LIMBURGS UNIVERSITAIR CENTRUM

IN HET CENTRUM VAN DE KENNIS

# *Voice over IP in networked virtual environments*

**Jori Liesenborgs**

promotor : Prof. dr. W. Lamotte

afstudeercoördinator : Prof. dr. E. Flerackers

**Academiejaar 1999–2000**

**Universiteit Maastricht**

Thesis voorgedragen tot het behalen van de graad
van doctorandus in de Kennistechnologie,
afstudeervariant Informatica–Multimedia

# Preface

When I first selected 'Voice over IP in networked virtual environments' as the subject for my thesis, I did not know exactly what to expect from it. I mainly chose the subject because it combined several things that seemed very interesting. First of all, there is the networking aspect. I already had some experience with network programming, but I did not know much about the internals of networking protocols. To me, this seemed an ideal opportunity to learn more about these matters.

Second, the 'chat' aspect also sounded appealing: instead of communicating over a computer network using textual messages, you would be able to simply talk to each other. This was relatively new to me, and I really wanted to try to make such applications myself. Creating applications to exchange textual messages was something which I had already done; the VoIP approach presented a new challenge.

Finally, I also found the use in virtual environments very interesting. Reading the description of this thesis subject made me wonder how one could add 3D effects to a sound to make it seem localised. Naturally, I wanted to learn more about it and try it myself.

I learned a lot about these items while working on this thesis. I also found out that VoIP in networked virtual environments is more complex than I first suspected. Furthermore, by trying to develop VoIP applications myself I noticed that you have to do quite some effort if you want to allow good quality conversations. This is especially so since systems often do not act the way you expect them to. Finally, I believe that working on these programs has helped to improve my programming style.

To conclude this section, I would like to say that besides having learned a lot, making this thesis and these VoIP applications also have been a lot of fun. So having come at the end of it, I can say that I am glad I chose this subject.

Jori Liesenborgs
24th May 2000

# Acknowledgements

This thesis would not have become what it is without the help of several people. First of all, I would like to thank my promotor prof. dr. Wim Lamotte for providing me with useful suggestions about the structure and contents of my thesis and for helping me test several VoIP programs.

Special thanks also go to Igor Kalders. Without his help I would not have been able to test my VoIP applications in a situation where there are at least several routers between the communicating parties and where there is less bandwidth available than on a LAN. Although we did not always find a good subject for a conversation, the tests were certainly useful.

Many other students also helped me test these VoIP applications. These are[1] (in no particular order): Koen Beets, Kris Luyten, Tom Van Laerhoven, Peter Quax, Daniël Teunkens, Stef Cant, Panagiotis Issaris and Silvio Sawicki.

I would also like to express my gratitude to the whole class for keeping a good atmosphere in our computer room. Without it, this year would not have been this much fun.

1 My apologies if I forgot somebody.

# Table of contents

# Abstract

Voice over IP is about transmitting voice information across an IP network, for example the Internet. The classical application of VoIP is as a telephone alternative. However, this thesis is about using VoIP in networked virtual environments.

The Internet Protocol (IP) is a part of the TCP/IP architecture. The protocol itself offers only a best–effort service: packets can be delivered out of order, corrupted, duplicated or not at all. Also, each packet takes a different amount of time to reach its destination. Applications normally do not use IP itself, but the higher level protocols: TCP, which offers a reliable byte stream service, and UDP, which offers a similar service as IP.

The speech signal is transmitted by digitising tiny pieces of it at regular intervals and sending these to the destination where an analogue signal is reconstructed. For good quality communication, the overall delay should be below 200 ms. Delay variance or jitter should be eliminated through buffering. Speech communication is fairly tolerant to lost or corrupted packets.

When the digitised speech signal is left uncompressed, a bandwidth of 64 kbps is needed for telephone quality communication. Various compression techniques can reduce this amount. The most successful among them model how the speech was produced rather than the signal itself. Various compression standards allow interoperability between applications.

To transmit the speech data, TCP is not a good choice: it has a lot of features which are unnecessary for VoIP, but which increase the overall delay. UDP itself is too simple, but we can extend it: this is the way RTP is used in the TCP/IP architecture. The Real–time Transport Protocol (RTP) provides information for synchronisation, flow and congestion control and identification. To provide some quality of service (QoS) guarantees, resources can be reserved by using RSVP, the Resource Reservation Protocol.

For VoIP in virtual environments, speech data will have to be sent to several destinations. This can be done in an efficient way by using multicasting. When a packet arrives at the receiver, the voice signal is extracted and a 3D effect is added to it, corresponding to the position of the sender. A sound appears to be localised because of interaural differences. These differences can be captured in Head–Related Transfer Functions (HRTFs) which can then be used to recreate localised sounds.

To be able to create VoIP applications myself, I first developed a RTP library which performs quite well. I also developed a VoIP framework in which different VoIP components can easily be tested. The applications I created with this framework include an Internet Telephony application and a 3D environment. Both allow good quality communication when sufficient bandwidth is available.

# Dutch summary

Hieronder volgt een Nederlandstalige samenvatting van deze thesis. De indeling die ik hierbij gevolgd heb, komt overeen met de verschillende hoofdstukken.

## S.1 Inleiding

'Voice over IP' (VoIP) gaat over het versturen van een stemsignaal via een IP netwerk (het Internet is hier een voorbeeld van). De context van dit signaal bepaalt de vereisten voor deze transmissie. Bijvoorbeeld, als het signaal deel uitmaakt van een conversatie tussen twee personen, moet men ervoor proberen te zorgen dat de real–time eigenschappen ervan behouden blijven: de vertraging tussen het spreken van één persoon en het horen wat er gezegd werd door de andere persoon moet zo klein mogelijk zijn om irritante stiltes in de communicatie te vermijden. Andere VoIP toepassingen – zoals een lezing via een netwerk – hebben deze vertragingsbeperking niet.

Deze thesis gaat over VoIP in genetwerkte virtuele omgevingen. Ze bevat zowel informatie over VoIP in het algemeen als over toepassing ervan in virtuele omgevingen. Verder geef ik ook een beschrijving van de applicaties die ik zelf ontwikkelde om verschillende aspecten van VoIP in virtuele omgevingen te kunnen testen.

De klassieke toepassing van VoIP is als vervangingsmiddel voor een gewoon telefoongesprek. Wanneer VoIP op deze manier gebruikt wordt, kunnen de kosten van een gesprek verminderd worden, maar de kwaliteit ervan is meestal wel lager dan bij een normaal telefoongesprek. Het gebruik van VoIP in genetwerkte virtuele omgevingen is relatief nieuw. Zulke toepassingen laten gebruikers toe met elkaar te 'chatten' zoals bij IRC, maar in plaats van tekstuele boodschappen naar elkaar te sturen kunnen ze gewoon met elkaar praten. Het toevoegen van een 3D effect aan het stemsignaal van een gebruiker zorgt voor een meer natuurlijke omgeving. Er zijn nog vele andere toepassingen die soortgelijke technieken gebruiken als VoIP, bijvoorbeeld het versturen van een videosignaal.

Er zijn heel wat componenten nodig om VoIP in virtuele omgevingen mogelijk te maken. Een stemsignaal wordt opgesplitst in kleine stukjes die afzonderlijk verzonden worden. Om zo een stukje van een signaal te kunnen versturen, moet het eerst gedigitaliseerd worden. Bij de ontvanger moet deze informatie weer gereconstrueerd worden tot een continu geluidssignaal dat naar de luidsprekers gestuurd kan worden. Merk op dat wanneer verschillende personen tegelijk spreken, hun stemsignalen gecombineerd moeten worden. Verder moeten er ofwel bij de zender, ofwel bij de ontvanger, 3D effecten toegevoegd worden aan het signaal. Om de nodige bandbreedte voor het versturen van de gegevens te verminderen, kan het gedigitaliseerde signaal best gecomprimeerd worden. Natuurlijk moet dan aan de andere zijde deze informatie weer gedecomprimeerd worden vooraleer het verder verwerkt kan worden. Tenslotte moet er ook een component zijn die het versturen en ontvangen van pakketten met steminformatie mogelijk maakt.

## S.2 Het Internet Protocol (IP)

Het Internet Protocol of IP is een deel van een gelaagde architectuur die het TCP/IP referentiemodel heet. Dit model bestaat uit vier lagen waarvan elke laag een aantal functies voorziet aan de laag erboven. De internet laag is de laag waarin IP gedefinieerd wordt. Deze maakt het mogelijk om blokken met gegevens, datagrams genaamd, te versturen van bron naar bestemming. Dit wordt gedaan door actief elk datagram over elk tussenliggend netwerk te versturen. Aangrenzende netwerken worden met elkaar verbonden door machines die ‘routers’ genoemd worden. Deze routers analyseren binnenkomende IP datagrams en sturen ze verder via het juiste netwerk. IP ondersteunt ‘multicasting’, een techniek waarmee een datagram efficient naar verscheidene bestemmingen gestuurd kan worden. Het sturen van een datagram naar slechts één bestemming wordt ‘unicasting’ genoemd.

Het Internet Protocol is een connectieloos pakketgebaseerd protocol dat geen garanties biedt over de aankomst van een datagram. Datagrams kunnen zelfs gedupliceerd of in de verkeerde volgorde afgeleverd worden. Andere karakteristieken van IP netwerken zijn de vertraging door wachttijden in routers en de verschillen in vertraging tussen opeenvolgende pakketten. Dit laatste wordt ook wel ‘jitter’ genoemd.

De transport laag – die boven de internet laag ligt – bevat twee veel gebruikte protocols. Het ‘Transmission Control Protocol’ of TCP, biedt een connectiegeörienteerde dienst aan waarvan de connectie kan beschouwd worden als een betrouwbare stroom van bytes. Het ‘User Datagram Protocol’ of UDP, is enkel een extensie van IP op het niveau van de transport laag en heeft dezelfde eigenschappen als IP.

De belangrijkste redenen voor pakketgebaseerde communicatie zijn de mogelijkheden om de informatie te comprimeren en stiltes uit te buiten. De alomtegenwoordigheid van IP is de voornaamste reden waarom dit protocol een goede kandidaat is voor het transporteren van steminformatie.

Omwille van een aantal redenen werd er een nieuwe versie van het Internet Protocol ontwikkeld. De belangrijkste reden was het feit dat er al gauw geen IP adressen meer beschikbaar zouden zijn op het Internet. De nieuwe versie van het protocol heet IP versie zes, of kortweg IPv6.

## S.3 Communicatie via de stem

Een aantal aspecten van het communiceren via de stem hebben een belangrijke betekenis voor ‘Voice over IP’. Om VoIP mogelijk te maken, moeten we een stemsignaal kunnen digitaliseren en reconstrueren. Een meetsnelheid van 8000 Hz waarbij men acht bit samples gebruikt, is voldoende om communicate met dezelde kwaliteit als een telefoongesprek mogelijk te maken. Hiervoor is dan een bandbreedte van 64 kbps nodig.

Mensen zijn meestal erg verdraagzaam tegenover incorrecte of verloren pakketten met spraakinformatie. Ze zijn echter veel minder tolerant tegenover vertraging en jitter. Jitter moet men trachten te vermijden door pakketten even in een buffer te plaatsen, aangezien het de kwaliteit van de communicatie sterk verlaagt. De vertraging van een pakket zou onder 200 ms moeten blijven om een conversatie met telefoonkwaliteit te kunnen houden.

Vanuit het standpunt van de communicatie zijn erg kleine pakketten wenselijk aangezien een incorrect of verloren pakket dan een kleinere onderbreking in de communicatie veroorzaakt. Vanuit het standpunt van de vertraging is een klein meetinterval belangrijk aangezien dit interval direct bijdraagt tot de algemene vertraging in de communicatie.

Tenslotte is er in de praktijk meestal slechts één persoon tegelijk aan het praten in een discussie. Dit betekent dat heel wat bandbreedte beter benut kan worden door geen pakketten te versturen die enkel stilte bevatten.

## S.4 Compressiemethodes

Voor communicatie met telefoonkwaliteit waarbij gedigitaliseerde spraakinformatie verzonden wordt, is een bandbreedte van 64 kbps nodig indien de gegevens niet gecomprimeerd worden. Zulke spraakgegevens kunnen vaak echter sterk gecomprimeerd worden waardoor de nodige bandbreedte drastisch vermindert.

Sommige compressiemethodes houden geen rekening met de aard van de gegevens. Zulke technieken zorgen wel voor wat compressie, maar resulteren meestal niet in hoge compressieratio's. Ze kunnen echter wel gebruikt worden om de vereiste opslagruimte nog verder te verminderen wanneer een andere methode de spraakinformatie al gecomprimeerd had.

'Waveform coding' technieken veronderstellen dat de gegevens een audio signaal representeren maar ze maken geen gebruik van de wetenschap dat het signaal enkel steminformatie bevat. Ze proberen gewoon het eigenlijke signaal zo goed mogelijk te modelleren. Deze manier van werken heeft meestal goede spraakkwaliteit tot gevolg waarbij een relatief hoge bandbreedte vereist is (16 kbps of meer).

'Vocoders' maken wel gebruik van het feit dat de gegevens feitelijk een gedigitaliseerd stemsignaal zijn. Ze coderen niet het signaal zelf, maar wel een benadering van hoe het signaal gevormd werd door het menselijke spraakmechanisme. Zulke methodes laten zeer hoge compressieratio's toe terwijl de verstaanbaarheid van de communicatie zeer goed blijft (aan 4.8 kbps of minder). Het gereconstrueerde signaal klinkt vaak wel wat synthetisch.

Een combinatie van technieken die waveform coders en vocoders gebruiken, wordt toegepast in hybride technieken. Ze maken nog steeds gebruik van een spraakproductie−model, maar ze kunnen het oorspronkelijke signaal veel beter reconstrueren door het gebruik van technieken die waveform coders toepassen. Hybride methodes kunnen goede spraakkwaliteit genereren en hebben meestal een bandbreedte van 4.8 tot 16 kbps nodig.

Het comprimeren en decomprimeren van spraakgegevens zorgt voor een zekere hoeveelheid vertraging in de communicatie. Omdat computers steeds sneller worden en omdat gespecialiseerde hardware beschikbaar wordt, is waarschijnlijk de hoeveelheid 'lookahead' die een compressiemethode nodig heeft de belangrijkste component van deze vertraging. De 'lookahead' beschrijft hoe ver de methode vooruit moet kijken om een deel van het signaal te kunnen comprimeren.

Om verschillende applicaties met elkaar te kunnen laten samenwerken, is het belangrijk dat standaarden vastgelegd worden. Bekende compressie standaarden in de VoIP wereld zijn onder andere de G. standaarden van ITU−T en de GSM standaarden van ETSI.

## S.5 Verzenden van stemsignalen

Wanneer we spraakgegevens willen versturen, zijn er een aantal dingen waar we aan moeten denken. Zo is er een of ander mechanisme nodig om intramedia synchronisatie te bewaren. Ook moeten er maatregelen genomen worden om een kleine vertraging te garanderen en moeten 'flow control' en 'congestion control' mogelijk zijn.

In de TCP/IP architectuur kan een applicatie TCP of UDP gebruiken om gegevens te versturen. Om steminformatie te verzenden zou TCP misschien een goede keuze lijken aangezien TCP een stroom van bytes betrouwbaar kan versturen waarbij flow en congestion control intern geregeld worden. Deze betrouwbaarheid wordt echter gegarandeerd door het opnieuw versturen van incorrecte of verloren pakketten en dit verhoogt dan weer de algemene vertraging in de communicatie. Extra vertraging kan ook veroorzaakt worden door de flow en congestion control mechanismen, waarover de gebruiker maar weinig controle heeft.

Het andere protocol, UDP, is niet voldoende voor real–time gegevens aangezien het op geen enkele manier intramedia synchronisatie of flow of congestion control mogelijk maakt. Een oplossing voor dit probleem is UDP een beetje uitbreiden. Dit is de manier waarop RTP in de TCP/IP architectuur gebruikt wordt.

Het 'Real–time Transport Protocol' (RTP) voorziet een pakket van extra informatie die gebruikt kan worden voor synchronisatie binnen een stroom van gegevens. Het 'RTP Control Protocol' (RTCP) geeft bijkomende informatie die gebruikt kan worden voor synchronisatie tussen verschillende media, voor flow en congestion control en voor identificatie.

Elk stukje gedigitaliseerde spraakgegevens wordt voorafgegaan door een aantal zogenaamde 'headers'. Deze headers nemen ook een deel van de bandbreedte in, dus om deze hoeveelheid relatief laag te houden, mag een pakket niet te klein zijn. Voor dial–up verbindingen kan extra bandbreedte bespaard worden door header compressie technieken toe te passen.

Om 'quality of service' (QoS) mogelijk te maken, zou men de prioriteitsinformatie in de IP header kunnen gebruiken. Deze methode kan de QoS vebeteren maar biedt geen garanties. Wanneer garanties noodzakelijk zijn, kunnen andere protocols zoals ST2 en RSVP gebruikt worden. Beide geven garanties door het maken van reservaties, bijvoorbeeld van bandbreedte.

Versie twee van het 'Stream Protocol' (ST2) gebruikt een zender–geïnitieerd reservatiemodel: de zender stuurt reservatie–aanvragen over het pad naar de ontvangers. Het is een connectiegeöriënteerd protocol dat als aanvulling bij IP dient.

Het 'Resource Reservation Protocol' (RSVP) daarentegen gebruikt een ontvanger–geïnitieerde aanpak. De zender verspreidt een beschrijving van de gegevens die hij zal zenden en de ontvangers kunnen dan reservaties maken voor deze gegevens. Het is een 'soft–state' mechanisme: zenders en ontvangers moeten deze informatie geregeld verspreiden omdat anders de gemaakte reservaties vrijgegeven worden. RSVP voegt ook reservaties samen wanneer mogelijk. Dit protocol lijkt efficienter te zijn dan ST2.

Het verzenden van pakketten zorgt voor een bepaalde vertraging in de communicatie. Deze vertraging is sterk variabel door de wachttijden in routers.

## S.6 VoIP in virtuele omgevingen

Om te doen lijken alsof een geluid vanaf een zekere positie komt, is het nodig om een stereo signaal te genereren. Hierdoor is het efficienter om 3D effecten toe te voegen aan de kant van de ontvanger, aangezien we dan enkel een mono signaal moeten verzenden. Op deze manier kunnen we ook IP multicasting gebruiken omdat exact dezelfde gegevens naar alle ontvangers gestuurd moeten worden.

Eén manier om de spraakgegevens te verspreiden is door unicasting te gebruiken. Dit laat toe dat de zender bepaalt wie deze gegevens ontvangt, maar verspilt bandbreedte. Efficientere distributie kan door het gebruik van multicasting. Het is dan aan de ontvangers om te bepalen wiens spraakgegevens ze moeten verwerken.

Geluiden lijken vanaf een bepaalde positie te komen doordat elk trommelvlies een licht verschillend signaal ontvangt. Uit deze verschillen bepalen de hersenen de positie van de bron van het geluid. Twee belangrijke aanwijzingen voor lokalisatie zijn 'Interaural Time Difference' (ITD) en 'Interaural Intensity Difference' (IID). Deze drukken respectievelijk het tijdsverschil en intensiteitsverschil uit van het waargenomen signaal aan elk trommelvlies. Het buitenoor (de oorschelp) speelt ook een zeer belangrijke rol bij de localisatie van geluiden.

Door ITD en IID te gebruiken, is het mogelijk eenvoudige 3D effecten te genereren. Het is op deze manier echter niet mogelijk om een onderscheid te maken tussen 'voor' en 'achter' of tussen 'boven' en 'onder'. Betere resultaten kunnen behaald worden door de transformaties van een signaal vooraleer het de trommelvliezen bereikt, te simuleren. 'Head–Related Transfer Functions' (HRTFs) beschrijven deze transformaties.

Omdat er meerdere geluidsbronnen tegelijkertijd kunnen zijn, is het mogelijk dat de berekeningen om 3D geluiden te genereren te zwaar zijn. Het kan dan nodig zijn om de localisatie van geluiden te laten doen door hardware. Omwille van dezelfde reden kan het zijn dat de nodige bandbreedte niet beschikbaar is, bijvoorbeeld wanneer een dial–up verbinding gebruikt wordt. Een oplossing is dan een machine vóór de trage verbinding de 3D effecten te laten toevoegen en het gecombineerde signaal over de verbinding te sturen.

## S.7 Gerelateerde onderwerpen

Verscheidene protocols en standaarden zijn gerelateerd aan VoIP. Een eerste voorbeeld is H.323, een aanbeveling voor het houden van multimedia conferenties over pakketgebaseerde netwerken zonder QoS garanties. Het is een onderdeel van een reeks standaarden die dezelfde diensten beschrijven over verschillende soorten netwerken.

H.323 definieert hoe verschillende componenten samen moeten werken om bepaalde functies mogelijk te maken. Door middel van H.323 kan een verbinding gemaakt worden, kunnen de mogelijkheden van de deelnemers uitgewisseld worden en kunnen de deelnemers met elkaar communiceren. Media vertrekken en komen aan in 'terminals'. Een 'gateway' maakt communicatie over verschillende soorten netwerken mogelijk. Conferenties tussen meerdere gebruikers zijn mogelijk door het gebruik van een 'multipoint control unit' (MCU). Een 'gatekeeper' maakt gevorderde zaken als beheer van bandbreedte en registratie van gesprekken mogelijk.

De H.323 aanbeveling werd ontwikkeld door ITU–T. De IETF werkt ook aan een architectuur die soortgelijke diensten mogelijk maakt. Net zoals bij H.323, worden de media binnen deze architectuur verzonden via RTP, dus is het grootste verschil tussen beide aanpakken de manier waarop een verbinding gemaakt en beheerd wordt. In de IETF architectuur wordt het 'Session Initiation Protocol' (SIP) gebruikt om dit te doen. De functionaliteit die SIP aanbiedt, komt overeen met de functionaliteit die door componenten van H.323 worden aangeboden. Een belangrijk kenmerk van SIP is de mogelijkheid tot persoonlijke mobiliteit. Hiermee wordt bedoeld dat een gebruiker een oproep die aan hem gericht is eender waar kan beantwoorden.

Wanneer we SIP en H.323 vergelijken, lijkt SIP minder complex te zijn, beter uitbreidbaar en beter bruikbaar bij conferenties met toenemend aantal deelnemers. Hun functionaliteit is gelijkaardig, maar het uitwisselen van mogelijkheden tussen terminals in H.323, is meer geavanceerd dan de overeenkomstige functie in SIP. Daarentegen voorziet SIP betere persoonlijke mobiliteit.

Het 'Real–Time Streaming Protocol' (RTSP) is een ander VoIP gerelateerd protocol. Dit protocol dient als een soort afstandsbediening voor een media server. Zo kan RTSP bijvoorbeeld gebruikt worden om een presentatie of mediabestand op een server af te spelen. Het kan ook gebruikt worden om een presentatie die bezig is op te nemen.

## S.8 JRTPLIB

Om RTP gemakkelijk in verscheidene applicaties te kunnen gebruiken, heb ik een library geschreven die RTP functionaliteit voorziet. Deze library heet JRTPLIB, wat staat voor "Jori's RTP Library". Hij is geschreven in C++, gebruik makend van een object–georiënteerde aanpak.

De library maakt het zenden en ontvangen van RTP pakketten makkelijker. De gebruiker kan een willekeurig aantal bestemmingen opgeven. Multicasting kan gebruikt worden voor het efficiënt verspreiden van de gegevens. De RTCP functionaliteit wordt volledig intern behandeld.

De structuur van de library is erg modulair, wat de broncode gemakkelijk begrijpbaar en uitbreidbaar maakt. De 'Application Programming Interface' (API) is relatief gemakkelijk te gebruiken. Door het gebruik van standaard socket functies is de library beschikbaar op een groot aantal platformen. Verschillende technieken worden toegepast om de library zo snel mogelijk te maken, wat natuurlijk erg wenselijk is bij applicaties met real–time vereisten.

Na de library een tijdje te hebben gebruikt, besloot ik hem beschikbaar te maken op het World Wide Web. Dit heeft me geholpen om de ondersteuning voor verscheidene platformen te verbeteren.

## S.9 Een VoIP framework

Om gemakkelijk wat VoIP test applicaties te kunnen maken, heb ik eerst een object–geörienteerd VoIP framework gemaakt in C++. Dit framework was ook ontworpen om verschillende technieken om een VoIP component te realiseren uit te kunnen testen, bijvoorbeeld verschillende compressiemethodes. De structuur van het framework weerspiegelt de componenten die in deze thesis beschreven worden.

De kern van het framework bevat veel abstracte klassen die VoIP componenten voorstellen, zoals bijvoorbeeld een transmissie of compressie component. Door overerving te gebruiken kan dan een component werkelijk gerealiseerd worden. Het is dit principe dat toelaat verscheidene versies van een component gemakkelijk uit te proberen.

