Final Thesis

MCapture; An Application Suite for Streaming Audio over Networks

by

Daniel Claesén

LITH-IDA-EX-ING--05/019--SE

2005-10-10
Final Thesis

MCapture; An Application Suite for Streaming Audio over Networks

by

Daniel Claesén

LITH-IDA-EX-ING--05/019--SE

2005-10-10

Supervisor: Dennis Andersson
Examiner: Henrik Eriksson
The purpose of this thesis is to develop software to stream input and output audio from a large number of computers in a network to one specific computer in the same network. This computer will save the audio to disk. The audio that is to be saved will consist mostly of spoken communication. The saved audio is to be used in a framework for modeling and visualization.

There are three major problems involved in designing a software to fill this purpose: recording both input and output audio at the same time, efficiently receiving multiple audio-streams at once and designing an interface where finding and organizing the computers to record audio from is easy.

The software developed to solve these problems consists of two parts; a server and a client. The server captures the input (microphone) and output (speaker) audio from a computer. To capture the output and input audio simultaneously an external application named Virtual Audio Cable (VAC) is used. The client connects to multiple servers and receives the captured audio. Each one of the client’s server-connections is handled by its own thread. To make it easy to find available servers an Automatic Server Discovery System has been developed. To simplify the organization of the servers they are displayed in a tree-view specifically designed for this purpose.
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© 2005, Daniel Claesén
Acknowledgements

I would like to thank my examiner Henrik Eriksson and my supervisor Dennis Andersson for their comments and support during this thesis. I would also like to thank Pär-Anders Albinsson and Mattias Johansson at the Department of Systems Engineering and IT Security for their help with MIND-related questions. Thanks also to Johan Allgurén, Richard Andersson, Andreas Frey, Mattias Johansson, Fredrik Mörnstedt, Sören Palmgren, Sofie Pilemalm and Joakim Stenius at the Department of Systems Engineering and IT Security for helping me test the software.
Abstract

The purpose of this thesis is to develop software to stream input and output audio from a large number of computers in a network to one specific computer in the same network. This computer will save the audio to disk. The audio that is to be saved will consist mostly of spoken communication. The saved audio is to be used in a framework for modeling and visualization.

There are three major problems involved in designing a software to fill this purpose: recording both input and output audio at the same time, efficiently receiving multiple audio-streams at once and designing an interface where finding and organizing the computers to record audio from is easy.

The software developed to solve these problems consists of two parts; a server and a client. The server captures the input (microphone) and output (speaker) audio from a computer. To capture the output and input audio simultaneously an external application named Virtual Audio Cable (VAC) is used. The client connects to multiple servers and receives the captured audio. Each one of the client’s server-connections is handled by its own thread. To make it easy to find available servers an Automatic Server Discovery System has been developed. To simplify the organization of the servers they are displayed in a tree-view specifically designed for this purpose.
Table of contents

1 INTRODUCTION .........................................................................................................................................1

2 TECHNICAL BACKGROUND ..................................................................................................................3
  2.1 ABOUT .NET ........................................................................................................................................3
  2.1.1 C# ..................................................................................................................................................3
  2.1.2 Common Language Runtime ........................................................................................................3
  2.2 VIRTUAL AUDIO CABLE ...................................................................................................................3
  2.3 DIGITAL AUDIO ...............................................................................................................................3

3 THE MCAPTURE APPLICATION-SUITE ..........................................................................................5
  3.1 OVERVIEW ........................................................................................................................................5
  3.2 MCAPTURESERVER, THE SERVER APPLICATION ..........................................................................6
  3.3 MCAPTURECLIENT, THE CLIENT APPLICATION .............................................................................7
    3.3.1 The status messages section .......................................................................................................8
    3.3.2 The available servers section .....................................................................................................8
    3.3.2.1 The tree-view ...........................................................................................................................8
    3.3.2.2 Buttons ....................................................................................................................................9
    3.3.2.3 The status bar ........................................................................................................................10
    3.3.3 Menu items ..................................................................................................................................10

4 THE MCAPTURE APPLICATION SUITE IN DEPTH .........................................................................13
  4.1 THE AVAILABLE SERVER DISCOVERY SYSTEM ...........................................................................13
  4.2 CONNECTION ....................................................................................................................................14
    4.2.1 Audio format negotiation ............................................................................................................14
    4.2.2 Capturing audio ..........................................................................................................................15
      4.2.2.1 Which buffer-parts contain audio? .......................................................................................16
      4.2.2.2 Down-sampling audio .........................................................................................................16
    4.2.3 Data communication ..................................................................................................................17
      4.2.3.1 The data protocol ................................................................................................................17
      4.2.3.2 The audio protocol .............................................................................................................17
    4.2.4 Abnormal termination ................................................................................................................18
    4.2.5 Disconnection ............................................................................................................................18
    4.2.6 File organization .........................................................................................................................18

5 THE SEQUENCE OF WORK ...............................................................................................................19
  5.1 RESEARCH .........................................................................................................................................19
    5.1.1 Capturing user desktop activities ..............................................................................................19
    5.1.2 Speech codecs ...........................................................................................................................21
  5.2 IMPLEMENTATION ............................................................................................................................21

6 THE TEST ..............................................................................................................................................23
  6.1 TEST SETUP ......................................................................................................................................23
  6.2 AUDIO RESULTS ..............................................................................................................................23
  6.3 PERFORMANCE RESULTS ...............................................................................................................23
  6.4 CONCLUSION OF THE TEST ..........................................................................................................25

7 CONCLUSION .......................................................................................................................................27
  7.1 RESULTS ............................................................................................................................................27
  7.2 FUTURE WORK ................................................................................................................................27
    7.2.1 Real-time playback of received audio .......................................................................................27
    7.2.2 Capturing user desktop activities ............................................................................................27
    7.2.3 Audio compression ....................................................................................................................28
    7.2.4 Sending data via multicast .........................................................................................................28
    7.2.5 Configuration of MCapture and VAC .......................................................................................28

