University of Western Australia

Electrical Engineering Final Year Project

Real-time Quality Monitoring and Control of Voice over IP

Todd Bayley
10326351

Supervisor: Bijan Rohani

Western Australian Telecommunications Research Institute (WATRI)

October 25, 2007
Dear Sir,

I submit to you this dissertation entitled “Real-time Quality Monitoring and Control of Voice over IP” in partial fulfilment of the requirement of the award of Bachelor of Engineering.

Yours faithfully,

.........................

Todd Bayley
Abstract

Voice Over IP is an increasingly popular replacement for traditional telephone service. Due to limitations of IP networks in regards to real time data, monitoring the quality of a VOIP call is essential in being able to provide an acceptable level of service. Current systems do not provide either the accuracy or the ability to run in real-time and so a new method of quality measurement is needed. The method proposed is to use a feedback loop between the two VOIP clients which will enable the sender to recreate the audio that was received by the receiver. This can then be compared to the audio that was sent by using an objective measurement algorithm and a quality score equivalent to a Mean Opinion Score can be calculated. This method is shown to provide accurate and almost real-time results over a variety of tested network conditions. As a result of this success an attempt was made to actively control the quality of the call to a specified target. The control system used is very simple, but the results show that it is possible to control a call to a target quality level. The ability to both actively monitor and control individual calls allows VOIP providers to have a much greater ability to control their network and enables them to minimise the bandwidth needed per call while still maintaining an acceptable quality.
Acknowledgements

I would like to first thank my supervisor Bijan Rohani for all his help with this project and for coming up with the original ideas this thesis is based on. For his technical and various other help I would like to thank WATRI Engineer Anders Johannson who also wrote ASPL of which much of this project is reliant on. Finally I would like to thank everyone at WATRI and all the others who helped me with some aspect during my project.
Contents

1 Introduction .......................... 1
   1.1 Voice over IP ..................... 1
   1.2 The Problems .................... 2
   1.3 Quality Measurement ............. 2
   1.4 Quality Control .................. 4

2 Background ......................... 5
   2.1 Quality Measurement ............. 5
   2.2 Feedback System .................. 6
   2.3 Implementation .................. 7
      2.3.1 The Acoustic Signal Processing Laboratory (ASPL) .............. 7
      2.3.2 New Components ............... 8
      2.3.3 libpesq ...................... 9
      2.3.4 libvoip ...................... 10
      2.3.5 Problems Encountered During Implementation .................. 16
   2.4 Network Simulation ............... 18
      2.4.1 Modeling Network Conditions .......... 18
      2.4.2 Netsimrt ..................... 20
   2.5 Control ........................ 21
      2.5.1 Control Strategies .............. 24
3 Results

3.1 Simulation Setup ................................................. 25
  3.1.1 Test Data ....................................................... 26
  3.1.2 Running the Simulation ..................................... 27
  3.1.3 Creating the Results ...................................... 29

3.2 Measurement of VOIP Quality ................................. 30
  3.2.1 Accuracy of Segment based PESQ ......................... 30
  3.2.2 Effects of Loss and Delay in Feedback Loop .......... 31
  3.2.3 Ideal Network Conditions ................................. 33
  3.2.4 Constant Delay’s ............................................ 33
  3.2.5 Variable Delays ............................................. 34
  3.2.6 Packet Loss .................................................. 35
  3.2.7 Measurement Summary ..................................... 36

3.3 Control of VOIP Quality ....................................... 37
  3.3.1 Control Algorithms ........................................ 38
  3.3.2 Control Summary ........................................... 40

4 Conclusions and Future Work ................................. 45

References .......................................................... 47

A Example Configuration File ................................... 49
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOS</td>
<td>Mean Opinion Score</td>
</tr>
<tr>
<td>PSTN</td>
<td>Public Switched Telephone Network</td>
</tr>
<tr>
<td>VOIP</td>
<td>Voice Over IP</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>MOS</td>
<td>Mean Opinion Score</td>
</tr>
<tr>
<td>ASPL</td>
<td>Acoustic Signal Processing Laboratory</td>
</tr>
<tr>
<td>WATRI</td>
<td>Western Australian Telecommunications Research Institute</td>
</tr>
<tr>
<td>POTS</td>
<td>Plain Old Telephone Service</td>
</tr>
<tr>
<td>PSQM</td>
<td>Perceptual Speech Quality Measure</td>
</tr>
<tr>
<td>PESQ</td>
<td>Perceptual Evaluation of Speech Quality</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>SIP</td>
<td>Session Initiation Protocol</td>
</tr>
<tr>
<td>RTP</td>
<td>The Real-time Transport Protocol</td>
</tr>
<tr>
<td>AIO</td>
<td>Audio Input/Output</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>PCM</td>
<td>Pulse-code modulation</td>
</tr>
<tr>
<td>PCMA</td>
<td>Pulse-code modulation A-law</td>
</tr>
<tr>
<td>PCMU</td>
<td>Pulse-code modulation $\mu$-law</td>
</tr>
<tr>
<td>GSM</td>
<td>Global System for Mobile communications</td>
</tr>
<tr>
<td>SDP</td>
<td>Session Description Protocol</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Feedback Model .............................................. 6
2.2 Simplified ASPL structure ............................... 8
2.3 PESQ Block Diagram ................................. 9
2.4 VOIP Implementation ............................... 11
2.5 SIP and Socket Feedback packet comparison .......... 14
2.6 Feedback and RTP Implementation ................. 15
2.7 The Clock skew problem ............................... 17
2.8 Netsim Operation Example ............................ 20
2.9 Netsim Packet Decisions .............................. 22
2.10 Control System ............................................. 22

3.1 Running the Simulation ................................. 28
3.2 Segment and Overall PESQ accuracy ................. 32
3.3 Feedback Accuracy vs Delay ........................... 34
3.4 Feedback and Actual MOS vs Network Delay ....... 35
3.5 Packet Loss Effect on Feedback ...................... 36
3.6 Control with target of 1.0 .............................. 38
3.7 Control with target of 2.0 .............................. 40
3.8 Control with target of 3.0 .............................. 41
3.9 Control with target of 3.5 .............................. 41
3.10 Control with target of 4.0 .................................................. 42
3.11 Control Accuracy as Percentage Difference between Target and Control  . 43
Chapter 1

Introduction

1.1 Voice over IP

Telecommunications play a pivotal role in everything we do and even with the increasing use of email’s and other text based communications there are still almost 4 billion mobile and land lines in use[1] thus the telephone is still very much alive. Along with these landlines there are now over 1 billion VOIP lines and this number is increasing rapidly. The reason for this is that VOIP can offer a number of significant advantages over the traditional Public Switched Telephone Network (PSTN) network with the two key advantages being scalability and cost.

An example of how VOIP makes some tasks simpler is comparing the process of adding a new phone to an office. In the case of the PSTN this means a new copper line must be connected and often it has to be run all the way back to the exchange. In comparison installing a new VOIP phone is as simple as plugging it into the existing network. Installing a new PSTN phone is clearly a much more difficult and costly exercise. Using a VOIP network also allows consolidation of resources as instead of needing both a voice and data network they can be merged into one IP network which both decreases cost and increases flexibility. However this comes with the risk of there now being a single point
of failure where before there were two. The main disadvantage of VOIP is that it is much harder to guarantee a level of service than it is on the PSTN. It is possible to do but due to the complexities involved it is difficult, and the reasons for this will be discussed in the next section.

