This degree project has been made in collaboration
with Nordicstation
Supervisor at Nordicstation: Håkan Rydin

NORDICSTATION

nsAnalyser - Speech quality testing application for telephone service

nsAnalyser – Talkvalitetstestapplikation för telefonitjänst

Alexander Stahl

Degree Project in
Computer Engineering
First cycle, 15 hp
Supervisor at KTH: Jonas Wåhslén
Examiner: Ibrahim Orhan
School of Technology and Health
TRITA-STH 2013:28

Royal Institute of Technology
School of Technology and Health
136 40 Handen, Sweden
http://www.kth.se/sth

Abstract

This degree project was made in collaboration with Nordicstation. The project task was to develop a testing application for a self-developed telephone survey service, which uses third party software. This third party software showed to be unreliable at higher loads. The purpose of the application is to analyse the speech quality of clients connected to the service. This report gives an introduction to the speech quality algorithms Perceptual Evaluation of Speech Quality (PESQ) and Single Sided Speech Quality Measure (3SQM). It also gives descriptions of the methods used to develop the application. The final chapters in this report are about the testing of the telephone service. The primary result of the testing was that the telephone service is unable to acceptably handle 80+ clients and recommendations to Nordicstation is to set a maximum of parallel connected clients to 80 or find an alternative to the third party software currently in use.

Keywords: *Speech quality measurement, 3SQM, PESQ, Ozeki, SIPSorcery, phone call simulation*

Sammanfattning

Detta examensarbete har gjorts i samarbete med Nordicstation. Projektets uppgift var att utveckla ett test program för at testa en egenutvecklad telefonundersökning-tjänst, baserad på tredjeparts mjukvara. Denna tredjeparts mjukvara visade sig vara opålitlig vid högre belastning. Syftet med programmet är att analysera samtals kvalitéten på de klienter som är anslutna till tjänsten. Denna rapport ger en introduktion till ljudkvalitetsalgoritmer så som Perceptual Evaluation of Speech Quality (PESQ) och Single Sided Speech Quality Measure (3SQM). Rapporten går även igenom de metoder som använts för att utvecklat programmet. De sista kapitlen i denna rapport är om själva testningen av telefonitjänsten. Det primära resultatet av testningen var att telefontjänsten inte kan hantera 80+ klienter acceptabelt och rekommendationer till Nordicstation är att sätta ett tak på maximalt parallellt anslutna klienter till 80 eller hitta ett alternativ till den tredjeparts mjukvara som nu används.

Nyckelord: *Talkvalitetsanalys, 3SQM, PESQ, Ozeki, SIPSorcery, telefonsamtalssimulation*

Contents

1 Introduction 1
1.1 Background 1
1.2 Goals 1
1.3 Limitations 1
1.4 Methods 1
1.5 Abbreviations 2
1.6 Chapters 2
2 Speech Quality Algorithms 3
2.1 Introduction 3
2.2 Perceptual Evaluation of Speech Quality 4
2.3 Single Sided Speech Quality Measure 5
2.3.1 Vocal tract analysis and unnaturalness of speech 5
2.3.2 Analysis of strong additional noise 5
2.3.3 Interruptions, mutes and time clipping 5
3 Methods 7
3.1 Frameworks 7
3.2 Design 7
3.3 Analysis 8
3.4 Configuration 9
4 Implementation 11
4.1 The testing module 11
4.1.1 Proxy 11
4.1.2 Client 12
4.1.3 Client manager 13
4.1.4 Configuration handler 13
4.2 The analysing module 13
4.3 Configuration file 15
4.3.1 Start 15
4.3.2 Run 16
4.4 Problems 17
4.4.1 Looping SIP replies 17
4.4.2 DTMF detection 17

5 Analysis 19
5.1 Testing 19
5.2 Result 19
6 Conclusion 21
6.1 Application 21
6.2 Testing 21
7 Further development 23
References 25
Picture references 26

1 Introduction

This chapter gives a short introduction to the assignment with background, goals and limitations. This chapter also contains abbreviations used in this paper and short descriptions of all the chapters in this paper.

1.1 Background

The assignment was conducted at Nordicstation, a software company that develops customised computer systems based on customer specification. One of the products developed by Nordicstation is a telephone survey service using third party software. At high load when the service processes a high amount of calls at the same time, the third party software became unreliable and the service showed negative trends in speech quality. At the time of this assignment, there was no way of stress testing this service locally before shipping it to the customer. This is a crucial problem as new updates of the third party software caused new problems in the service, such as crashing after a certain time interval. Therefore there was a need to develop a test environment that simulates calls and measures the speech quality of these calls, so that the service can be confidently shipped by Nordicstation to the customer.

1.2 Goals

The goal was to develop an application that connects a number of software phones to the telephone service and the speech quality. It should also be able to simulate an active human caller to create a test environment that is as close to real life as possible, meaning that application should be able to send speech, either from file or text-to-speech, and button signals. This call simulation and number of calls should be configurable.

A measurement should also be run on the current telephone service to test the speech quality at different load levels.

1.3 Limitations

I and the tutor at the company have jointly agreed upon the following limitations to make the assignment possible within the given time.

- The prototype should be able to handle at least 10 parallel connections.
- Only algorithms for digital audio analysis will be investigated, no analogue algorithms will be investigated.
- The development is done in C#.

1.4 Methods

An analysis of speech algorithms was conducted to decide upon which algorithm to implement in the application.

The following SIP/RTP frameworks was investigated and evaluated for use in the application:

- Ozeki SDK – SIP/RTP Framework for softphone development.
- SIP .NET – Lightweight SIP Framework for .NET applications.
- SIPSorcery – Open-source SIP managing project.

The use of a PBX/SIP proxy was evaluated, either Asterisk as a PBX or write a customised SIP proxy using SIPSorcery.

