provided. I would also like to thank Fred for all his ideas and input and Hannah proof reading my paper. Lastly I would like to thank the Department of Computer Science and all the sponsors - Telkom, Business Connexion, Comverse, Verso Technologies, and Thrip - for giving me the opportunity to do this project.

Abstract

With the merger of telephony and computer networks, telephony has become more flexible and adaptable to the needs of the user. Voice over IP allows for relatively inexpensive communication by sending the voice via computer networks such as company intranets and the internet. The project discussed in this paper will investigate how communication sessions can be re-routed via different networks and/or protocols depending on the state of the network that the session is currently routed over. The least-cost routing solution has been implemented as an application or module within Asterisk - an open source private branch exchange (PBX) - and was tested by re-routing sessions between an 802.11 based network and a global system for mobile (GSM) network. The objective was to provide a solution that will perform the re-routing on the last leg of the call and not to be deeply rooted within the telecommunication companies' networks.

Contents

List of Figures

Chapter 1

Introduction

1.1 Scenario and Problem Statement

With the merger of telephone and computer networks, Voice over IP (VoIP) is fast becoming a popular method of telephony as more bandwidth becomes available and data network speeds improve. A VoIP infrastructure is far cheaper to deploy, maintain and make calls on than legacy PBXs (private batch exchange) and legacy telephone networks. The deployment and maintenance is cheaper on a VoIP network as it makes use of the original internet protocol (IP) based computer network deployed in the organization [2, 3, 4, 5]. The calls are cheaper on the VoIP system as VoIP systems use an eighth of the bandwidth the traditional phone systems use.

The project discussed in this paper will investigate a way of implementing least-cost routing within Asterisk - an open source PBX - where the cost could be financial, the use of less secure links, bandwidth, or any criteria that the administrator would prefer to use. At the same time, the system will increase the quality of calls as well as improve the robustness of the VoIP system. This will enable the PBX to try and choose the cheapest network and/or protocol for communication, depending on the current conditions of the cheaper or more preferred network. In the case when an IP network is used, the conditions such as latency and packet loss of the cheaper network will be monitored.

The PBX will choose the optimal network for transmission before the establishment of the session and will continuously monitor the link while the link exists. If the current network and/or protocol's conditions deteriorate, the PBX will automatically re-route the session to an alternate network and/or protocol. The project will specifically investigate the re-routing of communication sessions between GSM and wireless VoIP phones as illustrated in Figure 1.1.

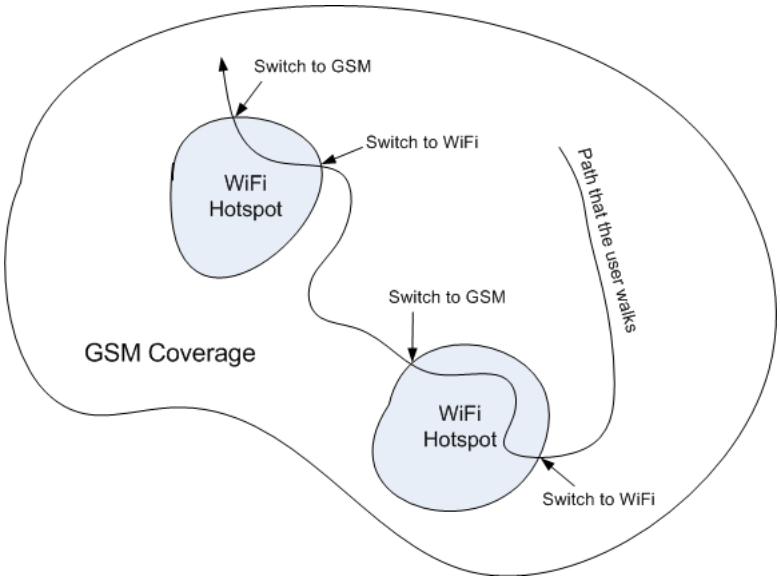

Figure 1.1: Showing how the session will be re-routed via GSM when the user exits the WiFi network and back to WiFi when the user enters a WiFi network

1.2 Technologies and protocols used in the project

This section will describe the protocols used in the project during experimentation and testing of the solution.

VoIP is the real-time delivery of voice between two or more parties across networks using IP as well as the exchange of information required to control this delivery [15]. Real-time communication sessions are comprised of two components: a signaling and a streaming component. The signaling component is responsible for all the administrative duties, such as negotiating the establishment of a session, redirection of a session and termination of a session, while the streaming component takes care of the actual end-to-end delivery of the media, be it voice or video. These VoIP protocols and how they interact are illustrated in Figure 1.2.

The Session Initiation Protocol (SIP) is a simple light weight signaling protocol that is most commonly implemented on top of the User Datagram Protocol (UDP), but can also be implemented on top of the Transmission Control Protocol (TCP) [3, 15]. SIP will be discussed in more detail in Section 1.3.

The Real-time protocol (RTP) is a streaming protocol and is generally used in conjunction with UDP, but can make use of any packet-based lower layer protocol [4]. RTP has many responsibilities as will be discussed in Section 1.4. The Real-time control protocol (RTCP) is a

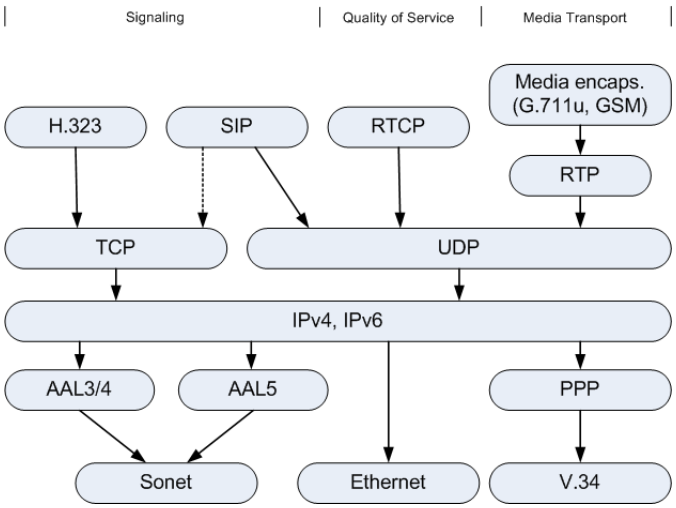

Figure 1.2: The Internet telephony (VoIP) protocol stack adapted from [15]

companion protocol of RTP. RTCP packets carry information regarding the state of end points as well as statistical information regarding the current RTP stream such as delay times, packet loss and jitter [1]. RTCP will be discussed further in Section 1.5.

802.11 is an IEEE medium range (in the range of 100 meters) broadband wireless networking standard that is used to connect wireless devices together as well as provide wireless access to a wired local area network. The 802.11b standard offers a shared data transfer rate up to 11Mbps, and 802.11g offers shared data transfer rates up to 54Mbps. 802.11 will be used to transport VoIP data and will be discussed in more detail in Section 1.6.

The remainder of the chapter will provide a more in depth discussion of the various protocols used in the experiments and testing of the solution.

1.3 The Session Initiation Protocol

The previous section introduced SIP and the other VoIP protocols and this section provides a detailed description of SIP and demonstrates its role in the VoIP architecture as a signalling protocol.

SIP [13] is the Internet Engineering Task-force's (IETF's) standard for multimedia conferencing over IP, where a conference involves one or more end-points. SIP is a basic text-based application layer protocol that allows end-points to negotiate the establishment, redirection and termination of real-time communication sessions as well as determine the location, media ca-

pabilities and availability of the target end point [3]. One should note however that a physical network resource or channel does not exist between the end points; instead a session is a signaling state at the two end points and a session indicates a RTP stream in either direction between end points [15]. Like other VoIP protocols, SIP is designed to address the functions of signaling and session management within a packet based telephony network. Signaling allows for calls to be established across network boundaries and session management provides the ability to control the attributes of the end-to-end call.

As illustrated in Figure 1.3, SIP is a peer-to-peer protocol where the peers in a session are called user agents. The user agents can either function as the user agent client or as the user agent server in a transaction. The client application that initiates the transaction will assume the role of user agent client and the server application that contacts the user when a SIP request is received assumes the role of user agent server [3].

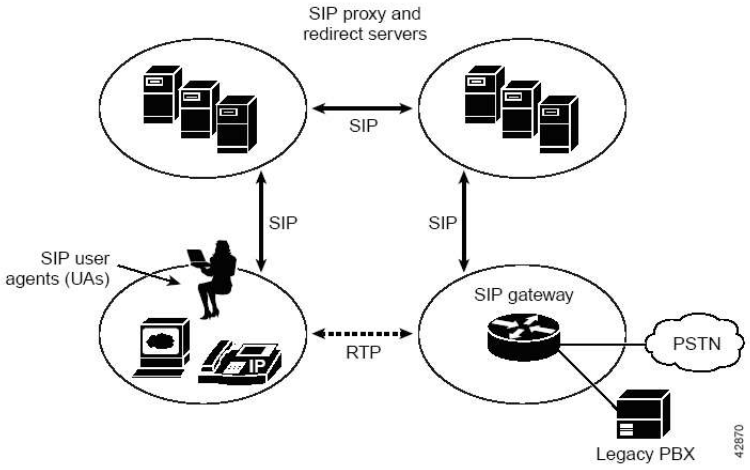

Figure 1.3: SIP Architecture [3]

The other components that assist in the functioning of SIP are the SIP gateway and the SIP proxy and redirect servers. The SIP gateway provides a signaling and streaming interface between the VoIP SIP network and a legacy PBX or public switched telephone network (PSTN). The proxy and redirect servers contain information about the other SIP users on the network, such as their location and capabilities [3]. These components do not fall within the scope of the project, so will not be discussed any further.

1.4 The Real-time Transport Protocol

This section provides a detailed description of RTP [14] and demonstrates RTP's role in the VoIP architecture as a streaming protocol.

Once a signalling protocols such as SIP has negotiated and established a session between two or more end points, RTP will start transporting the media between the end points, be it voice or video [4]. The real-time protocol is responsible for the sequencing of packets, intra-media synchronization, inter-media synchronization, payload identification and frame indication [15] .

RTP has two components, RTP itself and its companion protocol, RTCP. RTP is generally used in conjunction with UDP layer, but can make use of other packet-based lower layer protocols. When a host wishes to send a media packet, it takes the media, formats it for packetization, adds any media specific packet headers, prepends the RTP header and hands the packet to the UDP or lower-layer protocol for transmission [15]. RTP packets have been designed so that one can place any type of media in the RTP payload as long as both end points can encode and decode the media format. Various codecs can be used for the encoding and decoding of VoIP media as the different codecs are tailored for different network environments, such as networks that have high delay, high jitter or high packet loss. As illustrated in Figure 1.4, each RTP packet contains a sequence number which uniquely identifies the packet in the stream and indicates in which order the packets should be arranged for playback at the target end. RTP packets also contain a time-stamp, indicating when the media frame was generated relative to other RTP packets in the same session, allowing for media synchronization between the end-points [15].