1.2 The Problems

While VOIP’s voice quality is steadily improving many services are still found lacking when compared to the Plain Old Telephone Service (POTS). Most problems that occur with VOIP come down the basic design principles of the Internet versus the PSTN in that they were designed specifically for non-real-time and real-time data respectively. This means that the PSTN is designed to provide end-to-end delay constancy and handle a steady stream of data. This is in contrast to the Internet which is designed so that there is no loss of data and its data transfer characteristics tend to be bursty in nature. Of these differences it is the constant delay that is the most significant problem with VOIP, but one that can be overcome.

One other important point is that although the internet is designed for no data loss this does not mean that no packets are ever lost, but that if a packet is lost it will be resent until it arrive successfully. This is of no use to real-time applications as by the time the resent packets arrive they will no longer be of any use as the stream will have already progressed past it. Due to this the internet cannot be seen as a lossless medium when speaking about VOIP.

1.3 Quality Measurement

Due to these differences the quality of a VOIP call can vary significantly and so it is important that the quality of the call can be measured to enable the ability to gauge cur-
1.3. QUALITY MEASUREMENT

rent performance and help decide on improvements that could be made. Currently there are two main methods that are used to test quality, off-line measurements and statistical measurements. Both of these have their advantages but neither of them are ideal.

Off-line measurements have been used for a long time on the PSTN and normally consist of a pre-recorded reference stream that is transmitted at one end of the call and then recorded at the other as what is called the degraded stream. These two streams can then be compared using any of a large number of methods such as an real Mean Opinion Score (MOS)[2], where you simply ask a number of people to grade the quality from 0 to 5, or one of the many algorithms that can be used such as PSQM[3] or its successor PESQ[4]. In most cases one of the algorithms is used to evaluate the quality as this can be done relatively quickly and on the spot whereas a real MOS can take a significant amount of time and effort to organise. There are a number of companies selling off-line testing equipment or programs and they have been shown to be able to provide accurate quality measurements and various other statistics about the calls, but they have the significant disadvantage in that they need to use a pre-recorded signal and therefore can not provide real-time information about a call. This makes them good for setting up a VOIP system and doing occasional testing but means they are not appropriate for real-time call monitoring and hence are also inappropriate for controlling a call.

Statistical measurements are the other main group of VOIP quality measurements. They work by analysing the various statistics available about the call, such as jitter, packet loss, and overall delay. They then at their most basic level attempt to compare these to statistics that have been gathered earlier, generally using off-line measurements, in order to get a quality score. The main advantage of statistical over off-line is that a statistical quality can be calculated in real-time but this comes at the expense of accuracy as most statistical methods only give a general estimate of the quality.

This leads on to the first aim of this project, and that is to combine the advantages of both of the above quality measurement techniques into a single method. That is a method that can accurately measure the quality of a voice call in real-time. Having accurate real-time
1.4. QUALITY CONTROL

measurements of VOIP calls can provide a number of benefits to a provider as it means that they will always be able to see exactly how their network is performing in regards to voice data and will therefore be better equipped to make decisions on how to improve or expand their current systems. Although it is important and useful to be able to look at real-time quality measurements it is even better if these measurements can be used to accurately and rapidly control the quality of the call, and this is the second aim of the project.

1.4 Quality Control

Although not a new idea in regards to quality control, Quality of Service (QoS) was originally defined in ITU standard X.902[5] simply as "[a] set of quality requirements on the collective behaviour of one or more objects". What this means in an IP network is simply that QoS prioritises some types of traffic over others which in terms of VOIP means to maximise the quality of the VOIP call at the expense of all other traffic by giving its traffic a higher priority than what the rest of the data gets. This minimises any disruption to the call and works relatively well in a carefully planned network. However, it does not offer much in the way of flexibility, as the aim of traditional QoS is to maximise the quality of all VOIP calls equally, and this is not always the optimal solution.

In contrast, the control proposed in this project attempts to keep the quality at a set level in effect minimising the quality to an acceptable level. Instead of applying the QoS to all of the VOIP calls at once this method can apply different quality targets to each individual call, and this ability to control the quality of a single call means that it is possible for a provider to more accurately control how their whole system operates and will lead to both greater service options for their customers and more efficient bandwidth use for them. An example of the opportunities this creates is that a provider would be able to offer distinct quality levels to their customers based on MOS levels and then enforce them to minimise the bandwidth that they need while still providing the service level that is required.
Chapter 2

Background

2.1 Quality Measurement

To measure the quality of the call the Perceptual Evaluation of Speech (PESQ)[4] algorithm will be used. PESQ was designed by the ITU to objectively measure the quality of a voice call in real network conditions and provide a simple numerical value representing the quality. At its simplest PESQ works by taking in two audio streams, the reference and the degraded, from which it calculates a Mean Opinion Score (MOS) that represents the perceived listening quality of the degraded audio stream compared to the reference. This PESQ score can range from -0.5 to 4.5, with -0.5 being unintelligible to 4.5 being of excellent quality. This score range is different to a standard MOS which goes from 1 to 5, and the reason for this is that statistically no one ever gives a perfect 5 score, with the highest score statistically being 4.5. This fact means that in normal operation PESQ will give a score between 1 and 4.5, with the aim of being consistent with the results of a standard MOS test. In terms of accuracy, tests conducted by the ITU found that on average PESQ had a correlation of 0.935[4] when compared to the real MOS tests using the same data, and so for this project PESQ can provide a simple and accurate quality measurement.
2.2 Feedback System

In normal VOIP communications the two end points both transmit their audio and once it is transmitted they only get minimal feedback, such as total packet loss and average delay, as to what happened to the audio stream during its transmission over the network. This small amount of information is not enough to accurately estimate the quality of the audio being received. In order for PESQ to work it needs an accurate representation at the transmitter of the audio stream at the receiver which can then be inputted into the PESQ algorithm along with the original stream that was sent and thus the quality can be calculated. The simplified block diagram of the feedback system in Figure 2.1 illustrates the conceptual simplicity of the solution where only one extra component needs to be added.

The basics of the feedback system are that it stores a copy of the audio packets that it is sending and also send out information about which audio packets it received. When the clients receive these information packets they use them to create the degraded stream based on the saved reference stream and the packet loss information as shown in Figure 2.1. How this actually works will be described in more detail in the implementation section.
2.3 Implementation

2.3.1 The Acoustic Signal Processing Laboratory (ASPL)

The majority of this project is written within ASPL and so it is important that a basic understanding of it is gained before the rest of the project is explained. ASPL is the standard environment at WATRI for processing real-time audio and was written in the majority by Anders Johannson with extensions added by a various other contributors. It was designed to run on the Linux operating system and is primarily written in C but also consists of a number of Perl, Matlab and Unix scripts which both interact with and control the rest of ASPL. The main component of ASPL is the real-time kernel RTK which is in control of all of the audio I/O as well as the loading and execution of the other components.

In order for each component to be able to send and receive audio data they must use one of the two libport interfaces, sources and drains with the audio always travelling from a source to a drain. Both sources and drains can consist of a number of channels and it is possible to have one source feed multiple drains but not vice versa. As well as providing a way to route the audio libport also performs transparent bit depth conversion if needed between a source and a drain ie from 16bit float to 32bit int. Although the audio can have different bit depth all audio within the system must currently have the same sample rate, which for this project means everything must be either 8kHz or 16kHz depending on the codec that is being used.

Currently in ASPL there are two kinds of audio components algorithms and devices (AIO) both of which are similar in that they each can have sources and drains but have the important differences that while algorithms work on audio data that is already within ASPL devices handle moving the audio into and out of ASPL, hence Audio Input Output (AIO). By looking at Figure 2.2 it can be seen where each of the individual components fits into the overall system architecture, especially libpesq and libvoip which are
the new components that will be written for this project.