All these methods are more thoroughly explained in the chapter 3.

1.5 Abbreviations

This report uses the following abbreviations:

- SIP - Session Initiation Protocol
- ITU - International Telecommunication Union
- ITU-T - Telecommunication Standardization Sector
- PBX - Private branch exchange, a telephone switch
- VoIP - Voice over IP
- dBO - Decibel (overload)
- DTMF - Dual-tone multi-frequency signalling
- CPU - Central Processing Unit
- RTP - Real-time Transport Protocol

1.6 Chapters

Here is a short description of the chapters of this report.

Chapter 1: Introduction
Presents the project assignment with goals, limitations and methods.

Chapter 2: Speech Quality Algorithms
Introduction to speech quality algorithms and short descriptions of two such algorithms, Perceptual Evaluation of Speech Quality (PESQ) and Single Sided Speech Quality Measure (3SQM).

Chapter 3: Methods
Description of the methods evaluated and used by the application.

Chapter 4: Implementation
How the methods in chapter 3 was implemented in the application.

Chapter 5: Analysis
The testing of the telephone service and a analysis of the result of these tests.

Chapter 6: Conclusion
Conclusions of this assignment.

Chapter 7: Future Development
Authors thoughts about possible future work.

2 Speech Quality Algorithms

This section will give a short introduction to speech quality assessment and minimal descriptions of two algorithms needed to digitally analyse and measure speech quality, Perceptual Evaluation of Speech Quality (PESQ) and Single Sided Speech Quality Measure (3SQM). It is these algorithms that are used by the application developed in this degree project. More in-depth descriptions about these algorithms can be found in the respective algorithms standard documentation.

2.1 Introduction

Speech quality can be measured either by a subjective or objective testing. A subjective speech quality measurement uses humans for quality analysing, meaning that test subjects (humans) listen to the output speech signal and determine if the quality was good or not. An objective speech quality measurement does not use humans but instead it uses algorithms to model the test subjects and their analysis. Both subjective and objective testing results in a Mean Opinion Score (MOS), which is a scale from one to five where one being bad quality and five being excellent quality. [1]

Objective speech quality measurement algorithms can make use of two different methods. Figure 1 illustrates the differences between these methods:

- A “double-ended” or “intrusive” method has access to both the original signal on the talker side and the received degraded signal on the listener side. It then compares the degraded signal to the original signal.
- A “single-ended” or “non-intrusive” method uses the received degraded signal for quality estimation without information about the original reference signal.

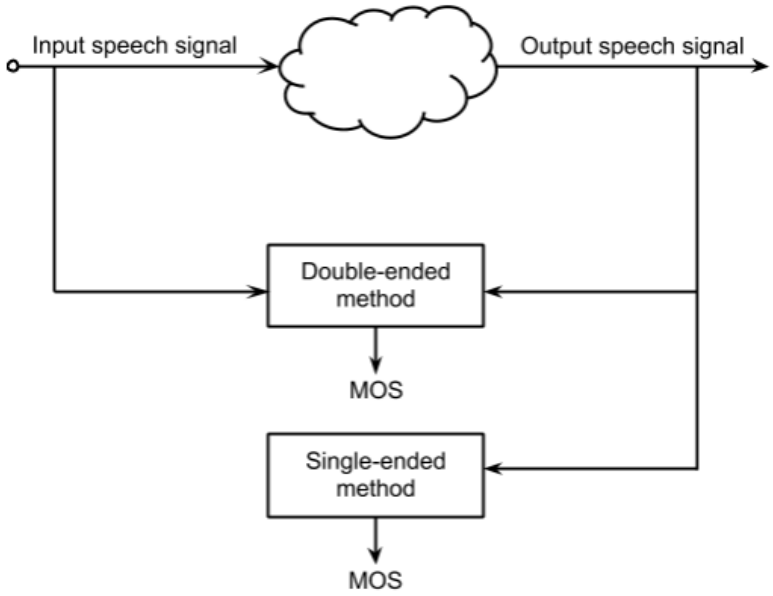

Figure 1 - Simple illustration of double-ended and single-ended methods. [F1]

2.2 Perceptual Evaluation of Speech Quality

This section is a simplified description of Perceptual Evaluation of Speech Quality (PESQ) described in ITU-T P.862 (02/01) [2] which this entire section uses for reference.

Perceptual Evaluation of Speech Quality (PESQ) is family of standards which together describe a double-ended algorithm for automated measurement of the quality of speech. PESQ is designed to model subjective tests. PESQ was succeeded by a new ITU-T standard in 2011, the Perceptual Objective Listening Quality Assessment (POLQA) described in ITU-T Rec. P.863 [3].

The first step of PESQ is to compute a series of delays between the original signal and the degraded (output) signal, one for each interval. A start and stop point is calculated for each of these intervals, in order to align the degraded signal with the original signal in time with limited delay.

The next step of PESQ is to compare the original signal with the aligned degraded signal through the perceptual model. To make this comparison both the original and degraded signal is transformed to an internal representation of how the human hearing system would perceive the speech found in the original and degraded signals. This internal representation is then passed to the cognitive model which proceeds in performing a number of measurements to finally omit a predicted MOS. Figure 2 show a simplified model of the PESQ algorithm.