## S.10 VoIP test applicaties

Om het VoIP framework te testen heb ik een aantal programma's gemaakt, waaronder een Internettelefonie applicatie en een 3D omgeving. Het VoIP gedeelte van de applicaties wordt in een aparte thread opgestart. Hierin wordt voortdurend een functie uit het framework aangeroepen om VoIP mogelijk te maken. Er wordt nog wat extra werk gedaan om synchronisatie tussen de deelnemers te verzekeren.

De Internettelefonie applicatie is relatief eenvoudig en laat makkelijk communicatie met een goede kwaliteit toe wanneer voldoende bandbreedte beschikbaar is. De 3D omgeving laat toe dat verscheidene personen met elkaar communiceren. Daarbij worden eenvoudige localisatie–effecten toegevoegd aan hun stemsignalen. Zowel unicasting als multicasting kan geselecteerd worden om spraakgegevens te versturen. Ook deze applicatie laat communicatie met goede kwaliteit toe wanneer er voldoende bandbreedte aanwezig is. Bij beide applicaties hangt de hoeveelheid bandbreedte die nodig is af van de gebruikte compressiemethode.

## S.11 Conclusie

Momenteel is VoIP in het algemeen al mogelijk wanneer het netwerk genoeg capaciteit heeft. Omwille van een aantal redenen verwacht ik dat in de toekomst zulke applicaties meer gebruikt zullen worden. Zo zullen QoS ondersteunende protocols op grotere schaal gebruikt worden, wat betere gesprekskwaliteit garandeert. Ook zullen hoogstwaarschijnlijk de beschikbare bandbreedte toenemen en zullen compressiemethodes krachtiger worden, wat alweer betere kwaliteit van de communicatie tot gevolg heeft. Wanneer standaarden als H.323 en SIP meer in gebruik genomen worden, zullen applicaties van verschillende ontwerpers met elkaar kunnen interageren. Een gebruiker kan dan een programma naar keuze gebruiken om met iemand anders te communiceren. Ook het kostenbesparend karakter van VoIP zal een rol spelen bij de ingebruikname ervan.

Specifiek voor VoIP in genetwerkte virtuele omgevingen bestaan er nog niet veel applicaties. Zulke applicaties vormen echter een aantrekkelijk alternatief voor tekstuele ‘chat’ programma’s. Betere QoS zal ook dit soort applicaties waarschijnlijk meer populair maken. Toenemende kracht van processoren zal voor steeds betere 3D effecten zorgen, wat het realisme van zulke applicaties sterk zal verbeteren. Dit kan ook weer bijdragen tot de ingebruikname ervan.

Wat mijn eigen programma’s betreft kan ik alvast stellen dat de RTP library zeer handig in gebruik is. Dit wordt bevestigd door de vele positieve reacties. Ook het VoIP framework is handig te gebruiken en laat toe gemakkelijk verschillende VoIP componenten te testen. De applicaties die ik hiermee geimplementeerd heb blijken een goede gesprekskwaliteit toe te laten wanneer voldoende bandbreedte aanwezig is.

# Part I: Introduction

# Chapter 1: Introduction

This first chapter begins with a brief introduction to Voice over IP, followed by a description of this thesis. Next I will give an overview of the components of a Voice over IP system and at the end of the chapter I will describe the outline of the rest of this document.

## 1.1 What is Voice over IP (VoIP)?

Before we start to discuss Voice over IP (VoIP) related topics, it is probably best to give a brief explanation of what it is. This way, the essence of what is discussed here will be clear throughout the document and the details can be worked out at the appropriate time.

Voice over IP is an extensive subject, but at the core it comes down to trying to transport speech signals in an acceptable way from sender to destination over an IP network. An Internet Protocol (IP) network is a computer network which uses the IP protocol to transmit information. I will give a more detailed explanation of this protocol in the next chapter, but for now it might be helpful to know that this is the basic protocol used on the Internet.

The definition of 'acceptable' depends on the particular situation we are dealing with. If, for example, speech signals are being transported as part of a real–time communication between two persons, it will mean that the real–time aspects of this conversation must be respected: the overall delay between sending and receiving should be low to avoid irritably long gaps of silence. If, however, speech signals are being transmitted as part of a one–way process – e.g. an on–line radio show or a lecture – the delay constraints are less strict since the interactive aspect is no longer present.

## 1.2 Thesis subject

Here, I will give the exact formulation of my thesis subject. This way there will be some clarity about what you may or may not expect to find in this document. The subject I have chosen is this one:

*"A conventional way to communicate with each other using IP–networks, is through the use of textual chat facilities. The purpose of this thesis proposal is to take this one step further by using voice communication instead of these textual facilities. The goal of this proposal is to perform research and development in order to let persons which are in the same virtual environment talk to each other as they would do in reality. Their positions and orientations can be used to vary the intensity of the words: persons close to each other will hear each other clearly; persons which are moving away from each other will understand each other less and less as their distance increases. The proposal encloses technical components (like grabbing, compression, buffering, transmission, decompression and regeneration of the signal) and also a study of what is happening in the Voice over IP world today. Also, a number of experiments will have to be conducted to justify the chosen techniques."*

It should be clear from this description that the real–time aspects of VoIP will be very important. We are talking about a virtual environment in which persons can communicate with each other and so the overall delay between talking at one end and hearing what is said at the other end should be as small as possible. Because of this, I will pay less attention to those types of VoIP that do not have this constraint, but the same principles can be applied to them.

## 1.3 Uses of Voice over IP

Currently, when you look at what literature can found about VoIP, you will find that most of it is about VoIP as a telephone alternative. This type of use is described first in this section, followed by a discussion about using VoIP in virtual environments.

### 1.3.1 Telephone alternative

The first kind of use is the 'telephone alternative'. This means that you would use some kind of VoIP system to make a voice call to another person. This can be done in several ways.

First of all, if a PC that can be connected to some kind of network is available, it can be used to make a call to somebody else who is also connected to that network. This PC would then be equipped with speakers and a microphone and some VoIP application would be used to make the call. The PC could have a direct connection to a computer network, like in figure 1.1, but a connection through a dial–up link is also possible.

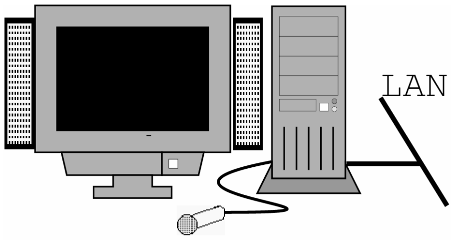


Figure 1.1 – PC to LAN configuration

The second case is a slight variation of the first one. In this case, a telephone is connected to the PC and used in a similar way as you would when making a normal call. The PC does all the necessary work to set up the call and to transmit the speech signals. This also means that the PC has to be switched on before the call can be made. This type of configuration might be easier to use for people who do not work with computers often. As with the previous case, the connection to the network can be either direct, like in figure 1.2, or through a dial–up link.

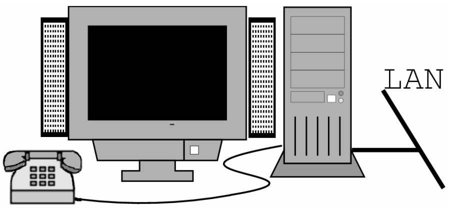


Figure 1.2 – Telephone to PC to LAN configuration

Finally, the use of a PC and the requirement of a network could be omitted by the use of a VoIP gateway. This is a special device that connects the public telephone network with a computer network and performs the necessary actions and conversations to make the call possible. To make a call to somebody, you would call the gateway and specify the destination for the call. The call will then be set up and if the other end is available, the conversation can start. This configuration would be best for persons who do not have a PC. It is probably also the easiest to use, since most people are familiar with using a telephone and there does not have to be a PC around. This configuration is illustrated in figure 1.3.

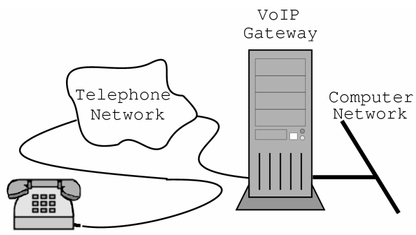


Figure 1.3 – Telephone to gateway configuration

There are probably a lot of variations to these configurations, but I believe that these three give a good idea of the possibilities. Combinations of these cases can also be worked out. A person could, for example, use his telephone to reach somebody through a VoIP gateway, while the latter uses a telephone to PC configuration.

Now, you may ask yourself: why use VoIP as a telephone alternative while the telephone itself is quite handy? Well, there are several arguments that can be made in favour of VoIP.

Suppose that somewhere – in a company or university for example – a computer network is needed. In that case, there are certain benefits by using Voice over IP instead of installing extra facilities to use telephones. The only requirement is that the IP protocol must be used, but nowadays this is almost always the case.

First of all, there is less cabling and equipment required. All the internal calls can be made using VoIP utilities. For outgoing and incoming calls, however, there still has to be some connection to the telephone network. This can be solved by installing a gateway that is connected to the computer network and the telephone network. This gateway will then perform the necessary signalling and conversations to make these calls possible.

Second, the capacity of the computer network will be better utilised. The available bandwidth of a network within an organisation is usually quite large and rarely fully used. By using VoIP, more of the network's capacity will be used.

At home, there is also an advantage in favour of VoIP. If Voice over IP could be used over a large distance, it would be much cheaper than making that same long distance call using the telephone network. For example, you could try to make the call by using the Internet.

With VoIP, not only the normal telephone features can be made possible, but also a wide range of new features could be created, especially when using VoIP on a PC. Whiteboarding could be used to make working together easier, a log book with information about incoming and outgoing calls could be kept, conversations could easily be recorded and security could be enhanced by using encryption algorithms.

When using VoIP over a Local Area Network (LAN), there is usually plenty of bandwidth available and the delay between sending and receiving is usually very low. Here, VoIP can often be used without problems. But when a Wide Area Network (WAN) is used – the Internet for example – problems can arise. One problem is the delay: while the delay on a LAN is usually very low, on a WAN this is not necessarily true. If the delay gets too large, the conversation will not be very pleasant. Another problem is the quality of the speech signals. When certain routes get too heavily loaded, packets on the WAN will be lost. These lost packets cause interruptions in the speech signal. In turn, these interruptions, when large enough, can also disturb the conversation. To alleviate the load, a lot of VoIP programs use compression techniques. However, compression often causes a certain degradation of the signal. This may or may not be disturbing to the listener, but with heavy compression, telephone quality will rarely be achieved.

### 1.3.2 In virtual environments

The use of VoIP for virtual environments can be seen as a replacement of the textual interface of chat facilities like Internet Relay Chat (IRC). The virtual environment can be made quite abstract by using the same kind of interface as IRC chat programs, but using voice input instead of text. There is, however, also the possibility of a three dimensional interface. This kind of application probably fits the term 'virtual environment' best. When you are using this kind of program, there will be some notion of a virtual world and the use of voice communication is very appropriate is this case. Note that now we are talking about facilities that do require a PC.

Because we are dealing with a virtual environment, several voice signals can be expected to go to several destinations, all at the same time. This means that considerable attention should be paid to limiting the required bandwidth. This is especially true when people can access the virtual environment through a dial–up link which has a very small capacity compared to a LAN for example.

Using VoIP this way is a rather new concept. This also means that currently, there is very little specific literature about it. However, it is obvious that a lot of the things that we have said in the previous section, also apply to VoIP in virtual environments.

### 1.3.3 Other

VoIP techniques can be used for a wide variety of other applications which require voice or sound in general to be transmitted over a computer network and where timing and synchronisation are important issues. The same techniques also work when it is not sound, but video information which has to be transmitted.

Several other applications can be thought of. One is the use of VoIP techniques to create an on–line radio station, or perhaps even an on–line jukebox, where you can select the song you want to hear, which is then played almost immediately. If enough bandwidth is available, it would even be possible to add video data to all this. This way, television broadcasts and video on demand over IP networks could be made possible. In a similar way, we could extend a VoIP telephone conversation with video information about the persons involved in the call, creating a videophone application.

Another kind of application would be fax over IP. This is a bit different since we are no longer transmitting speech data, but a digitised image. Like with VoIP, this service could be made possible by connecting a computer network to the telephone network using a gateway. For fax over IP, this gateway would perform similar functions as with voice over IP.

Note that the list of applications presented here is certainly not complete. A wide range of applications using VoIP related techniques are conceivable, but many of them will resemble the ones discussed above.

## 1.4 Components of a VoIP system

Here, the core components of a VoIP system for virtual environments will be illustrated. With ‘core components’ I mean the parts of the VoIP system that are at work during the conversations, so when the VoIP connection has already been established.

The entire process of the core VoIP system for virtual environments is depicted in figure 1.4. The arrows that point downward define the path which is followed when sending speech signals; the arrows that point upward define the processing sequence when speech signals are received. When the label of a box contains two items, the left one is about the sending of speech signals and the right one about the reception of such signals. They are grouped together because they operate at the same level: the right item does approximately the opposite of the left one.

This diagram can easily be adapted for VoIP applications which are not intended for virtual environments. The only thing that needs to be changed is the ‘3D effects’ step. In those applications the 3D effects are not needed, so the entire step can just be left out.

All these components will be described in more detail in the rest of this thesis, but below I will give a general description of each component of the system. This will create a general image of the workings of the VoIP system, which is useful to keep in mind when explaining each component in detail.

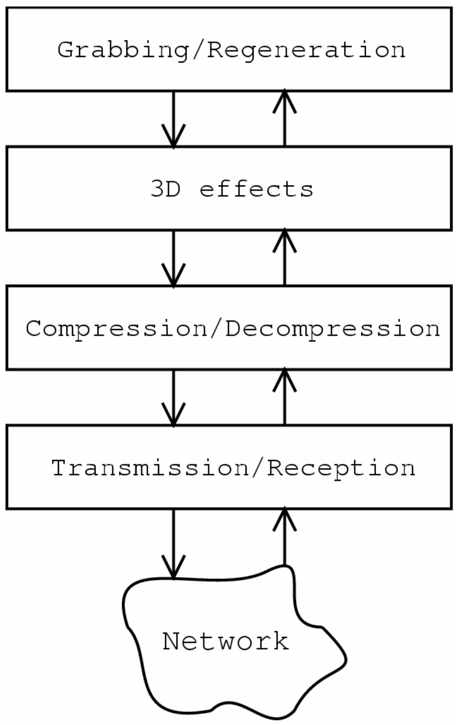


Figure 1.4 – VoIP process for virtual environments

### 1.4.1 Grabbing and regeneration

To be able to send speech information across a computer network, the speech signal has to be encoded into a digital representation. In general, the signal will be detected by a microphone and transformed into a digital one by a special device, a soundcard for example. This process is called 'grabbing' or digitisation and it is often also referred to as sampling[2].

To maintain the real–time aspects of the conversation, it is necessary for the receiver to start receiving the signal as soon as possible after the sender has started it. To accomplish this, at regular small intervals blocks of digitised speech information are sent across the network, where they can be processed by the receiver.

When a digitised block is received, it has to be transformed back into an audio signal. The output of the process will usually go to speakers, so that the receiver will be able to hear what the sender is saying. Like the digitisation step, this process is also done by a special device. In essence, regeneration is the reverse operation of grabbing.

Several things have to be considered before transforming the digitised signal. First of all, if multiple persons are allowed to talk at the same time, like in a virtual environment, the speech signals of those persons have to mixed together at the receiver.

Second, when sending blocks of data across a network, there will be tiny variations in the time it takes each block to get to the destination. If we are unlucky, these variations can even be rather large. The importance of these variations is this: suppose we start playing back the voice signal in a block as soon as we received it. Because of the jitter, it is possible that the next block has not yet arrived when the output of the first one is finished. To overcome this problem some buffering will have to be performed to make sure that when we are finished with one block, the next will be available. However, this buffering will introduce a certain amount of delay so care must be taken to avoid that the overall delay will be too large.

### 1.4.2 3D effects

To give the virtual environment a more realistic impression, it is important that some three dimensional (3D) effects are added to the voice signal. A participant should be able to determine roughly where the source of the voice signal is located.

Two general approaches can be thought of. Either the sender processes its own voice signal to appear as coming from a certain position, or the receiver adds the three dimensional effect to the sound. We will discuss later which one can best be used.

With the first approach the sender does the necessary transformations of the digitised signal. This signal can then be used by the receiver without any additional processing. The second approach requires that the receiver knows the position of the sender to modify the digitised signal accordingly. If necessary, this information can be added by the sender to the block containing the voice data.

### 1.4.3 Compression and decompression

The digitised information requires a certain amount of the available bandwidth of the connection. Very often compression schemes are used to reduce the required bandwidth for voice communication.

Several types of compression exist. Some of them use general compression techniques which are also used on other kinds of data; other types try to exploit the fact that we are dealing with voice information to achieve large compression ratios. Of course, combinations are also possible.

Once the compressed blocks with speech data reach the destination, they have to be decompressed. This means that given the compressed signal, the original digitised signal has to be reconstructed as good as possible. The decompression is very closely related to compression

2 This term is not entirely correct, since the entire process of digitisation consists of both sampling and quantising.

as it must be the inverse operation of the compression scheme that was used.

Compression is very important when the connection is slow, like with dial–up links for example. It is also an important issue when using VoIP in virtual environments, since the bandwidth requirements get larger as the number of senders increases.

### 1.4.4 Transmission and reception

Finally, the blocks have to be sent from source to destination, across the network. Some timing information should probably be added to the data, to make it possible for the receiver to reconstruct the exact order of the blocks. This is necessary because blocks may be lost, delayed or duplicated during the transfer. There are ways to assure a certain quality of the VoIP communication and to make the transfer more efficient when working with multiple destinations, but they will be explained later.

## 1.5 Outline of this document

This thesis is organised in four major parts. First, there is this introductory chapter. Next there are a number of chapters which can be categorised as research. Following these, there are some chapters in which I will discuss the development part of my thesis. Finally, the last chapter will contain an overall conclusion. Here is a short overview of the research and development parts.

### 1.5.1 Research

The next chapter is about IP networks. Since we are talking about Voice over IP, it is important to know some features of IP networks and the protocols used there. Therefore, that chapter will only discuss such items, without talking much about using IP for voice data.

In chapter three, we will talk about voice communication in general. Features which are important when using IP networks will be clarified here. The chapter will also include more information about grabbing and regeneration of voice signals.

Next, there is a chapter which contains a discussion about compression techniques. Several techniques will be explained and their use for VoIP will be clarified.

In chapter five the actual transmission of voice data is covered. Here, we will see a very useful protocol to transmit the data. Also, some techniques to provide quality of service (QoS) are discussed.

These four chapters were mostly about Voice over IP in general. Chapter six is about using VoIP in networked virtual environments. Here, techniques for the generation of localised sound will be detailed.

The last chapter of the research part is about subjects which do not belong to the core VoIP problem. However, to make sure that this thesis will produce a good image about what is going on in the VoIP world, some related topics will be discussed. These subjects include some related protocols and standards.

### 1.5.2 Development

The development part contains three chapters. The first one, chapter eight, contains information about the Real–time Transport Protocol (RTP) library that I wrote. Chapter nine describes the VoIP framework I created. Finally, chapter ten is about the VoIP test applications I developed. Since I have learned a lot while I was working on these programs, I will also discuss some of the design decisions I made and explain why certain changes were made.

## 1.6 Summary

Voice over IP (VoIP) is about transmitting a voice signal across an IP network (the Internet for example). The context of this voice signal determines constraints for this transmission. For instance, if this voice signal is a part of a conversation between two people, care must be taken to preserve its real–time characteristics: the delay between one person talking and the other person hearing what was said should be as low as possible to avoid irritable gaps in the communication. Other applications of VoIP – like an on–line lecture – do not have this delay constraint.

This thesis is about VoIP in networked virtual environments. It contains information about VoIP in general and its application in virtual environments. I will also describe the applications which I developed to test aspects of VoIP in virtual environments.

The classical use for VoIP is as a replacement for a telephone call. Using VoIP like this can reduce costs in various ways, but the quality of the conversation is usually lower than that of a normal telephone call. Using VoIP in virtual environments is relatively new. Such applications would allow users to chat with each other, like on IRC, but instead of typing messages to each other they could simply talk with other users. Adding a 3D effect to the speech signal of a user helps to create a more natural environment. Many other applications use similar techniques as VoIP, for example the transmission of a video signal.

Several components are required to make VoIP in virtual environments possible. The speech signal is split in tiny pieces which are transmitted separately. To be able to transmit a piece of the speech signal, it must first be digitised. At the other end, this digitised signal must be reconstructed into a continuous speech signal which can then be sent to some speakers. Note that several signals may have to be mixed together if several persons are talking at the same time. Also, either at the sender or at the receiver, 3D effects will have to be added to a speech signal. To reduce the amount of required bandwidth to transmit the signal, the digitised speech signal should be compressed. Of course, at the other end it must be decompressed before it can be processed. Finally, there must also be a component which handles the transmission and reception of packets containing speech data.

# Part II: Research

# Chapter 2: The Internet Protocol (IP)

Before we can really talk about Voice over IP, it is necessary to explain what IP is. The abbreviation IP stands for Internet Protocol. Version four is currently most in use and it is common to use the term ‘IPv4’ to indicate this version of the protocol. When no version number is mentioned, usually the discussion is about version four. This is also the case in this thesis.

The Internet Protocol is covered in this chapter. It begins with a discussion about network software architecture, followed by a description of the workings of IP. We will also see some characteristics of IP networks and I will describe the most used protocols which run on top of IP. Afterwards, some reasons will be given for the use of IP for voice communication. Finally, the chapter contains an overview of IPv6, the new version of the Internet Protocol. The information in this chapter was mostly obtained from [34], [9] and [41]. The official specification of IPv4 can be found in [19].

## 2.1 Network software architecture

Nowadays, network software is usually very structured. This section is about the way this software is organised. It also contains a discussion about the OSI reference model, which is a good example of this structured design, and about the TCP/IP reference model, in which – as the name suggests – IP plays a very important role.

### *2.1.1 Layered design*

To facilitate the design of network software, usually the approach of a ‘layered design’ is used. In this approach, each layer provides a certain functionality, which can be used by the layer directly above. There are several advantages to this approach.

First of all, the software is much easier to design. Trying to implement the desired functionality all at once will be very difficult and will probably result in many flaws in the program. Furthermore, these flaws will be difficult to track. By dividing the software in layers, you only have to worry about implementing some functionality for each layer. This does not mean that is will be an easy task, but by using a structured approach you will be able to tackle it more efficiently.

Another advantage is the adaptability. If you want to make some changes to the software, for example to correct a flaw or to improve an algorithm, you will only have to change the relevant layers if the interface with the layer above stays the same.

Closely related to this is portability. If the layers are well designed, only a few of them will have to be changed to be able to use the software with other networking hardware or on another operating system.

Finally, since many layers will probably be implemented as part of the operating system itself, the end–user applications do not have to contain those layers. This way, the size of those applications can be reduced.

To make communication between two hosts possible, they have to be connected to some kind of physical medium. All data will be sent over this medium, but only the lowest layer will have direct access to it. Conceptually, however, two layers on different machines but at the same level can be thought to communicate directly. The rules and conventions that are used in this communication are contained in the protocol for that level. The whole set of protocols is often referred to as the protocol stack. Figure 2.1 illustrates all this.

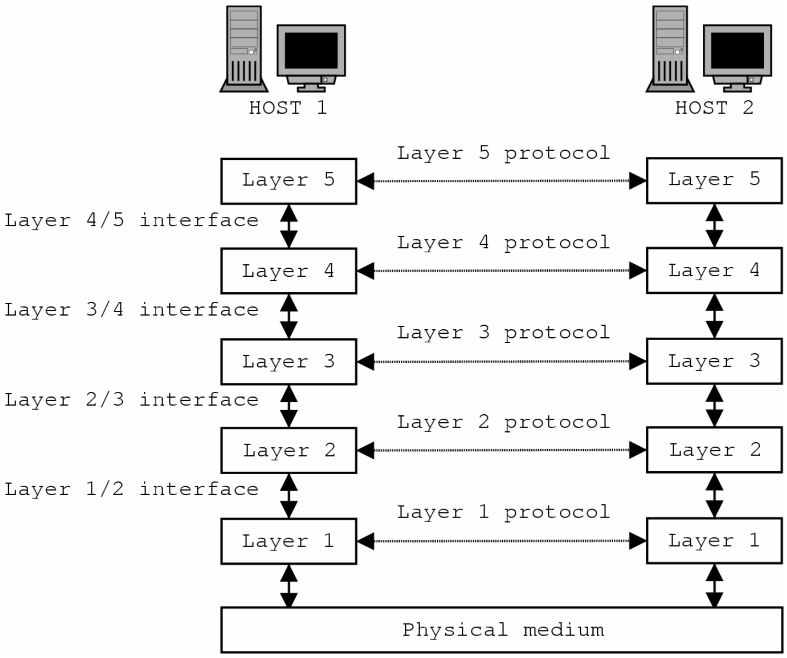


Figure 2.1 – Example of layered design [9]

When a layer wants to transmit some data to its corresponding layer at another host, it uses the functionality of the layer below to do this. That layer adds some control information, usually in the form of a header, to the data and uses the layer below to transmit the data. The whole process keeps repeating itself until the data is finally sent over the physical medium. When the data reaches the receiver, the first layer processes the control information and passes the data to the layer above. At each layer, this process then repeats itself.