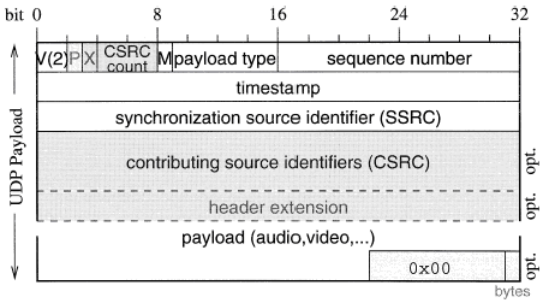

Figure 1.4: RTP Packet's structure[15]

1.5 The Real-time control protocol (RTCP)

As mentioned in the previous section on RTP, the real-time control protocol (RTCP) [14, 8] is a companion protocol for RTP and conveys information specific to a RTP media stream between end points. Media senders and receivers periodically send RTCP packets to each other containing a number of elements, usually a sender report or a receiver report followed by source descriptions [15]. The sender reports are generated by users who are RTP stream senders. Each RTCP sender report contains information regarding the amount of information sent so far as well as correlating the RTP sampling timestamps and absolute timestamps to allow for synchronization [1, 4]. Included in the receiver reports are delay times, packet loss quantities and jitter information. This information is used by the sending party to adjust their transmission speed and other parameters to minimize the packet loss, delay and/or jitter [1, 4, 15].

1.6 802.11 Wireless Networks

Wireless networking is fast becoming a popular medium for networking as it allows for a user to be connected to a network and ultimately the internet without having a physical network connection enabling greater mobility and flexibility [19].

802.11's biggest limitation is its limited range of approximately 100 meters without boosting through the use of high gain antennas or repeaters. This has proven to be rather problematic if the wireless network infrastructure is used for VoIP applications where the user can easily exit the coverage of the 802.11 wireless networks during a session as the session will be terminated. This is one of the problems that the project has addressed.

1.7 The solution

An application has been written for Asterisk which monitors the quality of the preferred link between the PBX and the end user. The application checks the status of the preferred link before a session is established. If the status is above 50%, the session is routed via the preferred link, else routed via the alternate link. If the quality or status of the link deteriorates below an acceptable level once the system has been established, the application will re-route the call to the alternate device or link, and if the quality or status rises above the upper threshold, the call will be re-routed via the preferred link. This will be discussed in more detail in Chapter 4 on the DialPref application.

Chapter 2

Asterisk and iLanga

2.1 Asterisk

Asterisk can be thought of as a ”middleware” as it sits between telephony technologies and telephony applications [17]. In other words, Asterisk provides a way for different telephony technologies to communicate with each other as well as interact with telephony applications. An example of this could be a VoIP protocol, such as SIP, communicating with legacy telephony technologies such as a integrated service digital network (ISDN) or an user phoning in on an ISDN channel and interacting with the Voicemail application within Asterisk.

Asterisk has a channel based architecture which allows for Asterisk to support any telephony protocol as long as a channel has been written for the specific protocol. The Asterisk channel structure translates the protocols such as SIP, ISDN and/or RTP from their native format to a generic structure called frames which can be understood by all channels regardless of the channel's type. This allows for the intercommunication of channels of different types such as SIP and H.323 (H.323 is a VoIP specification developed by the International Telecommunications Union), thus allowing for interoperability between telephony networks and the integration of modern and legacy telephony networks. Figure 2.1 illustrates how SIP, H.323 and ISDN networks can intercommunicate by using Asterisk as the middleware.

The remainder of this section describes the various APIs, applications, manager interface and configuration files within Asterisk.

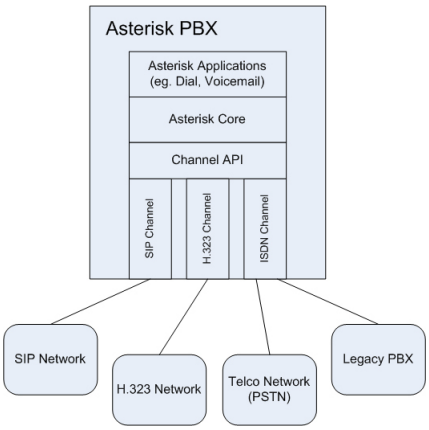

Figure 2.1: Different telephony networks intercommunicating via Asterisk

2.1.1 Asterisk's APIs

Asterisk has various APIs that enable the inter-communication of applications, channels, codecs and various other components as illustrated in Figure 2.2 below. These APIs are: the Application API, the Codec Translator API, the File Format API, and the Channel API.

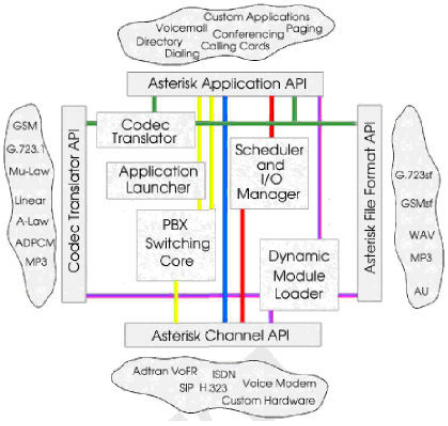

Figure 2.2: The Asterisk APIs

2.1.2 Asterisk Applications

Asterisk is shipped with various applications such as the Dial, Voicemail, MeetMe and Playback applications that can be called from within the Asterisk Dial Plan as shown in the extract below.

Extract from Asterisk Dial Plan:

```
[incoming-H323]
exten => 9000,1,Dial(SIP/3456)
exten => 9000,102,Voicemail(u3456)
```

C applications can be plugged into Asterisk by interfacing with the Application API and need to implement 5 mandatory functions namely: `load_module(), unload_module(), description(), app_exec()` and `key()`. When Asterisk starts up, the `load_module()` function is called which loads the module. The unload_module() is called when Asterisk is shutdown. The `description()` function is called when a user requests a description of a the application whilst using the Asterisk CLI (command line interface). The `app_exec()` function is called when Asterisk needs to execute that specific application and is the point of entry into the application [7]. Other functions may be defined within the application and can be called from the `app_exec()` function or any other function as one would normally do in programming of this style.

2.1.3 The Asterisk Manager Interface

When Asterisk starts up, a TCP port is opened on which Asterisk accepts external connections. This enables authenticated external applications to send and receive plain text commands to Asterisk via a TCP connection which can request various actions such as redirecting a call or establishing a call. Once Asterisk has performed the action, the results are returned to the client. The manager API can be configured in the manager.conf file.

2.1.4 Configuration files

Asterisk is configured by editing configuration files located in the `/etc/asterisk` directory on a Linux machine. Some of the commonly used configuration files include: `sip.conf, iax.conf, modules.conf` and `extensions.conf`. These configuration files are not within the scope of the project and so will only be discussed briefly in the remainder of this section, but more information can be obtained by consulting the Asterisk Handbook [17].

2.1.4.1 Modules.conf

Asterisk can be instructed to load or not load certain modules by editing the `modules.conf` file located at `/etc/asterisk/modules.conf` on a Linux machine. By entering `load`

`= app_dial.so` into the `modules.conf` file, Asterisk will load the Dial Application when Asterisk starts up and `noload = chan_sip.so` will instruct Asterisk not to load the SIP channel when starting up.

2.1.4.2 Sip.conf

The `sip.conf` file is the SIP channel's configuration file and is where all SIP users are defined and other configurations regarding the SIP channel are set.

2.1.4.3 Extensions

One of the core concepts within Asterisk is that of extensions The term extension has been inherited from legacy PBXs, where one extension can dial another extension by dialing a number associated with the extension. In Asterisk however, the term extension covers more. An extension is simply a number that has been associated with one or more actions within Asterisk as defined in the `dial plan` [7]. An example of this could be a user dialing a number, say 1289, this could first dial the other user at their extension and if they are unavailable re-direct the user to the Voicemail application.

2.2 iLanga

iLanga is a computer-based PBX currently being prototyped in the Department of Computer Science at Rhodes University. iLanga is composed of three open source components and a fourth component that has been written by students in the department. Asterisk - an open source PBX - forms the soft switching core of iLanga and SIP Express Router (SER) and Open Gate Keeper (OpenGK) provide points of entry into iLanga for SIP and H.323 users respectively as illustrated in Figure 2.3 below. SER is a high performance SIP proxy, location and redirection server and is suitable for a range of scenarios including small offices, enterprise PBXs and carrier services. In iLanga, SER is used in conjunction with Asterisk to handle SIP registrations and provide innovative SIP services. When SER is configured as a forking proxy, users can register multiple SIP clients at various locations and still retain the same SIP universal resource identifier. Asterisk supports the H.323 protocol by acting as a H.323 endpoint, thus not providing the H.323 management functionality that a H.323 gatekeeper such as OpenGK would provide [11].

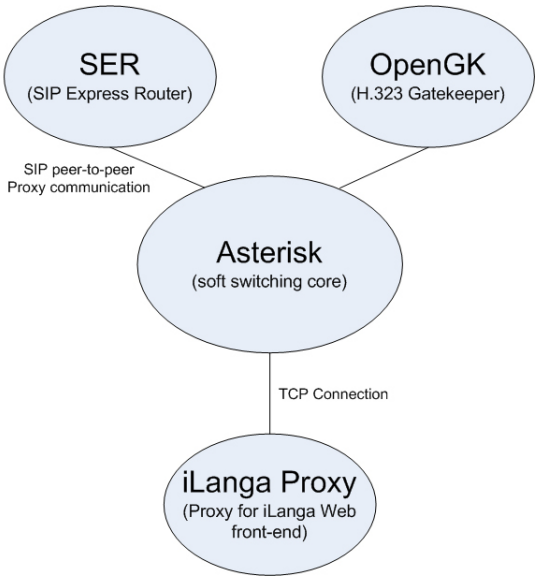

Figure 2.3: iLanga's architecture

The fourth component is the iLanga proxy which is the interface between Asterisk and the iLanga front-end. The iLanga proxy is needed as the Asterisk Manager API has been proven to be rather unstable when many TCP connections are made to it. The Asterisk Manager API also broadcasts all information to all the users connected to it which generates unnecessary network traffic and may prove to be a security risk. Thus by introducing the iLanga proxy, these problems are eliminated as only one connection to the Asterisk Manager Interface is made by the iLanga proxy and the proxy will only send information to a user that is of direct interest to that user [7].