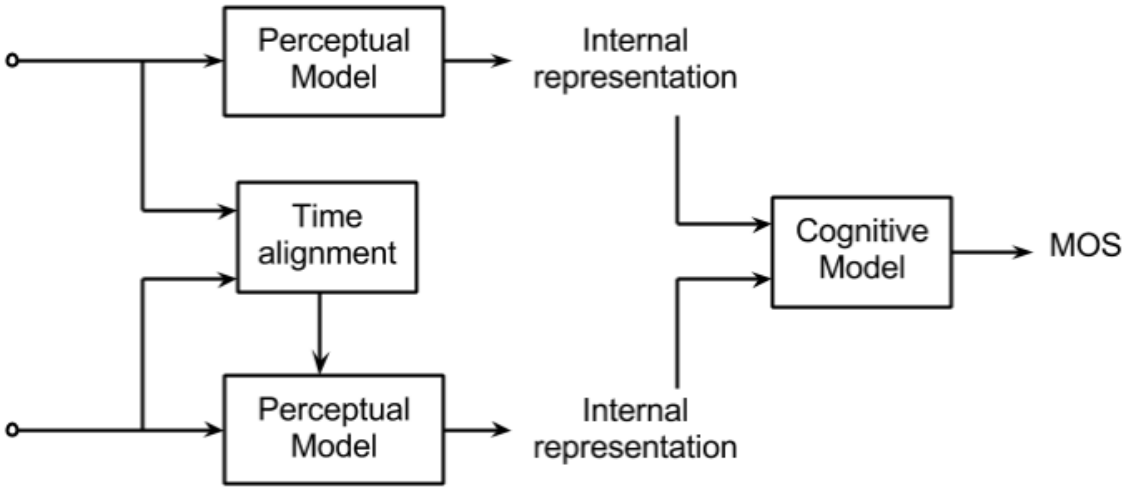

Figure 2 - Simple illustration of PESQ. [F2]

2.3 Single Sided Speech Quality Measure

This section is a simplified description of Single Sided Speech Quality Measure (3SQM) described in ITU-T P.563 (05/04) [4] which this entire section and subsections uses for reference.

Before a measurement of speech quality can be run the 3SQM algorithm must first pre-process the signal by running a voice activity detector (VAD) to identify the speech part of the signal to calculate the speech levels correctly. It then proceeds to analyse the pre-processed speech by using a set of key parameters, which together determine the predicted MOS.

The key parameters can be divided into three main blocks:

- Vocal tract analysis and unnaturalness of speech
- Analysis of strong additional noise
- Interruptions, mutes and time clipping

2.3.1 Vocal tract analysis and unnaturalness of speech

This block analyses the signal for unnaturalness in the speech. The unnaturalness of speech is rated differently for male and female voices and if the speech signal contains strong robotisation, i.e. too much periodicity, another gender-independent separation is made. During this phase the algorithm looks for the occurrence of tones like dual-tone multi-frequency (DTMF) signals, sent when a button is pushed during a call, and other high periodic non-speech signals.

A typical error in packet-based communication is packet loss. Some speech codecs implement concealment methods to increase received speech quality and some of these methods use packet repetition, i.e. replacing a lost packet with a previously successfully received packet. Due to this fact the 3SQM method takes packet repetition in account when measuring the unnaturalness of speech.

2.3.2 Analysis of strong additional noise

The noise analysis is used to measure different characteristics of signal noise. A decision is made if the noise is part of the main degradation. If this is the case then the type of noise is determined. The noise is either static and appears throughout the entire signal, i.e. no relation to the signal power, or the noise shows relation to the signal power envelope.

If the noise is classified as likely static then detectors try to qualify the extent of “local” noise and “global” noise. Local noise is used to describe signal parts found between phonemes, which are the small sounds are made to build words, and global noise is the signal between utterances, which is the continuous parts of speech beginning and ending with silence.

2.3.3 Interruptions, mutes and time clipping

Interruptions and speech mutes form a separate distortion block. These kinds of distortions can only partly be detected by vocal tract investigation, hence a separate analysis is done to detect unnatural mutes and time clippings in the speech signal. There are two forms of signal

interruption, temporal speech lipping and speech interruption both of which lead to loss of data.

Temporal clipping can occur if voice activity detection is used, or when the signal is interrupted. Temporal clipping cuts off a bit of the speech in the short time it takes to detect said speech. Speech interruption occurs during active speech parts. 3SQM method is able to differentiate between normal word endings and unnatural signal interruptions as well as abnormal intervals of silence in an utterance.

3 Methods

This section of the report describes the methods used to build the application and the design of the application.

3.1 Frameworks

The following frameworks were chosen by Nordicstation to be evaluated for use in the application. Not all of these frameworks are used in the final application:

- Ozeki SDK [5] - A C# .NET framework for development of softphones, web phones and call centres.
- SIP .NET [6] - Lightweight framework providing SIP communication for .NET applications.
- Asterisk [7] - An open source framework for private branch exchange (PBX), a telephone switch, and VoIP gateway applications written in C.
- SIPSorcery [8] - An open source application for SIP management written in C#.

Speech quality measurement algorithms will be implemented with the C programs provided by ITU-T.

3.2 Design

The initial system design, as seen in figure 3, was to set up a PBX server as a SIP proxy and have softphone threads managed by a separate server. The softphone server would register a number of softphones to the PBX. The PBX would receive SIP requests and forward them to the softphone server. This design was rejected by Nordicstation as a lightweight system with proxy and softphone handled by the same application, this to eliminate additional degradation of speech that may be caused the use of separate servers.