### *2.1.2 OSI reference model*

The Open Systems Interconnection (OSI) reference model is a model with seven layers which was developed by the International Standards Organisation (ISO). The model only specifies what each layer should do, without going into any detail about, for example, the protocols that should be used.

In actual implementations it turns out that some of the layers are almost empty and others are too elaborate. However, conceptually the model is quite nice and it is a good example of layered design. This is why I will describe it briefly.

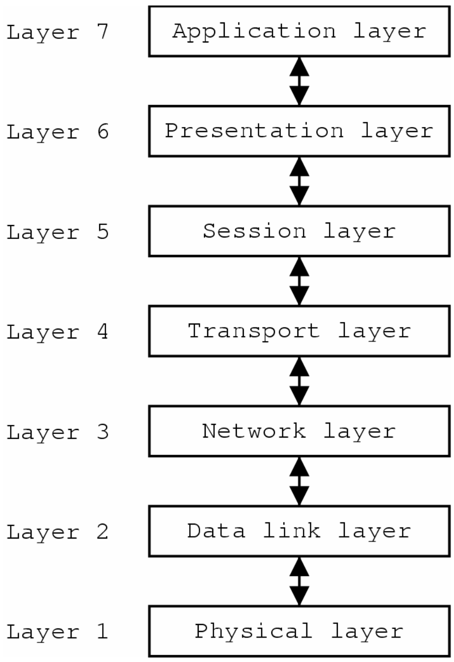


Figure 2.2 – OSI seven layer model

#### 2.1.2.1 The physical layer

The physical layer is the lowest layer in the model and this is the only one which has immediate access to the communication medium. It is responsible for the transfer of bits from the source to a destination which is connected to the same medium.

#### 2.1.2.2 The data link layer

The data link layer uses the facilities of the physical layer to create a more reliable communication channel. This layer makes it possible to send blocks of data, called frames, reliably from one host to an adjacent[3] one.

#### 2.1.2.3 The network layer

So far, the layers have only been concerned with transporting information between hosts connected to the same medium. The network layer's function is to make it possible to send packets to a host that does have a connection to the sender, but is not connected to the same physical medium.

This means that between the different physical media, there have to be devices which transfer data from one medium to another[4]. These devices are usually called routers or gateways. The use of such devices makes some extra work for the network layer necessary.

First of all, it is possible that between a certain source and destination there exist several possible routes. The network layer then has to determine which one to choose. These routes can be determined in advance but it is also possible that the network layer dynamically adjusts the routing information to achieve better performance.

Second, since the flow between adjacent networks can get very large, it is possible that a router cannot cope with all that traffic. The router then becomes a bottleneck for the data flow. The network layer tries to control such congestions.

#### 2.1.2.4 The transport layer

The previous layer made it possible to actually send data from source to destination. In that layer communication is done by exchanging packets. The transport layer makes it possible to consider the data as a stream of bytes, and not in terms of packets. The layer itself will divide the data in smaller units and hand it over to the network layer. If some packets get lost, the layer handles this and the receiver will still receive the correct stream of bytes. To be able to keep track of which data has already been sent and which not, the transport layer uses a connection–oriented approach.

The transport layer will also have flow control mechanisms, to prevent the flooding of a slow receiver, and congestion prevention mechanisms. Note that the network layer also has congestion control functionality. However, the best way to handle congestions is to prevent them from happening in the first place. This is what the transport layer does.

This layer is the first true end–to–end layer. The physical and data link layers were only able to communicate with an immediate neighbour. The network layer actively had to transport the packets step by step from source to destination. In this layer however, the underlying topology is transparent to its user.

#### 2.1.2.5 The session layer

The session layer makes it possible to establish sessions between two hosts. A session extends the capabilities of the transport layer with some extra services.

An example of such an extra service is synchronisation. During a transfer there would be certain synchronisation points. If the data transfer would be interrupted due to an error, the transfer could be restarted from the last synchronisation point rather  than starting the transfer all over again.

---

3 This means the two hosts are directly connected to the same medium.

4 There are also devices which perform a similar function, but operate at the level of the data link layer. These are normally called bridges or switches. They are usually used within an organisation to connect multiple LANs together.

#### 2.1.2.6 The presentation layer

The presentation layer takes the type of information which is being transferred into consideration. This layer could, for example, make the necessary transformations if one computer is sending ASCII characters and the other one is sending Unicode characters.

#### 2.1.2.7 The application layer

Finally, the highest layer in the model is the application layer. This is the layer in which most end–user networking applications reside. To communicate, such programs mostly use their own protocols. Examples of such applications are applications for file transfer and applications which represent a virtual terminal.

### 2.1.3 TCP/IP reference model

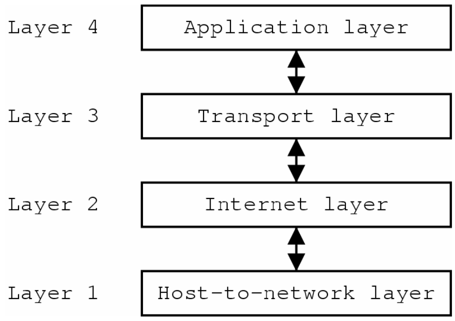


Figure 2.3 – TCP/IP four layer model

The Internet Protocol is a protocol which is used in the TCP/IP model. The TCP/IP model was originally designed for use on the ARPANET, a military network in the late 1960s. It is, in fact, this network which grew out to become the Internet as we know it today.

Because of its military background, there were two major requirements for the model. The first was robustness. The US Department of Defence (DoD) wanted to make sure that communication was still possible even if some routers or lines went down. The second requirement was interoperability. Since there were different types of hardware involved, for example copper wires and satellites, the DoD wanted a set of protocols which could not only handle these types of hardware separately, but which would also make it possible to connect them.

Compared to the OSI model there is a big difference in the way that the model came to existence. The OSI model was first carefully designed, and later protocols were designed to fit the model. This makes the OSI model a very general one. The TCP/IP model, however, originated in the opposite way. First the protocols were designed to meet the requirements of the DoD. Later, these protocols were described and it is this description which is the reference model. This means that the TCP/IP model does not really fit anything else but TCP/IP networks. Another point about TCP/IP is that the layered design is not followed very strictly. There are some violations to this principle in the model.

Despite of these arguments, the TCP/IP model has become very popular and very widely used. In contrast to the OSI model which has seven layers, the TCP/IP model only has four, as figure 2.3 shows. Here is a description of these layers.

#### 2.1.3.1 The host-to-network layer

The host–to–network layer is the lowest layer of the model. Sometimes it is also called the link layer or the network interface layer. There is in fact little to be said about this layer. The only requirement which is given by the model is that this layer should be able to transmit and receive the IP datagrams of the layer above over the network. The layer has somewhat the same function as the physical and data link layers in the OSI model. This means that this layer usually is only able to send data to hosts which are connected to the same medium.

#### 2.1.3.2 The internet layer

The internet layer corresponds to the network layer in the OSI reference model. Its job is to bring packets from source to destination, across different types of networks if necessary. There are, however, no guarantees that the packets will arrive or that their order will be preserved. The service that this layer offers is therefore called a best–effort service. There is no notion of a connection in this layer. The packets which are exchanged are called Internet Protocol datagram or IP datagrams and the protocol which is used is called the Internet Protocol or IP. The datagrams consist of a header and the actual data. The header will be described later on.

Like in the OSI network layer, intermediate devices called routers, are needed to make transmission of data across different types of networks possible. The IP datagrams can then be sent from source to destination, on a hop–by–hop basis. Again, like in the OSI network layer, this also means that routing algorithms and congestion control are important aspects of the internet layer.

#### 2.1.3.3 The transport layer

To make sure that multiple applications can use the network facilities at once, some extra naming mechanism is needed. The internet layer does contain a naming mechanism to identify different hosts, but there still has to be some way to differentiate between the processes which are using the network. This is done in the transport layer by the use of a port number. This layer has somewhat the same functionality as the transport layer in the OSI model. Here also, the transport layer is the first real end–to–end layer.

The TCP/IP model has two major transport layer protocols. One of them is the Transmission Control Protocol (TCP). This protocol transforms the connectionless unreliable packet based service of the internet layer into a connection–oriented reliable byte stream. It is a very important protocol since it makes reliable communication possible. This is why its name is also in the name of the reference model.

The other protocol is the User Datagram Protocol (UDP). This is a protocol for applications which do not need the service offered by TCP or want to use a protocol of their own. The User Datagram Protocol is merely a small extension to IP. It is also an unreliable packet based connectionless protocol and the only real extensions to IP itself are the presence of a port number and an optional checksum of the data.

#### 2.1.3.4 The application layer

Like in the OSI model, the application layer contains the protocols of networking applications. Among these are virtual terminal applications (TELNET protocol), file transfer utilities (FTP protocol) and electronic mail (SMTP protocol).

## 2.2 How IP works

Let us now take a closer look at the Internet protocol itself and how it makes communication between two hosts possible. First I will give a description of the IP packet format. Next, the addressing mechanism used by IP is discussed. We will then take a closer look at how packets are routed from source to destination. Finally, an explanation is given of multicasting, a technique which allows us to save bandwidth when the same data has to be sent to multiple destinations. This is, of course, a very interesting feature when using VoIP in virtual environments, since there will typically be many receivers for each talking participant.

### *2.2.1 Packet format*

Any packet sent by the IP layer consists of an IP header, followed by the actual data. The format of the IP header is shown in figure 2.4. The most significant bit is the one at the left, numbered zero. The least significant bit is the one at the right, numbered thirty–one. Transmission is done in network byte order, also called big endian format. This means that in each 32–bit word the most significant byte is sent first and the least significant byte is sent last.

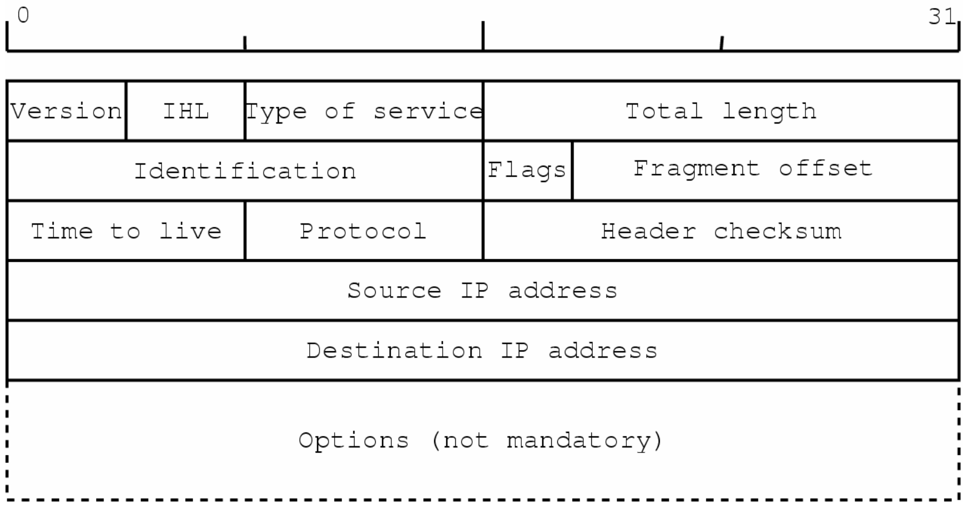


Figure 2.4 – IP header format

The **version** field should contain the value ‘four’ for the current version of the Internet Protocol. This field can be used to let different versions coexist, something which will make the transition to a new version much easier.

The **IHL** field contains the ‘Internet Header Length’. This specifies the length of the header in 32–bit words. Since it is a 4–bit value, the maximum length of the header will be sixty bytes. Also, since the mandatory part of the header consists of five words, the smallest legal value is five. The specification in 32–bit words also has as a consequence that the header must end on a 32–bit boundary, so it is possible that some padding is required if options are present.

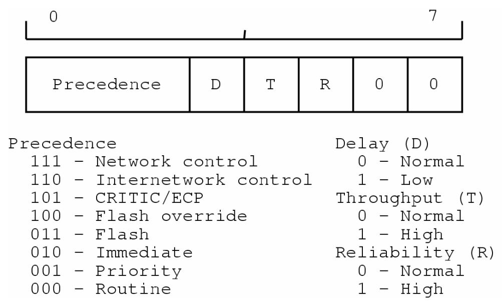


Figure 2.5 – The TOS field [19]

The next field is the **Type of service** (TOS) field. This field was meant to supply a quality of service (QoS) mechanism, but in practice it is rarely used. However, since voice data has real–time aspects, it may be necessary to pay attention to it if we want to keep the end–to–end delay in the communication low.

An overview of the TOS field is depicted in figure 2.5. The byte contains a three–bit precedence field which specifies the priority of the packet. A value of zero indicates a normal priority and a value of seven indicates the highest priority. Following the precedence field, there are three bits which stand for delay, throughput and reliability. Only one of the bits can be set to one. The last two bits in the field are currently unused and should be zero.

The size of the IP datagram is specified in the **Total length** field. It is a 16–bit field, so the maximum size is 65535 bytes. Most networks cannot handle this size so usually it is much less. All hosts are, however, required to be able to send and receive datagrams with a length of 576 bytes or less.

During the transmission of a packet it is possible that it has to traverse different kinds of networks. Each network has its own Maximum Transfer Unit (MTU) which specifies the maximum frame size it can handle, including the link layer header and trailer (if present). This means that there is always a possibility that the datagram, as it passes over the different networks, cannot be transmitted over a certain network. It then has to be fragmented and each piece has to be sent separately.

The **identification** field is an aid in reconstructing fragmented datagrams. Each datagram fragment will have the same value in this field. When sending IP datagrams, a host typically increments this field for each datagram sent.

Next, there are three **flag** bits, of which the first one is reserved and should be zero. The next one stands for ‘don’t fragment’ (DF) and the last one stands for ‘more fragments’ (MF). If a datagram cannot be transmitted across a network because it is too large and the DF bit is set, an error will be sent back to the sender[5]. All but the last the fragment of the original datagram will have the MF bit set.

Using the **fragment offset** field, the internet layer can reassemble fragmented datagrams. This 13−bit value specifies the offset of the fragment in the original datagram. The offset is given in units of 64−bit words.

The **time to live** (TTL) field is used to limit the lifetime of a datagram. In theory the value specifies the number of seconds the datagram is allowed to exist. There is also the requirement that each router must decrement the value by at least one. If the packets stays a long time in the queue of the router, the TTL value should be decreased with the number of seconds the datagram spent in queue. When the counter is zero, the datagram must be discarded. In practice, the value is just decremented at each router, which makes the field a hop counter.

The **protocol** field is used to specify to which protocol the data in the datagram belongs. This can be a transport layer protocol, but it can also be one of the control protocols of the internet layer.

The **header checksum** is used to check the validity of the datagram. Note that the checksum is only for the header, so higher level protocols will have to use their own checksums if they want to make sure their data is valid.

Finally, the minimal header contains the **source IP address** and the **destination IP address**. These addresses must be included in each datagram since the internet layer operates in a connectionless way. Each datagram is sent separately and therefore each datagram must contain not only its destination but also its source, in case an error has to be reported. The format of the addresses is described further on.

The **options** section can be used to record the route a datagram follows, possibly with timestamps. Another option is source routing, where you can specify the route a datagram should follow.

### 2.2.2 Addressing

Every host on an interconnection of networks – or internet – which uses IP, should have a unique IP address. An IP address is a 32−bit value and the complete address space is divided into five classes, named class A to class E. The way these classes are represented is shown figure 2.6.

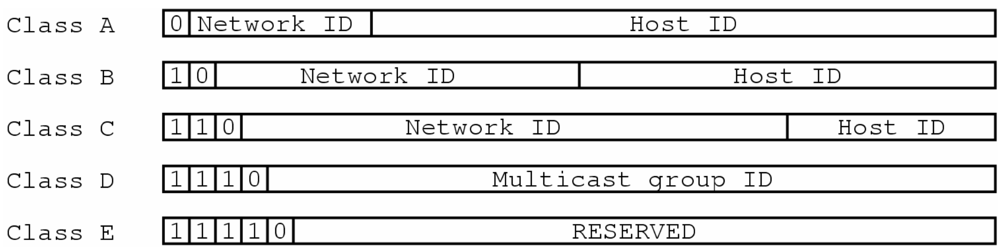


Figure 2.6 – Classes of IP addresses [34]

5 The error is reported by an ICMP (Internet Control Message Protocol) datagram. This is a control protocol used by the internet layer, but I will not discuss it any further.

The way an address is usually written, is in its dotted decimal form. To obtain this the 32−bit value is split in four 8−bit values. These four values are then written in decimal form, separated by dots.

The first three classes contain the addresses which can be assigned to hosts. Not all possibilities are allowed though; there are some reserved addresses. First of all, a host ID with value zero does not specify a host, but the network on which hosts with the specified network ID are located.

If the host ID is the highest possible value for its class (all one bits in binary format), the address is a broadcast address for a certain network. This means that if you send IP datagrams to that address, they are delivered to all hosts on that network.

When the network ID of an address is zero, it specifies the local network. This type of address is only used in initialisation procedures, when the local network ID is not known.

Other reserved addresses are 0.0.0.0 and 255.255.255.255. The first of these specifies the local host on the local network. It is also only used in initialisation procedures. The second address is the so−called limited broadcast address. This specifies a broadcast to all hosts on the local network.

Of the remaining two classes, only class D is actually used. Class E was meant for future use. Class D specifies a multicast address. Multicasting allows data to be sent to a group of hosts. This means that when you send an IP datagram to a multicast address, the datagram is sent to all hosts in the corresponding multicast group. Multicasting is explained in more detail later.

### 2.2.3 Routing

The internet layer uses the link layer to actually transmit its data. The link layer, however, can only deliver this data to hosts which are connected to the same medium. To be able to send this data across several networks, routers are used. These devices connect to several networks and make sure that incoming IP datagrams are forwarded to the appropriate network. We will now take a closer look at how this process works. Note that only the basic mechanisms of routing are explained here.

When the internet layer of the sending host has to transmit a datagram to a certain destination, it first examines the destination IP address. This is necessary because the internet layer has to tell the link layer to which machine the data has to be sent. If the destination IP address is on the same network, the machine which will receive the datagram will simply be the destination for the transmission.

If the address does not specify a host on the local network, the internet layer examines its routing table. The entries of such a routing table can be seen as pairs of a destination address and a router address. The destination address can be an address of a host or of a network.

The internet layer then starts looking for a router to send the datagram to. To do this, it compares the destination address of the datagram with the destination addresses in the routing table. If no complete match can be found, it checks if a matching network entry can be found. If not, it uses a default entry. If an entry was found, the internet layer takes the corresponding router address and tells the link layer to send the datagram to that address.

For example, consider a host with IP address 199.198.1.10 who wants to send a packet to 199.198.2.100. This destination host is not on the same network, so the internet layer of the sender will consult its routing table. Suppose that the table looks like this[6]:

6 This is a very simple representation of a routing table. Actual routing tables contain more information than is presented here.

| Destination | Gateway |
| --- | --- |
| 199.198.5.10 | 199.198.1.251 |
| 199.198.2.0 | 199.198.1.252 |
| default | 199.198.1.253 |

The internet layer first looks in the table for a complete match for address 199.198.2.100. It finds no such match, so it will check for a matching network address. This time, it does find a matching entry: the second one describes the network on which the destination host is present. The internet layer then takes the corresponding gateway entry – address 199.198.1.252 – and sends the packet to that router (gateway).

When the datagram reaches the router, it is passed on from the link layer to the internet layer. The internet layer then follows almost the same procedure to search for a destination machine to forward the datagram to. The only difference is that the router will usually be connected to several networks and this means that the appropriate interface to transmit the data also has to be chosen. The whole procedure is repeated until the datagram reaches its final destination.

To make sure good routes are chosen, many routers communicate with each other. They exchange their routing information and based upon this information each router updates its routing table to contain the best known route for each destination. The type of information and the way it is exchanged are determined by the routing protocol which is used. Examples of routing protocols are the Open Shortest Path First (OSPF) protocol and the Border Gateway Protocol (BGP).

### 2.2.4 Multicasting

Basically, there are three transmission modes that can be used when sending an IP datagram. They are called unicast, multicast and broadcast. Unicasting simply means sending a datagram from a source to one destination. The term broadcasting is used when you want to send a datagram to all hosts on a specific network. When you want to send a datagram to an arbitrary set of hosts, it is called multicasting.

A simple way to implement multicasting would be to unicast a copy of the datagram to each destination. This method obviously wastes a lot of resources. A better way would be to transmit one datagram which is copied only at points where it needs to follow different routes to reach its destinations. This is the way it is done on IP networks.

To be able to receive datagrams directed to a certain multicast address, a host must first join the multicast group associated with that address. Similarly, when it no longer wants to receive those datagrams, it leaves the multicast group. This group management is done according to the Internet Group Management Protocol (IGMP), which is formally specified in [18].

In general, the protocol works as follows. Each host maintains a list of multicast groups from which it wants to receive datagrams. Multicast routers periodically broadcast IGMP queries on the networks to which they are connected. The hosts then send IGMP replies, containing the groups in which they are interested.

Once these replies have been gathered using IGMP, multicast routers exchange this data with each other and use all this information to build their routing tables. When they receive a multicast datagram, they can then determine to which hosts and multicast routers the datagram should be sent.

## 2.3 Characteristics of IP networks

When datagrams have to travel across several networks, they will also need to pass through a number of routers. Each router has to examine all incoming packets and this will introduce a certain delay in the communication. Studies even show that the time it takes for a packet to reach its destination is much more affected by the number of hops the packet makes than the actual geographical distance covered [41].

When a router gets too heavily loaded, some packets will have to be discarded. This packet loss is usually bursty. This means that for a short period of time several consecutive packets will be lost.

Routers communicate with each other to dynamically adapt their routing tables to the current state of the network. This means that datagrams going to the same destination can sometimes follow different routes. Although it turns out that routes do not change very often during a transmission, it does happen. Such a change can cause datagrams to arrive out of order.

Besides packet loss and out–of–order arrival of packets, it can also happen that a datagram gets duplicated during its transmission. This will cause two or more identical datagrams to arrive at the destination, possibly with some delay between them.

Finally, another important feature of IP networks is the fact that when a source sends datagrams to a certain destination, the amount of time to reach the destination will differ for each datagram. This is usually called inter arrival delay, inter arrival jitter or simply jitter.

## 2.4 Higher level protocols

The two most common transport level protocols in the TCP/IP architecture are the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Each of these protocols offers a specific kind of service which applications can use to communicate across networks.

### *2.4.1 TCP*

Currently, TCP is undoubtedly the most used protocol of the two. This protocol transforms the unreliable packet–based service of the internet layer into a reliable byte stream. The protocol is designed for communication between two hosts, so it only supports unicasting.

To offer this kind of service, the TCP module has to do a lot of work. First of all, a connection has to be set up, and this has to be done in such a way that it is more or less safe: the module must make sure that connections cannot be established accidentally – for example because of duplicate packets.

The incoming stream of bytes then has to be split up at the side of the sender and the stream has to be reconstructed at the side of the receiver. Care must be taken to discard duplicate datagrams and to correct their arrival order if necessary. There must also be some kind of mechanism to cope with lost packets.

All this is handled quite effectively. To establish a connection the TCP module uses a handshake mechanism, called a three–way handshake. Duplicate and out–of–order datagrams are handled by using sequence numbers. Finally, lost packets are handled by an acknowledgement mechanism: all bytes of the stream have to be acknowledged by the destination. If the source did not receive an acknowledgement after a certain amount of time, it sends the necessary data again. The protocol also specifies flow control mechanisms, which prevent the swamping of a slower receiver, and congestion control mechanisms, which try to avoid congestions.

Note that the exact way in which the TCP module works, is a lot more complicated than this explanation makes it seem. For a complete specification of TCP, you should refer to [36].

### 2.4.2 UDP

Applications which do not require the functionality that TCP provides, can use UDP. To transmit data, the UDP module simply passes a UDP header followed by that data to the internet layer which then sends the datagram on its way. This means that just like IP itself, UDP is a best–effort service. No guarantees about delivery are given, datagrams can get reordered and datagrams can be duplicated. The exact specification of UDP can be found in [39]

The UDP header is shown in figure 2.7. The header contains the source and destination ports, which identify the sending and receiving applications. Next, it contains the number of data bytes which must be sent and finally the header contains space for an optional checksum.

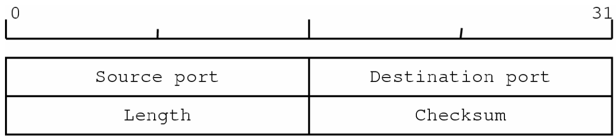


Figure 2.7 – UDP header

Since the service which UDP offers is almost identical to the service of IP itself, it is possible for applications to send UDP datagrams to a multicast address and to receive UDP datagrams from a multicast group.