REFERENCES ............................................................................................................................................29
APPENDIX 1: INSTRUKTIONER INFÖR TEST AV MCAPTURESERVER OCH MCAPTURECLIENT
Table of figures

FIGURE 1. THE SIGNAL TO BE SAMPLED. ................................................................. 3
FIGURE 2. THE SIGNAL WITH SAMPLES. .............................................................. 4
FIGURE 3. THE SIGNAL WITH ROUNDED OFF SAMPLES. ...................................... 3
FIGURE 4. THE RESULTING DIGITAL SIGNAL. ..................................................... 4
FIGURE 5. THE CLIENT RECEIVES AUDIO DATA FROM MULTIPLE SERVERS. .......... 5
FIGURE 6. THE SELECT CAPTURE DEVICE WINDOW .......................................... 6
FIGURE 7. THE MAIN WINDOW OF MCAPTURESERVER ........................................ 6
FIGURE 8. THE MAIN WINDOW OF MCAPTURESERVER, AFTER THE CLIENT HAS CONNECTED. .......... 7
FIGURE 9. THE MAIN WINDOW OF MCAPTURECLIENT ....................................... 7
FIGURE 10. THE AVAILABLE SERVERS SECTION ............................................... 8
FIGURE 11. THE STATUS BAR ............................................................................. 10
FIGURE 12. THE FILE MENU AND THE CONNECT MANUALLY DIALOG ............... 10
FIGURE 13. THE OPTIONS MENU ITEM .............................................................. 11
FIGURE 14. THE AUDIO FORMAT OPTIONS WINDOW ........................................ 11
FIGURE 15. TWO EXAMPLES OF THE MULTICAST COMMUNICATION IN ASDS. ..................... 13
FIGURE 16. THE AUDIO FORMAT NEGOTIATION .............................................. 14
FIGURE 17. THE AUDIO CAPTURE PROCESS ...................................................... 15
FIGURE 18. AN EXAMPLE OF HOW THE EXTRA BUFFER WORKS ....................... 16
FIGURE 19. THE DATA COMMUNICATION PROTOCOLS .................................... 17
FIGURE 20. AN EARLY VERSION OF MCAPTURECLIENT ..................................... 21
FIGURE 21. MCAPTURECLIENT BEFORE THE TREE-VIEW WAS IMPLEMENTED. ......... 22
FIGURE 22. BANDWIDTH USAGE GRAPH FROM SERVER .................................. 24
FIGURE 23. CPU-PERFORMANCE GRAPH FROM A SERVER ................................. 24
FIGURE 24. BANDWIDTH USAGE GRAPH FROM THE CLIENT ............................. 24
FIGURE 25. CPU-PERFORMANCE GRAPH FROM THE CLIENT ............................. 24
1 Introduction

This chapter will describe the main task and purpose of this thesis. It will also describe the approach used to solve the task.

The main task of this project is to develop prototype software to capture audio and distribute it over a network. The audio consists mainly of spoken communication. The audio sources are both the input (microphone) and the output (speaker) audio from a large number of computers. The audio destination is one specifically chosen computer in the same network. All received data are to be saved for later playback. The software should be configurable and able to record audio from a large number of computers (possibly over 100) at the same time. The programming language used should be C#.

In addition to the main task there are secondary tasks that should be implemented if time permits. One task is to include the ability to capture moving pictures of the computer desktop in addition to the audio and also to make it possible to view these in real time. Another task is to send the audio (and picture) data via multicast. This will allow several viewers to receive the data at the same time in a bandwidth-effective way.

The captured data will be used in a framework for modeling and visualization called MIND. MIND has been developed at FOI - Swedish Defence Research Agency to create models of operations involving the military, police and Swedish Rescue Services Agency. In these models all available information from the operation is saved in the form of text, pictures, audio, video etc. The models make it possible to present the course of events in a debriefing after the operation. The debriefing will be a summary of the operation and help all involved parties to better understand what really happened. The models can also be used in several other areas, for example as training material when educating new police officers and military personnel, or in command and control analysis. (Morin & Jenvald & Thorstensson, 2003)

An initial research will be made to see if already existing software can be used or modified to solve the task. If no such software is found, new software will be developed. When the software is developed it will be tested to measure the bandwidth- and CPU-usage. When all the previous steps are completed a detailed report will be written. A more detailed description of the approach can be found in chapter 5 – The sequence of work.
2 Technical background

This chapter will deal with techniques and programs that are used by the MCapture applications.

2.1 About .NET

Explaining what .NET is is not easy. No attempt will be made to explain everything that is included in .NET; instead this section will focus on the parts that are relevant to this thesis. For a more general explanation of .NET see Basics of .Net (Microsoft, 2005a).

2.1.1 C#

When .NET was launched it replaced Microsoft’s previous Component Object Model (COM). .NET comes with many new features; one of them, C#, is an entirely new programming language. C# is an object oriented programming language with a syntax closely resembling that of Java. Like Java, the .NET-code is compiled to an intermediate language which is then run by a virtual machine. The .NET virtual machine is called Common Language Runtime (CLR).

2.1.2 Common Language Runtime

The CLR can be seen as an additional layer between the software and hardware. Adding another layer has its upsides and downsides. One upside about the CLR is that it takes care of memory management. This is generally a good thing since hand coded memory management often contains bugs. The downside is that the CLR may be slower and use more memory than hand coded memory management.

2.2 Virtual Audio Cable

Virtual Audio Cable (VAC) is an application that can be thought of as a virtual soundcard. VAC is used to gain access to the output audio of a computer. In order to capture the output audio VAC must be chosen as the default audio device (see appendix 1 for a detailed description of how to configure VAC). This means all audio normally sent to the hardware soundcard will be sent to VAC instead. Through VAC the output audio can then easily be captured. The only problem remaining is that the user will not hear anything since no audio is sent to the hardware soundcard. This problem is solved with an audio repeater included with VAC. The audio repeater forwards the audio from VAC to the hardware soundcard which outputs it to the speakers. For more information see NTONYX – Virtual Audio Cable (NTONYX, 2005).