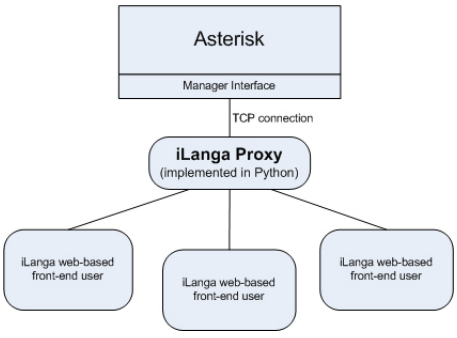

Figure 2.4: iLanga Proxy

Chapter 3

Asterisk Dial Application and Conference Room

The Dial application was used in this project as a tool to help understand how the Asterisk channel structure works and the steps that need to be followed to make a call in Asterisk at an application level. This proved to be quite a challenging task as there is no documentation on the Dial application apart from the sparse commenting in the source code and some very basic comments and discussions on the Asterisk forums. Thus the core operations that the Dial application performs had to be identified and understood by reading their source code before methods to re-route the session via a different link could be developed and implemented.

The remainder of this chapter will discuss the core functions used within the Dial application, the `ast_channel` structure and conference rooms.

3.1 The Dial Application and the Application API

The Dial Application is an Asterisk module or application written in C that allows for calls to be made between two devices via Asterisk. The Dial application is typically used by calling it from the Asterisk Dial Plan as shown below, but can also be called from within other Asterisk applications.

Extract from Asterisk Dial Plan (/etc/asterisk/extensions.conf):

```
exten => 9000,1,Dial(SIP/200,15)
exten => 9000,102,Voicemail(u200)
```

The Dial Application accepts several parameters. The first parameter of the parameters relevant

to this project is information needed to dial a user on a specific channel and the second parameter is the timeout value. The Dial application will dial the user and if the timeout value has been reached without the user having answered the device, the Dial application will exit. The Dial Application performs various checks and billing functions, which will not be discussed in this chapter; instead, the chapter will focus on the core activities that the Dial Application performs while setting up the call.

When an application is executed within Asterisk, it is passed two parameters: a pointer to the channel that called or owns the application, and a string of parameters passed from the Dial Plan. The Dial Application first tokenizes the input string that it received to extract the information needed to build the outgoing channel(s) as well as the timeout value. Once the Dial application has extracted the information needed from the input string, the core activities take place as illustrated in the flow chart in Figure 3.1.

The first function which is called is the `ast_request(char* channel_information)` function which will ask the Asterisk core to establish an outgoing channel and pass a pointer to the new channel created back. Once the outgoing channel has been established the `ast_call (struct ast_channel *chan, char *destination)` function is called which will establish a call to the end device using the channel and destination string passed to it. Next the `ast_channel_make_compatible(ast_channel *incoming, ast_channel *outgoing)` is called, which will find the lowest common denominator between the incoming and outgoing channels so that unnecessary work is not done by decoding and re-encoding information that was in the same format to start with and so that the two channels can intercommunicate. Next the `ast_bridge_call()` function is called which first performs some housekeeping work before it enters an infinite `for` loop which inspects each `ast_frame` to see what type of frame it is. Depending on the type of frame, the function will perform various tasks. First the function will check to see if the frame is a control frame or a special DTMF (Dual Tone Multi Frequency) tone frame such as a '#'. If the frame is a control frame or a special DTMF frame, the appropriate action is performed, else the frame is merely copied to the peer channel using the `ast_channel_bridge()` function. eg. If the frame is of type `AST_FRAME_CONTROL` and has a sub-type `AST_CONTROL_HANGUP`, then the function will break out of the infinite `for` loop and return to the calling function. The calling function, `dial_exec()` in the Dial application, will then check to see if the channels have been hung up and if not do a hangup on the channels, perform further cleaning up if necessary and exit the application. The `ast_channel_bridge` can be conceptually thought of as a bridge that picks frames up from the incoming channel and copies it to the outgoing channel as illustrated in Figure 3.2.

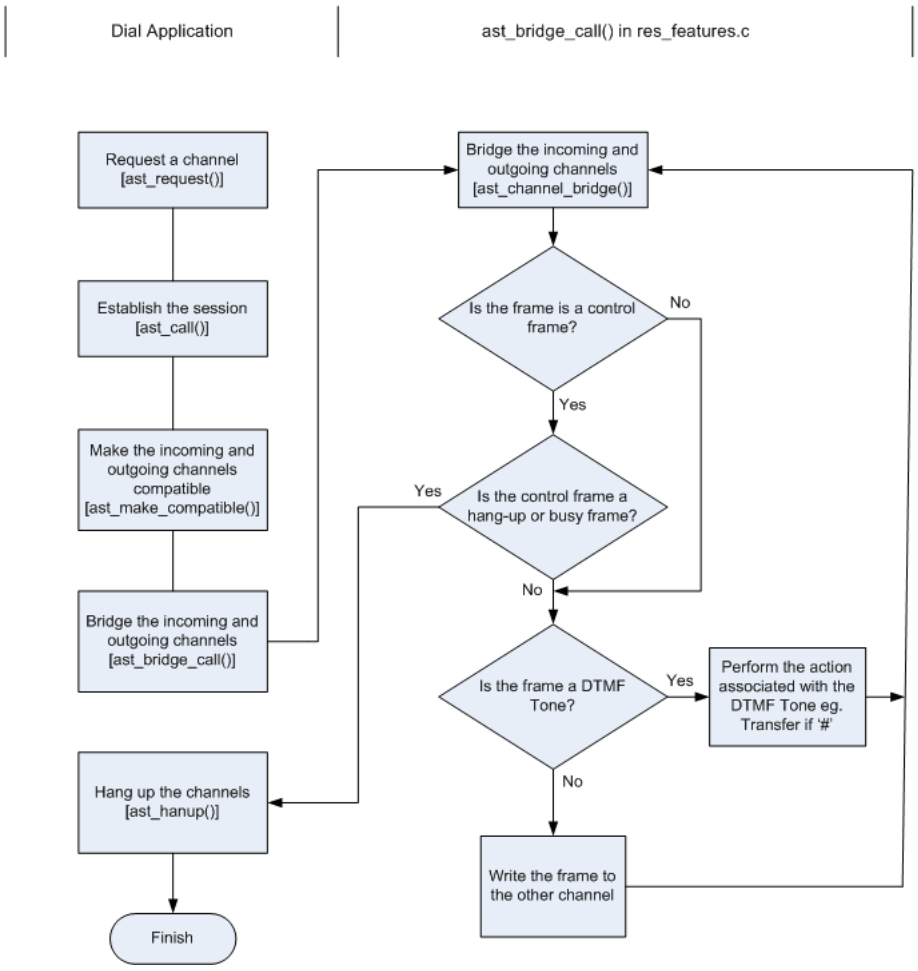

Figure 3.1: The actions performed to set up a call within Asterisk

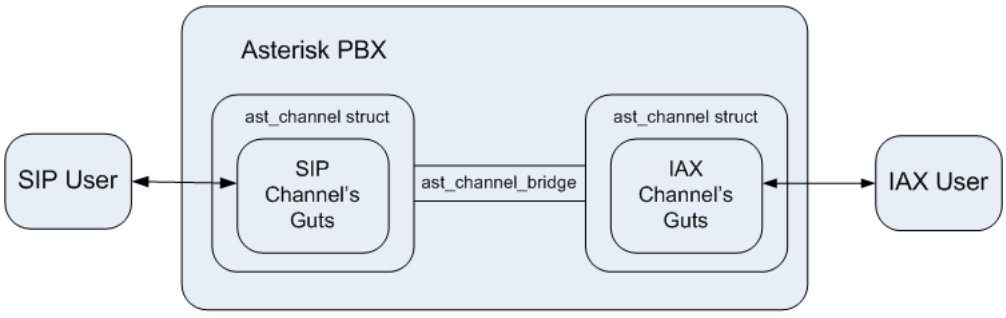

Figure 3.2: The Asterisk Channel Bridge

3.2 ast_channel struct and key functions

This section discusses the members of the `ast_channel` structure that have been used in this project.

```
struct ast_channel{
    ...
    char *name;
    char *type;
    struct ast_channel *dialing;
    int zombie;
    int _state;
    int readformat;
    int writeformat;
    char *callerid;
    ...
};
```

The extract of the `ast_channel` struct above only contains the members of the struct that were used in the implementation of the project. The complete version can be found in the `channel.h` file located in the `./include/asterisk/` directory within the Asterisk source directory.

The `name` is used to uniquely identify a channel, eg. SIP/2000-ef12. The first part, in this case, indicates which SIP users the channel is assosiated with and the hexidecimal digits (ef12) uniquely identify the channel as more than one call, and hence channel, can be made to the same user. `type` contains the type of the channel as each type of channel has its own implementation of a channel. The `*dialing` pointer points to the `ast_channel` that the current channel is calling or dialing. The `zombie` variable is set to indicate whether the channel is a zombie or not. A zombie channel is a channel that has a channel structure allocated, but has no guts. The `zombie` variable is used during channel masquerading as will be discussed later in the paper. The `_state` variable indicates the current state of the channel and can contain numerous values as defined in the `channel.h` file. Some of these states are: `AST_STATE_DOWN`, `AST_STATE_RING`, `AST_STATE_BUSY`, `AST_STATE_UP` and will be used in a discussion later.

3.3 Conference Room

In real life a conference room is an area where two or more people can meet for consultation, discussion and/or the interchange of opinions. In the world of telephony, a conference room has the same purpose and users can join a virtual conference room by dialing an extension from their telephone. Previously one could only join voice conferences, but video conferencing has become a more and more popular way of meeting.

Asterisk has built in conferencing facilities which work for voice and experiments are currently being conducted in the department to see whether video conferencing via Asterisk is possible. A conference room can be set up very easily within Asterisk by modifying the `meetme.conf` and `extensions.conf` files within the asterisk configuration directory - /etc/asterisk. One has to define a conference room in Asterisk by inserting `conf => <number>` into the `meetme.conf` file. The number of the conference room is used when defining an extension to be associated with the conference room in the `extensions.conf` file.

One of the possible architectures for implementing cross-protocol re-routing within Asterisk made use of a conference room. By simply adding a call to one of the user's device's interfaces, such as SIP, to the conference room one could easily re-route the session to another interface on the user's device, such as GSM. This re-route could be done by adding the GSM call to the conference room and then removing the SIP call from the conference room. Thus the conference room effectively acted like a switchboard where the outgoing links could be patched easily.