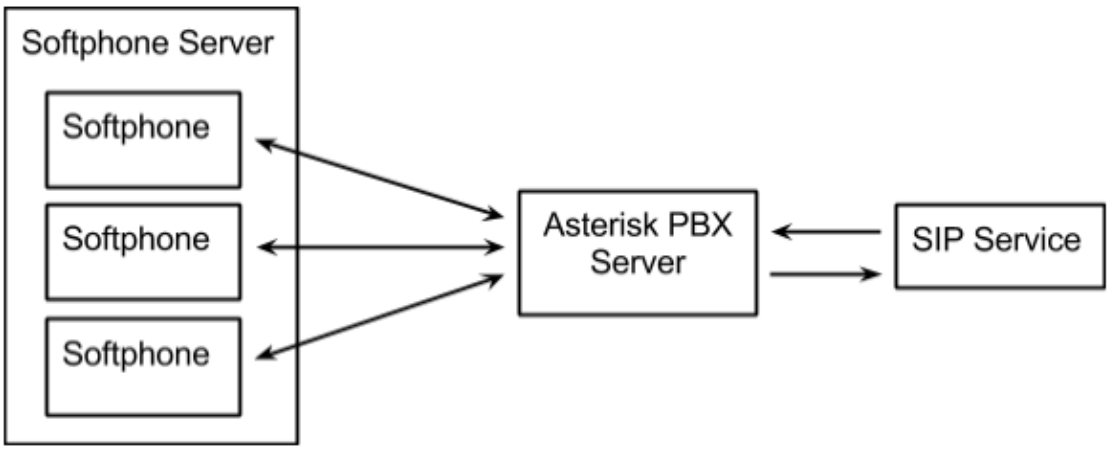

Figure 3 - Initial application design.

The final design, pictured in figure 4, was instead to integrate a proxy into the softphone application, which itself handles a number of softphone threads. The proxy receives SIP requests from the SIP service and forwards them internally to the right softphone thread. RTP connection is then established between the SIP service endpoint and the softphone thread.

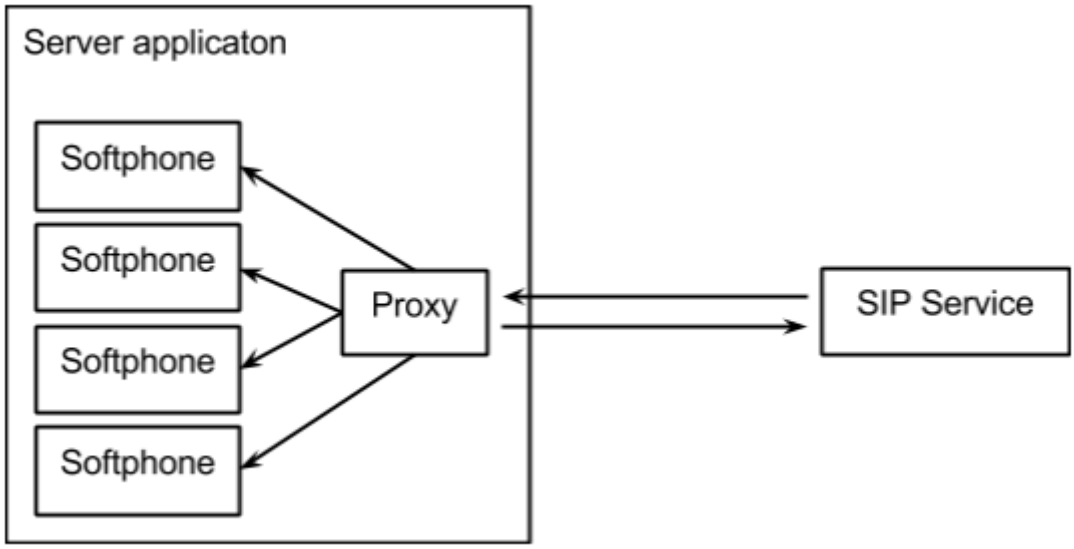

Figure 4 - Final application design.

3.3 Analysis

A decision was made to keep the analysis application separate from the testing module application. The testing module saves all test data, i.e. the speech, during testing but does not analyse it. Afterward, at an unspecified point in time, the analysing module can be run to evaluate the test data collected by the testing module, this to keep analysis processing time from affecting quality of the speech stream.

Due to the time limitation of this project the speech quality algorithm used in the system is Single Sided Speech Quality Measure (3SQM), as a single-ended method is easier to implement than a double-ended. To successfully analyse the speech quality with Perceptual Evaluation of Speech Quality (PESQ), a separation must be made and that was proven difficult because the speech files are sent from the service in small files and can be sent up to three files in succession without notable start and stop points.

3.4 Configuration

A request for an easy configuration of client behaviour and number of active clients was made by Nordicstation. To satisfy this condition, a lightweight custom scripting language was designed and implemented. The script is text based and uses a set of pre-defined keywords or commands, which are not case sensitive, to control the application. No particular scripting language was used as a base for this configuration script.

4 Implementation

This chapter will give a basic description on how the application was built and some of the problems that occurred during development. The application is split into two separate parts, a testing module and an analysing module. Both can be run independently however the analysing module must have recorded sound to analyse, which are recorded by the testing module.

4.1 The testing module

The testing module is the heavy part of the application, where heavy means that it is the most resource consuming part. Its purpose is model a testing environment where a number of human testers are actively using the service. The testing module is split into four main parts: Proxy, Client, Client manager and Configuration handler. Figure 5 shows the relation of these parts.

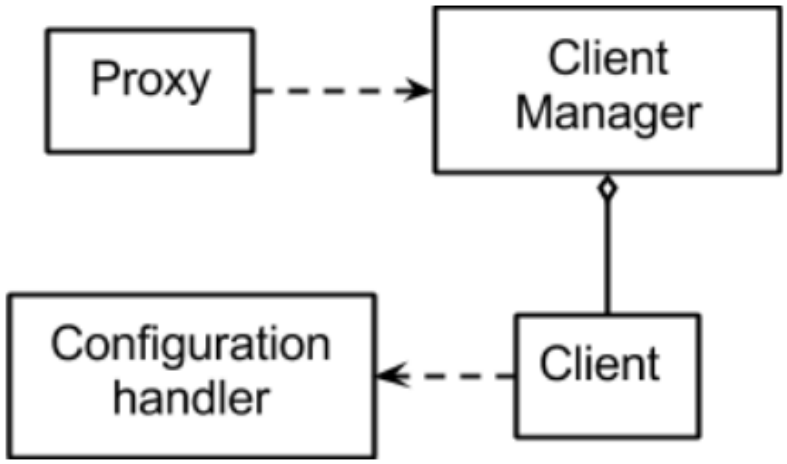

Figure 5 – Simple model of the applications parts.