## 2.5 Why use IP?

Delivering speech information in packets has some advantages to the classical telephone system. When you make a ‘normal’ telephone call, a path is set up between you and the destination of the call. You will then have a fixed amount of bandwidth you can use during the whole call.

The major advantage of that approach is that you will have some guarantees about the QoS, since you are certain to have a specific amount of bandwidth available. But this way, a lot of bandwidth is also wasted, because during a conversation there are a lot of silent intervals for each person.

Using VoIP, those silent intervals can be detected. The VoIP application can examine each packet and detect whether it contains speech information or only silence. If the latter is the case, the packet can simply be discarded.

Another advantage is the possibility of compression. With the compression methods available today, it is possible to reduce the requirement of 64 kbps[7] for uncompressed telephone–quality voice communication to amounts which are far lower. However, a high compression ratio often means that the voice signal will be of lesser quality. We will go deeper into the domain of compression in one of the next chapters.

So packetised voice has certain advantages to the classical telephone system. But IP is not the only packet–based protocol. Why exactly should IP be used? This protocol was designed mostly for data transport, and it has only limited QoS support. The main reason IP is so important is because of its omnipresence. The TCP/IP architecture has proved to be very popular and nowadays it is very widely used. This fact gives IP a great advantage over other protocols.

Alternatives for packetised voice include Voice over Frame Relay (VoFR) and Voice over ATM (VoATM). Both allow better support for real–time traffic than an average IP

7 ‘kbps’ stands for kilobits per second. A kilobit is equivalent to 1000 bits. Why this rate is 64 kbps will be explained in the next chapter.

network. However, these technologies are not used as widely as IP.

## 2.6 IPv6

With the growth of the Internet – on which IP is used – it has become clear that the current version of the Internet Protocol has some shortcomings. For this reason a new version of the protocol has been devised, now called IP version six, or just IPv6.

This section contains a brief description of the protocol, which was introduced in [20] and later redefined in [21]. The latter reference is the source of the information in this section.

### *2.6.1 Reasons*

Because of the enormous growth of the Internet, there will soon be a shortage of IP addresses. The current version uses 32–bit values, which can provide enough IP addresses in theory. However, because of the subdivision in classes and the way addresses are allocated within those classes, in practice there are far less addresses available. This lack of addresses was one of the most important reasons for the development of a new version.

Other reasons were the need for better QoS support and better support for security. Also, it turned out that some features of IPv4 were hardly ever used and bandwidth and processing time could be saved by redesigning the protocol. Finally, because the routing tables in routers kept growing, the reduction of their sizes was also an important reason for the design of an improved protocol version.

### *2.6.2 Description*

Let us now take a closer look at this new protocol. First I will describe the format of the IPv6 header. Next, we will see what exactly changed compared to IPv4.

#### 2.6.2.1 Header

The IPv6 header is shown in figure 2.8. In this version, the header has the fixed size of forty bytes.

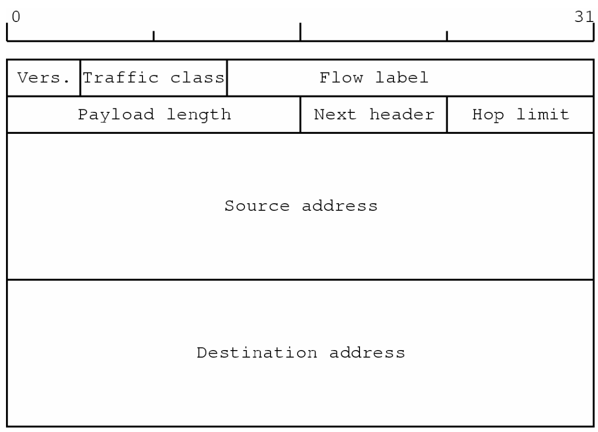


Figure 2.8 – IPv6 header [21]

The **version** field contains the value six. This way, the version of the protocol can be detected and IPv4 and IPv6 can coexist. This will make the transition to the new version easier.

The **traffic class** has somewhat the same function as the TOS field in the IPv4 header. Using this field, one could specify the type of traffic this datagram belongs to. This could then allow appropriate handling of the datagram.

A flow is defined as a sequence of datagrams which are sent from a certain host to a receiver or – in case multicasting is used – to a group of receivers, and for which the sender desires special handling by the routers along the way. The **flow label** field can then be used as an identifier for such flows.

The number of data bytes following the header is specified by the **payload length** field. This is a 16–bit wide field, so the maximum number of data bytes in a datagram is 65535. However, it is possible to create larger datagrams than this field allows. How this can be done is explained further on.

The **next header** field specifies of what type the header following the IPv6 header is. In the simplest case, this is a header from a higher level protocol. But it can also be one of the extension headers which IPv6 defines. It is because of these extension headers the IPv6 header is somewhat simpler than the header of IPv4. Some fields in the IPv4 header and the different options are now used through extension headers.

Several extension headers are defined. Fragmentation, security, authentication, source routing and many more are all made possible through these extension headers. For a complete description you should consult [21].

Earlier, I mentioned that the payload length of 65535 can be exceeded. Well, this can be done using an so–called ‘hop–by–hop’ extension header. This header has an option called ‘Jumbo Payload’ and allows lengths greater than 65535 to be specified. Such datagrams are often called ‘jumbograms’.

The **hop limit** field is a replacement for the TTL field in the IPv4 header. This field limits the lifetime of a datagram by requiring that the value in the hop limit field must be decremented by one by each node that forwards the packet.

Finally, the header contains the **source address** and the **destination address** for the datagram, which are 128–bit values.

#### 2.6.2.2 Important changes from IPv4

First of all, there is the larger address space. The 128–bit values should be enough to continue for quite some time. On the entire planet, these addresses would allow for 7 x $10^{23}$ addresses per square meter [9].

Furthermore, because of the way multicast addresses are represented, the scalability of multicast routing should be improved. Also, a new type of transmission, called ‘anycasting’, is available. This type of transmission is used to send a datagram to anyone of a group of receivers.

The header format is simpler than it was the case with IPv4. The IPv6 header has only eight fields, whereas the IPv4 header had at least twelve fields. This allows for faster processing of datagrams. The extension headers give the protocol great flexibility, certainly compared to the limited IPv4 options field.

The concept of a flow is also new to this version. This makes it possible for a certain stream of data to receive special treatment. This feature could prove to be useful for real–time services for example.

Finally, the added support for authentication and security are definitely an important improvement over version four.

## 2.7 Summary

The Internet Protocol or IP is part of a layered architecture, called the TCP/IP reference model. This model consists of four layers, each containing a number of functions which the layer above can use. The internet layer is the one which defines IP. This layer makes it possible to send blocks of data called datagrams from source to destination. It does this by actively sending the datagram across each intermediate network. Adjacent networks are connected by devices called routers, which examine incoming IP datagrams and forward them to the right network. IP supports multicasting, a technique to send a packet in an efficient way to any number of destinations.

The Internet Protocol is a connectionless packet based protocol which offers no guarantees about datagram arrival. Datagrams can even be duplicated or delivered out of order. Other characteristics of IP networks are the delay introduced by routers and inter arrival jitter.

The transport layer – the layer above the internet layer – contains two widely used protocols. The Transmission Control Protocol, or TCP, offers a connection–oriented service where the connection can be considered to be a reliable byte stream. The User Datagram Protocol, or UDP, is merely a transport layer extension to IP and has the same characteristics.

The main advantages of packet based telephony are the possibilities for silence suppression and speech compression. The omnipresence of IP is the main reason why this protocol is a good candidate.

For a number of reasons a new version of the Internet Protocol has been developed. The most important one was the fact that there would soon be no more IP addresses available on the Internet. The new version of the protocol is known as IP version six, or IPv6.

# Chapter 3: Voice communication

In the previous chapter the Internet Protocol was explained. This was done in a general way, without paying much attention to Voice over IP. Since we now know the most important features of the protocol, we can bring other components of VoIP into the picture.

In this chapter we will take a closer look at some aspects of digitised voice communication. The chapter starts with a discussion about grabbing and reconstruction of voice signals. Next, the requirements for a reasonably good form of voice communication are given. We will then take a closer look at communication patterns and finally we will see what the impact of all these things is on VoIP.

## 3.1 Grabbing and reconstruction

Before you can send voice information over a packet network, you must first digitise the voice signal. After the transmission, the receiver of this digitised signal has to convert it back to an analogue signal, which can be used to generate speaker output. The first stage is also called ‘grabbing’ of the voice signal and the second stage is called reconstruction. In general, these stages are also referred to as analogue–to–digital (A/D) conversion and digital–to–analogue (D/A) conversion, respectively.

As for terminology, it is useful to know that digitising an audio signal is often referred to as pulse code modulation (PCM).

Nowadays, digitisation and reconstruction of voice signals can be done by any PC soundcard, so this is not the most difficult step in creating VoIP applications. For completeness, however, I will give a brief description of the processes.

### 3.1.1 Sampling and quantisation

A continuous signal (a voice signal for example) on a certain time interval has an infinite number of values with infinite precision. To be able to digitally store an approximation of the signal, it is first sampled and then quantised.

When you sample a signal, you take infinite precision measures at regular intervals. The rate at which the samples are taken is called the sampling rate.

The next step is to quantise the sampled signal. This means that the infinite precision values are converted to values which can be stored digitally.

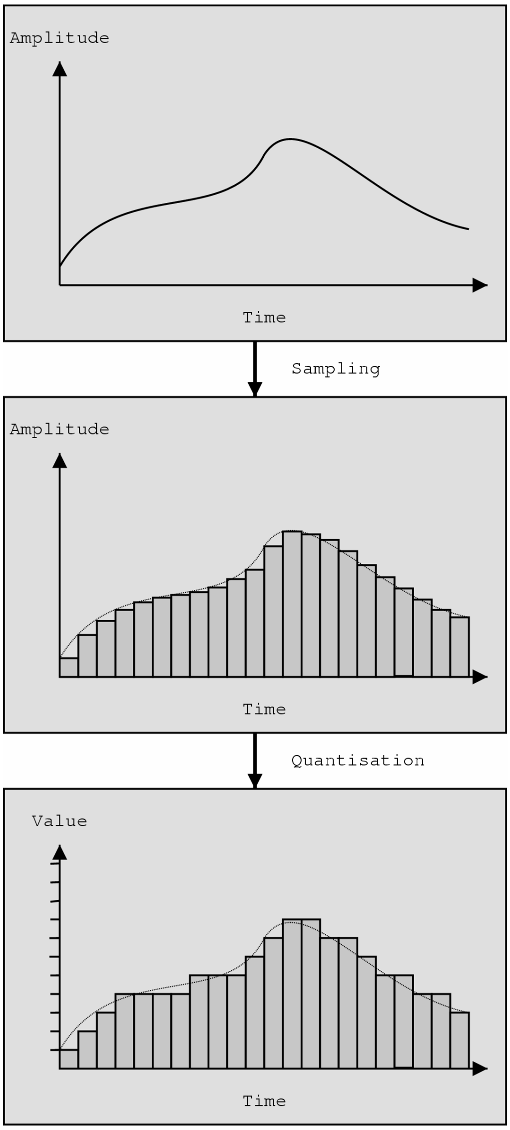


Figure 3.1 – Sampling and quantisation

In general, the purpose of quantisation is to represent a sample by an N–bit value. With uniform quantisation, the range of possible values is divided into $2^N$ equally sized segments and with each segment, an N–bit value is associated. The width of such a segment is known as the step size. This representation results in clipping if the sampled value exceeds the range covered by the segments. [10]

With non–uniform quantisation, this step size is not constant. A common case of non–uniform quantisation is logarithmic quantisation. Here, it is not the original input value that is quantised, but in fact the log value of the sample. For audio signals this is particularly useful since humans tend to be more sensitive to changes at lower amplitudes than at high ones [23].

Another non–uniform quantisation method is adaptive quantisation [10]. With such methods, the quantisation step size is dynamically adapted in response to changes in the signal amplitude. PCM techniques which use adaptive quantisation are referred to as adaptive PCM (APCM).

The sampling and (uniform) quantisation steps are depicted in figure 3.1. An important thing to note is that both steps introduce a certain amount of error. It is clear that a higher sampling rate and a smaller quantisation step size will reduce the amount of error in the digitised signal.

### 3.1.2 Reconstruction

Signal reconstruction does the opposite of the digitisation step. An inverse quantisation is applied and from those samples a continuous signal is recreated. How much the reconstructed signal resembles the original signal depends on the sampling rate, the quantisation method and the reconstruction algorithm used. The theory of signal reconstruction is quite extensive and goes beyond the scope of this thesis. A good introduction can be found in [11].

### 3.1.3 Mixing audio signals

When using VoIP in virtual environments, there is another thing that we must take into account. Each participant will send its own digitised voice signal which will be received by a number of other participants. If two or more persons are talking at the same time, their signals will have to be mixed somehow.

Luckily this is very simple: physics teaches us that for sound waves, the principle of superposition applies. This principle states that when two waves overlap, the amplitude of the combined wave at a specific time can be obtained simply by adding the amplitudes of the two individual waves at that time [38]. Practically speaking this means that we merely have to take the sum of the digitised versions of the signals.

## 3.2 Communication requirements

Nowadays everybody is used to telephone quality voice which typically has very few noticeable errors and low delay. Also, when using the telephone system there is no such thing as variation in delay. With packetised voice however, each packet will typically arrive with a slightly different amount of delay, resulting in jitter. There is also no guarantee about delay caused by the network and in general, some packets will contain errors on arrival or will not even arrive at all.

In this section, we will see what the requirements are for decent voice communication. With 'decent communication' a form of conversation is meant which does not cause irritation with the participants.

### 3.2.1 Error tolerance

In contrast to data communication, where even the smallest error can cause nasty results, voice communication is much more tolerant to the presence of errors. An occasional error will not seriously disturb the conversation as long as the error does not affect a relatively large portion of the signal.

### 3.2.2 Delay requirements

When you are using data communication, it does not really matter how much delay there is between the sending of a packet and its arrival. With voice communication however, the overall delay is extremely important. The time that passes between one person saying something and another person hearing what was said, should be as low as possible.

Studies show [10] that when the delay exceeds 800 ms, a normal telephone conversation becomes very hard to do. They also show that a delay of 200 to 800 ms is tolerable for short portions of the communication. However, in general a delay below 200 ms has got to be attained to hold a pleasant conversation.

### 3.2.3 Tolerance for jitter

If each block containing a part of a digitised voice signal would be played immediately on arrival, the quality of the communication would be rather low. Since each block typically arrives with a slightly different delay, sometimes a block would be played before the previous one was finished and sometimes there would be a small gap between the end of one block and the next. Since this jitter is not something that occurs for short periods, but continuously, this will be very annoying to the participants of the conversation[8].

## 3.3 Communication patterns

In a conversation between two persons, it it very unlikely that both are always talking at the same time. Usually, when one person is speaking, the other one listens, possibly giving short affirmations. The same principle applies when a group of people is holding a discussion: when one person is talking, the other ones listen.

It is because of this pattern that a normal telephone call wastes a large amount of bandwidth. When someone is not speaking, the bandwidth stays assigned without being used. With packetised voice this bandwidth could be used by other calls or applications.

Several speech models are presented in [10]. Such models could be used to predict arrival patterns of packets containing voice data. These predictions in turn could be used to create a network design with a more effective utilisation of resources. Although these models are obviously important, I feel that they are beyond the scope of this document. If you are interested in these matters, you should refer to [10].

## 3.4 Impact on VoIP

We have just seen some aspects of voice communication. In this section the importance of these aspects for VoIP is described.

### 3.4.1 Sampling rate and quantisation

At the start of the chapter we saw what sampling and quantisation is. It was also mentioned that a higher sampling rate and a smaller quantisation step implied a better representation of the

8 Actually, this depends on the amount of information contained in a packet. For example, if a packet contains two seconds of a voice signal, these delay variations probably will not disturb the conversation. If, however, a packet contains only twenty milliseconds of a voice signal, these variations will prove to be very irritable. Further on we will see why packets containing very small parts of the signal are to be preferred.

original signal. But this also means that more digitised information will have to be transmitted and more bandwidth is required. So we have to determine how much information is necessary to hold a telephone quality conversation.

The Nyquist theorem is important in this matter. It states that the minimum sampling frequency should be twice the maximum frequency of the analogue signal [26]. This is in fact quite logical: to be able to capture N cycles, you have to take measures at at least 2N points along the signal. Otherwise it would be impossible to capture the maxima and minima of the signal.

The speech signals that humans produce can contain frequencies of even beyond 12 kHz [10]. However, in the telephone system, only frequencies below 4000 Hz are transmitted and this still allows high−quality communication. Using this information together with the Nyquist theorem suggests that a sampling rate of 8000 Hz is adequate for the digitisation of speech.

To cover the range of amplitudes that a voice signal can produce, at least twelve bits are needed when a uniform quantisation scheme is used [23]. However, a uniform scheme which reduces each sample into an eight bit value is usually good enough to attain telephone quality conversations.

As for logarithmic quantisation, there are two schemes that are also worth mentioning at this point. In the United States and Japan, μ−law encoding is the standard for transmission over networks. It reduces a thirteen bit uniformly quantised signal into an eight bit value. In Europe, A−law is used. This scheme does the same but starts with a twelve bit uniformly quantised value. [23]

From this information we can calculate the required bandwidth for a telephone quality conversation. Above was explained that we needed a sampling rate of 8000 Hz and that an eight bit value is usually used to store a sample. So we will be transmitting 8000 eight bit values each second. This requires a bandwidth of at least 64000 bps or 64 kbps.

### 3.4.2 Packet length

With Voice over IP, packets can get corrupted or even lost. To reduce the amount of lost information, a packet should contain only a very small amount of the voice signal. This way, if a packet gets lost, only a tiny fraction of the conversation will be missing, which is very unlikely to disturb the conversation.

It is important to note that this argument is made from the point of view of the conversation. We will see later on that from the transmission's point of view, an argument can be made in favour of larger packets. Somehow, a compromise will have to be made.

### 3.4.3 Buffering

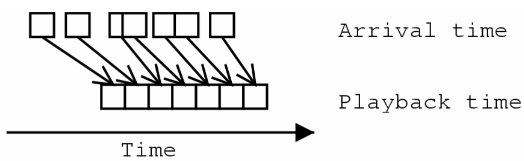

Figure 3.2 – Buffering to avoid jitter

In a previous section the negative effects of jitter were explained. A simple technique to avoid jitter in the playback of a voice signal, is to introduce an amount of buffering, as figure 3.2 illustrates.

Instead of playing the voice data of an incoming packet as soon as the packet arrives, a small amount of delay is introduced. Because this is done for all packets, there is a higher probability that when one packet has been played, the next one is immediately available.

In practice, normally only a small amount of buffering is needed. In the applications I developed, I have used jitter calculations to determine the amount of buffering needed. The amount of jitter is usually not very high and accordingly only a small amount of buffering is done. This method appears to have good results.

### 3.4.4 Delay

A large delay is disastrous for a conversation. The total delay can be categorised into two types [42]. The first type is fixed delay. This is the total delay which is always present due to buffering, link capacity etc. The second type is variable delay. This is the delay component which is caused by packet queuing in routers, congestions etc.

The components shown in figure 3.3 determine the amount of fixed delay for a VoIP system in virtual environments. For ‘normal’ VoIP, the 3D processing component can simply be left out.

| |
|---|
| Sampling delay |
| Compression delay |
| Transmission delay |
| Decompression delay |
| 3D processing delay |
| Buffering delay |

Figure 3.3 – Types of delay

The **sampling delay** is the delay introduced by the sampling of the voice signal. For example, if the sampling interval is one second, the total delay will be at least one second long since the digitised voice signal cannot be processed before the data is collected. This means that from the point of view of the delay, the sampling interval should be kept as small as possible. Again, from the point of view of the transmission, an argument will be made in favour of larger sampling intervals.

The **compression** and **decompression delays** are introduced by compression and decompression algorithms respectively. The **transmission delay** is the delay which is present due to link capacities. The **3D processing delay** is the delay caused by the algorithms which generate the three dimensional sound for use with virtual environments.

Finally, the **buffering delay** is the delay which is artificially introduced to compensate for jitter. This delay could be set manually or determined automatically, like it is done in my own VoIP applications[9].

### 3.4.5 Silence suppression

We saw earlier that in a conversation, there is usually only one person speaking at a time. In packetised voice, this gives us the opportunity to save bandwidth because packets containing only silence do not need to be sent.

However, before we can discard packets, we must first be able to determine whether they contain silence or not. One way this could be done is by calculating the amount of energy of the voice signal in a packet. Packets which do not contain a sufficient amount of energy are assumed to hold only silence and can be discarded.

A simpler technique which I have used is to check a packet for samples with a value above a certain threshold. If no such samples exist, the packet is considered to hold silence and can be discarded. This method has proven to be simple and effective.

Silence suppression does have a minor side effect. Because the ‘silent’ packets are discarded there is absolutely no sound at all at the receiving side, not even background noise. This is truly a deadly silence and it might even seem that the connection has gone. A solution is to artificially introduce some background noise at the receiver side.

9 In my own applications, the amount of jitter is calculated continuously and the amount of buffering is adapted accordingly. In this case the buffering delay could be considered as part of the variable delay rather than the fixed delay. For the sake of simplicity however, I will just consider it as part of the fixed delay.

## 3.5 Summary

Several aspects of voice communication have an important meaning for Voice over IP. To make VoIP possible, we must be able to digitise and reconstruct voice information. A sampling rate of 8000 Hz, using eight bit samples is sufficient to provide telephone quality communication and needs a bandwidth of 64 kbps.

Humans tend to be very tolerant to corrupted or lost voice packets. However, they are far less tolerant to delay and jitter. Jitter should be avoided through the use of buffering as it is disastrous for the quality of the communication. Delay should be kept below 200 ms to maintain a telephone quality conversation.

From the point of view of the communication, small packets are to be preferred since this way, a lost or corrupted packet causes a smaller interruption in the communication. From the delay's point of view, a small sampling interval is important since that interval directly contributes to the overall delay.

Finally, in practice there is usually only one person talking at the same time in a discussion. This means that a lot of bandwidth can be saved by not sending any packets containing only silence. This is called silence suppression.

# Chapter 4: Compression techniques

In the previous chapter we learned that for one–way voice communication, a bandwidth of 64 kbps is sufficient. For a LAN it is not a problem to achieve this rate, but for dial–up links at this time that rate is not possible. Even a LAN may get quite heavily loaded when VoIP is used in virtual environments, since there could be incoming packets from a large amount of senders at the same time. If we are considering a WAN, it is also not always possible to attain that rate since one slower link is enough to prevent it.

Clearly there is a need for compression and luckily voice information offers the possibility of large compression ratios. In this chapter we will take a look at several compression techniques and explain their importance for VoIP. We will start with some general methods and then we will take a look at some waveform coding and vocoding algorithms. The chapter ends with a discussion about some compression standards for Voice over IP.

## 4.1 Preliminaries

We will see in the next chapter that with VoIP usually unreliable protocols are used. It was already mentioned that Voice over IP can tolerate corrupted or lost packets very well, so an unreliable protocol normally does not cause a bad communication quality. Furthermore, reliable protocols rely on the retransmission of packets and these retransmissions add to the overall delay.

An important consequence of the use of unreliable protocols is that compression techniques can only rely on the data in the packet that is to be compressed. Compression algorithms which need information from previous packets cannot be used since it is possible that one of those previous packets did not reach the destination. This would either make decompression impossible or would reconstruct an invalid packet.

One could argue that because of the tolerability of voice communication to errors, this is not a problem. However, with such methods it would mean that one lost or corrupted packet can create several lost packets: all those which relied on the lost packet for compression. So unless the transmission path is highly reliable it is probably better to avoid such compression schemes.

The quality of voice compression is usually measured in terms of Mean Opinion Score (MOS). This is a value between one and five which expresses how close the voice quality is to real–life communication. Figure 4.1 shows this scale.

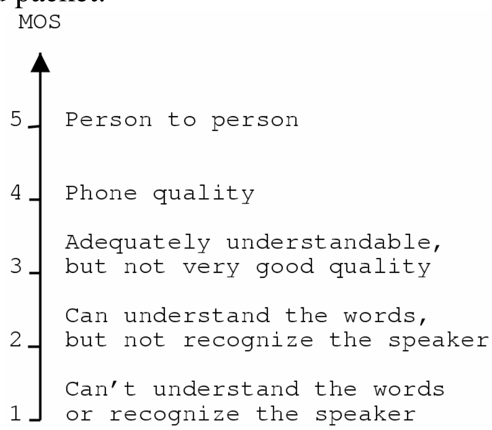


Figure 4.1 – Mean Opinion Score (MOS) [23]

## 4.2 General compression techniques

In this section I will discuss some widely used compression techniques: Lempel–Ziv compression and Huffman coding. These are both lossless compression techniques which means that when data is decompressed, the original data will be restored, without any modifications. This implies that these techniques can be used for both data in general and voice information.