The following 3 subsections will discuss why a Zaptel card is needed for by the conference room in Asterisk and how to install a Zaptel card.

3.3.1 The purpose of Zaptel cards within conference rooms

Zaptel cards come in many forms, from ISDN BRI (basic rate interface) cards to fxo (foreign exchange office) cards that can be used to connect to normal POTS (plain old telephone service) exchange. The conference rooms require a Zaptel card as the Zaptel cards have a chip on them which is dedicated to producing an 1000Hz clock frequency used when mixing the media from all the channels that are currently part of the conference. Conventional computers have chips that can produce these clock frequencies, but the Linux 2.4 kernel did not support the use of the clocks by applications other than the operating system.

ztdummy can be used to provide the 1000Hz clock frequency needed by some of the applications within Asterisk such as the MeetMe (conference room) and Queue that the Zaptel card would normally have supplied. The ztdummy module basically acts as a Zaptel device, but only

providing the clock signal and not any other functionality that the Zaptel cards would normally provide. Under the Linux 2.4 kernel, ztdummy makes use of a clock on one of the USB controller chips to generate the clock signal, whereas one of the clocks on the central processing unit is used under the Linux 2.6 kernel.

3.3.2 Signalling - fxo and fxs

The fxo (foreign exchange office) and fxs (foreign exchange station) are protocols used in legacy telephone networks. fxo is the protocol used by stations such as an exchange to communicate with offices (eg. telephones, fax machines or modems) and fxs is used by offices to communicate to the stations. Thus an fxo card will use fxs signalling and a fxs card will use fxo signalling [18].

3.3.3 How to install a Zaptel Card

Two configuration files need to be edited to configure a Zaptel card and the Zaptel channel within Asterisk, they are `/etc/zaptel.conf` and `/etc/asterisk/zapata.conf`. The `zaptel.conf` file is the configuration file for the zaptel module that is loaded at an operating system level. The `zapata.conf` file is the configuration file for Asterisk's Zaptel channel. In the `zaptel.conf` configuration file one needs to define which signalling protocol to use on the various ports or channels on the actual card. `fxsks=1` would tell the system to use the fxs kewlstart signaling on channel 1 on the card. The `zapata.conf` file is used to tell the Zaptel channel in Asterisk which context a channel needs to be associated with as well as set the caller id of the channels among other settings. For more information on this visit the voip-info.org website [18].

Chapter 4

The DialPref Application

The DialPref application is an Asterisk application, or module, that can be plugged into Asterisk, enabling Asterisk to perform least-cost routing. The DialPref application's architecture, its components as well as transparent handovers will be discussed in the remainder of this chapter.

4.1 How DialPref fits into the Dial Plan

Least-cost routing has been implemented within Asterisk by writing an application for Asterisk which interfaces with Asterisk's application API. The application, or module, is loaded into Asterisk when Asterisk starts up and can be called from within the Asterisk Dial Plan located in the `/etc/asterisk/extensions.conf` file. The DialPref application can be called from within the dial plan as follows:

```
exten => 100,1,DialPref(preferred channel,alternate channel)
```

The preferred link will in most cases be the link with the lowest costs and the alternate link will be a link with a higher cost, but at the same time have a guaranteed Quality of Service (QoS) that can be used in the event of the preferred link being in a state not conducive to transportation of media or data. When a user dials 100 in the example above, the DialPref application will be executed and start testing the quality of the preferred link. If the quality is below an acceptable predefined threshold, the alternate link is used, else the preferred link is used. Once the session or call has been established between the two end points, the application will constantly monitor the status of the preferred link as will be discussed in more detail in Section 4.6.

4.2 Architecture

This section will cover the architecture of the DialPref application and describe the roles of its two components: the Call Handler and the Link Analyzer.

As illustrated in Figure 4.1, the DialPref application has two main components, the Call Handler and the Link Analyzer. The Call Handler is responsible for checking the status of the preferred link which is computed by the Link Analyzer. The Call Handler is also responsible for performing the re-routing of the real-time communication session at the appropriate time depending on the status of the link. The system has been designed in such a way that a Link Analyzer specific to the type of channel can be implemented and associated with that channel type. This solution however has only implemented a Link Analyzer for an IP based channel such as SIP, H.323 or IAX where the round-trip-time to the end-point can be determined by using ICMP packets as will be discussed in Section 4.6.

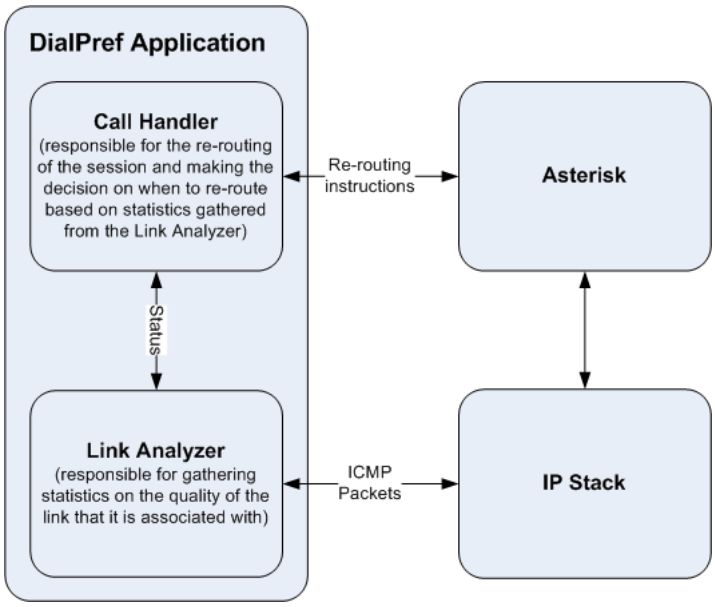

Figure 4.1: The Architecture of the solution

This solution has also made provision for the concept of a 'user', where a user may have many extensions with different priorities associated with each extension allowing one to dial a specific user rather than a physical device on the network. This concept has been implemented within iLanga, however the iLanga system will either ring all the devices associated with a user simultaneously or ring a single device until a timeout value has been reached and then try ring

the next device or until a user answers the ringing device. The DialPref application could enable the iLanga system to rather check to see if the device or extension with the highest priority is available and if it is, ring it, else check the availability of a device with a lower priority. When the application finds an available device, it will ring it and wait for the user to answer until a timeout is reached. If the timeout is reached, the application will assume that the user is unavailable and terminate. If the device's availability changes once the session has been established, the DialPref application will re-route the call to a device with a lower priority. The current implementation of iLanga has a priority field in the `userdevices` table in its MySQL database which can be used to indicate the priority of a users extension when the DialPref application is to be used in iLanga. The DialPref application currently only supports two extensions and one user.

By always trying to ring the user's device with the highest priority, one is assured that if the user answers the call, it will be answered on the device with the least cost associated with it. The configuration where all the devices ring simultaneously on the other hand allows the user to answer the call on a device which does not necessarily have the lowest cost associated with it. An example of this is where a user is in their office and both their desk and cellular phones ring. The user is most likely to answer their cellular phone as they might be too lazy to get up and answer the desk phone which in this case has the lowest cost associated with it.

4.3 The structures used

Several structures were needed to implement the concept of a user as well as allow for more than one pointer or parameter to be passed into a thread as will be discussed in this section.

4.3.1 The User structure

The `user` structure in this implementation provides for two extensions, the preferred and alternate extensions, named `extOne` and `extTwo` respectively. These extension pointers could be combined into an array of pointers if support for extra extensions is implemented. The `chan` pointer points to the incoming channel from the calling party.

```
struct user{
    char *name;
    int userID;
    struct ast_channel *chan;
    struct extension *extOne;
```

```
    struct extension *extTwo;
} ;
```

4.3.2 The Extension structure

The `extension` structure contains data relevant to each extension as shown in the code snippet below. The `status` is a value between 0 and 100 which indicates the current quality or status of the extension and is updated by the Link Analyzer associated with that type of extension. The `extensionType` and `number` are extracted from the string passed to the application when the DialPref application is called from the Dial Plan. The `upperThreshold` and `lowerThreshold` can be set for each device or extension as each and every device's networking capabilities and quality differ. The `channel` pointer points to the `ast_channel` associated with the extension and the `active` variable indicates if the session is being routed to the user via that extension or not.

```
struct extension{
    int status;
    char *extensionType;
    char *number;
    int upperThreshold;
    int lowerThreshold;
    struct ast_channel *channel;
    char *dialString;
    int active; //0 -> inactive and 1 -> active
} ;
```

4.3.3 The Call Handler Data structure

This structure seems rather trivial at first, but it allows one to simply send information to the Call Handler by adding members to the structure.

```
struct callHandlerData{
    struct user *theUser;
} ;
```

4.3.4 The Link Analyzer Data structure

The timeout value sets the time to wait between sending ICMP packets as will be discussed in Section 4.6.3. The `user` points to the user which the instance of this structure belongs to.

```
struct linkAnalyzerData{
    int timeout;
    struct user *theUser;
} ;
```

4.3.5 The Dialer Data structure

The `dialerData` structure is passed to the dialer thread when initially dialing a device as will be discussed in Section 4.5.1 and basically contains a pointer to the incoming channel and the dialstring of the device to call.

```
struct dialerData{
    struct ast_channel *incoming;
    char *data;
} ;
```

4.4 The Threading Structure

Asterisk has built in support for the use of `pthreads` and provides an `ast_pthread_create()` function that allows one to spawn `pthreads` within an Asterisk application. `Pthreads` have been used in this project to create the Call Handler and the Link Analyzer and to allow them to work independently of one another, allowing both the Call Handler and Link Analyzer to operate on the same data simultaneously. All the issues associated with threading such as locking and dead locks are not an issue for this solution as the Link Analyzer only updates the data structure, and the Call Handler only reads from the data structure.

`Pthreads` only allow one to pass a single pointer into the thread, thus using structs containing many members was essential to pass data and more than one pointer and/or variable into the threads. The Call Handler was passed a structure as shown in Section 4.3.3 and the Link Analyzer was passed the structure shown in Section 4.3.4.

4.5 The Call Handler

As mentioned earlier, the Call Handler is responsible for the re-routing of a session as well as the constant monitoring of the status of the preferred link as illustrated in Figure 4.6. The system currently only monitors the status of the preferred link and not the alternate link as it is assumed that the alternate link, in this case GSM, has a guaranteed QoS and if the end-point is unavailable on the GSM network, the probability that the device is not in the coverage of a 802.11 wireless access point is fairly high.

The remainder of this section will discuss how a session is established, the preferred link is monitored, and how a session is re-routed by the Call Handler.