2.3 Digital audio

Digital audio is made up of samples. The sample-frequency (number of samples per second) is one parameter that decides the quality of the audio. The other is the bit-
depth. The bit-depth decides the level of accuracy in which a sample is saved. When looking at digital audio in the form of a graph (like figure 4), the sample frequency decides the horizontal resolution and the bit-depth decides the vertical resolution of the digital signal. High bit-depth and high sample-frequency results in high quality audio. CD-quality audio has a bit-rate of 16 and a sample-frequency of 44100 Hz.

Figure 1 shows a simple audio signal. When an audio signal is converted from analog to digital it must first be sampled. Each sample is represented by a bar in figure 2. The width of each bar represents the time until the next sample is taken. The height of each bar represents the sample-value of that particular sample. This is where the bit-depth comes in. The bit-depth decides between how many values the samples can vary. 8-bit audio-samples can vary between 256 different values while 16-bit audio-samples can vary between 65536 different values. In this simplified example there are only 9 different values (-4 to 4). Figure 3 shows the samples rounded off to the nearest of these values. Figure 4 shows the resulting digital signal.

![Figure 1. The signal to be sampled.](image)

![Figure 2. The signal with samples.](image)

![Figure 3. The signal with rounded off samples.](image)

![Figure 4. The resulting digital signal.](image)

When converting analog audio to digital it is also important to bear the Nyquist theorem in mind. According to the Nyquist theorem, when sampling audio the sampling frequency must be at least two times as high as the highest frequency that is to be captured. (Webopedia, 2001)
3 The MCapture application-suite

This chapter will begin with an overview of how the MCapture applications work. It will then continue by describing the user interface of the client and server applications. A more detailed technical description will be given in chapter 4 – The MCapture application suite in depth.

3.1 Overview

As seen in figure 5 the MCapture application suite consists of two applications: a server and a client. The server application will capture the input and output audio from the computer running it. It will save the audio locally and transfer it to the client over the network. The server application must be installed on every computer that audio is to be captured from. The client application only needs to be installed on one computer in the network. A computer can run both the server and the client application at the same time.

*Figure 5. The client receives audio data from multiple servers.*
3.2  **MCaptureServer, the server application**

The first time MCaptureServer is started the select capture device dialog in figure 6 will pop up. This dialog is used to choose which devices to capture audio from. In the combobox labeled “Capture Device 1 (mic)” the user is to select the soundcard that the microphone is connected to. In the second combobox Virtual Cable 1 should be selected. If the “Remember my choice” checkbox is checked, the devices selected will be saved to a file named “mcconfig.txt”. The server will use this file to select the same capture devices the next time the application is started.

![Figure 6. The select capture device window.](image)

After the user has chosen which capture devices to use the main window of the server application will appear (figure 7). The interface is simple and no changes can be made by the user. The “Input Format” label shows the selected audio input format and the “Output Format” label will show the selected audio output format once the client has connected. The “Status” text-area displays important status messages; it starts with a header of four lines. The first line shows the date and time the application was started, it also shows the local machine name (in this case SOFIE1). The second line shows the operating system version. The third line shows the Common Language Runtime version and the fourth shows the audio input format. After the header, information about the application execution will follow.

![Figure 7. The main window of MCaptureServer](image)
Figure 8 shows the main window after the client has connected. If any exceptions had been triggered during the execution they would have been displayed in the status window as well. When the server is shutdown the contents of the status window is automatically saved to a log file in the directory where the server application was installed. The log file will be named after the startup date and time according to the following format “YYYY-MM-DD hhmmss.txt”.

**3.3 MCaptureClient, the client application**

When MCaptureClient is started the user will see the main window in figure 9. The window is divided into two main sections, the available servers section and the status messages section.

![Figure 8. The main window of MCaptureServer, after the client has connected.](image)

![Figure 9. The main window of MCaptureClient.](image)
3.3.1 The status messages section

The status messages section can be found on the right half of the main window. The text area in it starts with a three line header that contains information about the date and time the application was started on. It also contains information about the local machine name, operating system and Common Language Runtime version. Every time the user connects to or disconnects from a server, an entry will be made in the text area. An entry will also be made if an exception occurs. All entries start with a timestamp that shows when the event occurred.

3.3.2 The available servers section

The left half of the main window contains the available servers section (figure 10).

![Available servers section](image)

Figure 10. The available servers section.

3.3.2.1 The tree-view

In the leftmost part of the available servers section the tree-view can be found. The tree-view displays the machine-name and status of the servers in the network. It also displays any groups that may have been added by the user. The groups and servers are displayed in the form of a tree, much like the folders view in Windows Explorer.

Servers are displayed with a computer screen icon to the left of their name. The color of the computer screen represents the status of that server. A gray screen means the server is unavailable. A blue screen means the server is available for connection. A red screen means the server is currently connected to the client.

Group names always start with “#” and groups are always displayed above servers on the same branch of the tree. Groups have a circle icon to the left of their name. The circle is either gray, blue, red or any combination of these colors. The color of the circle icon represents the status of the members in that group. If a group contains only
unavailable servers the circle icon will be gray. If the group contains both available and unavailable servers the circle will be half gray, half blue. If the group contains both unavailable and connected servers the circle icon will be half gray, half red and so on. The circle icons are made to be able to show any combination of members, but the proportions of the colors in the circle will not change to represent the percentage of members with a certain status. An example: a group containing nine unavailable servers will have a gray circle icon. If a connected server is moved to this group the circle icon will change to half gray, half red to show that it contains both unavailable and connected servers.