4.1.1 Proxy

The proxy's purpose is to reroute the incoming and outgoing SIP traffic, mainly incoming SIP Invite. It also acts as a SIP Registrar server allowing the internal clients to register themselves in the proxy. It provides the outward interface for SIP messages through port 5060. The proxy is built using the core library found in the open-source project SIPSorcery. The proxy is a thread running parallel to the clients. Request handling occurs in an event based fashion, meaning that the thread is sleeping until a request or a response arrives at the listening port. There are two requests that require special handling by the proxy, invite requests and register requests. Figure 6 shows a flow chart how the proxy handles invite requests.

When an invite is received the proxy parses the information in the header and looks for the username, i.e. the phone number. It then asks the client manager to for the correct port of this

phone number. If no client with the specified phone number is found then the proxy answers the Invite request with a 404 Not Found response.

When a register request is received from an internal client, the proxy parses the phone number and port number from the request and passes it on to the client manager. The client manager saves the information in an array of registered clients. The proxy discards register request from other IP addresses than its own local address.

The proxy thread also keeps a record of active clients, i.e. the number of clients currently active in a call. The number is saved to an alignment file, which is a file with the number of active clients at certain points in time. This alignment file is used by the analysing module to map MOS and number of active clients. The thread actively checks every second if the number of active clients has changed since the last recording and if it has writes a new line in the alignment file with a timestamp and the number of clients. The alignment file is saved as a .txt text file.

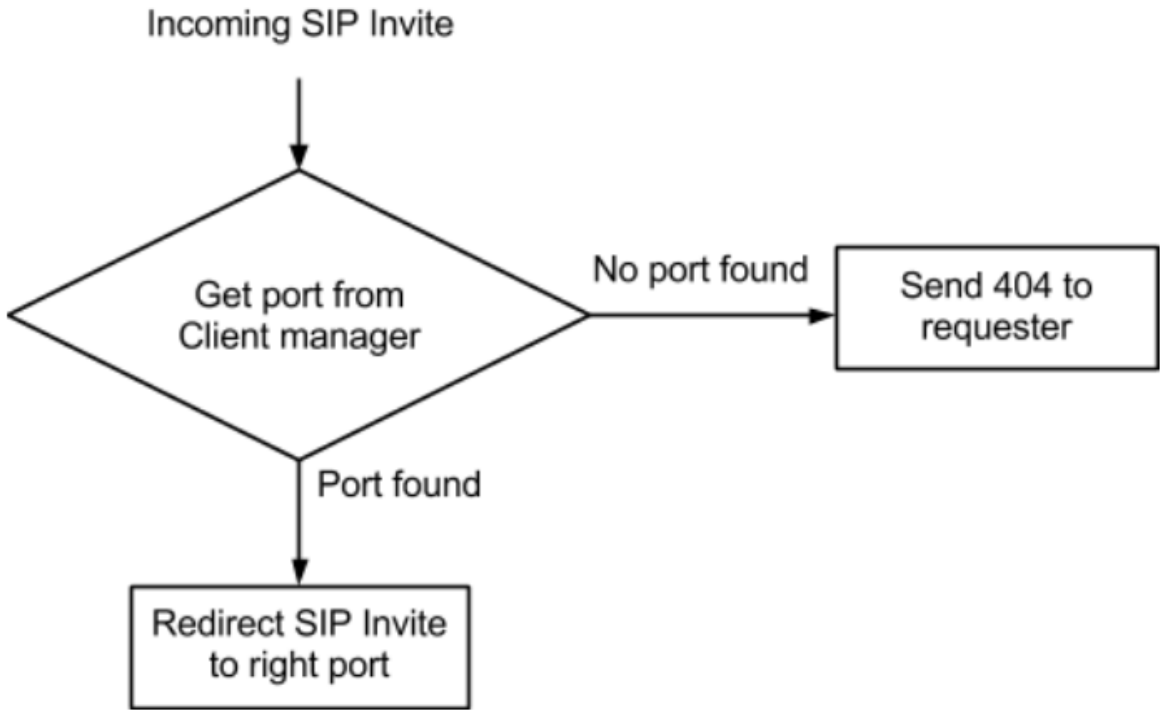

Figure 6 – Proxy behaviour on incoming SIP invites.

4.1.2 Client

The clients are threads simulating the human testers. Each client in the application is running in its own thread and does not share any memory with the other clients. Upon initiation the client opens a phone line using Ozeki SDK and sends a register request to the proxy. Then it waits for incoming calls, i.e. invite requests forwarded from the proxy.

Once a call has been initialised the client creates a wave file with a timestamp and the client number. The incoming speech stream is then written to this file continuously until the call has

ended. Once the wave file is created and ready for streaming the client proceeds to run each commands stated in the loaded run configuration.

After a call has been processed and simulated the clients disposes of allocated memory and reinitiate to start a new call.

4.1.3 Client manager

The client manager is a singleton object who is controlling an array of existing client's information, i.e. the client phone number and the port where this client sits. The clients are not in any particular order in the array, so when a request for a client's port arrives then the client manager matches the phone number against the array and when a match is found it breaks and returns the port number. If no matching phone number is found in the array the client manager throws a ***ClientNotFoundException***, which is picked up by the proxy who acts accordingly.

Each client number is unique so if a register request arrives with a number that already exists in the current client array, then the port number for that client will be overridden by the port number in the register request.