When these methods are directly applied to voice data they do not offer very good compression ratios and for this reason compression is almost never done this way. However, the algorithms can be used as a post–processing step for other compression methods to further enhance the compression ratio, so it is useful to explain them.

Note that several other lossless compression techniques exist. The reason that I will discuss Lempel–Ziv and Huffman compression, is that I have used these schemes in the applications I developed.

### 4.2.1 Lempel-Ziv compression

There are several Lempel–Ziv (LZ) compression algorithms, but they all work more or less according to the same principles. The algorithms in the LZ family all try to substitute a series of characters by a fixed length code. Two important members of this family are LZ77 and LZ78. An overview of these and other members of the LZ family can be found in [35].

Since I have used the LZ78 algorithm myself, this is the one I will explain. It gives a good idea about the way the LZ algorithms work. The LZ78 algorithm makes use of a 'dictionary'. This is a data structure which can hold a number of strings and their length. This dictionary is built both when coding and when decoding. At the start of the algorithm, the dictionary is empty.

The compression algorithm will output doubles <i,c> with 'i' being an index into the dictionary and 'c' being the next character. At the current position in the data which has to be compressed, the algorithm searches through the dictionary for a match. Suppose a match with length L is found. The current position is then incremented by L. Next, the algorithm outputs a double <i,c> with 'i' being the index of the found entry and 'c' being the character at the current position. Also, an entry is added to the dictionary. This entry is the concatenation of the found string and the current character. Then the current position is incremented by one and the algorithm again starts searching in the dictionary. The routine is repeated until there is no more data to compress.

If no match is found when searching through the dictionary, the algorithm outputs a double <0,c>. The zero indicates that there was no match and 'c' is the character at the current position. The entry which is added to the dictionary contains only the current character. The current position is then incremented by one and the algorithm starts searching the dictionary again.

The decompression algorithm reads doubles <i,c>. If the index 'i' is zero, the character 'c' is added to the decompression output and an entry containing 'c' is added to the dictionary. If 'i' is not zero, the concatenation of the dictionary entry at position 'i' and the character 'c' is made. This new string is added to both the decompression output and the dictionary.

### 4.2.2 Huffman coding

For a given set of possible values and the frequency with which each value occurs, the Huffman algorithm determines a way to encode each value binary. More important, it does this in such a way that an optimal encoding is created. In this section, I will only present the algorithm itself. Proof that it indeed constructs an optimal encoding can be found in [4], which is also the source of the information in this section.

The Huffman algorithm produces a binary string for each value that is to be encoded and these strings can have an arbitrary length. However, the decoding process has to know when a certain binary string has to be replaced by the appropriate value. To be able to do this, the strings must have the following property: none of the binary strings can be a prefix to another string.

Binary strings with this prefix property can be represented by a binary tree in which the branches themselves contain the labels zero and one. The strings which are created by traversing the tree from the root to the leafs are all strings with the prefix property. This is illustrated in figure 4.2.

The Huffman algorithm constructs such a tree in which the leafs are marked with the values that need to be encoded. To find out by which binary code a value has to be replaced, you only need to follow the path from the root to the leaf containing that value.

The decompression routine is also easy. The algorithm sets the current position at the root node and starts reading bits. For each bit the appropriate branch is followed. At intermediate nodes the same thing is done, but when a leaf has been reached, the value at that leaf can be output and the current position is reset to the root node. The algorithm then repeats itself.

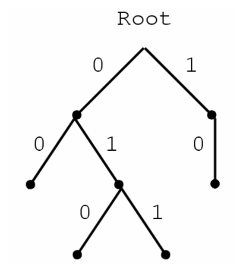

Figure 4.2 – Binary tree example

As you can see, once the tree has been constructed, the algorithm itself is fairly easy. To construct the tree, the algorithm starts with a number of separate nodes, one for each value that needs to be encoded. With each node, the frequency of occurrence of the corresponding value is also associated. For example, if we have got a file in which only the five characters a,b,c,d and e occur with certain frequencies, figure 4.3(a) illustrates a possible situation.

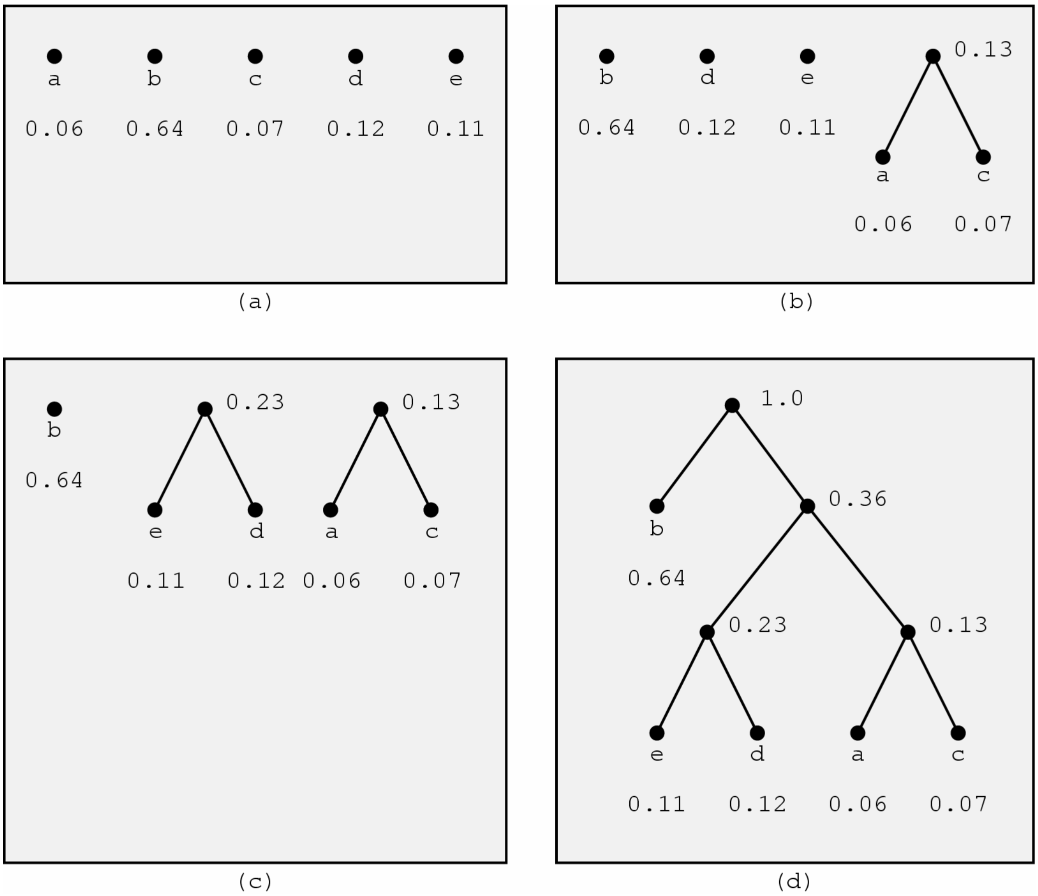

Figure 4.3 – Huffman tree construction

Next, the algorithm looks for the two nodes with the smallest associated frequencies. These two nodes are removed from the list of nodes. A new node is then added to the list and the two removed nodes are its children. The new node's associated frequency is the sum of the frequencies of its children. In the example, the result of this step is depicted in figure 4.3(b).

In this new list of nodes, the algorithm starts it search again for the two nodes with the smallest associated frequencies and the previous step is repeated. In the example, this leads to the situation in figure 4.3(c). When there is only one node left in the list of nodes, this node is the root node of the tree and the algorithm stops. In the example, this leads to the tree in figure 4.3(d).

## 4.3 Waveform coding

Waveform coding tries to encode the waveform itself in an efficient way. The signal is stored in such a way that upon decoding, the resulting signal will have the same general shape as the original. Waveform coding techniques apply to audio signals in general and not just to speech as they try to encode every aspect of the signal.

The simplest form of waveform coding is PCM encoding the signal. But a signal can be processed further to reduce the amount of storage needed for the waveform. In general, such techniques are lossy: the decoded data can differ from the original data. Waveform coding techniques usually offer good quality speech requiring a bandwidth of 16 kbps or more.

### *4.3.1 Differential coding*

Differential coding tries to exploit the fact that with audio signals the value of one sample can be somewhat predicted by the values of the previous samples. Given a number of samples, the algorithms in this section will calculate a prediction of the next sampled value. They will then only store the difference between this predicted value and the actual value. This difference is usually not very large and can therefore be stored with fewer bits than the actual sampled value, resulting in compression. Because of the use of a predicted value, differential coding is also referred to as predictive coding.

#### 4.3.1.1 Differential PCM (DPCM)

Differential PCM merely calculates the difference between the predicted and actual values of a PCM signal and uses a fixed number of bits to store this difference. The number of bits used to store this difference determines the maximum slope that the signal can have if errors are to be avoided. If this slope is exceeded, the value of a sample can only be approximated, introducing an amount of error.

In the applications I developed, I have tested a DPCM compression scheme. It used uniformly quantised PCM data as input and produced DPCM output. The predicted value I used was simply the value of the previous sample. Personally, I found that using five bits to store the difference still produced very good speech quality upon decompression. Even with only four bits, the results were quite acceptable.

#### 4.3.1.2 Adaptive DPCM (ADPCM)

An extension to DPCM is adaptive DPCM. With this encoding method, there are still a fixed number of bits used to store the difference. In contrast to the previous technique which simply used all of those bits to store the difference, ADPCM uses some of the bits to encode a quantisation level. This way, the resolution of the difference can be adjusted. [26]

#### 4.3.1.3 Delta modulation (DM)

Delta modulation can be seen as a very simple form of DPCM. With this method, only one bit is used to encode the difference. One value then indicates an increase of the predicted value with a certain amount, the other indicates a decrease.

A variant of this scheme is called adaptive delta modulation (ADM). Here, the step size used to increase or decrease the predicted value can be adapted. This way, the original signal can be approximated more closely. [10]

### 4.3.2 Vector quantisation

With vector quantisation, the input is divided into equally sized pieces which are called vectors. Essential to this type of encoding is the presence of a 'codebook', an array of vectors. For each vector of the input, the closest match to a vector in the codebook is looked up. The index of this codebook entry is then used to encode the input vector. [37]

It is important to note that this principle can be applied to a wide variety of data, not only to PCM data. For example, vector quantisation could be used to store an approximation of the error term of other compression techniques.

### 4.3.3 Transform coding

When we are considering PCM data, we are in fact looking at a signal in the time domain. With transform coding, the signal is transformed to its representation in another domain in which it can be compressed better than in its original form. When the signal is decompressed, an inverse transformation is applied to restore an approximation of the original signal. [37]

One of the domains to which a signal could be transformed is the frequency domain. Using information about human vocal and auditory systems, a compression algorithm can decide which frequency components are most important. Those components can then be encoded with more precision than others. Examples of transformation schemes which are used for this purpose are the Discrete[10] Fourier Transform (DFT) and the Discrete Cosine Transform (DCT).

Personally, I have experimented with transform coding using a wavelet transformation. A wavelet decomposition can be used to write a signal as a linear combination of certain wavelet basis functions. The coefficients used in the linear combination then form the wavelet representation of the signal. Using these coefficients for a certain wavelet basis, the original signal can be reconstructed.

A complete explanation of wavelets falls beyond the scope of this thesis. However, the theory of wavelets is very interesting and if you would like more information about it, a good introduction can be found in [44]. This reference is also the source on which I based the implementation of my compression scheme.

The wavelets which I used to transform the signal are called Haar wavelets. They form a very simple wavelet basis. Using these wavelets, decomposition and reconstruction can be done quite fast. They also possess a property called orthonormality which allows us to determine very easily which components of the transformed signal are most important.

In the scheme that I have tried, a uniformly quantised PCM signal was decomposed into its wavelet representation. Next, a number of coefficients with little importance were set zero and the other coefficients were quantised. The resulting sequence of coefficients were then run–length encoded[11] and finally, the data was compressed using Huffman coding.

For this last step, I first tried LZ78 compression, but this resulted in little extra compression. Sometimes the resulting code was even larger than before. With Huffman coding, the extra compression was much better. Typically, the code was compressed to sixty to eighty percent of its original size.

Unfortunately, the results are not very spectacular. If good speech quality is to be

10 The term 'discrete' is used because the transformation operates on a signal which has already been digitised and not on a continuous signal.

11 This is a compression technique which eliminates sequences of the same value.

preserved, this scheme cannot achieve good compression ratios. In fact, I have had better compression results using simple DPCM compression. I believe that this is probably due to the type of wavelets I used.

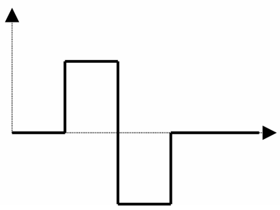

Figure 4.4 – A typical Haar wavelet

A typical Haar wavelet is shown in figure 4.4. As you can see, such functions are discontinuous at certain points, which is probably not a desirable property since speech is a relatively slow varying signal. Perhaps other types of wavelets which are more smooth would be more appropriate for speech compression.

## 4.4 Vocoding

Waveform coding methods simply try to model the waveform as closely as possible. But we can exploit the fact that we are using speech information to greatly reduce the required storage space. Vocoding techniques do this by encoding information about how the speech signal was produced by the human vocal system, rather than encoding the waveform itself.

The term vocoding is a combination of 'voice' and 'coding'. These techniques can produce intelligible communication at very low bit rates, usually below 4.8 kbps. However, the reproduced speech signal often sounds quite synthetic and the speaker is often not recognisable.

### *4.4.1 Speech production*

To be able to understand how vocoding methods work, a brief explanation of speech production is required.

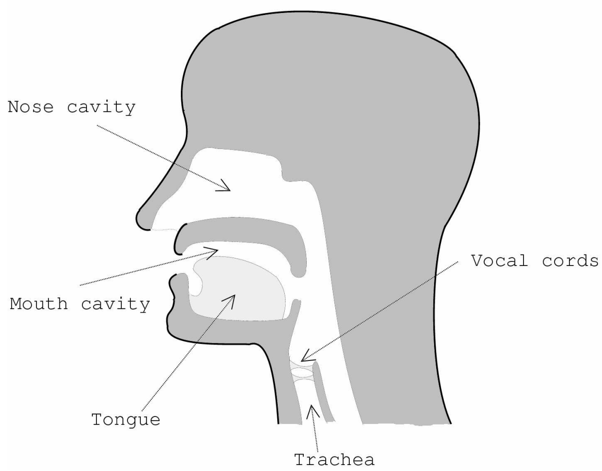


Figure 4.5 – The human vocal system (based upon [27] and [15])

Figure 4.5 shows the human vocal system. To produce speech, the lungs pump air through the trachea. For some sounds, this stream of air is periodically interrupted by the vocal cords.

The resulting air flow travels through the so–called vocal tract. The vocal tract extends from the opening in the vocal cords to the mouth. A part of the stream travels through the nose cavity.

The vocal tract has certain resonance characteristics. These characteristics can be altered by varying the shape of the vocal tract, for example by moving the position of the tongue. These resonance characteristics transform the flow of air originating from the vocal cords to create a specific sound. The resonance frequencies are called formants.

Basically, there are three classes of speech sounds that can be produced. Other sounds belong to a mixture of the classes. These are the classes: [33]

- **Voiced sounds** are created when the vocal cords vibrate open and closed. This way, periodic pulses of air come out of the opening of the vocal cords. The rate at which the opening and closing occurs, determines the pitch of the sound.
- To produce **unvoiced sounds**, the vocal cords do not vibrate, they are held open. Air is then sent at high velocities through a constriction in the vocal tract, creating a noise–like turbulence.
- **Plosive sounds** result from building up air pressure behind a closure in the vocal tract and then suddenly releasing this air.

An important fact is that the shape of the vocal tract and the type of excitation (the flow of air coming out of the vocal cords) change relatively slowly. This means that for short time intervals, for example 20 ms, the speech production system can be considered to be almost stationary. Another important observation is that speech signals show a high degree of predictability. Sometimes due to the periodic signal created by the vocal cords and also due to the resonance characteristics of the vocal tract. [33]

### *4.4.2 Vocoding basics*

Instead of trying to encode the waveform itself, vocoding techniques try to determine parameters about how the speech signal was created and use these parameters to encode the signal. To reconstruct the signal, these parameters are fed into a model of the vocal system which outputs a speech signal.

Since the vocal tract and excitation signal change only relatively slowly, the signal that has to be analysed is split into several short pieces. Also, to make analysis somewhat easier, the assumption is made that a sound is either voiced or unvoiced.

A piece of the signal is then examined. If the signal is voiced, the pitch period is determined and accordingly the excitation signal is modelled as a series of periodic pulses. If the speech signal is unvoiced, the excitation will be modelled as noise.

Like we saw in the previous section, the vocal tract has certain resonance characteristics which alter the excitation signal. In vocoders the effect of the vocal tract is recreated through the use of a linear filter.

Perhaps it is not entirely clear what a linear filter is. A filter is any system that takes a signal $f(x)$ as its input and produces a signal $g(x)$ as its output. The output of a filter is also referred to as the response of the filter to a certain input signal. The filter is called a linear filter when scaling and superposition at the input results in scaling and superposition at the output. [11]

A vocoding method will use a specific type of linear filter. The filter will contain certain parameters which have to be determined by the vocoder. This is so because the characteristics of the vocal tract change over time and the coder has to be able to model each state of the vocal tract approximately. Remember that the state of the vocal tract changes only relatively slowly, so for each piece of the input signal, the vocal tract can be considered to have fixed characteristics.

Due to this simple speech production model, speech can be encoded in a very compact way. On the other hand, this simple model is also the cause of the unnatural sounding speech which vocoders often produce.

Several types of vocoders exist, the oldest one being around since even 1939. They all use this simple representation of the speech production system. The main difference between the methods is the vocal tract model used. Below, I will only give a description of the Linear Predictive Coder (LPC) since this vocoder is often discussed in literature about VoIP.

### *4.4.3 Linear Predictive Coding (LPC)*

The LPC coder uses the simple model described above. The excitation signal is considered either to be a periodic signal for voiced speech, or noise for unvoiced speech.

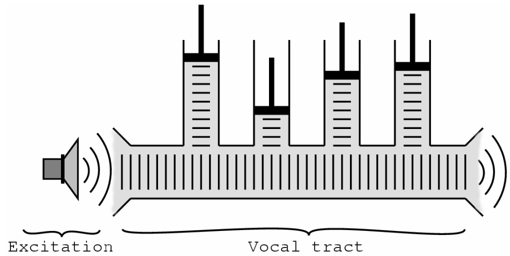

Figure 4.6 – LPC vocal tract model [25]

The vocal tract model which the LPC method uses, is an approximation of a series of concatenated acoustic tubes [10], as figure 4.6 illustrates.

The LPC vocoder examines its input and estimates the parameters to use in the vocal tract filter. It then applies the inverse of this filter to the signal. The result of this is called the residue or residual signal and it basically describes which excitation signal should be used to model the speech signal as closely as possible. From this residual signal, it is relatively easy to determine if the signal is voiced or unvoiced and if necessary, to determine the pitch period.

To determine the parameters for the filter, the LPC algorithm basically determines the formants of the signal. This problem is solved through a difference equation which describes each sample as being a linear combination of the previous ones. Such an equation is called a linear predictor, hence the name of the coder. [24]

The LPC method can produce intelligible speech at 2.4 kbps. The speech does sound quite synthetic however, like with most vocoding techniques.

## 4.5 Hybrid coding

Waveform coders in general do not perform well at data rates below 16 kbps. Vocoders on the other hand, can produce very low data rates while still allowing intelligible speech. However, the person producing the speech signal often cannot be recognised and the algorithms usually have problems with background noise.

Hybrid coders try to exploit the advantages of both techniques: they encode speech in such a way that results in a low data rate while keeping the speech intelligible and the speaker recognisable. Typical bandwidth requirements lie between 4.8 and 16 kbps.

The hybrid coders that will be discussed in this section are RELP, CELP, MPE and RPE coders. Here, only a brief description is given. A more detailed one can be found in [16], which is the main source for the information in this section.

The basic problem with vocoders is their simplistic representation of the excitation signal: the signal is considered to be either voiced or unvoiced. It is this representation that causes the synthetic sound of these coders. The coders discussed below try to improve the representation of the excitation signal, each in their own way.

### *4.5.1 Residual Excited Linear Prediction (RELP)*

The RELP coder works in almost the same way as the LPC coder. To analyse the signal, the parameters for the vocal tract filter are determined and the inverse of the resulting filter is applied to the signal. This gives us the residual signal.

The LPC coder then checked if the signal was voiced or unvoiced and used this to model an excitation signal. In the RELP coder however, the residual is not analysed any further, but will be used directly as the excitation for speech synthesis. The residual is compressed using waveform coding techniques to lower the bandwidth requirements. RELP coders can allow good speech quality at bit rates in the region of 9.6 kbps.

### *4.5.2 Codebook Excited Linear Prediction (CELP)*

The CELP coder tries to overcome the synthetic sound of vocoders by allowing a wide variety of excitation signals, which are all captured in the CELP codebook. To determine which excitation signal to use, the coder performs an exhaustive search. For each entry in the codebook, the resulting speech signal is synthesised and the entry which created the smallest error is then chosen. The excitation signal is then encoded by the index of the corresponding entry. So basically, the coder uses Vector Quantisation to encode the excitation signal.

This technique is called an analysis–by–synthesis (AbS) technique because it analyses a signal by synthesising several possibilities and choosing the one which caused the least amount of error.

This exhaustive search is computationally very expensive. However, fast algorithms have been developed to be able to perform the search in real–time. CELP techniques allow bit rates of even 4.8 kbps.

### 4.5.3 Multipulse and Regular Pulse Excited coding (MPE and RPE)

Like the previous method, MPE and RPE techniques try to improve the speech quality by giving a better representation of the excitation signal. With MPE, the excitation signal is modelled as a series of pulses, each with its own amplitude. The positions and amplitudes of the pulses are determined by an AbS procedure. The MPE method can produce high quality speech at rates around 9.6 kbps.

The RPE technique works in a similar fashion, only here the pulses are regularly spaced, as the name suggests. The GSM mobile telephone system uses a RPE variant which operates at approximately 13 kbps.

## 4.6 Other compression techniques

The compression principles discussed above cover pretty much the whole speech compression domain. Due to this fact I was unable to find much information about compression techniques which do not fall into the categories of either waveform coding, vocoding or hybrid coding.

But there is one technique which I find worth mentioning here, namely the use of artificial neural networks for speech compression. At this moment, there is not much information to be found about this particular use of neural networks, but there are documents which describe how neural networks can be used for lossy image compression. It is possible that similar techniques can be used for the compression of speech.

To do this, there are several ways in which artificial neural networks can be used. A neural net could be trained to predict the next sample, give a number of previous samples. This way, the network could perform the predictive function in differential coding schemes. If this prediction is done more accurately than regular predictive techniques this would result in better compression.

Another possible application is to use the neural network in a similar way as a vector quantiser. The network could be trained to map a number of inputs to a specific output. Then, either using a table lookup or another neural network, this number could be used to retrieve an appropriate waveform.

Perhaps a neural network could also be used to perform a speech analysis function which in turn could be used together with some vocoding or hybrid coding technique.

I realise that there is a lot of speculation in this section and unfortunately I did not have the time to conduct experiments using these techniques. However, I strongly believe that neural networks have great potential in a wide variety of applications, including speech compression.

## 4.7 Delay by compression

Like we saw in the previous chapter, to be able to preserve good communication quality, the overall delay has to be kept as low as possible. This means that we have to take the delay caused by compression and decompression into account: even if we are able to compress the signal in an excellent way, it has little use for real–time communication if it introduces an unacceptable amount of delay.

Delays during the compression stage can generally be divided into two categories. First of all, there is always some delay due to the calculations which need to be done. This amount of delay depends much on the capabilities of the system performing the compression.

Some compression techniques introduce a second type of delay: to compress a part of the speech signal, they need a portion of the signal which follows the part being handled. The amount of 'lookahead' needed determines the amount of delay introduced. For a specific algorithm this delay is fixed and does not vary among systems.

Decompressing the signal can usually be done much faster than compressing it. Of the compression schemes discussed in this chapter, transform coding probably introduces the most delay during decompression since, like during the compression stage, the signal has to undergo a transformation.

With computers becoming ever faster and specialised hardware becoming available, the fixed delay during the compression stage is probably the most important to consider.