4.5.1 Establishing the session

Once the Call Handler thread has been spawned by the main thread, the Call Handler thread will spawn the Link Analyzer thread and then sleep for 3 seconds, allowing the Link Analyzer to gather enough data so that the preferred link's status can be evaluated accurately. If the preferred link's status is greater than 50 after the 3 seconds have elapsed, the session is established using the preferred link, otherwise the call is established via the alternate link.

4.5.2 Monitoring the status of the preferred link

The Call Handler will check the status of the preferred link once per second, and then make a decision on whether to re-route the session or not. The decision is based on which link is currently active and what the status of the preferred link is. If the preferred link is active and the status of the preferred link has dropped below the lower threshold, the Call Handler will initiate a re-route to the alternate channel, else do nothing. If the alternate link is the active link and the status of the preferred link has risen above the upper threshold, the Call Handler will re-route the session via the preferred link.

4.5.3 Re-routing the session using channel masquerading

Several functions within Asterisk such as call parking and the `ast_async_goto()` function used during call transfers make use of channel masquerading which basically swaps the guts of two channels. After the channel masquerade has been performed, any applications which had pointers pointing to the original channel before the masquerade was performed are unaware that the channel that they are pointing to after the masquerade is not the original channel, making the

re-routing process invisible to the other applications as illustrated in Figure 4.2. This allows for any applications using the channel at the time of the re-route to continue functioning as if nothing had changed, allowing for seamless re-routing from the point of view of other applications.

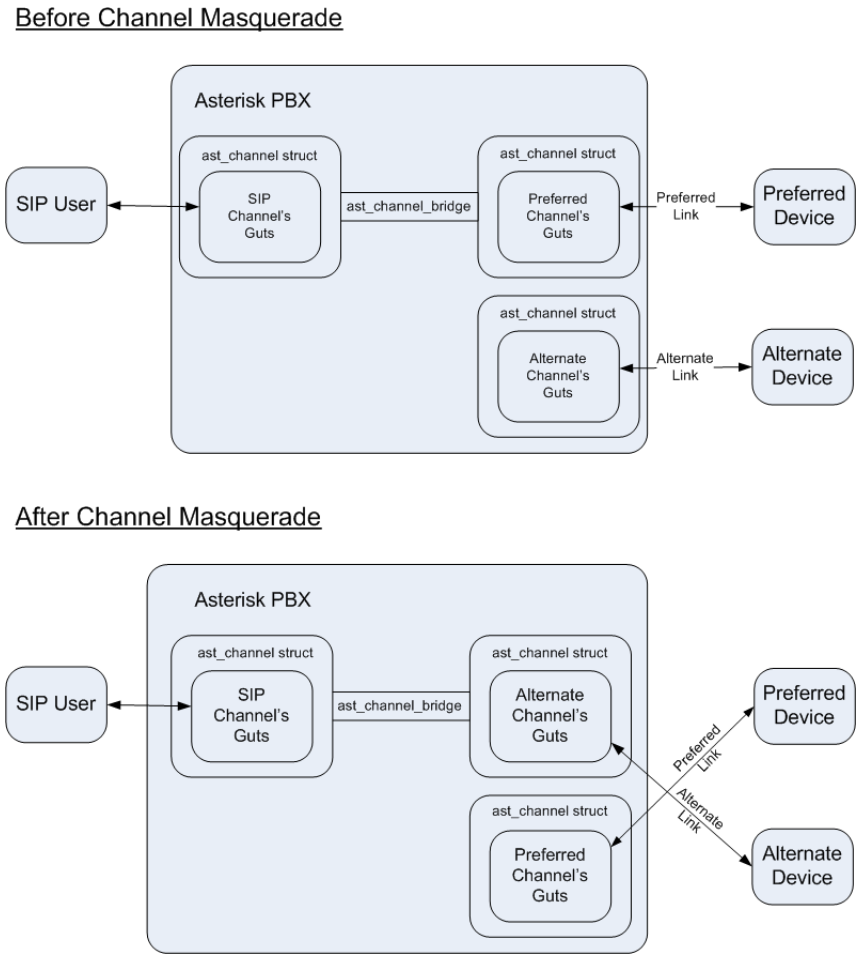

Figure 4.2: Before and after performing a channel masquerade

In order to perform channel masquerading one has to perform a sequence of actions as shown in the flow chart in Figure 4.3. One first has to create the new channel and then prepare both the new and old channel for the masquerade. To create the new channel, the `ast_request_and_dial()` function is called and passed the dialstring of the new extension. Once the channel has been created the function will ring the extension and wait for the extension to be answered. If the extension is answered, the channels are prepared for the masquerade. This first involves making the incoming channel and the new channels compatible by calling the `ast_channel_make_compatible()` function and passing it pointers to the incoming and new channel. Thereafter one has to set the

`name`, `_state`, `read_format`, `write_format` and `priority` variables of the new channel to those of the old channel. The channels are then ready to have the masquerade performed. To perform the masquerade, one first calls the function `ast_channel_masquerade()`, passing it a pointer to the new and old channels and then call the `ast_do_masquerade()` once a lock has been obtained on the new channel as illustrated in the flow chart below. Once the masquerade has been performed successfully, the new extension's `active` variable is set to active (1) and the old channel's `active` variable is set to inactive (0). After the masquerade is complete the session should be re-routed via the new link.

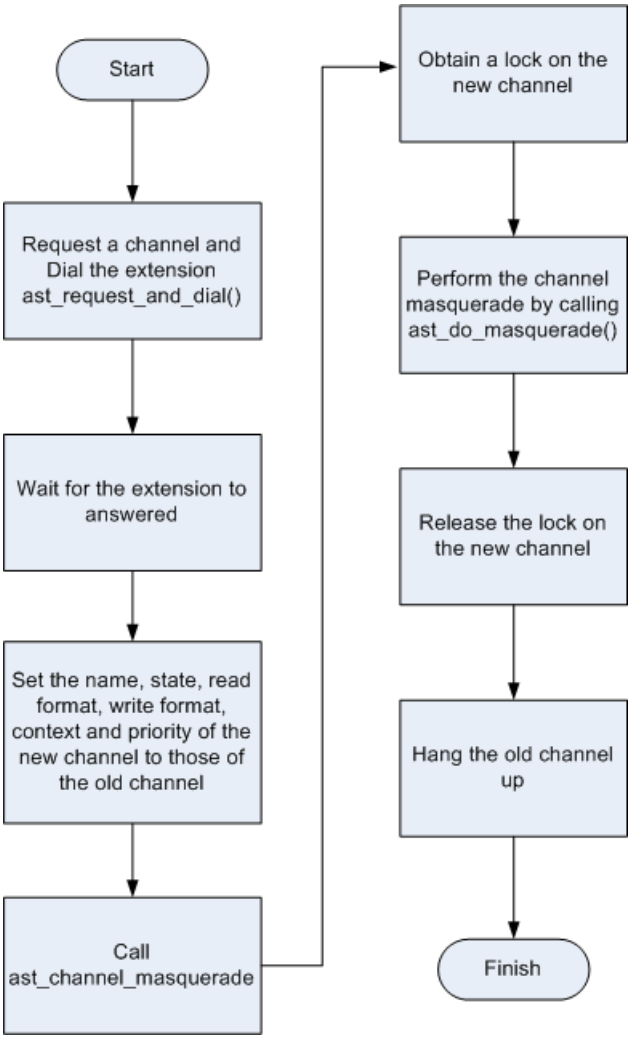

Figure 4.3: Channel Masquerade within Asterisk

4.5.4 Avoiding the ping pong effect

The ping pong effect occurs when the quality level at which the system re-routes is set at a single level, say 50%. If the quality momentarily drops below 50% and then jump above 50%, the system would re-route to the alternate link and then then back to the preferred link in a very short space of time which is undesirable.

An upper and lower threshold as well as a smoothing function was introduced to curb this problem. The Call Handler will only switch to the alternate link once the link's smoothed status dropped below the lower threshold and only switch back to the preferred link when and if the smoothed status of the link rose above the upper threshold. The smoothing function reduced the effect any sudden drops and rises, thus avoiding triggering the re-routing process unnecessarily and is discussed further in Section 4.6.4.

4.6 Link Analyzer

The purpose of a Link Analyzer is to analyze the link in terms of its quality and update a status variable associated with that link with the updated quality or status of the link. Each type of link will have its own link analyzer associated with it and each link will have a separate instance of a link analyzer to calculate the status of that link. Only a SIP link analyzer has been implemented in this project, but the supporting structures allow for link analyzers for other types of links to be plugged in quite simply. The SIP link analyzer uses ICMP echo requests and responses to determine the quality of the link and will be discussed in this section.

The remainder of this section will discuss the use of ICMP and RTCP packets in the analysis of the link, and how the results gathered from sending the ICMP packets are interpreted and smoothed by the Link Analyzer.

4.6.1 Using ICMP to calculate the status of a link

The round-trip-time or latency and packet loss of a link between an Asterisk PBX and an end-point can be calculated by sending an ICMP echo request from the Asterisk machine to the host on a IP network and time the time it takes to receive an ICMP echo reply from that host. If an ICMP echo reply as not been received after a predefined amount of time (usually 1000ms), the packet is assumed to have been lost. The round-trip-time and packet loss values are a good reflector of the status of the link and VoIP calls generally require a round-trip-time less than 300ms and a packet loss percentage less than 10%. Thus the round-trip-time and packet loss

percentages along with a smoothing function are used to calculate the status of a particular link.

4.6.2 Using RTCP to calculate the status of a link

Another way to monitor the quality of a link when using RTP streams to transport the media is to read the latency, packet loss and jitter values contained in the receiver reports available in some RTCP packets. This will avoid sending unnecessary ICMP echo request and reply packets between the end points as the information that is gathered during the process of sending ICMP echo request and reply packets (packet loss and latency) is already contained in the RTCP receiver reports. The RTCP receiver reports are also received more frequently than ICMP packets can be sent and received in the SIP link analyzer, thus producing a finer grained analysis of the quality of the link.

RTCP sender and receiver reports are send between the RTP end points reporting on the status of the current streams between them. After much investigation it was found that Asterisk version 1.0.9 handles RTCP packets, but the `ast_rtcp_read()` function in the `rtp.c` file returns a null frame when receiving the RTCP packets. Thus one would have to modify the function so that it returns frames of type RTCP so that the frames can be intercepted by the `ast_bridge_call()` function and be passed back up to the Link Analyzer in the DialPref application for analysis.

The current implementation of the SIP link analyzer does not make use of RTCP packets as it was found that ICMP packets provided a sufficient analysis of the link.