4.1.4 Configuration handler

The configuration handler purpose is to load the configuration file and check it for errors. The handler opens the provided file path and proceeds to read the entire document. For each line read the string is split on empty spaces, this to separate the command from the parameter. The read command is then checked using a switch with all the correct commands listed, if no matching command is found then the loader will throw a ***ConfigFileException*** with a message telling the user on which row the error occurred.

The loading is done in two steps. First the application loads and error checks the start configuration and saves it in the internal memory separate from the run configuration. Then the run configuration is loaded and saved. When clients requests the run configuration then copy of the run configuration will be given to the client.

Like the client manager the configuration handler is a singleton accessible by all other classes. The handler handles concurrency by using locks on the configurations so that only on client can access the run configuration at a time. The run configuration and start configuration are locked separately.

4.2 The analysing module

The analysing module is the speech quality assessment part of the application. The analysing module loads the recorded speech files and measures the quality of the signal using Single Sided Speech Quality Measure (3SQM). It makes use of the C-source implemented files provided by ITU-T [4]. The implementation of the analysing module is fairly simple, using P/Invoke [9], which is a library for calling C functions in C#, to call the compiled 3SQM library.

Modifications to the 3SQM source code were necessary to provide a correct interface for the C# written analysing module to invoke. A function called ***calculateMOS***, which takes the sound file path as an argument, was written which does the same job as the main function however returning a FLOAT value instead of printing it to the console. Then using P/Invoke the new function was exported to C#.

The analysing module starts by cleaning the speech file directory of files with a size less than 100 KB, because only files with longer speech time are to be analysed. Files that are smaller were cut by the exit chance and was unwanted because 3SQM includes random cuts in its analysis, meaning that the smaller files would corrupt the calculated MOS.

Then it proceeds to group the files based on the number of active clients at the moment of recording. The grouping is done by matching the timestamp in the file name to the alignment file created by the testing module. The matching reads then number of active clients in the alignment file until the timestamp of the speech file is smaller than that of the number about to be read.

After the cleaning and grouping is done the analysing module runs the actual evaluation of speech quality, using 3SQM. For each group an average MOS is calculated by adding up the MOS of every speech file in that group and then dividing it by the number of speech files. When the analysis is complete the average MOS for each group is saved in a CSV file [10]. The format of this file is each line in the file contains the number of active clients for that grouped data and the average MOS, separated by a semicolon. Figure 7 shows the flow of these steps.

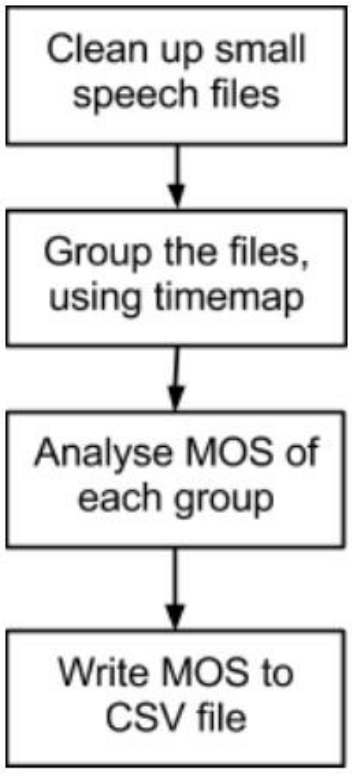

Figure 7 – Application flow of the analysing module.

4.3 Configuration file

The configuration file must contain two configuration blocks, a start configuration and a run configuration. The configuration file is compiled and loaded at runtime. Syntax errors in the configuration will be detected during loading and the application will not start. An example of a configuration file can be seen in figure 8, using keyword highlighting functionality of Notepad++.

4.3.1 Start

The start configuration is defined between the START keyword and the END keyword. This configuration contains parameters for application initialisation. Keywords that must be present in the start configuration are:

- **CLIENTS** <*parameter*> - Specifies the number of clients the application should create. Provided parameter must present and be an integer larger than zero. This configuration must be present in the start configuration.
- **PATH** <*parameter*> - Specifies the path where the recorded speech files should be saved. Provided parameter must be present and a string encased in quotation marks containing the desired path, e.g. "C:\Path\To\Speech\".

There are also optional keywords to employ in the start configuration:

- **EXITC** <*parameter*> - Specifies a chance of random exit during a call, where the parameter is the exit chance percentage. The exit chance percentage defines the chance of exit each second, e.g. a 43% chance means that each second there is a 43% chance of disconnection. Provided parameter must be present and an integer between 1 and 100.
- **INTERVAL** <*parameter*> - Specifies the interval in milliseconds between reconnections, i.e. after a client has completed a call how long should it wait to initiate a call again. If an interval is set then the client will randomise a number within that interval. Parameter must be present and an integer larger than zero or an interval, e.g. "1000-9000". Default reconnection interval value is 5000.

```
1   start
2   clients 132
3   path "C:\Path\To\Speech\"
4   end
5
6   run
7   say "Hello world"
8   wait 10000
9   dtmf 9
10  wait 3000
11  dtmf 1-4
12  wait 1000
13  play "C:\Sound\To\Play.wav"
14  end
15
```

Figure 8 - Configuration file example, from Notepad++.

4.3.2 Run

The run configuration is defined after the start configuration between the RUN keyword and the END keyword. This configuration defines the client behaviour during a call. The configuration is sequential and starts at the top and executes each command until the end. All the keywords within a run configuration are optional, however if no keyword is found then the call will be disconnected immediately. Possible commands for the run configuration are:

- **WAIT** <*parameter*> - This causes the client to wait, either specified milliseconds or wait for silence if the parameter is not set. Provided parameter is optional, must however be an Integer larger than zero if set.
- **DTMF** <*parameter*> - Client sends a DTMF signal. The provided parameter must be present and can either be a single number or interval, e.g. ‘9’ or ‘1-4’. If an interval is set then the client will send a random number within the specified interval. The number and interval must be an integer between 0-9.
- **SAY** <*parameter*> - Send speech using Text-to-speech. Parameter must be present and a string encased in quotation marks containing desired text to be played, e.g. “Hello World”.
- **PLAY** <*parameter*> - Send speech by playing a sound file. Parameter must be present and a string encased in quotation marks containing the path to desired sound file, e.g. “C:\Sound\To\Play.wav”.