All groups and servers that are added to the tree will be placed at the root in alphabetical order and groups above servers. To order them the user must manually drag and drop the nodes to their right place. Groups can hold servers and other groups (servers cannot hold groups or other servers).

3.3.2.2  Buttons

There are a number of buttons in the available servers section. In the bottom left corner the “Update” button can be found. The “Update” button is used to find new servers and update the status of the servers already in the tree-view. Servers that are started while the client is running will automatically be added to the tree. Servers started before the client was started will not be added to the tree-view until the “Update” button is pressed. If the “Update servers in list only” checkbox is checked no new servers will be added to the list even if found. This can be useful if the server application is installed and running on all the computers in a network, but the client application user is only interested in recording audio from some of them.

The “Clear files” button can be found to the right of the “Update” button. Pressing it will cause all locally saved audio files on all computers currently running the server application to be purged. Since these files are saved only for backup reasons (in case the computer running the client application hangs or something else goes wrong) they can be safely deleted once the recording session is over.

In the bottom right corner of the available servers section the “Save” and “Load” buttons can be found. These are used to save and load the structure of the tree to/from file. No status information will be saved in the file. Instead, a status update will be automatically triggered once the tree is loaded. The application will also add all currently connected servers to the tree (if they are not already in it). This ensures that the loaded tree will reflect the current status of the servers.

In the middle right part of the available servers section the “Connect” and “Disconnect” buttons are found. The “Connect” and “Disconnect” buttons can be used when a server or a group is selected. If a group is selected all members of the group will be connected or disconnected. The “Connect all” and “Disconnect all” buttons affect all the servers currently in the tree. The “Disconnect all” button has a confirmation dialog to avoid unintentional disconnection.

The edit tree button section can be found in the top right corner of the available servers section. The “Add group” button will add a group to the tree with the name
typed in the textbox above it. Group names must be unique, but since “#” is always added to the beginning of the name a server can have the same name as a group (without the “#” of course). The “Remove node” button is used to remove the currently selected node in the tree, either a server or a group. If a group is removed all members in it will be removed as well.

3.3.2.3 The status bar
The status bar can be found at the very bottom of the main window (figure 11). It displays different status messages depending on the user’s actions. For example it will display messages regarding the load/save file status. It will also display messages regarding the drag and drop actions in the tree-view.

![Figure 11. The status bar.](image)

3.3.3 Menu items
As figure 12 shows the main menu has two items in it, “File” and “Options”. In the “File” menu item there are two items to choose from, “Connect manually” and “Close”. “Close” will close the application and “Connect manually” will bring up the connect manually window (figure 12). This is useful if the user wants to connect to a server that, for some reason, is missing from the list.

![Figure 12. The file menu and the connect manually dialog.](image)

The options menu item contains five items, “Always on top”, “Show detailed exception info”, “Tell server to save audio data locally”, “Tell server to send audio data” and “Show Audio Format Options” (figure 13). If “Always on top” is checked the MCaptureClient window will remain the topmost window on the user desktop even if it is not currently selected. If “Show detailed exception info” is checked additional debug info will be printed to the status window if an exception occurs. The option “Tell server to save audio data locally” sets whether the server will save the audio data locally or not. The “Tell server to send audio data” option sets whether the server will send audio data to the client or not. At least one of the “Tell server…” options must be checked. Otherwise the client will be unable to connect to any server.
“Show Audio Format Options” will bring up the Audio Format Options window (figure 14). This is where the user chooses the format to save the captured audio in. “Frequency” sets the number of samples per second to save. “Bitdepth” sets the number of bits per sample to use. “Clipping level” sets the minimum level of sound intensity that will start an audio capture. “Extra buffer” sets the delay (in seconds) to use when stopping an audio capture. If the sound intensity goes below the clipping level the server application will wait for one second (default) before stopping the audio capture (4.2.2.1 - Which buffer-parts contain audio). This is useful to capture the end of a spoken sentence and it also decreases the fragmentation of the audio. It is important to find a good balance between the “Clipping level” and “Extra buffer” values.
4 The MCapture application suite in depth

This chapter will explain what goes on behind the scenes during a normal MCapture session. The first part will explain the Available Server Discovery System. ASDS is used to find all available servers and display them in the tree-view of the client application. The second part will focus on the connection and data transfer procedure. This is a large section that explains the core features of the MCapture application suite. The reader is assumed to have some basic understanding of computer networking and programming.

4.1 The Available Server Discovery System

In order to be able the find available servers in an easy way, an Available Server Discovery System (ASDS) was developed. The system uses multicast-groups to allow for easy communication without the need to know individual IP-numbers. The UDP-protocol is used to transfer all multicast messages since multicast does not allow the use of the TCP-protocol. See Computer Networking (Kurose and Ross, 2005) for more information about multicast and computer networking in general.

Once a server is started it automatically joins the server multicast-group with the IP-address 224.168.100.2. The client will join the client multicast-group with the IP-address 224.168.100.3. A message sent to a multicast-group will automatically be distributed to all members of that group by the router. Only four different types of messages are sent to these multicast-groups, these are: add-, remove-, update- and clear-messages.

The update- and clear-messages are sent from the client to the router. The router forwards the message to the server multicast-group. The add- and remove-messages are sent from the servers to the router which forwards the messages to the client multicast-group (figure 15). The messages are very simple. The server-to-client messages are strings with the message tag followed by the local machine-name of the sender. The machine-name must be present in order for the client to determine which machine that sent the message. The client-to-server messages are strings consisting only of the message-tag. The machine-name of the client is not included since it is not needed by the client.

Figure 15. Two examples of the multicast communication in ASDS.
When the server application is started it sends an add-message to the client multicast-group to let the client application know that a new server is available. This requires that a client application is running and actually receiving the message, otherwise the information will be lost. For the client application to find servers that were started before the client the user must manually press the update button. This sends an update-message to the server application multicast-group which will cause all available servers to send an add-message to the client multicast-group. When a server is shutdown it sends a remove-message to the client multicast-group. This causes the client application to set the server status to unavailable. When the user presses the clear button a clear-message is sent to all currently running servers, this causes them to delete all locally saved audio files.