4.6.3 Sending the ICMP Packets

An `ICMPPacketSend(char *hostAddress)` function has been created to send an ICMP echo request to an host on an IP network and wait for the ICMP echo reply from the host and then return the round-trip-time back to the calling function. The round-trip-time is calculated by subtracting the system time when the request packet was sent from the system time when the reply packet was received.

The `select()` function had to be used within the `ICMPPacketSend()` function to prevent the thread from blocking when a reply packet had been lost. The `select()` function waits for a buffer associated with a certain file descriptor, in our case the socket used to receive the ICMP echo reply packet, to be filled. If the buffer has been filled before the timeout value has been reached, the `select()` function will return a value greater than 0. If the timeout has been reached it returns 0, otherwise it returns -1 when an error has occurred. Based on these return

values, the function will set the value of the latency which is returned to the calling function. If a 0 is returned by the `select()` function, the actual round-trip-time is returned, else 1000 is returned, indicating that no reply packet was received or an error has occurred.

4.6.4 Interpreting the results gathered from the ICMP Packets

If an ICMP echo reply packet was received before the timeout value had been reached, the formula below is used to calculate the raw status of the link, else the raw status was set to 0. In the following formula, X indicates the raw status, λ is the latency or round-trip-time in milliseconds and r is the round-trip-time limit in milliseconds:

$$X = 100 - ((\lambda/r) * 100)$$

The round-trip-time limit may be adjusted to change the sensitivity of the system, the lower the round-trip-time limit, the more sensitive the system will be to changes in the latency. The typical value for the round-trip-time limit in a LAN environment is 300ms as it is the highest acceptable round-trip-time for VoIP sessions in LAN environments. The latency is calculated by the `ICMPPacketSend()` function discussed earlier in Section 4.6.3.

Owing to the nature of the wireless networks and access points used in the experiments, the quality of the network stayed relatively constant at an acceptable level of around 90%. But when the signal strength between the wireless access point and the endpoint decreased below a certain level, a sudden drop-off was observed when packets were being lost as illustrated Figure 4.4. These areas are referred to as “dead” spots or shadow zones.

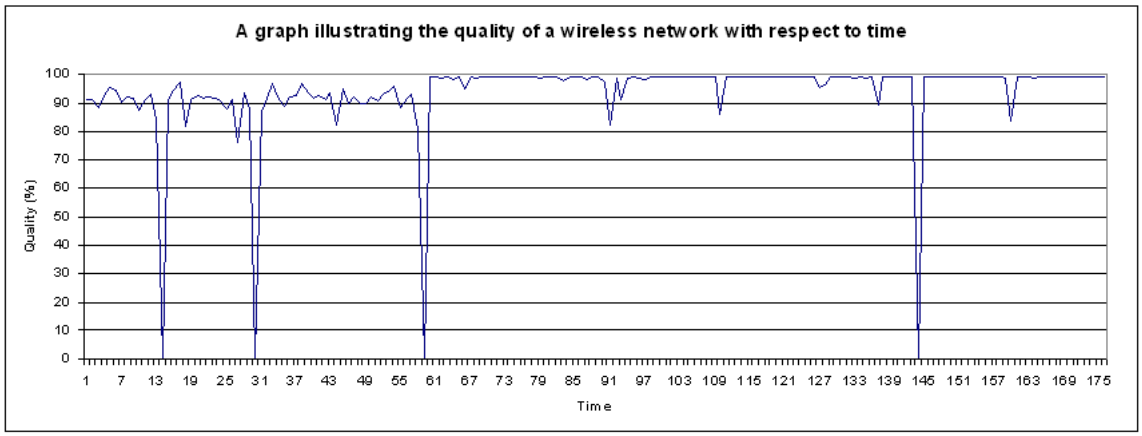

Figure 4.4: A graph illustrating the quality of a wireless network with respect to time

The raw status values obtained by using the formula discussed earlier in this section were very erratic and did not show any definite trend, so it was decided to use a single exponential smoothing function to smooth the raw status. A property of single exponential smoothing is that the weight of the previous values decrease exponentially [16] which helps decrease the effect "dead" spots. The following exponential smoothing function was used where S indicates the smoothed status, X indicates the raw status and α is a value greater then 0 and less than 1.

$$S_n = \alpha X_n + (1 - \alpha) S_{n-1}$$

By applying the smoothing function to the dataset, the effect of the sudden drop-offs caused by lost packets was reduces and a more usable dataset was generated which was used to indicate the status of the link as shown in Figure 4.5.

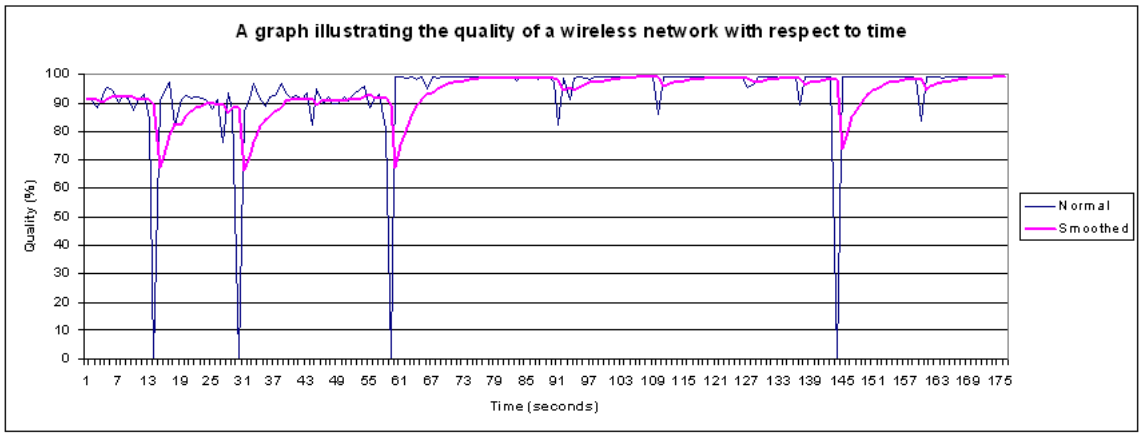

Figure 4.5: A graph illustrating the quality of a wireless network with respect to time with smoothing applied

4.7 Transparent hand-overs

The DialPref application only handles the server side of the hand-over or re-routing process and not the client side. In order to achieve a re-routing process that appears seamless or transparent to the users, the client application would have to be involved in the re-routing process as the new link needs to be answered and the old link needs to be hung-up on the client side. With the current server-side implementation of the solution the end user has to physically answer the new link, say GSM, and physically hang the old link up, say SIP, which does not allow the process to

appear seamless to the user.

To get the server and client to co-ordinate the hand-over from one link to the other, the client and server would have to communicate using some protocol such as TCP if on an IP based network. The server will have to tell the client that it is about to perform a hand-over from one link to the other, for example SIP to GSM. This will enable the client to match the caller id of the incoming call on the GSM interface with the caller id of the server. If the caller ids match, the chances are very good that the call on the GSM interface is that of the re-routing process that the server has just performed. The client application can then automatically answer the new call on GSM and hang the old call up on SIP, completing the re-routing process and making the re-routing process seamless.

4.8 The DialPref application's flow chart

The flow chart in Figure 4.6 below shows the flow of events that occur from the time that the DialPref application is executed until it is terminated. The Main thread is on the left, the Call Handler thread in the middle, and the Link analyzer thread on the right of the flow chart in Figure 4.6. All three threads run independently of one another, but operate on the same data set allowing for inter-communication between the threads. The Link Analyzer is the only thread that updates the status of the preferred link and the Call Handler reads the status of the link and the Call Handler then bases its decision on whether to re-route or not depending on the value of the preferred link's status as well as which link is active at the time.