4.4 Problems

There were some problems that occurred during development of the application. Below are descriptions of problems that occurred and how they were solved or avoided.

4.4.1 Looping SIP replies

During early tests of the application the sip proxy crashed after a period of time. Trying to identify the problem it was discovered that the CPU usage of the application increased over time until it finally crashed. The source of the problem was found during debugging of the application when it was discovered that SIP replies from internal clients were looped back to the proxy indefinitely. The replies were processed and the destination was computed from the top VIA header, which was set to the proxy's IP address which caused the request to be sent to the proxy again.

The problem was solved by changing the processing algorithm to determine if the reply was sent from an internal source and if so ignore all VIA headers that point to the internal address.

4.4.2 DTMF detection

During early testing of the application a problem with DTMF detection occurred. After sending approximately 25 DTMF signals the telephone service started to ignore further DTMF signals. The problem was at first thought to be service side, that the server somehow had an error causing the service to ignore DTMF signals. This theory was proven wrong when the service was tested with an actual phone, were well beyond 25 DTMF signals was detected properly by the service. The conclusion was that the problem must lie somewhere in the client.

The problem was debugged by using *Wireshark* [11] to capture and analyse the DTMF packets sent from the client to the service. What was discovered when the packets was analysed was that the timestamp started as normal but then quickly grew abnormally, within 25 packets the timestamp reached the maximum value and stayed as the maximum value for all consecutive packets. And as RFC 4733 [12], which is the default way to send DTMF signals with Ozeki SDK, states that RTP packets with the same timestamp are considered the same RTP event the service treated all packets sent after the 25 packet as the same packet.

This problem was solved by changing the way the client sends DTMF signals. Instead of sending them as RFC 4733 signals Ozeki SDK has the option of sending them as SIP Info messages, RFC 2967[13]. This change meant that an unlimited number of DTMF signals could be sent by the clients.

5 Analysis

This section describes the testing of the telephone service and the primary result of these tests.

5.1 Testing

The purpose of testing was to identify a breakpoint where the telephone service speech quality is greatly decreased or if such a breakpoint is non-existent evaluate the degradation rate with increasing number of clients.

The testing was conducted on a laptop computer and run over on separate occasions during a week long period. A total of seven tests were run, each with duration of 30 minutes. The application configuration during the test was to connect to the service and send random 1-9 DTMF signals with a 15 seconds interval for a total of one minute. The service responds by sending speech reading the survey questions, and with each DMTF signal a new question is sent. The speech received during the call is recorded and saved. Once the call is completed the client rested for seven seconds then proceeded to repeat the call again. If a call is dropped then the client will wait seven seconds and then reinitiate the call.

The recorded speech of these calls were analysed using the analysing module, which runs the implemented Single Sided Speech Quality Measure (3SQM) algorithm and compiles an average MOS for that test.

Each of the seven tests was run 30 times to get an average value with a reliable confidence interval for each test.

5.2 Result

The average MOS, calculated with a confidence interval of 99%, of each client during the above described test were as follows:

Clients	MOS
20	2,53 ± 0,10
40	2,55 ± 0,10
60	2,56 ± 0,10
80	2,17 ± 0,12
100	1,84 ± 0,14
120	1,82 ± 0,15
140	1,22 ± 0,15

Table 1 - Test results.

A graphical chart was created with Excel. The chart shows the average MOS and the upper and lower bounds of the confidence interval. Figure 6 show the graphical chart with the test results shown in Table 2.