## 4.8 Voice compression standards

To make interoperability between applications possible, it is important that standards are established. The most widely known standards in the VoIP domain, are the G. standards of the ITU−T[12]. Other well known standards are the ETSI[13] GSM standards. Here is a list of some standards: [12]

| Standard | Description | Bit rate | MOS |
|---|---|---|---|
| G.711 | Pulse Code Modulation using eight bits per sample, sampling at 8000 Hz | 64 kbps | 4.3 |
| G.723.1 | Dual rate speech coder designed with low bit rate video telephony in mind [41]. The G.723.1 coder needs a 7.5 ms lookahead and used one of these coding schemes: | | |
| | − Multipulse Maximum Likelihood Quantisation (MP−MLQ) | 6.3 kbps | 4.1 |
| | − Algebraic CELP (ACELP) | 5.3 kbps | 4.1 |
| G.726 | Coder using ADPCM. Contains obsolete standards G.721 and G.723 | 16,24,32 and 40 kbps | 2 – 4.3 |
| G.727 | Five, four, three and two bits per sample embedded ADPCM. The encoding allows bit reductions at any point in the network without the need for coordination between sender and receiver [10]. | 16,24,32 and 40 kbps | 2 – 4.3 |
| G.728 | Low Delay CELP (LD−CELP) | 16 kbps | 4.1 |
| G.729 | Conjugate Structure ACELP (CS−ACELP) | 8 kbps | 4.1 |
| | Annex A: Reduced complexity algorithm | 8 kbps | 3.7 |
| | Annex D: Low rate extension | 6.4 kbps | |
| | Annex E: High rate extension | 11.8 kbps | |
| | These coders need a 5 ms lookahead. | | |

12 The ITU is the International Telecommunication Union. The ITU−T is the sector which focusses its efforts on telecommunication standards.

13 European Telecommunications Standards Institute

| | | | |
|---|---|---|---|
| GSM 06.10 | Full rate speech transcoding using Regular Pulse Excitation–Long Term Prediction (RPE–LTP) | 13 kbps | 3.71 |
| GSM 06.20 | Half rate speech transcoding using Vector Sum Excited Linear Prediction (VSELP) | 5.6 kbps | 3.85 |
| GSM 06.60 | Enhanced full rate speech transcoding using ACELP | 12.2 kbps | 4.43 |

Some remarks have to be made at this point. First of all, unfortunately I was not able to find MOS information about some coders. Second, the Mean Opinion Scores are rather subjective and it is probably due to this fact that the MOS values often differ according to different sources. Sometimes these differences are even quite large. For example, in [40] it was mentioned that G.729 annex A had a MOS of 3.4 while [32] claimed that it was 4.0. In this particular case I chose to make a compromise and took the value of 3.7 from [43].

## 4.9 Summary

For telephone quality communication using digitised speech, a bandwidth of 64 kbps is needed if the speech data is left uncompressed. But speech data can often be greatly compressed and this can drastically reduce the amount of required bandwidth.

Some compression schemes do not take the nature of the data into account. Such techniques offer some compression, but usually they do not result in high compression ratios. However, they can be used to further reduce the amount of storage needed when another compression technique has already compressed the voice information.

Waveform coding techniques assume that the data is an audio signal, but in general they do not exploit the fact that the signal contains only speech data. They just try to model the waveform as closely as possible. This results in good speech quality at relatively high data rates (16 kbps or above).

Vocoders do exploit the fact that the data is in fact digitised speech. They do not encode the waveform itself, but an approximation of how it was produced by the human vocal system. Such techniques allow very high compression ratios while still providing intelligible communication (at rates of 4.8 kbps or below). However, the reproduced speech usually sounds quite synthetic.

A combination of waveform coding and vocoding techniques is used in hybrid coding schemes. They still rely on a speech production model, but they are able to reproduce the original signal much more closely through the application of waveform coding techniques. These methods can give good speech quality at medium data rates (between 4.8 and 16 kbps).

Compressing and decompressing speech data introduces a certain amount of delay into the communication. Because computers are becoming ever faster and because specialised hardware is becoming available, the amount of lookahead that a compression scheme requires is probably the most important delay component.

To be able to provide interoperability between different applications, it is important that standards are established. Well known compression standards in the VoIP world include the ITU–T's G. series standards and the ETSI's GSM standards.

# Chapter 5: Transmission of voice signals

We now know how to send voice information without wasting a lot of bandwidth so we can move on to issues relating to the actual transmission of the speech data. The Internet Protocol only offers a best–effort service without any QoS guarantees. For decent voice communication it is, however, necessary to have certain guarantees since too much delay or too many lost packets will seriously affect the quality of the conversation.

This chapter discusses how we can transmit voice information while preserving the communication quality. First we will talk about some general requirements. Next, we will see what protocols can be used to transmit the speech data. Also, some resource reservation methods will be discussed since this can help to improve the transmission quality. Finally, we will discuss the transmission delay.

## 5.1 Requirements

When transmitting packets containing voice data, there must be some mechanism to preserve synchronisation within the speech signal. The consecutive packets should be played at the right time, in the right order. This type of synchronisation is called intra–media synchronisation.

We have seen in chapter three that for real–time voice communication, the overall delay has to be kept as low as possible. Since an IP network in general only offers a best–effort service, there is no guarantee that the delay will meet the requirements. Similarly, the amount of lost packets can be quite high, for example during periods of congestion. To be able to deliver telephone quality speech, there will have to be some quality of service (QoS) mechanism which offers guarantees about these things.

The speech data which has to be sent is typically generated at regular small intervals. It is possible that a receiving end cannot cope with this data flow, so somehow the sender should know whether the receiver can handle the incoming stream or not. A method that does this is often called a flow control method.

Also, due to the fact that data is sent at a regular basis, it is not unlikely that a link becomes overloaded and congestion occurs. In turn, congestion causes the loss of packets and an increase in delay which are not desirable features for voice communication. The transmission component should be able to detect an arising congestion and take appropriate actions. The mechanism to prevent and control congestions is called congestion control.

The appropriate action for flow and congestion control is to decrease the amount of data sent. Typically, this is done in cooperation with the compression module: when the data rate has to be lowered, the compression module is signalled to increase the amount of compression. This will usually result in a degradation of speech quality, but it is still better than having a lot of lost packets and a large delay.

## 5.2 Transmission protocols

If an application wants to transmit data, it uses a certain protocol to do this. Recall that in the TCP/IP architecture, TCP and UDP are the protocols which an application can use.

First, we will see why the bare TCP and UDP services are not sufficient. Then, a description of the Real–time Transport Protocol (RTP) is given. This is a widely used protocol for real–time data like speech and video.

### 5.2.1 Why not TCP or UDP?

When we are thinking about VoIP applications which should offer a telephone–like service, TCP could seem a good candidate to transmit the speech data. It offers a service in which the connection can be seen as a reliable byte stream. To use TCP, a connection is set up, data is exchanged and the connection is torn down again. This procedure immediately reminds one of the way a telephone call is made.

When it comes to synchronisation, the reliable byte stream service seems like a very good starting point: all data arrives nicely in the exact same order as it was transmitted. Also, data is guaranteed to arrive correctly, which is also good for voice communication. The protocol also has built–in flow and congestion control mechanisms which offer good protection against overloading the network.

There are however, several disadvantages to the use of TCP. One of the basic problems is that to offer this reliable byte stream service, the protocol relies heavily on the retransmission of lost or corrupted packets. While this offers a reliable service in which order is preserved, the waiting for retransmitted packets adds extra delay to the communication. Usually, it is better to have an occasional lost or corrupted packet than having a large amount of delay.

A related issue is that one lost or corrupted packet effectively prohibits the application of receiving any packets which come after it, since TCP preserves the order of the packets. The application has to output speech data at regular intervals, so if one packet stays lost for a sufficient amount of time, this will block the playback of other packets, even when they have already arrived.

The flow and congestion control features might seem very useful, but an application has very little control over these things. TCP can easily decide by itself to decrease the rate at which data is sent, and this again would increase the overall delay.

The key point to be made here is that TCP has a lot of features and a lot of complexity which are not very useful for VoIP. We could easily get an equal or better performance using far less elaborate protocols.

When speech data has to be distributed to several users at the same time, TCP has another major disadvantage. While IP offers the efficient distribution of data using multicasting, TCP has no support for this. If data has to distributed to several destinations using TCP, it has to be done using separate TCP connections. This, of course, wastes a lot of bandwidth.

When we eliminate TCP, the only basic protocol we can use is UDP. This protocol has almost no complexity at all. It is simply a minor extension to IP, so it offers only a best–effort service.

The protocol has the advantage of not having to wait for retransmissions of lost packets. Also, since it is only a small extension to IP, it can make use of the IP multicasting features and save bandwidth when data has to be sent to multiple destinations. As good as all this may seem, there are also some disadvantages: UDP provides no mechanism for synchronisation whatsoever and there are no means for flow or congestion control.

A solution to these problems is to extend UDP somewhat: we can add extra information to the speech data and use UDP to distribute this control and speech information. This is in fact how the Real–time Transport Protocol (RTP) works in the TCP/IP architecture.

### 5.2.2 Real-time Transport Protocol (RTP)

The Real–time Transport Protocol is formally specified in [30]. There, it is defined as a protocol which provides end–to–end delivery services for data with real–time characteristics, such as interactive audio and video. So this protocol can also be used for VoIP applications.

The RTP specification actually defines two separate protocols. The first one is the Real–time Transport Protocol (RTP). The second one is called the RTP Control Protocol (RTCP). The function of RTP is to transfer the real–time data. The control protocol supplies information about the participants in the session. The protocols are defined in such a way that they can be used on a lot of network architectures and not just on TCP/IP networks. However, if RTP is used on a TCP/IP network, it is typically run on top of UDP.

The protocols themselves do not provide mechanisms to ensure timely delivery. They also do not give any QoS guarantees. These things have to be provided by some other mechanism.

Also, out of order delivery is still possible, and flow and congestion control are not directly supported. However, the protocols do deliver the necessary data to the application to make sure it can put the received packets in the correct order. Also, RTCP provides information about reception quality which the application can use to make local adjustments. For example if a congestion is forming, the application could decide to lower the data rate.

In the following sections, an introduction to RTP and RTCP is given. These sections are based on the specifications in [30], which is also the appropriate source to consult for more detailed information about these items.

Personally, I have used RTP in the VoIP applications which I developed. For this purpose I first wrote a RTP library, which is described in chapter eight.

#### 5.2.2.1 RTP

A RTP packet consists of a RTP header, followed by the data to send. In the RTP specification this data is referred to as the payload. The header is transmitted in network byte order, just like the IP header. Figure 5.1 below shows the RTP header format.

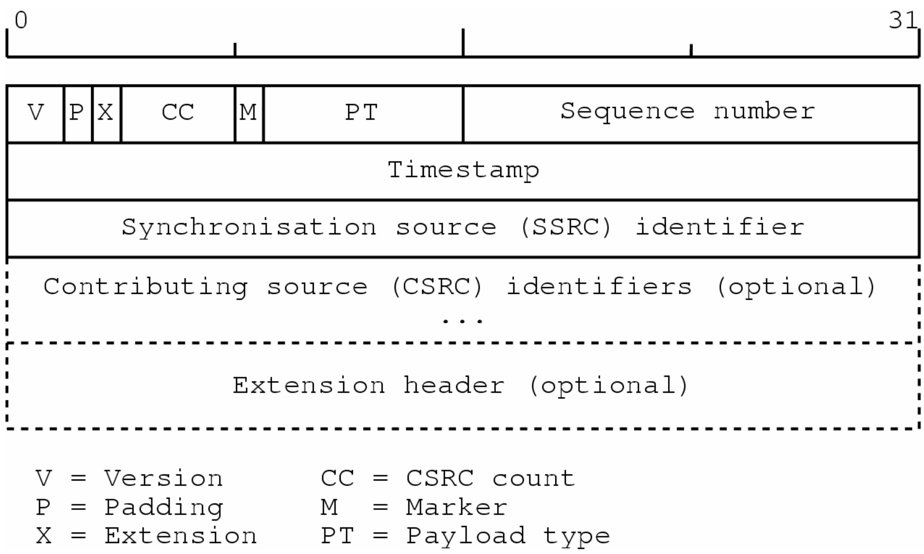


Figure 5.1 – The RTP header [30]

The first two bits of the header contain the **version** number. The current version of the protocol is ‘two’. Next, there is the **padding** bit. If this bit is set, the packet contains some padding bytes which are not part of the payload. The last padding byte then contains the number of padding bytes. For example, padding may be necessary for some encryption algorithms which need the payload to be aligned on a multiple byte boundary. The **extension** bit specifies if the header contains an extension header. Then, there is the **CSRC count** which specifies how many contributing sources are specified in the RTP header.

The **marker** bit can be used by an application to indicate a talkspurt for example. The exact interpretation is not defined in the RTP specification, it is left to the application itself. Next, there is the **payload type**. This defines the type of data the packet contains, so it defines the way in which the application will interpret the payload.

The **sequence number** can be used by an application to place received packets in the correct order. The numbering starts at a random value for security reasons.

The **timestamp** contains the synchronisation information for a stream of packets. This value specifies when the first byte of the payload was sampled. For example, for audio, the timestamp is typically incremented with the amount of samples in the packet. Based on this value, the receiving application can then play the audio data at exactly the right time. Just like with the sequence number, the initial value of the timestamp is random. Note that several packets can have the same timestamp value: with digitised video for example, one image will usually have to be sent in several pieces. These pieces will all have a different sequence number, but their timestamp value will be the same.

The **synchronisation source (SSRC) identifier** is the identification number of the sender of the packet. If an application wishes to send different media at the same time, for example audio and video, there have to be separate RTP sessions for each of the media. This way, an application can group the incoming data according to the SSRC value. The identifier is chosen randomly; the chance that two communicating parties accidentally end up having the same SSRC value is extremely small. In the rare case that this should happen, the specification gives the appropriate course of actions to resolve this problem.

Next, there are possibly a number of **contributing source (CSRC) identifiers**. For example, if at some point different audio streams have to be mixed together, the original SSRC identifiers can be put here. The SSRC identifier of this packet then becomes the identifier of the source which forwards the mixed packet.

Finally, the header can contain extra information through the use of an **extension header**. The RTP specification only defines the extension mechanism, not the possible extensions. This is left to the application.

Note that the header does not contain a payload length field. The protocol relies on the underlying protocol to determine the end of the payload. For example, in the TCP/IP architecture, RTP is used on top of UDP, which does contain length information. Using this, an application can determine the size of the whole RTP packet and after its header has been processed, it automatically knows the amount of data in its payload section.

#### 5.2.2.2 RTCP

The RTP protocol is accompanied by a control protocol, RTCP. Each participant of a RTP session periodically sends RTCP packets to all other participants in the session. According to [30], RTCP has four functions:

- The primary function is to provide feedback on the quality of data distribution. Such information can be used by the application to perform flow and congestion control functions. The information can also be used for diagnostic purposes.
- RTCP distributes an identifier which can be used to group different streams – audio and video for example – together. Such a mechanism is necessary since RTP itself does not provide this information.

- By periodically sending RTCP packets, each session can observe the number of participants. The RTP data cannot be used for this since it is possible that somebody does not send any data, but does receive data from other participants. For example, this is the case in an on–line lecture.
- An optional function is the distribution of information about a participant. This information could be used in a user–interface for example.

There are several types of RTCP packets which are used to supply this functionality. Sender reports (SR) are used by active senders to distribute transmission and reception statistics. If a participant is not an active sender, it still distributes reception statistics by sending receiver reports (RR). Information which describes a participant is transmitted in the form of source description (SDES) items. There is also a packet type to allow application specific data (APP). Finally, when a participant is about to leave the session, it sends a goodbye (BYE) packet.

The transmission statistics which an active sender distributes, include both the number of bytes sent and the number of packets sent. It also includes two timestamps: a Network Time Protocol (NTP) timestamp, which gives the time when this report was created, and a RTP timestamp, which describes the same time, but in the same units and with the same random offset of the timestamps in the RTP packets.

This is particularly useful when several RTP packet streams have to be associated with each other. For example, if both video and audio signals are distributed, on playback there has to be synchronisation between these two media, called inter–media synchronisation. Since their RTP timestamps have no relation whatsoever, there has to be some other way to do this. By giving the relation between each timestamp format and the NTP time, the receiving application can do the necessary calculations to synchronise the streams.

A participant to a RTP session distributes reception statistics about each sender in the session. For a specific sender, a reception report includes the following information:

- The fraction of lost packets since the last report. An increase of this value can be used as an indication to congestion.
- The total amount of lost packets since the start of the session.
- Amount of interarrival jitter, measure in timestamp units. When the jitter increases, this is also a possible indication of congestion.
- Information that can be used by the sender to measure the round–trip propagation time to this receiver. The round–trip propagation time is the time it would take a packet to travel to this receiver and back.

The source description items give general information about a participant, like name and e–mail. But it also includes a so–called canonical name (CNAME). This is a string which identifies the sender of the RTP packets. Unlike the SSRC identifier, this one stays constant for a given participant, independent of the current session and it is normally unique for each participant. Thanks to this identifier it is possible to group different streams coming from the same source.

Since these packets are sent periodically by each participant to all destinations, we have to be careful not to use too much of the available bandwidth for RTCP packets. The RTCP packet interval is calculated from the number of participants and the amount of bandwidth which RTCP packets may occupy. To prevent that each participant would sent its RTCP packets at the same time, this value is multiplied by a random number.

### *5.2.3 Packet size*

Now that we have a decent protocol which we can use to transmit the digitised speech, we need to address another matter. With RTP, we can transmit packets containing voice information, but what should the size of these packets be?

We have already seen in a previous chapter that packets containing only a small amount of voice data are desirable for two reasons. First, if a packet gets lost, it does not cause a severe distortion in the communication. Second, to reduce the overall delay, the sampling interval should be as low as possible and each piece of the digitised voice signal should be transmitted as soon as possible. This automatically implies small packet sizes.

But we have to keep in mind that when this speech data is transmitted, a part of the bandwidth will be occupied with headers of the underlying protocols. So, the smaller the time interval captured in a packet, the larger is the overhead caused by headers.

Consider the following example. A voice signal is sampled at regular intervals of one millisecond. After each sample interval we will transmit the digitised signal using RTP. This means that every millisecond, at least a RTP, UDP and IP header is actually transmitted over some medium. Their total size is at least forty[14] bytes. Sending forty bytes each millisecond needs a bandwidth of 320 kbps!

It is obvious that if we increase the sampling interval, the bandwidth occupied with only header information will decrease. This will result in a larger overall delay, so we must be careful not to make the sampling interval too large. It will also result in larger packet sizes which causes the communication to be more vulnerable to lost packets.

Clearly, somehow a compromise will have to be made. Usually, sampling intervals of ten to thirty milliseconds are used, since these are the sampling intervals which a lot of compression techniques use. When bandwidth is scarce, perhaps even a larger values are advisable.

With dial–up links, the available bandwidth is very low compared to the available bandwidth on a LAN for example. In this case, we would like to have as much bandwidth available for the actual data as possible. Luckily, there exist methods to greatly reduce the bandwidth occupied by header information on such links.

When you are using a dial–up link, a lot of consecutive packets will go to the same destination application. This means that many IP and UDP header fields stay the same. When RTP is used on top of UDP, a number of fields in the RTP header will also stay the same, while the values of other fields change with a fixed amount for each packet. Using this information, the aggregate header size of forty bytes can be reduced to two to four bytes! This, of course, greatly reduces the bandwidth occupied by header information. The exact way to do all this is specified in [8].

## 5.3 QoS mechanisms

We mentioned before that RTP itself offers no way to achieve certain levels of QoS, it relies on external methods to provide this. There are several ways in which this can be done. This section gives an overview of such methods.

### *5.3.1 Assigning priorities to packets*

Both IPv4 and IPv6 have a way to specify the priority of a datagram. In the IPv4 header some level of QoS can be specified in the TOS field. The IPv6 header has a similar feature through the use of the traffic class field.

If all routers take such priorities into account, this could help real–time data to be

14 The IP header is at least twenty bytes long, the UDP header has a fixed size of eight bytes and the RTP header has a minimum size of twelve bytes.

delivered with, for example, low delay. The main advantage of this approach is that no additional protocols are needed. The only thing that needs to be done is to adjust routers so they can take the priorities of packets into account.

But these mechanisms can only help to give a better service, they cannot give any guarantees whatsoever. For example, if the whole network is filled with high−priority traffic, the quality will still be poor.

So we have to rely on other means if we want to be able to provide guarantees about QoS. In the following sections we will explore two protocols which are designed for this purpose. First, an explanation of version two of the Stream Protocol (ST2) is given. Next, I will describe the Resource Reservation Protocol (RSVP). To give guarantees about QoS, both of these methods rely on the reservation of resources. The way this is done in each of these cases differs greatly. Therefore, a section with a comparison of their basic techniques is presented afterwards.

### 5.3.2 Stream Protocol version two (ST2)

The Stream Protocol version two (ST2) was first specified in [13]. This document was released in October 1990. Five years later, after gaining experience with the protocol, it was revised and redefined in [22]. The main goal of the revision was to simplify the protocol and to clarify some issues. Also, some extensions were added. The basics of ST2 remained the same however. The information in this section was obtained from the two mentioned references.

Within the TCP/IP architecture, ST2 is situated in the internet layer. Its purpose is to provide an end−to−end guaranteed service across an internet. The protocol is not intended as a replacement for IP, but as an addition. This way, general data transfers can still use IP while real−time data can be transmitted using ST2.

Unlike IP, which is a connectionless protocol, ST2 is connection−oriented. This implies that to transfer data, there are three stages involved. First, the connection has to be set up. During this stage, resources are reserved to be able to provide certain QoS guarantees. When the ST2 connection has been established, the actual data transfer can take place. When all data has been transmitted, the connection has to be released again.

A connection will only allow the flow of data in one direction: from the origin to the destinations. If communication in the other direction has to be possible, a different ST2 connection will have to be made. This way, the connection can be represented by a directed tree, from the origin to the destinations. Using this model, the distribution of data will be done in such a way as to minimise the amount of duplicate packets sent.

To create a connection, an application must first know a number of targets to connect to. For this purpose, the application cannot use ST2 itself, it must use some other means. Probably, some IP based protocol will be used to do this.

When the application knows the destinations and the necessary QoS constraints, it can deliver this information to the ST2 module and ask it to set up a connection to these destinations. Within ST2 the QoS constraints are distributed by means of a flow specification, also called FlowSpec.

Based upon the information in this FlowSpec, the intermediate ST2 supporting routers can make the necessary reservations. If these reservations do not correspond to the desired QoS, the information in the flow specification is updated to reflect the actual obtained QoS at the current point.

When a destination receives the connection request, this FlowSpec can be investigated by the application and it can decide whether to accept the connection or not. If the connection is accepted, the FlowSpec with the acquired QoS values is propagated back to the origin of the connection request.

If all goes well, the application which attempted the connection, receives a confirmation of each destination. These confirmations also contain a FlowSpec describing the obtained QoS for each target. The application can then decide whether to start sending its data or to abort the connection.

Note that at different stages along the path from origin to destinations, the reservations can differ. The application has to explicitly release any excess reservations.

During the lifetime of the connection, there is still a possibility to add destinations. The procedure is similar as the creation stage and it is started by the origin of the connection. Destinations can also be removed from the connection. This can be done by the origin or the destination itself.

Only the data packets are in fact transmitted using ST2. Other functions are provided through the use of a control protocol: the ST Control Message Protocol (SCMP). These functions include connection creation and adding destinations. All control messages are transmitted reliably, using acknowledgements and retransmissions if necessary.

To be able to detect the failure of network elements, each ST2 capable machine periodically sends a 'hello' message to each of its neighbours. If necessary, recovery procedures will be initiated.

Note that ST2 itself does not specify how the reservations should actually be made or how the QoS itself should be provided. It only presents a way to distribute the desired QoS specifications.

### 5.3.3 Resource Reservation Protocol (RSVP)

Another way to reserve resources is by using the Resource Reservation Protocol (RSVP). This protocol is specified in [29]. The protocol is a part of an Integrated Services model, described in [17]. The term Integrated Services refers to the fact that several kinds of services can be offered, for example both real–time and best–effort services.

Unlike with ST2, RSVP does not provide its own data transmission protocol. This function is still performed by IP. RSVP is merely a control protocol which can be used to help provide QoS guarantees to applications. The protocol can be used with both IPv4 and IPv6.

If a host is going to transmit data which should arrive with a certain QoS, it periodically sends a so–called path message to the destination of the data. This address can be both a unicast or multicast address. The path message contains information about the characteristics of the traffic that will be generated by this sender. Also, the format of its data packets is described. Using this information it is be possible to select packets from this specific sender out of others.

On its way to the destinations, RSVP capable routers store the information in the path message. This way, it can be used when reservations will be made for data coming from the sender of the path message.

When an application receives a path message, it can decide it wants to receive the sender's data with a specific QoS. It can determine the necessary QoS constraints from the information contained in the path message. To request this QoS, it will periodically send a reservation request along the reverse path of the path message. The exact reverse path can be followed because of the saved information in RSVP capable routers in response to the path message.

The reservation request contains two items. First, it contains the QoS which the receiver would like to obtain. Second, it contains a description of the set of data packets which should be received with this QoS.

Inside each router, the necessary reservations can be made to supply the QoS. An important feature of RSVP is that reservations may be merged. After a merge, the router checks if there is a net change for the link upstream, and if so, an appropriate reservation request is sent to the previous router.