![Simplified ASPL structure](image)

**Figure 2.2: Simplified ASPL structure**

### 2.3.2 New Components

Firstly the PESQ algorithm was implemented as an algorithm library in ASPL that takes two audio streams as input and outputs the quality values at regularly defined intervals to both a data file and a network socket. The other library is the actual VOIP implementation and it consists of 4 main subcomponents: SIP, RTP, and AIO which together implement a standard VOIP client. As well as these there is also Feedback which is an extra nonstandard component that deals with all of the feedback information. Each of these components are designed to handle their own specific aspect of the VOIP call and are described in more detail in the following sections.
2.3.3 **libpesq**

The implementation of PESQ is based on the reference code provided by the ITU standard P.862[4] with some modifications made to how the code is called so that it will interface with the rest of the system. Originally the PESQ code was called as an executable and run on two input audio files as defined on the command line with the result being output to the console. To be of use in ASPL it needed to be modified to be able to take its inputs from the audio already available within ASPL. This meant reimplementing the initialisation functions to both read from a ASPL port and output the MOS score to a variable instead of the console. This score is then written to both an output file and a TCP socket to allow both off-line examination of the quality using the file and real-time data analysis using the socket. The output file is of the ASPL matrix format mtx and can be read into Matlab using the provided \textit{m} file and also output to the console using the \textit{readmtxf} program that was written for this project. The MOS values are written to the socket as soon as they are available and take the format of a float with accurate to 3 decimal places followed by a new line.

The two functions that needed to be reimplemented to allow PESQ to interface with the rest of ASPL are \textit{pesq\_measure} and \textit{main} which were replaced by \textit{aspl\_pesq\_measure} and \textit{perform\_pesq} respectively. It is important to note that other than these two functions the rest of the PESQ code is exactly the same as the reference and no attempt has been made to customise it to better suit the needs of this project.

![PESQ Block Diagram](image)

**Figure 2.3: PESQ Block Diagram**

It is likely that significant performance gains could be achieved by both rewriting and removing certain parts of PESQ to better suit the specific nature of this project. PESQ
was designed to be a multipurpose algorithm and one of its key benefits is that it can compensate for the two input signals having variable delays, which is the Time Alignment block in Figure 2.3, this is very important factor when evaluating a normal VOIP or telephone call but is not needed in this case enabling the whole block could be removed. The reason for this is that since both of the feedback audio streams are created together there will be no time variance between them and so this step of PESQ is redundant and time consuming. Removing this would be a simple step to improve the performance of the current algorithm, but the main limitation of PESQ is that it was not designed as a real-time algorithm. Currently we just run PESQ independently on small durations of audio and output the results of each segment and while this works well it is not optimal and a better solution would be to reimplement PESQ as a real-time algorithm that could run on a continuous stream of audio and output MOS values on a continuous basis. Unfortunately the algorithm is complex and this task would require a significant amount of both knowledge of PESQ and time to implement which meant it was not attempted as part of this project.

2.3.4 libvoip

Audio Input/Output (AIO)

This is where the VOIP library interacts with the rest of the ASPL system and is in charge of both setup and shutdown of the library including reading all of the configuration options. Once the call is setup it controls the transfer of the audio stream between the VOIP call and the rest of ASPL. The code for this is contained in voip.c as well as the header files voip.h and defines.h.
2.3. IMPLEMENTATION

Real-time Transport Protocol (RTP)

The RTP[6] component is in charge of sending and receiving audio on the network and this involves both encoding/decoding the audio and sending the actual RTP packets. The actual RTP network transport is implemented using the open source oRTP[7] library and is run in a separate thread to minimise the audio delays.

Codecs

When transmitting RTP data a payload type must be specified, this payload type contains information relating to both the encoder used on the audio and its other parameters such as clock rate and the number of channels. The standard payload types are defined in RFC1890[8] and of these PCMU[9], PCMA[9] and GSM[10] are currently implemented as well as two additional non standard codecs raw PCM at 8 and 16kHz which are used in testing. The GSM code is from The Communications and Operating Systems Research Group (KBS) at the Technische Universitaet Berlin with a simple wrapper so that it can
be used as a codec for ASPL and the same goes for the two G.711 codecs PCMU and PCMA where the code is based on that available from Sun Microsystems. Although there are many other codecs available these were three were chose for both their common use and most importantly readily available and unencumbered source code. If other codecs were needed it would be trivial to implement them as the codecs system is designed to be modular and easily extensible, for example the wrapper to implement GSM is simple and short as it only needs to implement 3 methods and a codec structure which summarises the codec as is shown in Listing 2.1. This structure contains in order the RTP payload type, the name of the codec, the frequency in hertz and then the size of the encoded data, which for GSM is 33 bytes. After these values come the references to the codec initialisation function as well as the encode and decode functions.

```
Listing 2.1: GSM structure

codec_t codec_gsm = {3, "GSM", 8000, GSM_FRAME_SAMPLES, &init_gsm, &encode_gsm, &decode_gsm};
```

**Session Initialisation Protocol (SIP)**

This is in charge of the initialisation of the call, sending the feedback information and then the termination of the call. The SIP protocol is defined in RFC3428[11] and is commonly used on VOIP applications and hardware including the Snom 100 Hardware phone that was the primary test target. The majority of the SIP work is done by the Nokia Sofia-SIP stack which is an open source LGPL library for writing SIP applications and was chosen as it is the most complete and usable C library of its kind.

The SIP stack’s main job is to control the call. Initially this means that it must either make a call or answer an incoming call and once this is done it must negotiate the audio codec to use. This negotiation is described in RFC3264 [12] and RFC4566 [13] both of which Sofia-SIP implements and so all ASPL has to do is create a Session Description Protocol (SDP) [13] string and then pass it to Sofia-SIP which will then negotiate with
the other client and return another SDP string which will contain the codecs that are allowed. These SDP strings are quite simple and the only part that ASPL needs to worry about is that which contains the codecs available and the RTP ports they correspond to. For example the ASPL client by default supports GSM, PCMU and PCMA on port 8000 which gives the codec part of the SDP string shown in Listing 2.2. This string is generated dynamically depending on what codecs are available and preferences for certain codecs can be set in the configuration file.

Listing 2.2: Example SDP string for ASPL

```
//m=audio 8000 RTP/AVP
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:3 GSM/8000
```

Feedback

In order to recreate the audio streams for feedback each end of the connection needs to know what packets the other end received, and the simplest way to do that is to use the SIP MESSAGE\[11\] extension which can send simple text messages between the clients. These messages consist of a simple text string that contains the timestamps of received packets and an example is given in Listing 2.3.

Listing 2.3: Example SIP Feedback packet

```
/* Example: */
```

Using SIP to transfer this data is not optimal as SIP has relatively high size overheads and is by default an unreliable protocol. A much better choice would be to use either raw TCP or UDP sockets which would both have the advantage of significantly lower overheads. As well as this TCP is also a more reliable transmission method as it supports
retransmitting lost packets unlike UDP. The amount of extra bandwidth needed by the SIP method is displayed in 2.5 which shows the size of the current SIP packets compared to a direct Socket which would simply send the data structure outlined in Listing 2.4.

Listing 2.4: Proposed data structure for feedback packet

```c
struct feedback_packet {
    uint32_t length;
    uint32_t ts[length];
}
```

This new structure is of size \(4 + (\text{length} \times 4)\) where by default the length is 10 which gives a size of 44 bytes that when added to the UDP/IP header gives a total packet size of 72 bytes. This is only 15% of the 465 bytes that are needed to transmit this same data using a SIP message.