4.2 Connection

When the client application connects to the server application, they start to exchange information about the audio format to use. Once the audio format negotiation is done the server and client are considered to be properly connected. The client will enter a waiting state where it will remain idle until the server sends it some data to process. The server in turn will start scanning the audio sources. If audio is detected the server will tell the client to create a new file to hold the data the server is about to send. The server will then start sending the data to the client. The server will also save a backup copy of all the audio data locally. When audio is no longer detected from the audio source the server will tell the client to close the file. This process repeats until the client disconnects or the server is shutdown.

4.2.1 Audio format negotiation

The client starts the negotiation by sending a request for a specific audio output format (frequency, bit-depth, clipping level and extra buffer size). The values sent in the request are taken from the audio format options window (see 3.3.3, figure 14). The server uses these values to calculate an output audio format that it can down-sample the input audio format into. The server chooses a format as close to the requested one as possible and sends the chosen frequency and bit-depth as a reply to the client (figure 16). No down-sampling will be made if the client requests an output format of the same or higher quality than the input format. If this happens the server will send the audio in the original input format.

![Figure 16. The audio format negotiation.](image-url)
4.2.2 Capturing audio

The server uses a circular buffer divided into 16 parts to capture audio. A circular buffer means that when the buffer is full, the server will start over and write data to the beginning of the buffer again. The buffer size is set to be able to hold two seconds of uncompressed audio at the currently selected input audio format. The actual size of the buffer will vary depending on which input audio format is used.

The server will start filling the buffer from the beginning. Once the first of the 16 parts is full the server will check if it contains audio. If it does the server will check if a new file needs to be created. To create a new file the server gets the time when the recording of this particular part of the buffer began. The server then sends the time information to the client. Both the server and client create a new local file named after the start-time. These files will be used to hold the audio information captured by the server. The server continues by down-sampling the audio data to the chosen audio output format. The down-sampled data is sent to the client and then the audio is saved to file by both the client and the server. The server will now wait for the next part of the buffer to fill. Once that part is full the server checks it for audio. If this buffer contains audio as well, the server will not need to create a new file. It will simply down-sample the data, send it to the client and then they will both append the data to the previously created file. The client and server will continue to append the audio data to the same file until a buffer that does not contain audio is found. Once this happens the server will tell the client to close the file. The server and client will both respectively close their local files and then the process will start over (figure 17).

Figure 17. The audio capture process.
4.2.2.1 Which buffer-parts contain audio?

A part of the buffer is considered to contain audio if any sample-value in it is larger than the clipping value. A buffer that has no sample-value larger than the clipping value is considered to be empty. But a buffer considered to be empty may still be recorded. This can occur if the buffer examined before the empty one had a sample-value above the clipping level and the extra buffer value is set to at least 0.125s. In other words, the extra buffer value sets the number of empty buffers that will be considered to contain audio after a buffer with audio has been found (figure 18). An extra buffer value of one second equals eight buffers.

![Diagram of buffer parts](image)

Extra-buffer value = 0.5 seconds (4 buffers)
- Contains audio, will be recorded
- Empty, but will be recorded because of the extra-buffer value
- Empty, will not be recorded

Figure 18. An example of how the extra buffer works.

4.2.2.2 Down-sampling audio

MCaptureServer can down-sample the frequency, bit-depth or both. It can down-sample to any frequency that is evenly dividable by the input frequency. The bit-depth can be down-sampled from 16- to 8-bits.

Down-sampling the frequency from 44100 samples per second to 22050 is done by removing every other sample. Down-sampling from 44100 samples per second to 11025 is done by saving one sample and removing the following three samples, and so on. The down-sampler in the server application is capable of down-sampling to any sample rate that is evenly dividable by the audio input sample-rate.

Down-sampling the bit-depth from 16- to 8-bits is done by adding 32768 to the 16-bit value and then dividing the result by 257. The resulting value is then rounded off to
the nearest integer. This is done because 8-bit samples are unsigned integers (values ranging from 0 to 255) but 16-bit samples are signed integers (values ranging from -32768 to 32767).

For more information about digital audio see 2.3 - Digital audio.

### 4.2.3 Data communication

All data communication messages are delivered by the TCP-protocol for reliable transfer. The data communication is divided into two parts, the data protocol and the audio protocol (figure 19).

#### 4.2.3.1 The data protocol

The data protocol is very simple; it consists of only one part, the stream-number. The stream-number is a 32-bit signed integer that tells the client which type of data that will follow. This version of the data protocol only supports one type of data (audio data) but there are two different sources. A stream-number value of 1 means that audio data from the microphone will follow. A stream-number value of 2 means that audio data from Virtual Audio Cable will follow. A stream-number value of 0 means that the connection will be closed, no audio data will follow in this case. This approach was used to allow for an easy way to add more audio streams and also other types of streams (for example video streams).

#### 4.2.3.2 The audio protocol

The audio protocol consists of two parts, the first part decides the type of the second. The first part is a 32-bit signed integer called message-size and the second part is called message. Three different scenarios exist depending on the value of message-size. A message-size value of -1 means that a new audio file should be created. In this case message will be a string containing the start time of the audio; this string will be used as a filename for the audio file. A message-size value larger than 0 means that message-size number of bytes of audio data will follow in standard PCM-format. Depending on the audio format negotiation this format will vary. A message-size value of 0 means that the current audio file is to be closed. In this case no message will follow.

![Figure 19. The data communication protocols.](image-url)
4.2.4 Abnormal termination

The client and server applications are both designed to be able to handle abnormal terminations of one another. If a server computer crashes for some reason, the client will save the last received audio-data and then continue with its other tasks. If the client computer crashes, the servers will stop sending the captured audio-data over the network. The servers will continue to capture audio, but it will only be saved to disk locally.