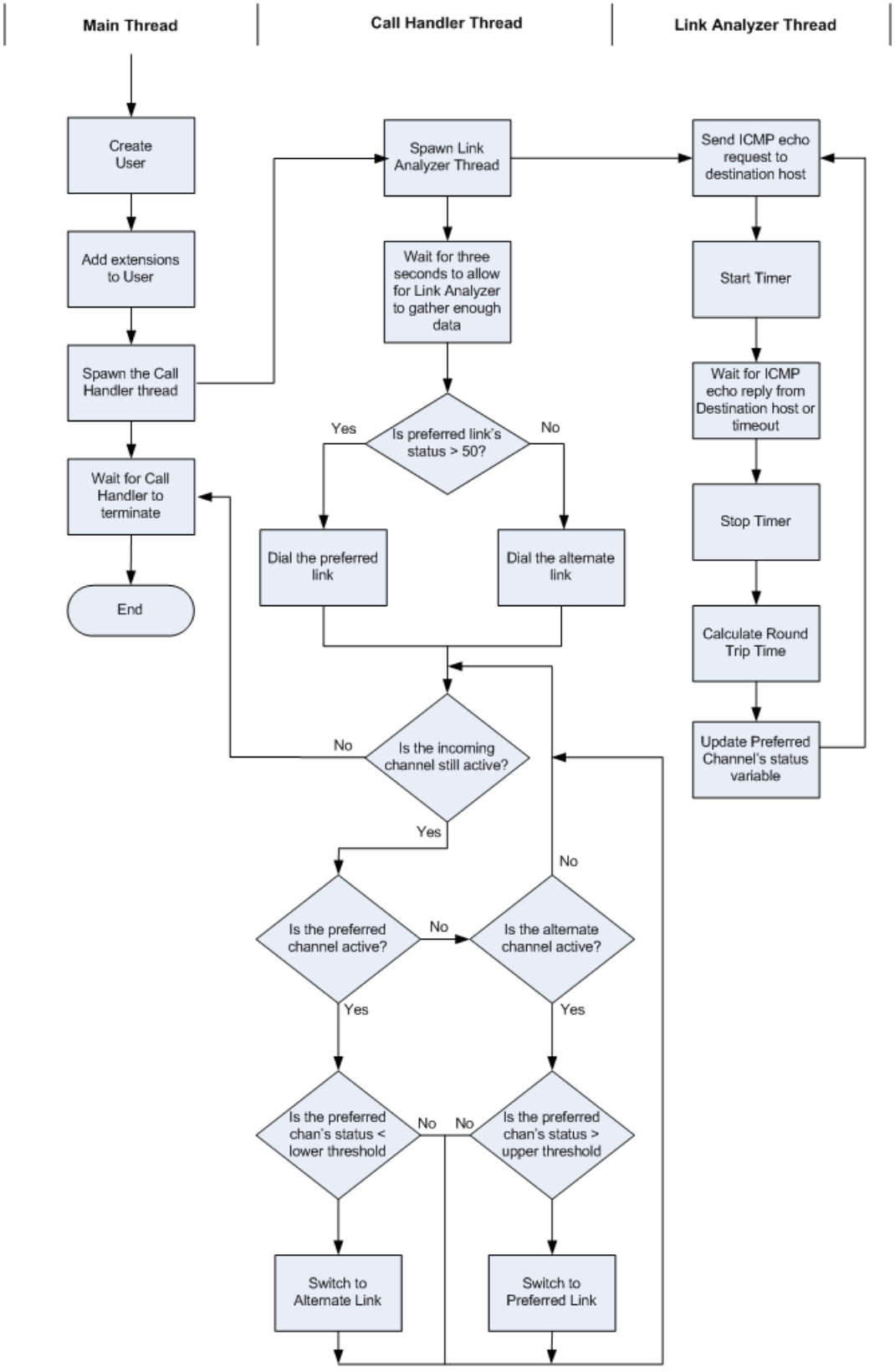

Figure 4.6: Flow chart of the DialPref application

Chapter 5

Testing the solution

Two environments were used in the testing of the solution (the DialPref application). The first was a test network that was setup in the lab and the second made use of the wireless network on the Rhodes University campus. Two mobile clients, either a HP iPAQ h6340 PDA [6] or an imate PDA2K PDA [9] with both a 802.11b interface and a GSM interface were used in the experiments. Once SJ Phone - a SIP client for Pocket PC - was installed on the PDAs, the PDAs were both a GSM phone and a VoIP phone. The other combination of GSM and VoIP phones used to simulate an end point were a PulverInnovations WiSIP [12] phone and a Nokia 6670 cellular phone [10]. The WiSIP phone is a wireless (802.11b) SIP phone used for the testing the SIP link and the Nokia 6670 is a GSM cellular phone and was used to test the GSM link.

The computer based PBX was running Gentoo as its operating system and had Asterisk version 1.0.9 installed on it. By plugging the DialPref application into Asterisk, Asterisk was able to perform least-cost routing between SIP and another protocol such as IAX. The current implementation only allows for the preferred link to use the SIP channel.

The remainder of this chapter describes the testing processes used, and the results obtained during the tests performed in the two environments.

5.1 Testing in the lab

5.1.1 Disconnecting the wireless access point

The first test was performed using the imate PDA2K PDA as the client. This test involved disconnecting the Test wireless access point that the PDA was connected to from the wired network which simulated 100% packet loss. This proved to be a rather drastic way to simulate

packet loss, but it guaranteed a 100% packet loss to initially test the re-routing process. The only problem with this test was that the break in the network connection was between the wireless access point and the computer which did not reflect reality where the break would normally occur between the wireless device and the wireless access point. When the break occurs between the access point and the mobile device, other problems are introduced such as associating the mobile device with an access point.

Once the system worked by detecting packet loss, it was decided to rather move the mobile device out of the wireless access point's coverage to not only test the effect of packet loss, but also the effect of latency on the Link Analyzer. This then simulated a break in the network connection between the wireless device and the wireless access point, as will be discussed in the following section.

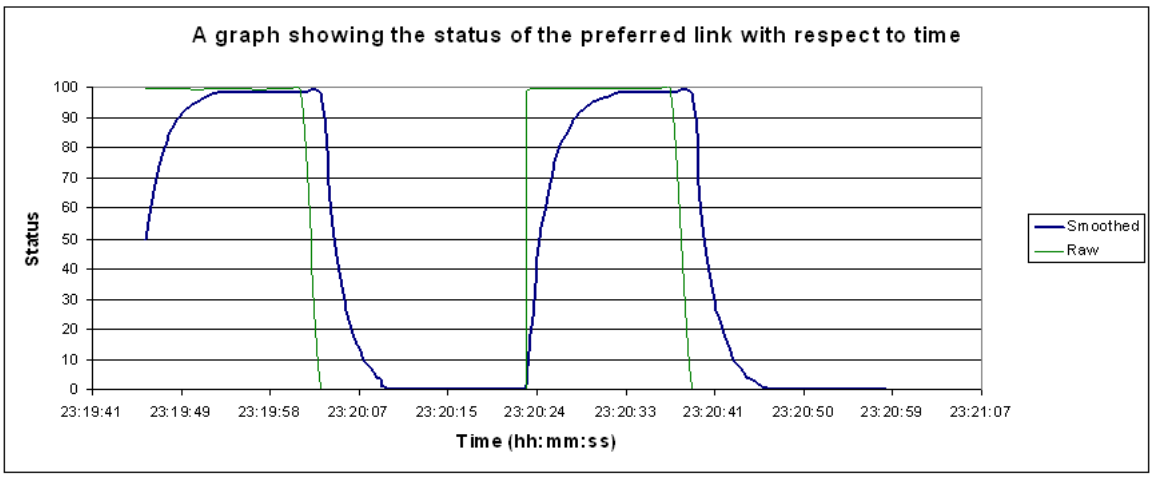

Figure 5.1: A graph showing the status of the preferred link with respect to time when the access point was unplugged from the network

5.1.2 Walking out of range

The second test was performed by connecting the PDA to the Test access point in the lab, establishing a call and then walking out of the wireless network's range. Next, a session was established whilst outside of the wireless network coverage and then entering the coverage of the wireless network and observe how the re-routing takes place.

The results were better than expected as the system actually started the re-routing process before completely losing the wireless connection. This showed that by choosing the correct α value for the smoothing function (in this case 0.4) the system could actually predict when the

device is close to the boundary of the wireless network or hot-spot and initiate the re-routing process. The smoothing function also decreased the effect of the so called “dead” spots within the wireless coverage as the weight of the previous status measurements decrease exponentially as discussed in Section 4.6.4.

The graph in Figure 5.2 illustrates the results obtained from walking in and out of the coverage of the Test wireless network. It can be seen that the sudden drop-offs caused by “dead” spots or shadow zones in the wireless network were taken into account, but did not have a major impact on the status of the network. These drop-offs occur randomly within the wireless network and is effected by the physical environment that the wireless network operates in. However, “dead” spots do tend to occur more frequently at the edges, so the fact that they can assist in predicting when the mobile device is near the edge of the wireless network cannot be ignored.

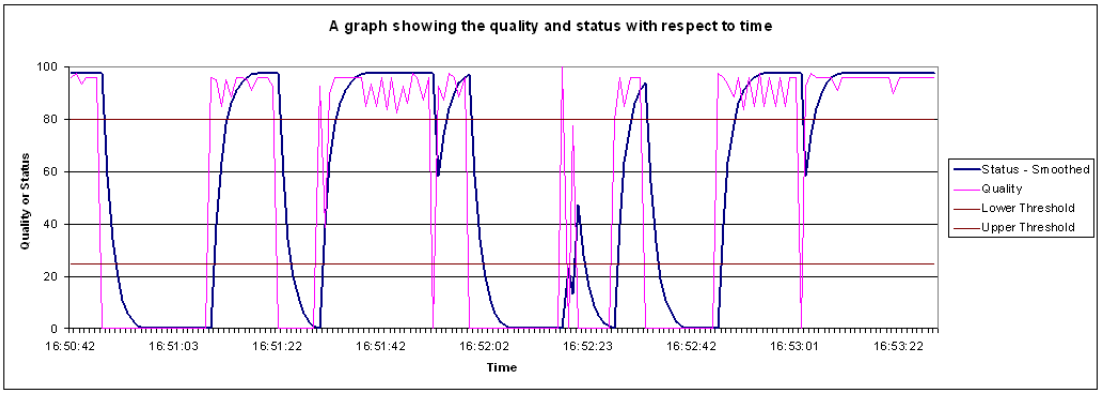

Figure 5.2: Results obtained when performing a walk around while the PDA was attached to the Test wireless access point

5.2 Testing around campus

Rhodes campus has several wireless hot-spots which can be used to connect to the Rhodes network. The Rhodes wireless network was used to test the system by entering and leaving various hot-spots on campus. This proved rather problematic with the PDAs. When the PDA lost the Rhodes access points’ signal it searched for the next accessible wireless access point and tried to connect to it. Thus when re-entering the coverage of a Rhodes access point, the PDA did not automatically connect to that Rhodes access point and one had to explicitly choose and connect to the proper, Rhodes, network. In an attempt to avoid this problem, a dedicated wireless SIP (PulverInnovations WiSIP) phone was used as the VoIP phone as one could specify the SSID

(Service Set Identifier) of the access points to connect to and a Nokia 6670 was used for the GSM link. The system now re-routed the call or session to VoIP when entering the hot-spot and once approaching the edge of the hot-spot when leaving, the quality of the call started deteriorating as packets were being lost more frequently. Before the call was lost, the system re-routed the call to the GSM phone. The re-routing also worked well when one entered the wireless network with the WiSIP phone and the re-route back to the preferred link took approximately 4 seconds. Figure 5.3 shows the results obtained when walking around campus while a session had been established.

The graph illustrated in figure 5.3 below shows the status of the preferred link while walking through the various hot-spots on campus. When the status rose above the upper threshold, 80, the session was routed to the preferred link, else if the status dropped below the lower threshold, 25, the session was routed via the alternate link. The "dead" spots in the wireless hot-spots' coverage, where packets were lost, did not have a big effect on the status of the preferred link, thus avoiding the ping pong effect effectively, yet still detecting the edge of the wireless hot-spots where packet loss frequency increased substantially.