Currently SIP is an adequate solution for testing purposes as the results will be the same except for the bandwidth taken by the feedback packets, but if further work was to be done
implementing the feedback as a UDP socket would be important and should not involve too much extra work.

![Feedback and RTP Implementation Diagram](image)

**Figure 2.6: Feedback and RTP Implementation**

### Compatibility

It is important that the main VOIP implementation maintains compatibility with other clients that support the same standards and so from the beginning every aspect was designed in accordance to the various standards, with the exception of the feedback loop which is custom to this implementation. Although the feedback loop will not work with any other client the call will still be able to be made and answered and will work as normal, but there will be no feedback statistics available. In order to ensure compatibility the ASPL implementation was tested against a number of different VOIP devices both software and hardware and in all cases performed as expected.
2.3. IMPLEMENTATION

2.3.5 Problems Encountered During Implementation

Writing the implementation was a considerable task with the final VOIP client consisting of just over 2800 lines of C code which meant there were a number of problems that needed to be overcome as it was written. In a VOIP client minimising delay is essential and therefore a lot of time was spent removing sources of delay in the client which was measured by using a generated square wave as the source and comparing both it and the received signal on an oscilloscope in order to measure the delay in milliseconds. Changes were then made to the system and compared with the original until delays were reduced to almost the minimal level possible using the available equipment.

Other than delay the major problem facing a VOIP call is that no two systems have the same hardware clock and without some method to synchronise or allow for the clock-skew the call will get out of sync very quickly. For example on the two WATRI lab computers after a VOIP conversation of 5 minutes the two systems would be out by more than a second, which is not acceptable. In order to fix this clock-skew some audio samples must be either dropped or repeated so that the two computers stay in sync. The method chosen to do this is one of the simplest and works by having a target buffer size and if the actual size is greater than this plus a margin then samples are dropped at certain intervals until the buffer size is again acceptable and similarly when the buffer size is too low samples are repeated. This dropping and repeating of samples enables the two systems to stay synchronised and since the number of samples either dropped or repeated is very small the audio quality difference is negligible to a persons hearing.

The problem that arose from doing this is that in order to verify the accuracy of the PESQ and feedback algorithm’s in ASPL it needed to be compared with the actual audio output and of the program. This meant recording the output of a call and then splitting it up into one second segments as is done by ASPL and then running PESQ on each segment and comparing these values with those calculated during the call by ASPL. If the ASPL feedback implementation was working correctly both MOS values should be identical as they will be comparing the same signal, but what was originally seen was that the two sets
of values started off the same but gradually got further and further away until the recorded MOS was -1, indicating no similarities between the two files. The problem was that due to one computer’s clock being consistently slower than the others which led to it having to drop samples in order to keep up and because of this its output file was shorter than the input file.

This would not be a problem if PESQ was run over the entire file as it is designed to handle time alignment and clock skew, but because the files were being broken up into one second segments after a period of time the segments from the two files would no longer match which explains why the MOS scores of the recorded value gradually get worse. This is easier to explain using Figure 2.7 where it can be seen that if trying to compare the Input file with ASPL everything matches up properly but when it is compared to the recorded file it does not line up and gets worse the further along it goes. The solution to this is simply to disable the synchronisation when doing this kind of testing. This will have no affect on the overall quality as is shown later in the results section, except for slowly increasing delay, but will allow for the tests to be performed. With this gradually increasing delay comes the problem of buffer overflows and this was countered in the tests by both using a large buffer size and limiting the duration of the calls. These two fixes would not be necessary in real usage but were needed to be able to run the tests which verified the accuracy of the algorithm.

![Figure 2.7: The Clock skew problem](image-url)
2.4 Network Simulation

To be able to test the accuracy and limitations of the quality measurements tests need to be run that simulate many different types of real world network conditions, and to do this we need to be able to model a real world network in the lab. There are two key parameters that determine a network connection and these are packet loss and delay, so in order to model a real network we need to be able to both delay and discard the packets between two hosts in a predetermined manner.

2.4.1 Modeling Network Conditions

Delay

In both Corlett[14] and Bolot[15] it is shown that a shifted gamma distribution best models the delays of the data that was recorded, and due to this it will be the preferred method of simulating delay in the test network. In this case we have the following distributions.

\[ X \sim \text{Gamma}(s, m) \]  

Which has density function:

\[ f_s(z) = \frac{z^{s-1}e^{-z}}{\Gamma} \]  

and with shift

\[ X = mZ + c \]  

Where m is scale, s is shape and c is shift.

To use the this distribution as the delay model appropriate values for s and c need to be determined and this will also vary depending on the type of link involved and the current utilisation and a summary of the results Mukherjee[16] found follows.
2.4. NETWORK SIMULATION

**Backbone** Low to medium $s (< 5)$

**Cross Country** Medium $s (3$ to $7)$

**Regional** Low $s (0.06$ to $0.20)$

For all network types $c$ is based on the minimum delay possible and is determined primarily by the distance of the link. In terms of VOIP the $s$ parameter is the most important as it determines how much variation (jitter) there is in the packet arrival times and this jitter can have a significant impact on the quality of the call.

**Loss**

Packet loss is the other network parameter that needs to be modelled, and once again there are a number of different model that have been proposed. The main aspect that is important in modelling loss is that it tends to be bursty and not evenly distributed, ie a 5% packet loss is more likely to occur all at once than spread evenly over the whole time period.

There are a number of different models that can be used to simulate this and two popular ones are the Gilbert[17] and the Gilbert-Elliot [18] Models. Both of these models are simple two state models which consist of a good state, 0, and a loss state, 1, and the state transition probabilities of $P_{01}$ and $P_{11}$. It is important to note that in the plain Gilbert model the loss state has its own loss probability $P_L$, and that the Gilbert-Elliot model extends this so that both states have their own separate loss probabilities. One problem with both Gilbert models is that they do not account for longer periods of packet loss, ie when a link is down, but for the experiments being conducted this is not an issue as any long term loss will mean that the call will fail and there is no need to analyse it.

In order to use these models a Matlab script was made which takes the parameters of the model and outputs an array of ones and zeros representing if a packet should be kept or
discarded. This data file can then be used as the input to \texttt{netsimrt} which will then apply it to the network.

### 2.4.2 Netsimrt

Currently \texttt{ASPL} runs between two computers, the same as a normal VOIP call, and so in order to test how it performs a method is needed to simulate different network conditions between them. It was decided that the best way to do this would be to overlay the simulation on top of the existing network so that the it occurs transparently to \texttt{ASPL}. This works by a program called \texttt{netsimrt} intercepting each network packet and applying the parameters defined in the model to it, which are Delay and Loss, where delay is in milliseconds and Loss is a boolean defining if the packet is to be dropped.

![Figure 2.8: Netsim Operation Example](image)

An example of how \texttt{netsimrt} works is shown in Figure 2.8 with Example A showing the original packet stream across the network, and Example B showing the modified stream. In Example A the packets enter Client B at regular intervals and with a small constant delay, which is almost ideal network conditions. Once the model has been applied packets arrive at varying intervals depending on the individual delays and some packets such as 3 and 8, do not arrive at all. This example is in essence how \texttt{netsimrt} simulates a network.