4.2.5 Disconnection

The client disconnects from the server by sending it a 32-bit signed integer value of 0 while it is capturing. This will cause the server to stop capturing, it will then continue by sending the remaining buffered audio (if any) to the client. The server will then send a stream-number value of 0 to the client (which means the connection will be closed).

4.2.6 File organization

The backup audio files saved by the server are stored in a subfolder of the folder where the server is installed. The subfolder is named MCDATA and in it both the input and output audio files are saved. The files are named after the date and time of the capture. To distinguish input audio files from output audio files a special tag is added before the file extension. The tag is “mic” for input audio files and “VAC” for output audio file. The exact format of an input audio file is: “YYYY-MM-DD hhmmss_mic.wav”.

The audio files saved by the client are organized in subfolders to the folder where the client is installed. The folders will be named after the server and the audio source they come from. The input audio files from a server named SOFIE1 will thus be saved in a folder named: “SOFIE1_mic”. The files will be named only after the date and time they were captured. This is to allow for easy integration of the files into MIND. The exact format of the files is: “YYYY-MM-DD hhmmss.wav”.


5 The sequence of work
This chapter will give a detailed description of the sequence of work. The chapter will start with the research and then move on to the implementation of the application.

5.1 Research
In the beginning of the research no priority between the tasks of this thesis had been given. The author made the decision to focus mainly on the desktop capturing task because it seemed to be the more important task. Therefore most of the research time was focused on finding suitable techniques to capture pictures of the computer desktop. More than halfway through the research, the author was told that the priority had changed. Capturing the audio was now the top priority. This is why most of the research section will be about techniques used for capturing the desktop activities.

5.1.1 Capturing user desktop activities
There are many ways to capture user desktop activities. Perhaps the most simple and obvious way is to take screenshots at regular intervals. While this approach is simple to implement it has several disadvantages. It does not capture the mouse-cursor and it is bandwidth-ineffective. A much better way to capture desktop activities would be to take an initial screenshot and then only save the changes made to it. To get the position of the mouse cursor the coordinates of it could be captured and added separately.

There are several applications that take this approach to showing the desktop. For example: Windows Remote Desktop and Virtual Network Connection (VNC). Both of these applications are really used for controlling a computer remotely, but it is their desktop viewing feature that is interesting in this case.

The idea to use VNC-like software to capture desktop activities had been brought forth at an early stage by Dennis Andersson at FOI. Therefore VNC and Remote Desktop were examined to find the more suitable of the two.

Windows Remote Desktop is developed by Microsoft and uses the Remote Desktop Protocol (RDP). RDP gives excellent performance with almost no perceivable delay. The downside is that it is very poorly documented. The most promising source of information found was an open source software that uses RDP (rdesktop: A Remote Desktop Protocol client, 2005).

VNC was originally developed by Olivetti Research Laboratory (Richardson et al., 1998). Later on it was released as open source under the conditions of the GNU General Public License (GNU General Public License – GNU Project – Free Software Foundation (FSF), 2005). VNC uses the Remote Frame Buffer Protocol (RFB). RFB seems to perform slightly worse than RDP but is instead very well documented (Richardson, 2005).
RFB seemed like the better choice because of the documentation available. Therefore the RFB protocol was chosen as a basis to start from in order to create a desktop capture program. One problem with the RFB protocol is that it is made for viewing and controlling a desktop in real-time. Not for saving the sessions to disk and viewing them at a later time. However there are other programs that have made modifications to the RFB protocol to allow this.

After an extensive search several programs were found that had made modifications to the RFB protocol to allow saving and viewing it from disk. The most promising of these were TeleTeachingTool (TTT).

The TeleTeachingTool is a piece of software to record, transmit and replay of [sic!] multimedia-based lectures, speeches and documentations.

Features:

- free software
- platform independent (Java)
- recording and internet-transmission of...
  - ... computer-desktop with any application (screen recording based on VNC)
  - ... video and audio from the speaker
- multicast- and unicast-transmission

The TeleTeachingTool was/is developed at the University of Trier (chair for programming languages and compilers) within the scope of the ULI-Projekt by Dipl. Inf. Peter Ziewer. (TeleTeaching @ University of Trier – TeleTeachingTool, 2005)

TTT has many of the features required. It can capture desktop activities in an RFB-based format, it can capture audio and it can send the captured data to multiple receivers via multicast.

A lot of time was spent testing TTT to see if it could be modified to fit the requirements of this thesis. Initially TTT seemed very promising but after some disappointing performance tests this changed. TTT was found to have serious performance issues. Often when a capture was started TTT would use up all the CPU-resources of the test-computer. Restarting the program did not help. Sometimes rebooting the computer would solve the problem, sometimes not. Different versions of the Java runtime and Java Media Framework were tried but nothing seemed to help. This was a serious problem. Modifying the program to fit the thesis requirements seemed possible, but there were no guarantees that the performance issues could be solved. Therefore the decision was made to develop new software using some of the ideas from TTT.

Capturing the user desktop activities would be done by modifying a VNC-client and creating a new receiver application. The VNC-client would then connect to a VNC-server running on the same computer. The VNC-server would capture the desktop activities and send the data to the VNC-client. The VNC-client would add timestamps to the data, save it locally for backup and then send it to the receiver application. Upon receiving the data, the receiver application would save it to disk as well. This idea is inspired by the way TTT captures the user desktop which is very similar (Ziewer & Seidl, 2002).
5.1.2 Speech codecs

There are a number of audio compression algorithms specifically designed for voice recordings (GSM, WM9 Voice). Initially the thought was to use some sort of compression on the audio data to lower the network-bandwidth usage. Since MIND currently does not support compressed audio, audio compression got a lower priority and in the end there was no time to implement it.