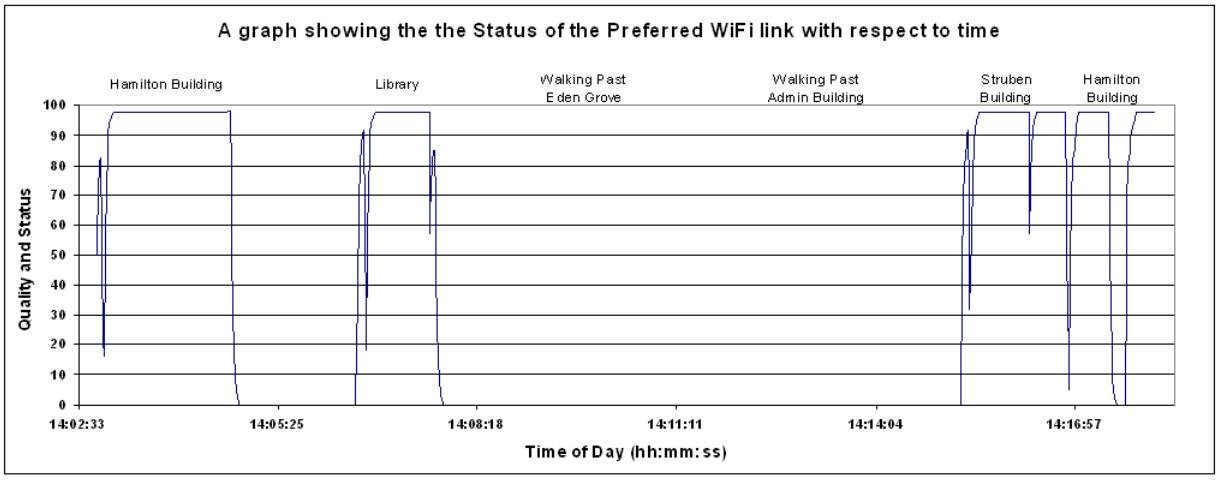

Figure 5.3: Results obtained when performing a walk around while the wireless SIP phone was attached to the Rhodes wireless network

5.3 Equipment related issues

When sending ICMP echo requests to the HP PDA as well as the PulverInnovations WiSIP phone while the devices' network interfaces did not have a load on them, the return-trip-times were

around 800ms and 250ms respectively. As soon as the devices' network interfaces were loaded with a VoIP call, the return-trip-times dropped to around 3ms under good wireless networking conditions. The only explanation for this is that the devices must have some kind of automatic network performance scaling in order to save battery power as 802.11 is known to consume a lot of power. This made it very difficult to use the initial return-trip-times to calculate the initial status of the link as the devices were not assisting in reflecting the true condition of the link by returning an ICMP echo reply immediately. The imate PDA however has manual wireless network performance scaling and so this problem did not occur with the imate.

One problem that the imate had that the HP and WiSIP phone did not have was an extremely broken up audio transmission despite near perfect received audio. When browsing forums related to this problem, people mentioned that the problem could be caused by poor quality microphones in the imate PDA, but this fact had not been proven.

5.4 Overlapping calls

During the tests that involved walking out of the range of the wireless access points, it was found that by choosing a good round-trip-time limit and α value, the Link Analyzer could accurately estimate when one was close to the edge of the wireless network and start the re-routing process. A round-trip-time limit of 300ms and an alpha value of 0.4 was used during the tests. The re-routing process from SIP to GSM took approximately 10 seconds from the time the decision was made to switch until the the GSM phone rang. During the re-routing process, communication via the SIP link was possible even though the call was very broken. Once the GSM call had been answered, the session was re-routed from the SIP channel to the GSM channel immediately, and the time that elapsed during this process was not noticeable to the user, thus no overlapping is necessary.

One could try improve the system by informing the caller that the re-routing process is taking place and that the call will be "re-established" in a few seconds using a voice over. But the time taken to inform the other person of the re-route may be longer than the actual re-routing process takes, making the voice over rather pointless as it will increase the re-routing time.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

As a whole the project achieved its goals, and least-cost routing has been proven to be possible within Asterisk. The cost that determines when to re-route may be a financial cost, but could also be any other cost such as the security of the link between the end points. If a less secure link is used, a higher cost could be associated with that link than the cost of using the most secure link. The following sections will more specifically conclude the core areas of the project.

6.1.1 The Call Handler

The use of channel masquerading to perform the actual re-routing of the real-time communication sessions worked well, as the time taken to do the actual preparation and masquerade is less than 500ms. The use of channel masquerading proved to be stable as long as the correct checks on the channels were performed before the masquerade. The Call Handler also integrated nicely with the Link Analyzer and the separation into separate threads assisted in the smooth co-functioning of the two.

The other option was to make use of a Conference Room as a switchboard. The conference room had a delay associated with it which seemed to be introduced when the various channels' audio or media that are part of the Conference Room were mixed. This delay reduced the quality of the communication session and a Zaptel card or ztdummy had to be set up before a conference room could be used. This did not provide a clean solution and so it was decided not to use the conference room for the re-routing process.

6.1.2 The Link Analyzer

The concept of a Link Analyzer proved to be effective, however the actual implementation of the SIP link analyzer used in this project has room for improvement. The use of ICMP echo request and reply packets works, but are not necessarily the most effective method of determining the status of an IP-based link. The effectiveness of the use of ICMP echo packets decreased as the end-points such as the HP PDA and PulverInnovations WiSIP phone perform automatic performance scaling on the wireless network, which directly effects the accuracy of the calculation of the status of the link by the link analyzer.

6.1.3 The equipment used

The PDAs worked well in a very controlled laboratory environment where there was only one wireless access point with a good signal strength. When the PDAs were moved outside of the lab into an environment with many wireless access points, the PDAs tried to connect to other non-Rhodes wireless access points as there is no way to explicitly state which SSID to use and as a result the re-route back to the preferred link did not occur when entering the wireless hot-spots. The PulverInnovations WiSIP phone allows one to explicitly state which SSID to use and so it was always scanning for the correct wireless networks, which improved its effectiveness and efficiency as it immediately connected to the correct network when re-entering one of the Rhodes wireless hot-spots.

6.2 Future Work

The area of cross-protocol re-routing has several sections in itself that need research done as discussed below.

6.2.1 Adding support for RTCP to the SIP link analyzer

RTCP messages are sent between RTP senders and receivers to inform each other about the currents status of the end-points and the streams flowing between them. The receiver reports in RTCP packets contain the values such as jitter, latency and packet loss percentages which can be used to better determine the status of the session between the end-points once a session has been established. This will allow the Link Analyzer to more accurately calculate the status of the link and avoid creating unnecessary traffic on the network by sending ICMP echo requests.

6.2.2 Improving the DialPref application

Several functions need to be added to the DialPref application if it is to be used in a production environment.

Support for billing needs to be added to the DialPref application as the Dial application is currently used by the DialPref application to set up the initial session. Thus after a re-route, the user will be billed using the incorrect model which will cause the system to not accurately measuring the users usage. To solve this problem, the billing module needs to be informed when a re-route has taken place, so that the billing model used can be updated.

A second improvement that can be made to the DialPref application is to create a configuration file where parameters such as the upper and lower threshold, alpha value and round-trip-time limit can be set.

6.2.3 Develop link analyzers for other channel types

The supporting structures within the DialPref application allow for different link analyzer to be plugged into the DialPref application. One could develop link analyzers for other protocols such as IAX and H.323 where information on the status or quality of the link can easily be determined.

One could also develop sensors that feed data back to a link analyzer. Such a sensor could be a device that detects when a mobile device such as a cellular phone is within the range of the sensor. When it is, the session can be re-routed to another device such as a wireless SIP phone or a normal wireless phone. These sensors would be used with protocols or channels where the status of the link cannot be calculated easily such as a Zaptel card.

6.2.4 Develop an intelligent client that assists in the functioning of least-cost routing

An intelligent client needs to be developed before cross-protocol re-routing can function seamlessly. Currently the server performs the back-end re-routing and the user has to physically answer the one device or interface and physically hang the other device or interface up to re-route the session. If the server and client could communicate to co-ordinate the re-routing process, it would allow for close to seamless if not seamless re-routing of the communication session.

References

[1] K. BROWN, *The rtcp gateway: scaling real-time control bandwidth for wireless networks*, Computer Communications, 23 (2000), pp. 1470–1483.

[2] A. CALVAGNA, A. LA CORTE, AND S. SICARI, *Mobility and quality of service across heterogeneous wireless networks*, Computer Networks, 47 (2005), pp. 203–217.

[3] CISCOSYSTEMS, *Guide to Cisco Systems' VoIP Infrastructure Solution for SIP*, http://www.cisco.com/univercd/cc/td/doc/product/voice/sipsols/biggulp/bgsip.pdf.

[4] S. GRUBER, J. REXFORD, AND A. BASSO, *Protocol considerations for a prefix-caching proxy for multimedia streams*, Computer Networks, 33 (2000), pp. 657–668.

[5] J. HARDEN, P. FENTON, M. HEHIR, AND E. DUGGAN, *Voip and wireless networking*, tech. rep., 4BA2 - Technology Survey, http://ntrg.cs.tcd.ie/undergrad/4ba2.05/group9/.

[6] HEWLETTPACKARD, *HP iPAQ h6340 Pocket PC Product Specification*, http://h10010.www1.hp.com/wwpc/za/en/sm/WF06b/1781533-1781535-1781535-1781535-1781563-4873191-11568027.html.

[7] J. HITCHCOCK, *Decorating asterisk: Experiments in service creation for a multi-protocol telephony environment using open source tools*, Master's thesis, Department of Computer Science, Rhodes Unoversity, RSA, www.cs.ru.ac.za.

[8] C. HUITEMA, *RFC 3605 (Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP))*, ftp://ftp.is.co.za/mirror/ftp.isi.edu/rfc/rfc3605.txt, October 2003.

[9] IMATE, *imate PDA2K Product Specification*, http://www.imate.com/detailspda2k.htm.

[10] NOKIA, *Nokia 6670 Product Specification*, http://europe.nokia.com/nokia/0,,63733,00.html.

[11] J. PENTON AND A. TERZOLI, *iLanga: A Next Generation VoIP-based, TDM-enabled PBX*, South African Telecommunications Networks and Appliances Conference, spiers ed., September 2003.

[12] PULVERINNOVATIONS, *PulverInnovations WiSIP Phone Product Specifications*, http://pulverinnovations.com/wisip.html.

[13] J. ROSENBERG, H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY, AND E. SCHOOLER, *RFC 3261 (SIP: Session Initiation Protocol)*, ftp://ftp.is.co.za/mirror/ftp.isi.edu/rfc/rfc3261.txt, June 2002.

[14] H. SCHULZRINNE, S. CASNER, R. FREDERICK, AND V. JACOBSON, *RFC 3550 (RTP: A Transport Protocol for Real-Time Applications)*, ftp://ftp.is.co.za/mirror/ftp.isi.edu/rfc/rfc3550.txt, July 2003.

[15] H. SCHULZRINNE AND J. ROSENBERG, *Internet telephony: architecture and protocols - an ietf perspective*, Computer Networks, 31 (1999), pp. 237–255.

[16] P. SOUSA AND V. FREITAS, *A framework for the development of tolerant real-time applications*, Computer Networks and ISDN Systems, 30 (1998), pp. 1531–1541.

[17] M. SPENCER, M. ALLISON, AND C. RHODES, *The asterisk handbook*, March 2003.

[18] VOIP INFO.ORG, *Asterisk config zaptel.conf*, voip-info.org, http://www.voip-info.org/wiki-Asterisk+config+zaptel.conf.

[19] J. YIN, X. WANG, AND D. P. AGRAWAL, *Modeling and optimization of wireless local area network*, Computer Communications, 28 (2005), pp. 1204–1213.

Appendix A

The DialPref application

The Code on the following page is an extract from app_dialpref.c - a least-cost routing application for Asterisk.