When simulating a network using \texttt{netsimrt} it is important to note that the minimal
2.5. CONTROL

delay will not be the same as that which is defined in the delay input distribution but will instead be that plus the nominal delay of the network. All of the tests I will be running are on an unloaded LAN with an averaged measured delay of 0.087ms which is a number of orders of magnitude smaller than the delays from the delay distributions and so can be ignored. If the network delay was much larger it would need to be taken into account and so the new delay distribution would be:

\[ X = mZ + c + d \]  \hspace{1cm} (2.4)

Where \( d \) is the average delay of the local network.

Implementation

Netsimrt is implemented as a C program that interfaces with the linux kernels iptables in order to control certain packets, in this case the VOIP call. Netsimrt takes three arguments, first of which is the netfilter queue to bind to and the second two are the files to load the delay and drop data from respectively.

It controls the packets by connecting to the kernel through libnetfilter_queue and then intercepting every packet that passes through queue specified. Each of these packets can be dropped or delayed as specified in the input files. Figure 2.9 shows the decisions that are made for each packet that passes through netsimrt.

2.5 Control

The controlling of the call can be easily split into two separate component, the first of which is ASPL which provides the MOS scores that are needed to evaluate the current quality of the call and secondly there needs to be a way to dynamically control the quality of the call. This second component, the limiter, consists of another linux program based
2.5. CONTROL

Figure 2.9: Netsim Packet Decisions

Figure 2.10: Control System
on **netsimrt** that once again interacts with the kernel’s networking subsystem through **libnetfilter_queue** and decides if each packet that has previously matched an iptables rule should be accepted or denied. The reason for the iptables rules is to allow only the packets for a single call to be passed to the limiter and this is simple to do as each call will use a unique port and source address combination, which for these experiments is port 1200 and an address of 192.168.1.1 which gives the iptables command seen in Listing 2.5. When looking at the command the **--queue-num** argument is important as the number given for this argument is what the limiter uses to identify this particular call. For example if there where 5 calls occurring at the same time each would have a unique queue number that would identify it and each queue would have its own limiter controlling it allowing for completely independent control of multiple calls.

**Listing 2.5: Iptables section of run script**

```
iptables -I INPUT -p udp --src 192.168.1.1 --dport 1200 \
         -j NFQUEUE --queue-num 0
```

The current implementation provides a simple control method where a certain ratio of packets can be dropped in order to control the overall bandwidth utilisation of the call. This method was chosen since it provides the simplest way to dynamically control the quality of the call.

The limiter talks to ASPL through a standard TCP socket from which ASPL transmits the MOS scores as they are calculated. A TCP socket was chosen both for its simplicity and the fact that this means that the limiter does not need to run on the same system as the rest of the components. This means that many systems could all report back to the one router which would then control all of the calls. In allowing the traffic to be controlled from a different system to the which the measurements are taken gives this form of control the advantage that the traffic can be controlled at the entrance to a network instead of being controlled at an endpoint as is normally done and this can further reduce network utilisation.
2.5. CONTROL

2.5.1 Control Strategies

Since this is more of a proof of concept the original control strategy chosen is quite simple, it consist of one case where the calculated quality is close to the target and the drop rate is not changed. On either side of this case are from one to three cases that change the drop rate in increasing amounts as the calculated value gets further away from the target.
Chapter 3

Results

3.1 Simulation Setup

To run the simulations two computers are needed one for each end of the call and they need to be networked preferably on a dedicated 100Mbps or greater network so that any network congestion will not influence the results. Three main test setups were used to run the simulations and these are outlined in Table 3.1. For each simulation a large amount of variables need to be setup and the majority of these are in the ASPL configuration files which are simple XML files, an example of which is in Listing A.1. As well as controlling the variables these config files also specify which parts of the call are to be recorded, and for all of these simulations the following output audio files are produced by each computer:

- **COMPUTER_out.wav** Audio sent, 1 channel.
- **COMPUTER_in.wav** Audio received, 1 channel.
- **COMPUTER_feedback.wav** Audio created by the feedback implementation, 2 channels (ref,deg).
3.1. SIMULATION SETUP

COMPUTER\_aspl  Output of ASPL Quality measurements for each time interval.

As well as these audio files all of the network traffic during the simulation is saved as COMPUTER\_cap so as it can be analysed later to calculate network performance such as average jitter and delay, and if needed the whole simulation can be recreated from the network capture.

<table>
<thead>
<tr>
<th>WATRI A</th>
<th>2 x IBM Servers alice and abob</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intel Pentium 4, 1.4 GHz</td>
</tr>
<tr>
<td></td>
<td>1024Mb RAM</td>
</tr>
<tr>
<td></td>
<td>Debian Linux 3.0</td>
</tr>
<tr>
<td>WATRI B</td>
<td>2 x Shuttle Boxes artemis and athena</td>
</tr>
<tr>
<td></td>
<td>Intel Pentium 4, 2.4 GHz</td>
</tr>
<tr>
<td></td>
<td>2048Mb RAM</td>
</tr>
<tr>
<td></td>
<td>Debian Linux 3.0</td>
</tr>
<tr>
<td>Home</td>
<td>server consisting of:</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon 3060 2.4GHz</td>
</tr>
<tr>
<td></td>
<td>2048 Mb RAM</td>
</tr>
<tr>
<td></td>
<td>client consisting of:</td>
</tr>
<tr>
<td></td>
<td>2 x AMD Opteron 246 2.0Ghz</td>
</tr>
<tr>
<td></td>
<td>2048 Mb RAM</td>
</tr>
<tr>
<td></td>
<td>Both running Gentoo Linux</td>
</tr>
</tbody>
</table>

Table 3.1: System Setup

3.1.1 Test Data

In order to run a simulations audio needs to be inputted into ASPL and since this needs to be done in a consistent and repeatable way a number of audio files are needed. It is very important that these files are of both male and female voices of different types, as in general female voices tend to achieve higher listening scores than male, therefore both are needed to be able to properly evaluate the system. The three sources of audio I used were WATRIs own sound archives which consisted of a female speaker, a variety of audio files from the ITU that were used in the conformance testing of PESQ, and lastly a number of
freely available audio files from various sources. The problem with many of these files is that they are relatively short in length, varying from around 10 seconds for the ITU files to 30s for WATRI’s, and so in order to simulate a real telephone conversation these files were either repeated or a number of them were joined together to create one long audio file. When joining these files care was taken so that there was still a normal amount of silence in the overall file as if they were simply joined to make one continuous section of speech it would not be an accurate representation of a conversation. The amount of speech per silence chosen is based on that used in PESQ testing[4] where a typical burst of speech will last for around 3 seconds and the overall speech to silence ration should be from 40% to 80%. As well as English some of the test files were spoken in different languages to ensure that the results obtained were not specific to the English language.

3.1.2 Running the Simulation

Since the simulation needs to be run over two computers a script is used to both start the instances of ASPL and also prepare to collect any extra data that will be needed. The Simulation stages are displayed in Figure 3.1 and consist of setting up the simulation, which entails copying over the config files, and then setting up the iptables rules. These iptables rules are needed so that the VOIP parts of the network traffic are passed to \texttt{netsimrt} or \texttt{limiter}. It is possible to pass all of the traffic but this would cause all of the other communication to the outside network to be delayed and this is not necessary. In Listing 3.1 is an example of the iptables rules used on \texttt{server} when running and it is important to note that there are two queues setup an input and output as if this is not done the network will only be simulated in one direction. This could also be done by using one queue that contains both directions but then the simulated network will not be quite as expected since it will be attempting to simulate both directions from one directions data. Once the simulation is finished all of the data and config files are collected and copied to one location and the temporary files are cleaned up and everything is reset to the original configuration.
3.1. SIMULATION SETUP

Figure 3.1: Running the Simulation
3.1. SIMULATION SETUP

Listing 3.1: Iptables section of run script

```bash
echo "Setup iptables"
sudo iptables -F
sudo iptables -I OUTPUT -p udp --dest client -j NFQUEUE \    --queue-num 0
sudo iptables -I INPUT -p udp --src client -j NFQUEUE \    --queue-num 1
```

3.1.3 Creating the Results

Once the simulation has been run there is a large amount of data that needs to be processed and this is done differently depending on the kind of simulation that is being run. For the quality measurements the following values need to be obtained:

1. Read the segmented MOS values calculated in ASPL.
2. Calculate the segmented MOS values from the audio files recorder by the receiver.
3. Calculate the overall MOS values for the received audio.
4. Retrieve the network statistics from the capture.