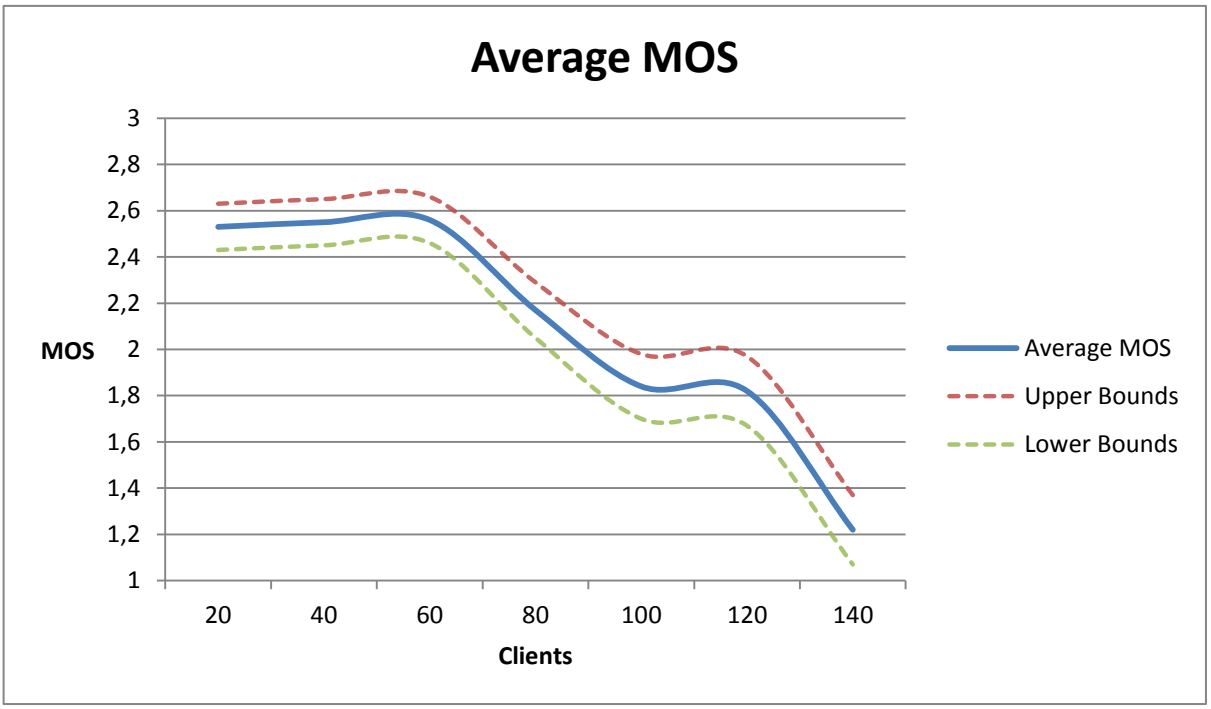

Figure 9 – Chart representation of test results.

Looking at the chart one can clearly see a clear degradation in speech quality after 60 clients and then another after 120 clients. However in this case a speech quality of above 2 is still acceptable in means of human perception, tested by manually listening to the recorded speech files. The result of this test shows that the service is unable to handle 80+ clients with acceptable speech quality.

6 Conclusion

The goal of this degree project was to build a test application for a telephone survey service, and run a test using the developed application to get a sense of the speech quality during different loads.

6.1 Application

The primary goal for the application was to be able to run several softphone threads and connect and conduct the survey provided by Nordicstations telephone service. Another goal was that the application should be configurable.

Both these goals were met by the developed application as a number of softphone clients can be created and run against the service. Basic configuration such as number of clients and client behaviour can be specified in the application configuration file.

6.2 Testing

As the testing shows a drop in speech quality with a load of 80 clients or more, a recommendation for Nordicstation is to debug the telephone service with a longer stress run. A solution to the problem is to set a maximum of 80 parallel calls, so that users will not have to experience a call with poor speech quality, or test which version of the third party software that handles scalability acceptably so that the service can handle multiple clients. Alternatively Nordicstation can investigate other alternatives than the third party software that is currently in use.

The testing should also be run on a dedicated server, as during the testing the computer was used as normal, to browse and code parallel with the testing. This may have caused the test results to be slightly incorrect.

A recommendation to Nordicstation is also to increase memory and CPU power on the server handling the telephone service to see if the problem is hardware related or not.

7 Further development

This chapter gives some personal points in what future development of the application could be. These are points are solely based on the writer's thoughts.

Further development would be to implement a user friendly graphical interface for configuration to complement the configuration script. The interface could also have the ability to show MOS charts directly in the application.

Another further development could be real time analysing on services in production environments where one client connects to the service and continuously measures the MOS and shows the data in chart.

For the analysing module a further development would be to save the average MOS to a database instead of saving it to a CSV file. This would mean that the accumulated over a longer period of time, by several separate runs of the test application. As it is now the analysing module recreates the CSV file each time the test is run, meaning that the previous test data is lost.

A choice to just stress test the application would be an additional development. When the application is set to just stress test no speech files are recorded, it instead focuses solely on client simulation.

References

[1] ITU-T, “Methods for subjective determination of transmission quality” 1996. [Online] Available: http://www.itu.int/rec/T-REC-P.800-199608-I [Taken May 6, 2013]

[2] ITU-T, “P.862 Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs” 2001. [Online] Available: http://www.itu.int/rec/T-REC-P.862-200102-I/en [Taken May 3, 2013]

[3] ITU-T, “P.863 Perceptual objective listening quality assessment” 2011. [Online] Available: http://www.itu.int/rec/T-REC-P.863-201101-I/en [Taken May 3, 2013]

[4] ITU-T, “P.563 Single-ended method for objective speech quality assessment in narrow-band telephony applications” 2004. [Online] Available: http://www.itu.int/rec/T-REC-P.563-200405-I/en [Taken May 3, 2013]

[5] “Ozeki VoIP SIP SDK” [Online] Available: http://www.voip-sip-sdk.com/ [Taken May 3, 2013]

[6] “Asterisk” [Online] Available: http://www.asterisk.org/ [Taken May 3, 2013]

[7] “Independentsoft SIP .NET” [Online] Available: http://www.independentsoft.de/sip/index.html [Taken May 3, 2013]

[8] “SIPSorcery project” [Online] Available: http://sipsorcery.codeplex.com [Taken May 3, 2013]

[9] “Platform Invoke Tutorial” [Online] Available: http://msdn.microsoft.com/SV-SE/library/aa288468%28v=vs.71%29.aspx [Taken May 3, 2013]

[10] Y. Shafranovich, “RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files” Available: http://tools.ietf.org/html/rfc4180 [Taken May 7, 2013]

[11] Wireshark Network Protocol Analyzer [Online] Available: http://www.wireshark.org/ [Taken May 15, 2013]

[12] H. Schulzrinne & T. Taylor, “RFC 4733: RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals”, 2006. [Online] Available: http://tools.ietf.org/html/rfc4733 [Taken May 15, 2013]

[13] S. Donovan, “RFC 2976: The SIP INFO Method”, 2000. [Online] Available: http://tools.ietf.org/html/rfc2976 [Taken May 15, 2013]

Picture references

[F1] Alexander Stahl, *"Simple illustration of double-ended and single-ended methods." 2013*
Based on Figure 1 in ITU-T, "P.563 Single-ended method for objective speech quality assessment in narrow-band telephony applications*"* 2004.

[F2] Alexander Stahl, "*Simple illustration of PESQ." 2013*
Based on Figure 1 in ITU-T, "P.862 Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs" 2001.