Lets illustrate this with a small example. Consider the situation in figure 5.2. Suppose host A is the sender and is distributing digitised speech as part of an on–line conference. This data is being distributed by sending it to a multicast address to which path messages have also been sent. Host B in response, has issued a reservation request which caused the reservation of 32 kbps along the path from host A to B.

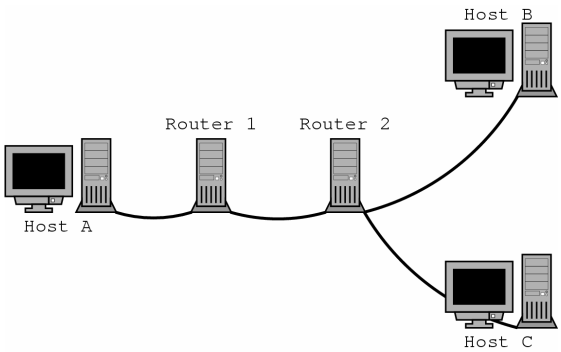


Figure 5.2 – Reservation example

Now host C also joins the conference and wishes to have a guaranteed 16 kbps link with host A. The host creates a reservation request which specifies this and sends it to router 2. There, the router investigates the request and reserves bandwidth of 16 kbps downstream over the link to host C. It also notices that upstream, there is already a reservation of 32 kbps to host A, so no extra reservation request will have to be sent to router 1. If, however, host C would have requested 64 kbps, a new reservation would have been forwarded to router 1.

Several reservation styles are supported by RSVP. For example, if a host receives data from many sources, it could issue a reservation request which would allocated separate bandwidth for each source. But it is also possible to specify that the allocated bandwidth should be shared by all senders. This is useful in case of an on–line discussion where there will be usually only one speaker at a time. It would then be sufficient to allocate bandwidth to accommodate only one or a few speakers.

Note that the path messages and reservation requests are sent periodically. This is because the RSVP information within a router will time out after a while. To keep the path information and reservations in place, they have to be updated regularly. This is called a soft–state mechanism. When a reservation or path state times out, the associated resources can be released. Resources can also be released explicitly when a sender or receiver quits.

Like ST2, RSVP does not specify how actual QoS guarantees have to be enforced. The protocol is only used to distribute the QoS related information.

### 5.3.4 ST2 vs RSVP

Both ST2 and RSVP provide a mechanism which can be used to make resource reservations along the path from sender to receivers. These resource reservations are intended to supply QoS guarantees. With both protocols, the resources are allocated for data distribution in one direction only: from sender to receivers. This can be modelled as a directed tree, with the root being the sender.

But the approach that these protocols follow differs significantly. So the question arises which one is the most efficient. In [5] a comparison is made between the two protocols. Here, I will summarise the key differences.

An important difference is from where the reservation requests originate. With ST2, it is the sender which makes the necessary reservations along its distribution path. In contrast, with RSVP the receivers request the reservations. For this reason, ST2 is often said to have a sender initiated reservation style, while RSVP 's reservation style is called receiver initiated.

For some applications, there will be several senders, but there will usually be only a few of them sending at the same time. Like was mentioned above, with RSVP this knowledge can be exploited through the use of a specific reservation style. ST2 however, does not have such a feature. Here, reservations will have to be made for each possible sender and this, of course, wastes a lot of bandwidth.

When data is distributed to a number of receivers, it is very unlikely that all these receivers will have the same QoS demands. Since ST2 is sender–initiated, the sender will have to request reservations to satisfy the needs of the most demanding receiver. Even branches that lead to less demanding receivers will all have the same reservations.

With RSVP, the heterogeneity of receivers is handled much more efficiently. The requests originate from the receivers themselves and if possible, requests are merged. This means that branches of the distribution tree which lead to less demanding receivers, will have less reservations. A previous example already illustrated this.

When a receiver is unable to accommodate data streams from all active senders, it may wish to be able to dynamically select from which sources to receive data. This is called channel selection. With ST2, the only possibility for channel selection is to make a separate reservation for each sender. The actual channel selection will have to be done at the receiver.

Recall that a RSVP reservation request contains a description of which packets should receive the associated QoS. This way, when a reservation request is sent, a new set of sources can be selected and filtering can be done inside the network.

Network failure detection in ST2 is done by periodically sending messages to neighbouring machines which participate in the same stream. If an error is detected, a recovery procedure is started. RSVP completely relies on the soft–state mechanism to automatically adapt to any failures. Note that both protocols send messages periodically. With RSVP however, the overhead of these messages is reduced by merging reservation requests were possible.

When a receiver joins a ST2 session, the reservations for this receiver have to be requested by the sender. This way a message will have to travel all the way from the sender to the new receiver. With the receiver–initiated approach of RSVP, a reservation request is only propagated towards the sender until it can be merged with other reservations. This results in less protocol overhead. However, the receiver may have to wait a while to send its request until it receives a path message.

## 5.4 Transmission delay

When resource reservation methods are supported in routers, transmission delays can probably be kept low enough to satisfy the overall delay constraint of 200 ms. But at this time, routers which are currently used in general do not have such capabilities.

When data is transmitted there is always a minimal amount of delay due to the capacity of the links along which the data travels. But the most significant part of the delay by transmission is usually due to queuing of packets inside routers. This delay is highly variable and depends both on the number of routers along the path and the load of the routers.

It is not possible to make a general claim about transmission delay in IP networks, although one–way transmission delays rarely tend to exceed 100 ms [41]. However, it is not inconceivable that this delay can exceed 200 ms.

## 5.5 Summary

When we want to transmit speech data, there are several things which we have to keep in mind. Some mechanism has to be used to preserve intra–media synchronisation. Measures should be taken to guarantee a low overall delay and somehow flow and congestion control should be possible.

In the TCP/IP architecture, an application can use either TCP or UDP to transmit data. To transmit voice information, TCP might seem like a good choice because it offers a reliable byte stream service with flow and congestion control. But this reliability is achieved by the retransmission of lost packets, which causes the overall delay to increase. An increase in delay can also be caused by the flow or congestion control mechanisms, over which the user has little control.

The other protocol, UDP, is not sufficient for real–time data, as it does not provide any means for intra–media synchronisation or flow or congestion control. A solution to this problem is to extend UDP somewhat. This is the way RTP is used in the TCP/IP architecture.

The Real–time Transport Protocol (RTP) provides extra information which can be used for synchronisation within a data stream. The RTP Control Protocol (RTCP) provides additional information which can be used for inter–media synchronisation, flow and congestion control and identification.

Each piece of speech data which is transmitted will have a number of headers sent along with it. These headers also occupy a part of the bandwidth, so to reduce the header overhead, packets should not be too small. For dial–up links, extra bandwidth savings can be achieved by using header compression techniques.

To provide quality of service (QoS), one could use the priority information in the IP header. This method can improve the QoS, but it cannot offer any guarantees. Other protocols like ST2 and RSVP can be used to do this. Both of these give QoS guarantees by making resource reservations.

The Stream Protocol version two (ST2) uses a sender initiated reservation model: the sender issues the resource reservation requests along the path to the receivers. It is a connection–oriented protocol, which serves as an addition to IP.

The Resource Reservation Protocol (RSVP) on the other hand, uses a receiver initiated approach. The sender distributes a description about the data which is being sent and the receivers can make resource reservations for this data. It is a soft–state mechanism: the senders and receivers have to send this information periodically because the associated resources will be released otherwise. RSVP also merges reservations where possible. This protocol seems to be more efficient than ST2.

The transmission of a packet introduces an amount of delay into the communication. This delay is highly variable due to the queuing delays in routers.

# Chapter 6: Voice in virtual environments

The information in the previous chapters gives an idea about the workings of a general VoIP application. We can now extend this to VoIP in virtual environments.

This chapter begins with an explanation of where the 3D sound should be generated: at the sender or at the receiver. Then, I will present several possible methods for distributing the data. This is followed by a description of how 3D sound can be generated and finally, the involved processing delay is discussed.

## 6.1 Where to generate the 3D sound?

When we are using voice in virtual environments, adding a 3D effect to the speech signal will create more realism. But this 3D effect can be generated both at the receiver and at the sender side. Which approach is the most efficient?

When it comes to processing delay, it makes no difference where the 3D sound is created. If it is generated at the sender side, processing will have to be done on outgoing packets for each possible receiver, since they will all need a slightly different effect. If it is generated at the receiver side, processing will have to be done on incoming packets from each sender. So, the net result for these methods will be the same.

But when it comes to bandwidth utilisation, processing at the receiver side has some advantages. To create a 3D effect, a stereo sound signal is needed. This means that when the 3D sound is generated at the sender, the mono speech signal will be converted into a stereo one, which needs at least twice the amount of storage space. Consequently, when this data is transmitted, it will need at least twice the bandwidth as the unprocessed data. Furthermore, due to the 3D effect, the data which has to be transmitted will differ for each receiver. This eliminates the possibility of multicasting the data to reduce the required bandwidth.

In contrast, when the 3D sound is generated at the receiver side, the sender can distribute the mono data which is the same for all receivers. This requires less bandwidth than a stereo signal and it also allows the sender to multicast this information, making very efficient use of the available bandwidth. Note that using this approach, the receiver must somehow know the position of the sender of the data to be able to generate the 3D effects.

## 6.2 Distribution mechanisms

Now that we know that it is best to distribute the mono speech data to the necessary receivers, we have to determine a way to do this. Note that not all participants in the virtual environment will be interested in this data: some will be so far away that after the 3D processing step, the resulting sound will not be audible.

In this section I will describe some ways to distribute the speech data. First, a method using unicasting is described and next, some methods involving multicasting are given.

### *6.2.1 Unicasting*

When you are using unicasting to distribute the speech data, you will send a copy of the data to the appropriate receivers. It is obvious that multicasting would be more efficient, but it is possible that this service is simply not available.

The advantage of unicasting is that the sender can control exactly who receives the data. The sender simply looks up the participants who are close enough to 'hear' him, and sends the voice information to those destinations.

### 6.2.2 Multicasting

Multicasting is a more efficient way to distribute data, since the sender only has to transmit one copy of the data. This data is duplicated only when it has to be forwarded over separate links. Still, there are several approaches that can be taken.

#### 6.2.2.1 A single group

One possibility is to use one multicast group for the whole virtual environment. In this case, each participant will receive all the data that is being transmitted, even the data from senders which are too far away. The receivers themselves should then determine whether to process the incoming data, based upon the distance of the sender.

The main disadvantage of this method is that possibly a lot of participants receive unnecessary data, which obviously wastes bandwidth. However, when the virtual environment is quite small, this approach can prove to be very useful, since almost every participant then needs to receive the data from other participants. Also, when only one multicast address can be used, this technique might still prove to be more efficient than using unicasting to transmit the data.

#### 6.2.2.2 One group per participant

The most efficient way to distribute the data is by assigning a single multicast group to each participant. Each participant then only sends data to its own multicast address, and only if there are other participants within a certain range. To receive the appropriate data, a participant joins the multicast groups of other participants which are in range.

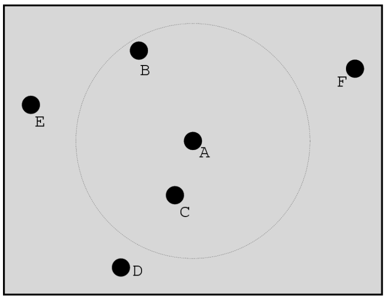

Figure 6.1 – Multicasting example

As an example, consider the situation in figure 6.1. Here, the black dots represent participants in a virtual environment and the dotted circle marks the range for participant A. Like the other ones, participant A has its own multicast group and will send its voice data to it since there are other participants – namely B and C – in range. Participant A will also join the multicast groups of B and C, since he wants to be able to 'hear' them. The other participants use the same technique.

In contrast to the unicast technique, where it is the sender who decides who receives its speech data, in this case the receivers decide for themselves whose data they want to receive by joining the appropriate multicast groups. From the point of view of security, this solution is not as safe as the unicast solution since basically everyone can hear what everyone has to say. However, the distribution of data is far more efficient than it is in the unicast case.

## 6.3 Generating 3D sound

When speech data arrives at the receiver, 3D effects have to be added to it; we have to spatialise the sound. How this can be done, is covered in this section. First, we will see how sound is perceived as coming from a specific position. Next, it is explained how it is possible to generate 3D sound. The following information is mostly based on [1], which presents an excellent introduction into these matters.

### 6.3.1 Perception of 3D sound

The reason that we can localise the source of a sound quite accurately is that we have two ears. At each ear, a slightly different signal will be perceived and by analysing these differences, the brain can determine where the sound originated.

### 6.3.1.1 Primary cues

When a sound source produces a sound wave, the length of the path to each ear can differ. This is illustrated in figure 6.2. How much this difference is, depends on the relative angle of the head to the sound source.

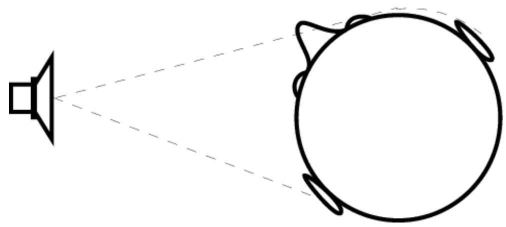

Figure 6.2 – Sound reaching the head

In normal circumstances, the speed of sound in air is about 343 meters per second. So when the length of the path to each ear differs, the sound will reach one ear before the other. This effect is called the Interaural Time Difference (ITD). The ITD is a first indication to the position of the sound source.

The importance of the ITD is frequency dependent. This is demonstrated in figure 6.3. The figure shows two signals: a low frequency signal (above) and a high frequency signal (below). The full line represents the sound wave reaching the ear closest to the sound source (at time *t1*), the dotted line represents the signal at the other ear (at time *t2*).

In the low frequency case, the time difference is accurately given by the displacement in the signals. However, when a high frequency signal is perceived, the information is ambiguous. This is because the actual difference in the signals covers several cycles. Once the signals are overlapping, this cannot be determined anymore since the displacement will always seem to be less than one cycle.

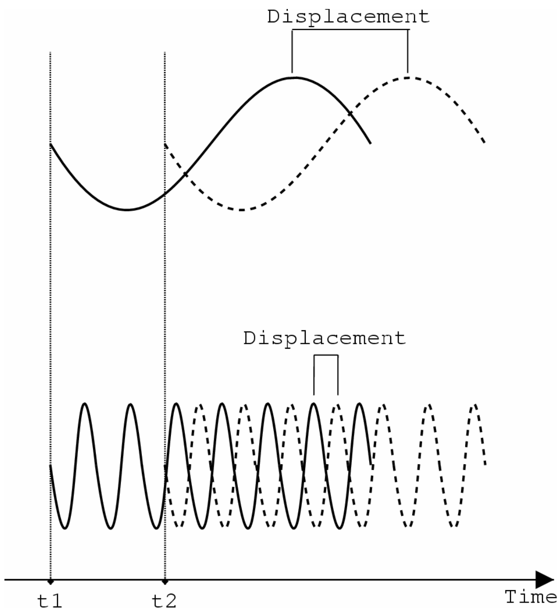


Figure 6.3 – Frequency dependence of the ITD

The intensity of a sound wave decreases as it travels through the air. Since the path length to each ear can differ, this implies that the intensity of the signal at each ear will differ. This difference is called the Interaural Intensity Difference (IID).

When the head is 'in the way' of the sound wave, the IID is also frequency dependent. When the wavelength of the sound is large relative to the diameter of the head, the intensity difference will be rather low. But when the wavelength is small, the intensity difference can become quite large. This is called the head–shadow effect.

Note that the ITD and IID are complementary. At high frequencies, the IID is the most important cue for localisation, but at low frequencies the ITD provides the most accurate information.

The fact that ITD and IID are the primary cues for localisation and the fact that they are complementary is stated in the so–called Duplex Theory. This theory was developed by Lord Rayleigh about a century ago.

### 6.3.1.2 Effect of the outer ear

When we only take the distance of the sound source to each ear into account, it is clear that there is no way to make a distinction between front or back and above or below. So there must be some other factors which allow us to localise sounds.

The outer ear or pinna also plays an important role in the localisation of sounds. Because of its shape, it boosts some frequencies, while others get dampened. With which frequencies this happens, highly depends on the position of the sound source. This effect causes the pinna to help a lot in localising the sound source.

Because of the rather small sizes of the pinna and its folds, it is mainly the higher frequencies which are transformed. The brain is therefore better able to localise higher frequency sounds than lower frequency sounds.

#### 6.3.1.3 Estimating range

Range estimation is not yet well understood, but there are several known factors. A first indication of the distance to the sound source is given by the loudness of the sound. A sound coming from far away tends to sound a bit muffled, while a near sound is more clear. But it is not only the loudness that is important, but also the nature of the sound. For example, if you are far away, you will not seem to be close if you yell. This is because sounds produces by yelling and talking have different characteristics.

Turning the head also helps to determine the range of a sound source. The change in angle to a sound source that is close is larger than the change for a source that is further away. This is illustrated in figure 6.4. Part (a) of the figure shows two sound sources with the same angle to the right ear. When the head is turned, shown in part (b), the change in angle to the closest sound source is larger than to the other source. This helps the brain in determining the distance to a sound source. The effect is called 'motion parallax'.

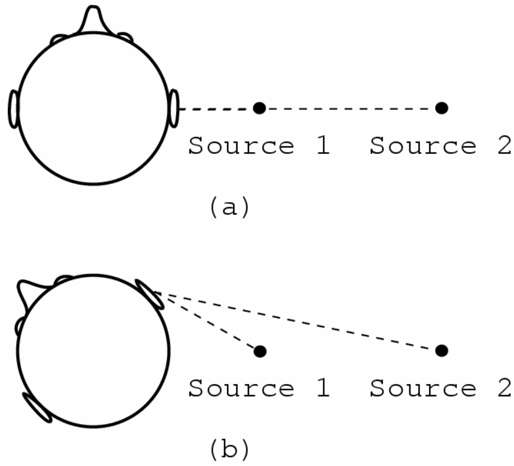

Figure 6.4 – Motion parallax

The IID also provides some information to determine the range of the sound source. The intensity of the sound decreases inversely with the square of the range. This causes the IID to be large for close sound sources. An example of this is when you hear an insect buzzing in one ear.

When you are in a room, you do not only perceive the sound wave coming directly from the sound source, but also a lot of sound which has been reflected off objects. This type of sound is referred to as reverberant sound. Reverberant sound does not differ as much with the distance to the listener as the direct sound. Therefore, the ratio of direct to reverberant sound is also a cue to the range of the sound source.

### 6.3.2 Generating spatialised sound

With the knowledge of how sound is perceived as coming from a specific location, we can try to simulate this effect. We do this by transforming a mono sound signal into one which seems to be coming from a certain position. Since the key to 3D sound is the different signal at each ear, the resulting sound signal will be stereo.

#### 6.3.2.1 Using ITD and IID

A sphere can be used as a simple model for the head. Given the positions of a sound source and the listener, this model can be used to calculate the path of the sound to each virtual ear. This information can then be used to calculate ITD and IID. For accurate results, the curvature of the sphere should be taken into account.

Figure 6.5 illustrates this model. The figure shows how the path of the sound differs for each ear. When the sound source is at an angle θ and the head radius is R, the sound will have to travel an extra distance of R.θ + R.sinθ to reach the ear which is furthest away.

Using this information together with the knowledge that the speed of sound is about 343 meters per second, we can calculate the time it takes for the sound to reach each ear. We then simply have to insert the appropriate amount of delay for each channel in the stereo signal to give it the correct amount of ITD.

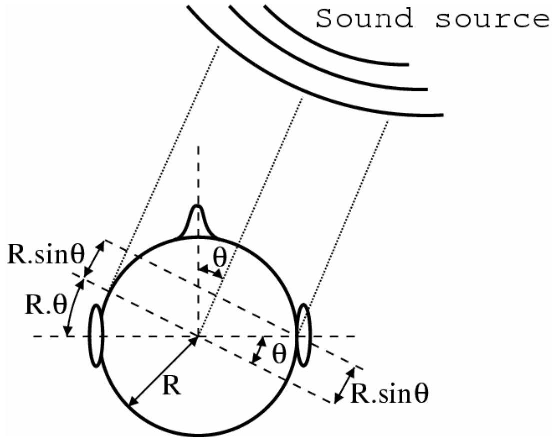

Figure 6.5 – Spherical head model [1]

Using the path length for each ear, it is also possible to calculate the basic part of the IID (without taking the head–shadow into account). It is known that under normal circumstances the intensity level of sound decreases with approximately six decibels as the distance increases with a factor of two [2]. The relationship between the decibel scale and the amplitude is given by

$$D = 20 \cdot \log_{10}|A|$$

where A is the amplitude and D is the corresponding intensity level. After a bit of calculating, you will find an expression which you can use to adjust the amplitude of a signal, given a certain distance:

$$A' = A \cdot 10^{\frac{-3}{10} \cdot \log_2 \frac{distance}{headradius}}$$

This formula can be simplified further, and this gives us approximately the following relationship:

$$A' = A \cdot \frac{headradius}{distance}$$

Note that 'distance' is calculated from the centre of the virtual head of the participant who produced the sound. This means that it will always be at least 'headradius', which is the radius of the virtual head of that participant.

As you can see, this formula indicates that the original amplitude simply has to be multiplied with a factor that depends both on the distance and the head radius. When this formula is applied for the distance to each ear, this will result in a certain IID.

As was mentioned above, these calculations do not take the head–shadow effect into account. To do this, we have to determine a filter which depends on the angle to the sound source and which dampens the high frequencies when the head is in the way. More information about modelling the head–shadow effect can be found in [1].

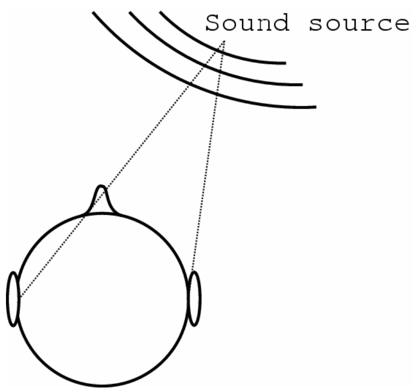

Figure 6.6 – Simplified spherical head model

In my own implementation I have used a simpler model than the one above. To calculate the distance, I have not taken the curvature of the head into account. I have simply calculated the distance from each ear to the centre of the sound source. This is illustrated in figure 6.6.

In my application, the 3D effect was created by using this model and the ITD and IID calculations mentioned above. I also added some reverberation to make the sound a bit more spatial. Personally, I found the result quite good.

By using only ITD and IID in the creation of a 3D sound, it is not possible to make a distinction between front and back or above and below. Also, the sound still seems to be coming from inside the head: there is no sense of externalisation. To solve these problems, more sophisticated techniques have to be used.

#### 6.3.2.2 Head-Related Transfer Functions (HRTFs)

Basically, what we need to know is what a sound signal looks like when it reaches each eardrum. We already know that ITD and IID will be introduced, but a lot of other effects also occur, like reflections by the pinna for example.

We can model the transformation which occurs at each ear as a variable linear filter. It is variable because the effect differs according to the position of the sound source. For a linear filter, it can be shown that if we know the filter's output to an impulse, we can calculate its output to any signal [11]. The output of a filter to an impulse is called the impulse response for that filter.

So if we can find out what signals reach the eardrums in response to an impulse, we can determine its response to any signal. This impulse response is called the Head−Related Impulse Response (HRIR). Note that the HRIR still depends on the position of the sound source and in general will be different for each ear.

The representation of a HRIR in the frequency domain is called a Head−Related Transfer Function (HRTF). Like with the HRIR, with each position there are two related HRTFs: one for each ear. When you look at these signals, you can clearly see which frequencies get boosted and which ones get dampened.

One way to obtain the HRTF information is by actually measuring them. Usually, this is done by using a model of the human head, in which microphones are present at the ears. A well known model is the KEMAR model. KEMAR stands for 'Knowles Electronic Manikin for Acoustic Research'. Measurements which are made with this model are available to the public, which makes this method quite easy to use.

There is a disadvantage to the use of these measured HRTFs. Because the shapes of the head and the pinna differ greatly from person to person, these standard measurements will not create a good 3D effect for several people. The ideal solution would be taking measurements of the listener's HRTFs, but this consumes a lot of time and effort.

Another solution is not measuring the HRTF information, but modelling it. This way, a number of parameters could be set by the user to generate a good 3D effect. More information about such models can be found in [1].

#### 6.3.2.3 Speakers vs headphones

If headphones are used, it is not difficult to generate a specific signal for each ear, since there is no interference of the two signals. But some headphones tend to transform the signal somewhat, which makes localisation less accurate. Also, localised sounds coming from a headphone often seem too close because of the closeness of the actual sound source.

Speakers generally do not cause such problems, but there is another problem: the signals from the speakers interfere with each other. It is possible to create the signals for each speaker in such a way that the resulting signal at each ear is still correct, but it is computationally quite intensive. Also, the listener has to be sitting in the right spot and cannot turn his head too much.

Personally, I have used headphones and found the results quite adequate. This is computationally very simple, especially since I only used a very simple model to create 3D sound. Another advantage of headphones is that no echo is generated when a microphone is near.