Number 1 is simply a matter of reading the mtx file written by ASPL and saving it as the first column in a results text file, 2 is more difficult as the two files first need to be split up into the segments and then each of the corresponding reference and degraded segments need be run through PESQ and a MOS calculated and then added into the second column of the results file. 3 is also simple as it just means running the transmitted and received data through PESQ and getting the MOS value. Now that all the MOS values have been collected the network capture is read and the Packet Loss and Jitter are saved to a file. All of the needed information has now been collected and the results can be used as in the rest of the results section.
3.2 Measurement of VOIP Quality

It is of not much use to have real-time quality measurements if they are not accurate and that is what will be tested in this section. Before any real network testing of the measurement system can be done the PESQ algorithm in ASPL needed to be verified against the reference implementation and once they produced the same results the next stages of testing over real networks can be performed. Since VOIP traffic in general travels over the internet the network parameters between two hosts can vary dramatically in different circumstances and in order to get an accurate representation of how the feedback system works the system must be tested under a range of different network conditions the results of which will be examined in the following sections.

3.2.1 Accuracy of Segment based PESQ

In the original PESQ algorithm it was intended that it be run over the whole conversation or a significant part of it. However, this is not how it is being used in this instance. Instead we are running PESQ at regular short intervals, the default being one second and with a minimum of 0.25 seconds as defined by the PESQ algorithm, and it is important to be able to measure the difference this makes to the overall quality. To do this tests were run using a variety of different audio files, both male and female voices, and a number of different packet losses varying from 0% up to 48% so as to simulate different quality levels and the results of both the overall MOS and the averaged segmented MOS were compared. The results of this test are displayed in Table 3.3. What was found was that in general the segmented MOS was slightly higher than that which was calculated overall, which can be seen in Figure 3.2, and had an average error of 7%. Since the error is fairly small and relatively constant it is possible to apply a scaling function to it in order to bring it closer to the expected overall MOS. The scaling function chosen in this case was simply to subtract 0.1 from the segmented score and this reduced the percentage error to 1.45%.

The fact that the overall MOS score was lower than the segmented score is not surprising
3.2. MEASUREMENT OF VOIP QUALITY

<table>
<thead>
<tr>
<th>Packet Loss</th>
<th>PESQ Segment Average (MOS)</th>
<th>Overall PESQ (MOS)</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0%</td>
<td>3.451</td>
<td>3.393</td>
<td>1.7%</td>
</tr>
<tr>
<td>1.4%</td>
<td>3.191</td>
<td>3.038</td>
<td>5.0%</td>
</tr>
<tr>
<td>4.8%</td>
<td>2.823</td>
<td>2.673</td>
<td>5.6%</td>
</tr>
<tr>
<td>9.0%</td>
<td>2.426</td>
<td>2.268</td>
<td>7.0%</td>
</tr>
<tr>
<td>24.2%</td>
<td>1.852</td>
<td>1.690</td>
<td>9.6%</td>
</tr>
<tr>
<td>47.3%</td>
<td>0.907</td>
<td>0.806</td>
<td>12.5%</td>
</tr>
</tbody>
</table>

Table 3.3: Accuracy of Segment Based PESQ

as PESQ does not use a normal average to calculate the overall MOS, but a weighted average that is balanced so that the bad portions of the audio stream have a greater impact on the overall score. If it were necessary to increase the accuracy of our predictions even more it would be advantageous to use a similar weighting as PESQ but this was not done in this case as the added complexity is large compared to the small increase in accuracy that would be gained.

3.2.2 Effects of Loss and Delay in Feedback Loop

In previous tests done it was chosen that the feedback loop should not be affected by the simulated network conditions and this leads to near perfect accuracy. This result is expected since with no packet loss the feedback loop should be able to recreate the exact stream that was received. Since the system was shown to work in this simple case it is now important to test if it can be used to accurately reconstruct a VOIP call that has been affected by both packet losses and delays over the whole network, that is feedback traffic will experience the same losses and delays as the voice traffic.

The first test that was performed was the constant delay which used identical network data to the previous test. From these results it could be seen that where before the feedback buffer needed to be increased by roughly the maximum delay now it needed to be increased by two times the delay. This is to be expected as before the delay was only...
3.2. MEASUREMENT OF VOIP QUALITY

Figure 3.2: Segment and Overall PESQ accuracy
acting on the voice signal but now it acts on both and so the delay will be twice as long.

### 3.2.3 Ideal Network Conditions

The first test performed is to evaluate the accuracy in ideal network conditions as this establishes a baseline for the rest of the tests. From the results obtained we can see that in ideal network conditions where there is a constant low delay and no packet loss the MOS values calculated by the real-time feedback are identical to those calculated off-line. This is as expected as in the case of no packet loss the audio stream that is created from the feedback should be a bit for bit copy of the stream at the receiver.

### 3.2.4 Constant Delay’s

A common situation in backbone links is to have a constant delay of varying size depending on mainly the distance between the two links. This is designed to test performance over these kinds of links. An important factor in the accuracy of these results will be how long ASPL waits for the feedback packets as since the end to end delay has increased if the feedback wait is too low the feedback packets will be discarded as arriving too late and it will be assumed that the other end is not receiving any of the voice packets. In order to accommodate various delays the size of the feedback buffer can be changed, and hopefully from these tests an optimal size for varying delays can be found. This optimal size is important as if it is too small the results will be inaccurate and for increasingly large a buffer the results will become more delayed so it is in essence a trade-off between accuracy and delay. This can be seen in Figure 3.3 where as the feedback delay is increased the feedback calculated MOS becomes more accurate until at a delay of 570.0ms it exactly matches the actual MOS.
3.2.5 Variable Delays

In terms of feedback a call with highly variable inter-packet delays could be treated the same as if it had a constant delay, where the constant delay is equal to the largest variable delay. This is not the optimal solution and it comes down to statistics and accuracy requirements as to how much feedback delay should be used. For instance in Figure 3.3 it can be seen that at 570.0ms the feedback quality matches the actual quality, and this would be the feedback size chosen if 100% accuracy was needed. If a smaller delay was needed some accuracy could be sacrificed and a feedback size of 500ms may be chosen. The problem with choosing a too small a delay is illustrated in Figure 3.4 which shows the feedback results from a number of simulations with a small feedback delay. In this case it works well when the one-way network delay is less than 70ms, but after this the performance drops of rapidly emphasising the need to choose a good value for the feedback delay.