5.2 Implementation

To start off a simple program to capture audio from the microphone was developed. This was based on the Capturesound example found in the Microsoft DirectX SDK (Microsoft, 2005b). The first feature to be added was the ability to capture the output audio as well as the input audio. Capturing the output audio was not at first expected to be a large problem. It was not until the output audio capture feature was about to be implemented that the problem became apparent. Up until then, most of the audio research had been focused on compression and down-sampling.

A lot of time was spent searching forums and mailing lists to find a solution to capturing the output audio. A solution to only capture output audio was found, but no input audio could be captured at the same time using this solution. Some research had already been done in this area by Dennis Andersson at FOI, and it was known to be a problem. To solve it an external application named Virtual Audio Cable (VAC) had been purchased (2.2 - Virtual Audio Cable). Since no other solution was found VAC was used to solve the problem.

The work was continued by extending the application to allow sending of the captured audio over the network. Once this feature had been added, the first rough version of the server application was complete.

In order to be able to receive the audio, a client application was developed. The client was developed entirely from scratch. At first the client was very simple; it could only receive audio from one server at a time and had no control over the capture process. The starting and stopping of the audio capture was controlled at the server end. To move this control to the client, changes had to be made to both the client and server application. The audio format negotiation was added at both ends and the data being sent was divided into the data protocol and the audio protocol. At the same time exception handling was added to both the server and the client, a feature that was long overdue. A screenshot of this early version of the client can be seen in figure 20.

![Figure 20. An early version of MCaptureClient.](image)
The client and server were now working quite well together, but there was still work to be done. One problem was that the server continuously transmitted audio once connected. To solve this problem the server was modified to allow audio detection (See 4.2.2.1 – Which buffer parts contain sound for a more thorough explanation). Modifications were also made to the client to allow changes to the clipping level and extra buffer.

Another problem was that the client only could handle one connected server at a time. To allow several server-connections at once, threading was used. Each server-connection was started as a separate thread where the connection and audio file writing was handled. While this was working very well it quickly became apparent that a suitable interface was needed to manage all the connections. The connected servers were placed in a list to make it easy to see which servers the client was connected to. From this list one or several servers could be selected and disconnected from by a press of the disconnect button.

To allow for an easier way to find available servers the Automatic Server Discovery System was added to the client. This required quite a few additions to the server as well. Once it was done the interface of the client had to be changed. At first another list was added. This list displayed the available servers. One or several servers could be chosen from the list and connected to. When the servers were connected they were moved to the list of connected servers. Once they were disconnected from they were moved back to the list of available servers again. This was a working solution for up to ten or maybe twenty servers, but it was not the easy-to-use interface requested in the task description. Figure 21 shows a screenshot of the client at this stage.

![Figure 21. MCaptureClient before the tree-view was implemented.](image)

The tree-view interface was developed to replace the dual list interface. Developing it required more time and effort than expected, but it resulted in an interface that makes it a lot easier to manage large numbers of servers.
6 The test

To get a rough estimate of the amount of network bandwidth and CPU-time used by the client and server application a test was conducted.

6.1 Test setup

In total nine computers connected to the same local network were used. The server application was installed on eight computers. Two computers were using Windows 2000 as an operating system; the other six were using Windows XP. The ninth computer was used to run the client application and was using Windows XP as an operating system.

Each of the eight server computers was controlled by an operator with a microphone. The operators had received written instructions that described their tasks (see appendix 1). When the client had connected to the servers the operators were to start playback of a prerecorded audio file and also start reading a text into their microphone. When done, they were to continue by taking a screenshot of the networking tab in Windows task manager. In Windows 2000 there is no networking tab in the task manager. On the two computers running Windows 2000 screenshots were taken of the performance tab instead. On the computer running the client application two screenshots were taken. One of the performance tab and one of the networking tab in Windows task manager.

All audio was recorded at 44100 Hz, 16-bit mono and sampled down to 11025 Hz, 8-bit mono. A clipping level of 18.3% and an extra buffer of 0.5 seconds were used. The VAC version used was 3.08.

6.2 Audio results

During the test two operators had microphone problems that resulted in some audio not being recorded. These problems were most likely caused by the audio input level not exceeding the clipping level. Another operator had a microphone problem which resulted in no input audio being recorded at all from it. All other audio sources were working as expected.

6.3 Performance results

Figure 22 shows that the network bandwidth usage of the server application on the QVINTUS computer never exceeded 0.2% of a 100Mbit connection. Figure 23 shows that the maximum CPU-time used by the server application during the test was measured to 20% on a 1.7 GHz Pentium 4 processor running Windows 2000 with 512MB of RAM. The results were very similar on the other server computers in the test.
Figure 22. Bandwidth usage graph from server.  

Figure 23. CPU-performance graph from a server.  

Figure 24 and 25 show pictures of the networking and performance tab on the computer that was running the client application. Figure 24 shows a peak network bandwidth usage of 1% of a 100Mbit connection. In figure 25 the red square shows the part of the graph which represents the time of the test. The maximum CPU-time used was approximately 35% on a 2.8 GHz Pentium 4 processor running Windows XP with 512MB of RAM. It’s worth noting that the CPU-performance graph stays below 10% except for two relatively short spikes. These spikes were most likely caused by the user-interactions with the GUI.
6.4 Conclusion of the test

The test shows that the client application works under heavy load from eight simultaneously connected servers. The server used a maximum of 20% of the available CPU-resources on a 1.7 GHz Pentium 4 with 512 MB of RAM. This leaves at least 80% of the CPU-resources left for other applications. The client CPU-performance graph is a bit harder to read. Except for two short spikes it stayed below 10% on a 2.8 GHz Pentium 4 with 512 MB of RAM. Since the CPU-load is very low it will be possible to have quite a few more servers connected, unless there are other limiting factors.