It is important to note that the estimated feedback quality will never be higher than the actual quality. This is as expected from the algorithms implementation where since no feedback packet was received then ASPL assumes the other end did not receive any of the
audio packets corresponding with the feedback packet. At best the feedback and actual quality could be the same, but this will only occur if the receiver is not actually receiving any audio packets. With more work the feedback algorithm could attempt to compensate for lost feedback packets, with one simple possibility being just to repeat what happened in the last feedback packet. A more advanced method would be to use statistics gathered during the rest of the call to attempt to create a new feedback pattern to use in place of the missing one that would hopefully closely match the original.

![Figure 3.4: Feedback and Actual MOS vs Network Delay](image)

3.2.6 Packet Loss

Losing feedback packets can dramatically reduce the accuracy of the feedback system, as since each feedback packet carries data on multiple audio packets if one is lost ASPL assumes that all the audio packets that were referenced in it were also lost and so this could mean that between 50ms and 500ms (depending on the audio and feedback packet size) of the degraded audio stream will be silent when it should likely contain audio data.
3.2. MEASUREMENT OF VOIP QUALITY

an example of this occurring is displayed in Figure 3.5.

![Figure 3.5: Packet Loss Effect on Feedback](image)

When dealing with packet loss in the feedback loop it becomes much more important to have a robust system for retransmitting lost feedback packets and by default ASPL will resend feedback packets quite aggressively and so occasionally there may be two of the same packet received in quick succession and one will need to be discarded. Although this introduces a small increase in bandwidth used it also increases the performance of the feedback loop from increased accuracy by reducing delay. Once again there is a trade-off that needs to be made, this time being between bandwidth utilisation and accuracy.

3.2.7 Measurement Summary

Packet loss is the most serious problem for the feedback loop as originally the SIP messages were not retransmitted if lost and so any loss of a feedback packet caused a significant drop in measurement accuracy. In order to overcome this the SIP settings were modified so that SIP Message packets are retransmitted when lost so that given enough time all the feedback packets would be received correctly. From this point and the experiments done using varying delays it can be seen that the common conclusion can be made that measurement accuracy is a direct trade-off with the delay of the measurements.

\[ \text{Accuracy} \propto \text{Delay} \] (3.1)
3.3 Control of VOIP Quality

When assessing the success of the control the important criteria are that the resultant quality of the call averages to the target quality and that the controlled quality does not vary from the target by excess amounts. If these criteria have been met then it is also useful to calculate what the bandwidth savings are from using the specified target. In the experiments conducted seven MOS values were targeted, varying from 1 to 4 and which represents audio quality from Bad to Good according to the MOS scale and the results are displayed in Table 3.5.

<table>
<thead>
<tr>
<th>MOS</th>
<th>Target</th>
<th>Average</th>
<th>Overall</th>
<th>% Diff</th>
<th>RTP</th>
<th>Total</th>
<th>Lost</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.70</td>
<td>1.28</td>
<td></td>
<td>22%</td>
<td></td>
<td>3156</td>
<td>2836</td>
<td>47%</td>
</tr>
<tr>
<td>1.5</td>
<td>1.70</td>
<td>1.61</td>
<td></td>
<td>7%</td>
<td></td>
<td>3460</td>
<td>2533</td>
<td>42%</td>
</tr>
<tr>
<td>2.0</td>
<td>2.13</td>
<td>1.94</td>
<td></td>
<td>3%</td>
<td></td>
<td>4158</td>
<td>1835</td>
<td>31%</td>
</tr>
<tr>
<td>2.5</td>
<td>2.66</td>
<td>2.50</td>
<td></td>
<td>0%</td>
<td></td>
<td>5444</td>
<td>549</td>
<td>9%</td>
</tr>
<tr>
<td>3.0</td>
<td>3.19</td>
<td>2.93</td>
<td></td>
<td>2%</td>
<td></td>
<td>5962</td>
<td>259</td>
<td>4%</td>
</tr>
<tr>
<td>3.5</td>
<td>3.54</td>
<td>3.37</td>
<td></td>
<td>5%</td>
<td></td>
<td>6013</td>
<td>96</td>
<td>1%</td>
</tr>
<tr>
<td>4.0</td>
<td>3.70</td>
<td>3.54</td>
<td></td>
<td>13%</td>
<td></td>
<td>6094</td>
<td>23</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table 3.5: Simple Control Results

Currently the control algorithm used is very simplistic and this causes problems for very low target values, as can be seen in Figure 3.6 where the algorithm cannot get the quality low enough to meet the target. This problem is caused by the fact that the algorithm can only drop a certain ratio of packets that is less than or equal to 50%, and losing this amount of packets is not sufficient to reduce the quality down to 1.0. One solution to this would be to allow larger ratios of packets to be dropped ie $\frac{3}{2}$, but using a target of 1.0 or less is not very applicable to the real world as the audio is only barely comprehensible.

As the quality target is increased to 2.0 we can see a dramatic improvement in the control algorithm which as shown in Figure 3.7 exhibits much better behavior with the actual quality being close to the target for the majority of the call. When using this quality target the bandwidth that the voice data is using has for this simulation been reduced by 31%, a
significant reduction.

### 3.3.1 Control Algorithms

During the testing a number of control algorithms have been tried and many discarded for various reasons I will outline below. The first attempt was very basic and simply had an allowed band around the target MOS and if the measured MOS was out of the band the drop rate was increased or decreased by 1 as appropriate. The code for this is shown in Listing 3.2 and it worked surprisingly well except that the initial drop ratio needed to be accurate or else it would take a long time to settle.

**Listing 3.2: Original Control Algorithm**

```c
/* cmos is the calculated mos value */
int deadband = 0.5;
if (drop_ratio > 1 && cmos > target + deadband) {
```

![Figure 3.6: Control with target of 1.0](image-url)
In order to fix the settling time a new modified algorithm was used which halved the drop rate if it was significantly too high and this lead to a much shorter settling time while still having the same performance characteristic as Listing 3.2. The main limitation of these first control methods is that while they do provide control the average quality for the call can be different by up to 20% from the target. This means that a more aggressive strategy is needed and the first one tried is to simple have multiple bands of control, where the further the quality is from the target the more the drop rate will change. Up to a point this leads to better control, but due to the unpredictable nature of speech quality it can also lead to very bad drops in call quality for short periods of time, as can be seen in Figure 3.7 at 45 seconds.

In Plot 3.10 the limitations at the upper end the current control strategy can be seen where it is unable to control the call to a MOS of 4.0, a value just below the original calls average quality of 4.18. From the plot it can be seen that the Drop Ratio is continually increasing but it is not having any affect on the output quality. The reason for this is that it only takes a very small number of dropped packets to reduce the quality to below 4.0, and the current control algorithm can not handle this as it uses the same algorithm for all target qualities when it would ideally use different algorithms for different targets. These different algorithms would tend to be more aggressive for lower MOS targets and make only small changes for higher targets. This would allow more accurate control over a larger range of quality targets than can currently be achieved.

The results of the call’s using a targets of 3.0 and 3.5 are a good example of some of the shortcomings of the current control strategy. In Plot 3.8 it is seen that the drop ratio slowly decrease during the whole call and it is only in the second half of the call where it is low
3.3. CONTROL OF VOIP QUALITY

Figure 3.7: Control with target of 2.0

enough to be able to actually be able to control the quality to near the 3.0 mark. This is in comparison to Plot 3.9 which is using the same control algorithm, but this time the starting drop ratio happens to be about the same as what is needed to keep the call quality at 3.5 and so during the call only small alterations to the drop rate need to be made. This poor performance in Plot 3.8 is an excellent example of the need for a better algorithm to cover all MOS targets as the performance at a target of 3.0 is significantly worse than either 3.5 or 2.5.

3.3.2 Control Summary

From the results obtained using a very basic control algorithm it has been shown that it is possible to control the quality of a VOIP call in real-time using the MOS values calculated by the feedback method. With the current method of control the best results occur when the target value is between 1.5 and 3.5, as can be seen in Figure 3.11, which shows that
3.3. CONTROL OF VOIP QUALITY

Figure 3.8: Control with target of 3.0

Figure 3.9: Control with target of 3.5
for all of these targets the quality can be controlled to within +/-10%. With more research into the control algorithm it is likely that these results could be improved. The first area to improve would be a better way to drop the packets instead of the currently used drop ratio. Using a drop ratio is not a very good way to do this as the amount of packets dropped does not decrease linearly as the drop rate is increased but instead is an inverse relationship. This leads to small changes in the drop ratio having a large effect for values less than 8 and a very small effect when the drop ratio is above 32. The drop ratio method was chosen originally since it was simple to implement and the control method had not yet been proven to work. Now that the control method has been proven a better method with a linear drop rate should be used.

Even with new control algorithms there will always be the problem of speech quality being very unpredictable, which means that it will be very difficult to control the call well enough that there are no times when the controlled quality varies by a considerable amount from the target. One way to help overcome this is by using statistics as well as the current MOS values in order to attempt to predict the future MOS values based on the
3.3. CONTROL OF VOIP QUALITY

Figure 3.11: Control Accuracy as Percentage Difference between Target and Control

% Difference

Target MOS

0 1 2 3 4 5

0% 10% 20% 30%

-10% -20%

Figure 3.11: Control Accuracy as Percentage Difference between Target and Control
past and current values. With some changes to the system like those mentioned the results
of the control system could be greatly improved.
Chapter 4

Conclusions and Future Work

In this report both the implementation and the corresponding results of a method to accurately measure the quality of a VOIP call in real-time and then to actively control it to a set quality level have been discussed and analysed.

In terms of performance this new method of measuring VOIP quality shows very good accuracy as long as it is set up correctly. This means that the feedback packets are given enough time to arrive even in the event of network loss. The relationship as stated before between measurement accuracy and delay is proportionally linked up to when the accuracy is at its maximum. Currently the delay is manually configured and this is one area where improvements could be made as instead of setting it at the start of the call it would be better if it could vary dynamically as network condition change. This and other parameters in the feedback system would benefit if they could be tuned automatically and although this would dramatically increase the complexity of the system it would also simplify its use and allow it to operate over a wider range of networks without needing to individually tune the configuration.

Since the measurement method worked as expected it was possible to use it to attempt to dynamically control a VOIP call to a specified target quality. From the tests that were carried out it was shown that it is possible to keep a VOIP call’s quality close to the target
level for a moderate range of targets. Now that is has been proven to work there is a large amount of future study that could be done on both the control algorithms and how well they work in larger networks. For this to be practical the PESQ algorithm would need to be modified to better suit this use of it. The primary reason for doing this would be to dramatically reduce the processing needed to calculate a MOS score allowing for more calls to be monitored at the same time by the system. It is also possible that a different algorithm could be used instead of PESQ as the system was designed to be modular and so multiple speech quality algorithms could be implemented and compared.

Once these improvements to the control system have been made tests could be carried out on networks with multiple links and nodes each carrying a number of VOIP calls all of which are controlled to certain target qualities. With this kind of setup research into how well the control algorithms can control multiple calls could be performed and the overall performance of the network compared to one in which no control is performed. The individual control and monitoring nodes of the network could also report to a central monitoring server where a complete view of the entire network and the ongoing quality of every call could be monitored and changed if necessary. These are just some of the possibilities that could be implemented with further research using the method described in this project.

Overall the feedback method of monitoring VOIP calls is highly accurate as long as it is set up properly and the control part has been proven to work. With more research it will be able to provide even more advantages over current methods and overcome the limitations of the current design.
Bibliography


[10] GSM Standard. 06.10, prl-ETS 300 036.


Appendix A

Example Configuration File

Listing A.1: Example Configuration File

```xml
<?xml version="1.0"?>
<!--Shared entities-->
<!DOCTYPE aspl SYSTEM "config.dtd">
<!--XML data-->
aspl

<!--Information about this file-->
<information>
  <author>Todd Bayley</author>
  <datewritten>Mar 15, 2007</datewritten>
  <purpose>VOIP Simulation config for server</purpose>
</information>

<!--Global information read by all processes-->
<globals>
  <tempdir>$tempdir</tempdir>
  <basedir>$basedir</basedir>
  <datadir>/home/todd/work/watri/data/sim</datadir>
  <subbands>1024</subbands>
  <microphones>1</microphones>
  <speakers>1</speakers>
  <verbosity>0</verbosity>
  <samplerate>8000</samplerate>
  <soundspeed>160</soundspeed>
</globals>

<!--Configuration for Real Time Kernel-->
<rtk>
```
<blocksize>128</blocksize>
<timeout>1000.0</timeout>
<!— Input and output drivers —>
<driver name="snd">
  <outfile>server_in.wav</outfile>
  <infile>../f12-8.wav</infile>
  <repeat>1</repeat>
  <realtime>0</realtime>
  <source>
    <channels>1</channels>
    <name>filesource</name>
    <sync>1</sync>
  </source>
  <drain>
    <channels>1</channels>
    <name>filedrain_in</name>
    <sync>0</sync>
  </drain>
</driver>
<driver name="snd">
  <outfile>server_out.wav</outfile>
  <drain>
    <channels>1</channels>
    <name>filedrain_out</name>
    <sync>0</sync>
  </drain>
</driver>
<driver name="snd">
  <outfile>server_feedback.wav</outfile>
  <drain>
    <channels>2</channels>
    <name>filedrain_pesq</name>
  </drain>
</driver>
<driver name="voip">
  <local_address>192.168.1.1</local_address>
  <local_rtp_port>1200</local_rtp_port>
  <iscaller>0</iscaller>
  <local_sip_url>sip:192.168.1.1:5070</local_sip_url>
  <remote_sip_url>sip:192.168.1.10:5070</remote_sip_url>
  <username>ASPL</username>
  <jitter>512</jitter>
<jitter_threshold>256</jitter_threshold>

<feedback>1</feedback>
<feedback_calculate>1</feedback_calculate>
<feedback_delay>600</feedback_delay>
<sample_algorithm>1</sample_algorithm>
<payload_type>0</payload_type>
<source>
  <channels>3</channels>
  <name>rtpsource</name>
  <sync>1</sync>
</source>
<drain>
  <channels>1</channels>
  <name>rtpdrain</name>
  <sync>0</sync>
</drain>
</driver>
<algorithm name="pesq"/>
</algorithm>

<!—-Port connections—->
<connection>
  <source>filesoure</source>
  <drain>rtpdrain</drain>
  <sindex>0</sindex>
  <dindex>0</dindex>
</connection>
<connection>
  <source>filesoure</source>
  <drain>filedrain_out</drain>
  <sindex>0</sindex>
  <dindex>0</dindex>
</connection>
<connection>
  <source>rtpsource</source>
  <drain>filedrain_in</drain>
  <sindex>0</sindex>
  <dindex>0</dindex>
</connection>
<connection>
  <source>rtpsource</source>
  <drain>filedrain_pesq</drain>
  <sindex>1,2</sindex>
  <dindex>0,1</dindex>
```xml
<connection>
<connection>
<source>rtpsource</source>
<drain>pesqin</drain>
<sindex>1,2</sindex>
<dindex>0,1</dindex>
</connection>
</rtk>
</aspl>
```