Figure 22 shows a server bandwidth usage slightly below 0.2 Mbit/s when both input and output audio was transmitted. This translates into that the data sent was less than 0.1 Mbit/s per channel. This is to be compared to the audio data, which at the output format used in the test will use 0.0882 Mbit/s per channel (11025 samples per second multiplied by 8 bits per sample). This leaves about 0.01 Mbit/s for protocol overhead which seems reasonable.

Figure 24 shows a peak bandwidth usage of 1Mbit/s for the client. This may seem strange since eight servers were supposed to be connected and sending simultaneously. Since each server generated about 0.2 Mbit/s the eight servers together should generate about 1.6 Mbit/s, not 1 Mbit/s. The main reason for this is most likely that some of the operators were done sending before others had begun. The microphone problems described in 6.2 – Audio results are also at least partly responsible.

When calculating the bandwidth usage of a given number of connected servers there are some things to bear in mind. The maximum bandwidth one server will need is 0.2 Mbit/s. This requires that both input and output audio is being transmitted at the same time. In real world situations this scenario is not very likely. An operator will seldom be speaking if being spoken to (at least not for greater lengths of time). It is also unlikely that all operators will be speaking or listening at the same time. In a real world scenario it is therefore more likely that a server will need a maximum bandwidth of 0.1 Mbit/s.
7 Conclusion
This chapter will present the results of the thesis and it also gives some suggestions to possible areas of future work.

7.1 Results
The result of this thesis is a working prototype software suite consisting of a server and a client. The server captures the input and output audio of a computer, saves it to disk and sends it over the network. The client receives the audio data sent by multiple servers in a network and saves it to disk. The software also features a server discovery system designed to make it easy to organize a large number of servers into manageable groups.

The software has been successfully tested with eight servers simultaneously connected to one client. The test shows that all the parts of the main task of the thesis have been completed. This thesis also presents a working solution showing how to capture user desktop activities. Unfortunately there was no time to implement this solution or any of the other secondary tasks. However, this was to be expected when new software had to be developed from scratch. If TTT had been more stable it would have been an excellent starting point for this thesis. Then it probably would have been possible to implement most of the secondary tasks as well.

As previously stated the software has been successfully tested with eight servers connected to one client. The “real” test however, will be when the application-suite is used to capture the audio of a large military exercise in Enköping scheduled for this autumn.

7.2 Future work
This section covers some possible areas of future work.

7.2.1 Real-time playback of received audio
The client application could be extended to allow the user to listen to the audio from one or several chosen servers in real-time. The playback in itself would probably not be hard to implement. The real problem lies in choosing the audio-streams from their respective threads.

7.2.2 Capturing user desktop activities
This topic has already been covered to some extent in the research chapter. A suitable solution to how to capture the desktop has been found, but implementing it will require a lot of additions to both the server and client which will likely take some time. Once the capturing has been implemented, real-time viewing of a server desktop could also be implemented.
7.2.3 Audio compression
Adding some sort of compression to the audio would lower the network bandwidth usage. The downside is that it would increase the CPU-load of the server. Another downside is that MIND currently does not support compressed audio; however, this might change in the future.

7.2.4 Sending data via multicast
Sending the audio data via multicast would allow several recipients to receive the data in a bandwidth effective way. This would be especially interesting if playback of the data was also possible.

7.2.5 Configuration of MCapture and VAC
There are a lot of configuration options that could be added, for example the option to choose the path to where the audio files should be saved. Another welcome feature would be the possibility to have VAC and MCaptureServer start automatically every time Windows starts. This is, however, mostly a VAC related problem.
References


Appendix 1: Instruktioner inför test av MCaptureServer och MCaptureClient

1. Avsluta alla program.

2. Packa upp och installera MCaptureServer.

3. Packa upp och installera Virtual Audio Cable, välj **inte** ”I am an advanced user”.

4. Starta Sound and Audio Devices via startmenyn (Settings\Control Panel\Sounds and Audio Devices).

5. Välj Virtual Cable 1 Out som default device för sound playback och voice playback. Välj ditt ljudkort som Default device för sound recording och voice recording. Välj därefter Apply och sedan OK.
6. Starta VAC Control Panel via start-menyn (Programs\Virtual Audio Cable\Control panel) och ställ in mode: Sync, tryck sedan på set-knappen.

7. Starta Audio Repeater via startmenyn (Programs\Virtual Audio Cable\Audio Repeater).

8. Välj Virtual Cable 1 In som Wave in och ditt ljudkort som wave out. Ställ in följande ljudformat: Sample rate: 44100 Hz, Bits per sample: 16, Channels: 1, Total buffer (ms): 40, Buffers: 4. Välj därefter Start.

9. Starta Sound Recorder via startmenyn (Programs\Accessories\Entertainment) och testa att spela upp den bifogade ljudfilen. Obs. använd inte Windows Media Player, den spelar nämligen inte upp ljud via Virtual audio cable (trots att man väljer Virtual audio cable som default device).

11. Starta MCaptureServer via startmenyn (Programs\MCaptureServer\MCaptureServer).

12. Välj ljudkortet som första capture-enhet och Virtual Cable 1 In (emulated) som andra capture enhet. Kryssa i rutan ”Remember my choice” om du vill slippa göra om valet av capture-enheter nästa gång programmet startas.

13. Kontrollera att audio input stämmer överens med det bestämda ljudformatet.

14. När servern har blivit kontaktad av klienten, starta uppspelningen av Testljud.wav och läs samtidigt upp exempeltexten som du finner i slutet av detta dokumentet i mikrofonen. Detta bör ta ca 15 sekunder.
15. När du har läst upp texten och ljudfilen har spelats upp tar du en skärmdump på nätverksfliken genom att trycka på printscreen på tangentbordet medan nätverksfliken i Task manager är synlig på skärmen.

16. Starta Paint via startmenyn (Programs\Accessories\) och välj i menyraden Edit\Paste.

![Paint](image)

17. Spara bilden i PNG format och maila den till dancla@foi.se


Exempeltext:


Tack så mycket för din medverkan i testet!