## 6.4 Processing delay

Transforming a mono speech signal into a stereo one which seems to be coming from a certain position will require several calculations which, in turn, introduce delay. Depending on the realism to be achieved, the delay can vary greatly.

The simple technique I have used in an application requires almost no CPU power. The distance from each ear to the sound source is first calculated. Then, the appropriate amount of delay is inserted for each channel and the amplitude of the signal is adjusted. This does not require many calculations.

When HRTFs are used, the calculations are more demanding. Note that for each ear the calculations have to be done separately. Depending on the desired realism, the required calculations still vary a lot. When the calculations have to be done for only a few sound sources, it can normally easily be done in real–time by software. However, when many sources have to be processed, it may not be possible to do this anymore. Fortunately, many soundcards already have the ability to generate 3D effects, so we can relieve the CPU of this task.

## 6.5 Bandwidth considerations

When several participants in the virtual environment are speaking at the same time, a receiver needs to have enough bandwidth available to receive their voice data. When you have direct access to a LAN, this is probably not a problem. But when you are using a dial–up link, the necessary bandwidth might simply not be available, even when severe compression is used.

A solution to this problem is to place a machine which mixes the signals for a specific participant before the link. This is depicted in figure 6.7. In this figure, the dial–up link is directly to the mixer, but this is not necessary. The only thing the mixer needs to do, is to generate the appropriate 3D effects for the user’s position, mix the signals from the sound sources together and transmit the resulting data to the user. The user will then only need bandwidth for one stream, which is achievable over a dial–up link. Note that this will introduce some extra delay.

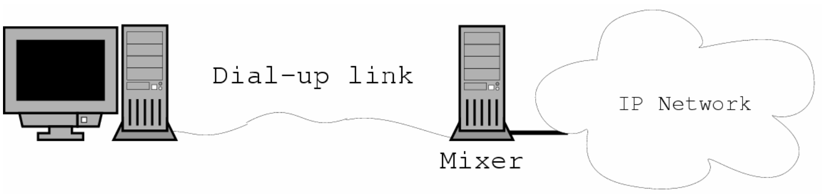


Figure 6.7 – Using a mixer for a dial–up link

## 6.6 Summary

To make a sound appear to come from a specific position, it is necessary to generate a stereo signal. Because of this, it is more efficient to add 3D effects at the receiver side, since then we only need to transmit a mono signal. This way, we can also make use of IP multicasting because the same data can be sent to all receivers.

One way to distribute the speech data is to use unicasting. It allows the sender to determine who receives the data, but it wastes bandwidth. More efficient distribution can be achieved by using multicasting. However, this way the senders cannot determine who receives the data. It is then up to the receivers to decide which data they need to process and which not.

Sounds are perceived as coming from a certain point because each eardrum receives a slightly different signal. From these differences, the brain determines the position of the sound source. Two important cues for localisation are Interaural Time Difference (ITD) and Interaural Intensity Difference (IID). The outer ear or pinna also plays a very important role in the localisation of sounds.

Using ITD and IID, it is possible to create basic 3D effects. However, this way it is not possible to create distinction between front and back or above and below. Better results can be achieved by simulating the transformations of the sound signal before it reaches the eardrums. These transformations are described by Head−Related Transfer Functions (HRTFs).

Because there can be several sound sources at the same time, it is possible that the calculations to generate 3D sounds are too demanding. It may then be necessary to let hardware perform the localisation of the sound. For the same reason, the necessary bandwidth may not be available, for example when using a dial−up link. A solution is to let a machine before the slow link apply the 3D effects and let it send the mixed signal over the link.

# Chapter 7: Related subjects

In this chapter we will see some VoIP related topics. There is a lot going on in the VoIP world and it would be impossible to describe every item which is related to VoIP. Therefore, I will give a brief overview of some subjects which I found interesting. They also give a good idea of what is currently being worked on.

First, a description is given about H.323 and the Session Initiation Protocol (SIP). Because these protocols are somewhat related, next I will present a short comparison of them. Finally, a brief explanation of the Real−Time Streaming Protocol (RTSP) is given.

## 7.1 H.323

The ITU−T document about H.323 is a recommendation for multimedia conferencing over packet based networks without QoS support. It is a part of the H.32X series of recommendations which all describe multimedia conferencing but over different types of networks. These recommendations are: [6]

| Recommendation | Network |
|---|---|
| H.320 | Narrowband Integrated Services Digital Network (N−ISDN) |
| H.321 | Broadband Integrated Services Digital Network (ISDN) |
| H.322 | Guaranteed bandwidth packet switched network |
| H.323 | Non−guaranteed bandwidth packet switched network |
| H.324 | The analogue phone system |

This section presents an overview of the H.323 recommendation, mostly based upon the information in [6] and [14].

### 7.1.1 Functionality

End systems conforming to the H.323 recommendation can communicate with each other, either point−to−point or in a multipoint conference. These end systems may have different capabilities, but each must at least support G.711 audio encoding. Video support and other audio coders are optional. H.323 also defines how to do general data transfers, but this feature also is optional.

The recommendation allows communication with end systems on a different type of network, conforming to other H.32X standards. This requires special devices which connect to the different networks and do the necessary conversions.

Management and accounting support are also provided. This way it is possible to specify for example the maximum amount of bandwidth that may be occupied with H.323 calls. Accounting is provided to support billing of the callers.

The H.323 recommendation defines a framework for the development of supplementary services. Currently, two such services are already defined: call transfer and call forwarding.

Finally, since packet based networks − like IP networks − are often not very secure, H.323 defines several mechanisms to provide better security.

### 7.1.2 Components

Four components are specified in recommendation H.323: terminals, gateways, gatekeepers and multipoint control units (MCUs). A terminal is a system where H.323 data and signalling streams originate and terminate. It was already mentioned that such a system must at least be capable of handling G.711 audio.

A gateway is a device which allows H.323 capable systems to communicate with other H.32X systems. Gateways connect the different networks together and perform the necessary transformations. For example, it may be necessary to change signalling information or to use another audio encoding. A gateway is optional in a H.323 enabled network.

A gatekeeper is an optional component, but is very useful when present. When a gatekeeper is present, all terminals, gateways and MCUs must be registered with it. Two important services are provided by a gatekeeper. The first one is address translation from an alias – an international phone number for example – to a network address – an IP address for example.

The second major service of a gatekeeper is bandwidth management. A gatekeeper could be configured to limit the bandwidth used by H.323 calls or to only allow a certain amount of simultaneous calls.

An optional feature of a gatekeeper is to route calls. When a call is routed through a gatekeeper, this allows more effective control and more information about the call. This could be used to bill calls or to re–route a call to another system when a user is unavailable at the called endpoint.

A MCU is used for conferences between three or more endpoints. It contains a multipoint controller (MC) and possibly a number of multipoint processors (MPs). Participants send their control information to the MC so that endpoint capabilities can be exchanged and communication parameters can be negotiated. A MP is used to process the incoming media, for example to mix several streams together.

Three models for multipoint conferencing are defined. In all models each participant sends its control information is to the MCU, where it can be processed by the MC. In the centralised model, each participant also sends its media to the MCU. In the decentralised model the different media are distributed by multicasting them. In the hybrid model, some participants use multicasting to distribute the media, others send their media directly to the MCU.

### 7.1.3 Architecture

The H.323 recommendation is often called an ‘umbrella specification’. This is because it uses several other ITU–T recommendations to provide its functionality. The structure of the H.323 architecture is illustrated in figure 7.1

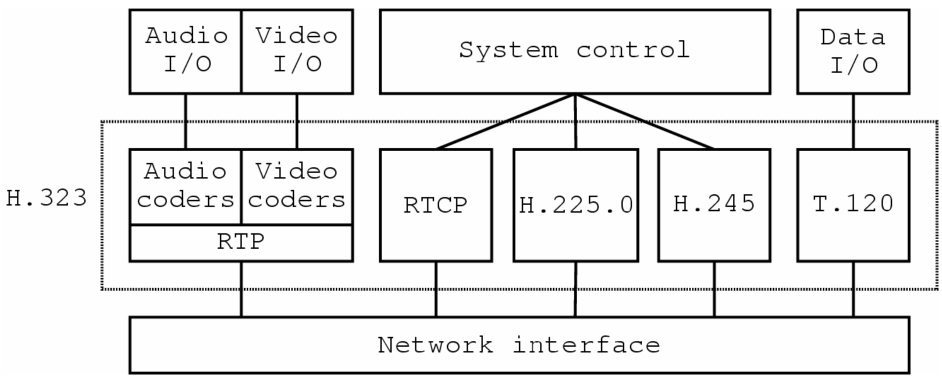


Figure 7.1 – H.323 architecture [14]

The audio coders are the ITU–T G. standards which were already described in chapter four. The video coders defined in the recommendation are H.261 and H.263. The H.263 coder was designed for low bit rate transmission but is more complex than H.261. Both audio and video are encapsulated in RTP packets and then transmitted across the network. Additional information about these transmissions is provided by RTCP.

Before two or more parties can communicate with each other, the call first has got to be set up. This is done using mechanisms defined in H.225.0 and H.245. A part of the H.225.0 recommendation specifies how a call should be set up and torn down. When the call has been established, the capabilities of the involved end systems are exchanged so that each end system can select the appropriate coders. This capability exchange is done by H.245, which also defines other functions, for example the opening and closing of logical channels to transport audio and video.

Another part of the H.225.0 recommendation specifies how the interaction with a gatekeeper should be done. This is a done by a protocol called RAS, which stands for Registration, Admission and Status. The RAS functions include gatekeeper discovery and endpoint registration with a gatekeeper. Functions like bandwidth management and admission control are also done by RAS messages.

H.323 end systems can also exchange general data with each other. How this should be done is specified in the T.120 recommendation. Like H.323, this is also an umbrella recommendation, defining how to use other protocols to exchange data.

How security services should be provided is defined in recommendation H.235. Authentication is provided by admission control of endpoints, which is done by a gatekeeper. Data integrity and privacy are implemented using encryption techniques. Finally, non–repudiation is also provided by a gatekeeper. Non–repudiation means that nobody can deny that he participated in a call.

## 7.2 Session Initiation Protocol (SIP)

The Internet Engineering Task Force (IETF) has also been working on protocols to provide multimedia communication. Like with H.323, the media themselves are transported with RTP, so the main difference between the approaches of the ITU–T and IETF is how call signalling and control is done. These functions are covered by the Session Initiation Protocol (SIP).

SIP is formally specified in [31] where it is described as an application–layer control protocol that can establish, modify and terminate multimedia sessions or calls. Although no real assumptions are made about the underlying network and protocols, SIP has been designed with the TCP/IP architecture in mind. Unlike call signalling and control protocols in the H.323 recommendation, SIP is a text based protocol. It resembles somewhat the Simple Mail Transfer Protocol (SMTP) and the Hypertext Transfer Protocol (HTTP), the protocols used to transfer e–mail and World Wide Web pages respectively.

This section describes the Session Initiation Protocol. The information herein is mostly based upon the SIP specification in [31].

### *7.2.1 User agent (UA)*

A user agent (UA) is an application which resides at a SIP end station. It consists of two parts: a user agent client (UAC) and a user agent server (UAS). The UAC is responsible for sending SIP requests when a call needs to be established. The UAS is a server application which contacts the user when there is an incoming request and responds to it.

### *7.2.2 Network servers*

Three types of network servers are defined. The first one is a redirect server. A user can send a call invitation request for another person to a redirect server. This server will then locate the user and return the necessary information to enable the caller to establish a call with the intended person.

The second type of server is a proxy server. Like with a redirect server, a user can send an invitation request to a proxy server. The proxy server will also try to locate the destination of the call, but unlike with a redirect server, it will not simply return possible locations of the called person. Instead, based upon that information, a proxy server will try to establish a connection on behalf of the caller.

Finally, the last server type is a called a registrar. A user can send information about its current location to a registrar; the user can register himself. This information can then be used to contact him. Thanks to registration information, personal mobility is allowed, which means that a person should be able to accept calls directed to him at any end system. The information sent to a registrar describes at which system a user should be contacted.

### 7.2.3 Operation

Somehow, you must specify to who you want to make a call. A SIP user is identified by a SIP Uniform Resource Locator (SIP−URL). Such a URL looks somewhat like a World Wide Web URL or an e−mail address. An example is ‘sip:me@home.net’.

When a user wants to invite someone into a session or wants to make a call to someone, the user can send an invitation request to the end system specified in the destination’s SIP−URL. In the example above, the request would be sent to ‘home.net’. If the called user is available at that system, he can send a response, indicating whether he wants to participate in the communication or not. When the caller receives this response, he sends an acknowledgement to the other system.

The invitation request could also be sent to a redirect server. This redirect server would then look for possible locations of the called user and send the corresponding SIP−URLs back to the caller. Based upon this information, the caller could then try to contact the other user directly, as described above.

Finally, the caller could also send its invitation request to a proxy server. This proxy server then looks for possible locations of the other user and tries to invite that user itself. When the proxy knows that the invitation was either accepted or denied, it can send an appropriate response back to the caller. This way, a proxy acts as both a client and a server.

The invitation request normally contains information about the media that will be sent. If the invitation was successful, the response will also contain a description about the media that the other user will use. The SIP specification does not demand a specific format, but the Session Description Protocol (SDP) was designed for this purpose.

Note that SIP can be used to invite parties to both unicast and multicast sessions and that the initiator of the invitation does not actually have to participate in the session. SIP also offers services to provide secure communications.

## 7.3 H.323 vs SIP

Since H.323 and SIP offer similar services, which solution should be used? Comparisons of these protocols are given in [7] and [3]. In this section, I will summarise the key points made in [3]. For a more complete discussion you should consult these references.

When we compare the complexity of the two protocols, it seems that SIP is far less complex than H.323. The specification of H.323 is more extensive than that of SIP and defines a lot more elements. Furthermore, H.323 uses a binary encoding mechanism for call signalling and control, whereas SIP is text based. This textual format is easy to decode and much easier to debug than a binary representation. A part of the complexity of H.323 stems from the interaction between several components which are not cleanly separated. Also, in H.323 there may be several ways to accomplish a single task and some of the functionality is present in several parts of the protocol.

Considering the extensibility of the protocols, the experience with other protocols like SMTP and HTTP has been used to make SIP very extensible: new features can easily be incorporated into the protocol. H.323 also allows some extensions, but only at predefined places within the protocol. SIP is quite modular which allows its components to be changed quite easily. H.323 on the other hand, is less modular. Since various protocol components usually need to work together to accomplish a task, it will be harder to simply replace one component.

H.323 was originally intended for use on a single LAN. Currently, this restriction is no longer present, but H.323 can have some difficulties in detecting looping messages. SIP can be used over wide area networks without any difficulties, easily detecting loops when they occur. H.323 also has some difficulties when the conference size keeps increasing. The use of a Multipoint Controller (MC) is a bottleneck for the conference. When the conference size keeps growing, eventually another protocol will have to be used: H.332. Since SIP does not have something similar to a MC, it does not suffer from such scalability problems.

Like was mentioned before, the services provided by H.323 and SIP are roughly the same. However, when it comes to capability exchange services, it seems that H.323 has a much richer set of functionality than SIP. Also, H.323 has various conference control services for which SIP has to rely on external protocols. On the other hand, the personal mobility services provided by SIP are more extensive than similar support in H.323.

## 7.4 Real-Time Streaming Protocol (RTSP)

The Real−Time Streaming Protocol (RTSP) is also related to VoIP, but in a whole other way than H.323 and SIP, which can be used to establish VoIP calls in a standardised way. RTSP on the other hand does not have such a direct relationship with VoIP, it merely uses a lot of the same techniques.

The official specification can be found in [28], where RTSP is defined as a protocol which establishes and controls either a single of several time−synchronised streams of continuous media. Like SIP, RTSP is an application level protocol and is a part of the overall IETF multimedia data and control architecture.

Like with VoIP, the continuous media which are transmitted, are divided into tiny pieces which are separately sent across the network. At the other end, the continuous media will have to be reconstructed from those pieces. Usually RTP will be used for the transmission of such packets, although RTSP does not require it.

RTSP has some resemblance to HTTP, the protocol used to transfer World Wide Web pages. Whereas HTTP provides functionality to transmit text and images, RTSP tries to provide similar services for audio and video. RTSP provides a VCR−style remote control for audio and video. For example, a user can start, pause or stop the playback of media across a network.

A media server offers playback or recording functionality of media. A client can use RTSP to interact with such a media server. The following operations are provided:

- A client can retrieve media stored on the media server. This media will then be sent back to the client.
- The media server can be invited into a conference. The media server can either record a presentation or play back media.

- The server and client can notify each other when additional media has become available.

A presentation or a media stream is identified by a RTSP Uniform Resource Locator (RTSP URL), which looks somewhat like a HTTP URL. An example of a RTSP URL is ‘rtsp://example.com/audio’. The overall presentation and the properties of the different media are defined in a presentation description which has to be obtained by means other that RTSP, for example via a World Wide Web page.

## 7.5 Summary

Several protocols and standards are related to VoIP. A first example is H.323, a recommendation for multimedia conferencing over packet based networks without QoS guarantees. It is a part of a series of standards which describe similar services over different kinds of networks.

H.323 is an umbrella specification: it defines how different components should work together to allow certain functions. Using H.323, a call can be set up, capabilities can be exchanged and parties can communicate with each other. Multi−user conferences are also possible. Media originate and terminate in terminals. A gateway makes communication possible over different kinds of networks. Multipoint conferences are made possible through the use of a multipoint control unit (MCU). A gatekeeper makes advanced features like bandwidth management and accounting possible.

Recommendation H.323 was developed by the ITU−T. The IETF has been working on an architecture which provides similar services. As in H.323, the media are transmitted using RTP, so the main difference between the two approaches lies in call signalling and control. In the IETF architecture, the Session Initiation Protocol (SIP) can be used for these services. The functionality which SIP provides is similar to that of signalling and control components in the H.323 architecture. An important feature of SIP is the possibility of personal mobility. This means that a user can answer calls directed to him at any place he wants.

When we compare signalling and control in H.323 and SIP, it seems that SIP is less complex, better extensible and better scalable. Their functionality is similar, but the capability exchange method of H.323 is more advanced. SIP on the other hand, provides better personal mobility.

The Real−Time Streaming Protocol (RTSP) is another VoIP related protocol. The protocol serves as a sort of remote control for a media server. For example, RTSP can be used to play back a presentation or media file stored on a server. It could also be used to record an ongoing presentation.

# Part III: Development

# Chapter 8: JRTPLIB

It was already mentioned in chapter five that RTP is a very useful protocol for transmitting real−time data. To make it easy to use RTP in several applications, I decided to write a library which provides the necessary functionality. This library was given the name JRTPLIB, which stands for "Jori's RTP Library". At the time of writing, the latest version of the library is version 2.3.

The RTP library is described in this chapter. First, I will explain what the library can do and then some implementation related issues are discussed. Finally, I will give some reactions of people when the library was made available on the World Wide Web.

## 8.1 Features

The RTP library was made according to the specifications in [30]. It performs several necessary functions automatically, to make using RTP easier for the programmer. The library was written in C++ using an object−oriented approach.

### *8.1.1 Functionality*

Working with the library, you can easily select to which destinations data packets have to be sent. A destination can be either a unicast or a multicast address. Default values can be set up in case a lot of packets need the same parameters.

To be able to receive packets, you have to call a 'poll' function periodically. This will make the library check for incoming RTP and RTCP packets, which are then processed. Functions are provided to be able to join and leave multicast groups. When you are in a multicast group, all packets which are sent to that group will be received[15].

You can select one of three receive modes. The first mode simply accepts all incoming packets. The second receive mode accepts only packets coming from user−specified origins. Finally, the last mode accepts all packets except those coming from origins specified by the user.

The incoming RTP and RTCP packets are processed and the resulting information is stored per participant. To access this information you first have to select the appropriate participant. This can be done by either iterating over all participants until the right one is found or by selecting one using its SSRC identifier.

The RTCP protocol is handled entirely internally. Each time you send a RTP packet or poll for incoming packets, the library checks if it is time to send RTCP packets. The time at which RTCP information should be sent is calculated from the number of participants and from the maximum bandwidth RTCP data may occupy, as specified in [30]. As was already mentioned, when you poll for incoming data, any new RTCP packets are processed and the information is stored with the corresponding participant.

For certain events, the user can specify handlers which should be called by the library. A handler is a user−specified function. For example, a user could specify a function which the library should call when a new participant joins the session.

Finally, the user can set its own source description (SDES) information as well as the SDES information of contributing sources. The SDES information about another participant is stored with the other RTCP information about that participant.

---

15 Note that this is not entirely correct. The only packets which are actually received by the library are the ones with the right port numbers.

### 8.1.2 Platforms

The first versions of the library were tested on a Linux and a MS−Windows platform. When the library was made available through the World Wide Web I was able to add support for several other platforms, thanks to the help of many people. Currently, the library is known to work on the following platforms:

- MS−Windows 95, 98 and NT
- Linux
- FreeBSD
- Solaris
- HP−UX
- VxWorks

Normally, it should be possible to compile the library on other UNIX−like platforms too.

## 8.2 Implementation

Like I already mentioned, the library was implemented in C++ using an object−oriented approach. In this section, I will first give an overview of the library's structure and next I will discuss some design decisions I made.

### 8.2.1 Overview

The Application Programming Interface (API) of the library consist of four parts:

- RTPSession class
- RTPSourceData class
- RTPPacket class
- Structures to hold event information

The RTPSession class is the central part of the library. Through this class, the user can select destinations, send RTP packets, poll for incoming data etc. It also provides a mechanism to ask for information about a participant. This information is stored in an instance of the RTPSourceData class. Received packets are passed to the user as instances of the RTPPacket class. Finally, when an event handler is called, event related information is passed through an argument of the handler, which can be one of several structures.

The first version of the library was quite monolithic: almost all of the library's functionality was implemented in the RTPSession class. After completing the original version, I immediately started a new one, which has led to the current structure. Now, the library is much more modular. The RTPSession class uses several other classes to perform the necessary functions; it merely provides the links between the different components, without actually doing much by itself.

This design is a lot more object−oriented than the original one. This way, the structure of the library is better, the code is easier to read and the library can easily be extended.

### 8.2.2 Design decisions

The library is intended for the use of RTP in the TCP/IP architecture, so UDP is used to encapsulate the RTP packet. The network routines are implemented through the use of the Berkeley socket routines. These functions are available on most UNIX−like systems and also on MS−Windows platforms by using the WinSock library. Because I used these standard functions, the library can be used on a wide range of platforms.

Since the library is most likely to be used in real–time applications, I have tried to make the library as fast as possible. First of all, I have used hash–tables to store information that is likely to be consulted frequently. For example, the information about participants is stored this way. Also the list of destinations is stored using a hash–table. This allows the destination list to be adapted quickly. This is desirable for VoIP in virtual environments, where the destinations might change.

Calls to dynamically allocate memory are only done when absolutely necessary. For example, when you want to transmit data, a RTP header has to be attached to it. This means that some extra memory is needed. Instead of dynamically allocating it, I use a statically allocated buffer. The size of this buffer is set to the maximum size of an IP packet. The buffer is not declared in the send function, but in the appropriate class. This way, it only has to be created once.

I have also tried to make as little copies of data as possible. This is done by passing pointers to data instead of duplicating it. Finally, many functions are declared 'inline', which also makes several functions perform better.

For some applications, it might be desirable to receive data as soon as it comes in. Now, to receive data, the poll function has to be called. But how do you know when to call this function? To solve this problem, you can retrieve the used socket descriptors from the library. This way, you can use these descriptors in a call to 'select'[16], which can inform you when data is available.

### 8.2.3 Testing

During the first implementation of the library, I have used the 'rtpdump' utility to check if the library sent correct packets. The 'rtpdump' program was written by H. Schulzrinne, one of the creators of RTP. I have also let several applications send data to each other to see if they were processed correctly. Later, the library has also been tested in the VoIP programs I wrote, where it seemed to be working fine.

## 8.3 Publication and reactions

At the end of December 1999, I made the library available to the public through the World Wide Web. To announce this, I sent an e–mail to the mailing list 'rem–conf@es.net', where topics like VoIP are discussed. This mailing list was suggested by H. Schulzrinne, to whom I asked where I could best make the announcement.

Currently, several people are using the library. Thanks to their responses and suggestions, some improvements were made and the RTP library is now supported on many platforms. They also led to the removal of some bugs in the library.

To be able to give you an idea of what projects the library is being used in, I sent some e–mails to people who are using JRTPLIB, asking them what they were using the it for. Here are the responses I received:

- From Dave Osborne (dosborne@argoneng.com), I received the following reply:

  *"Argon Engineering Associates, Inc., located in Fairfax Virginia, is a systems engineering and development company providing full service information solutions to a wide range of customers. The company's primary business area is the design and development of communication systems that search, identify, and capture low*

16 The 'select' function is one of the standard socket functions. It can inform the user to which descriptors can be written or on which descriptors data is available.

*probability of intercept signals. This includes sensor development, data collection and decision support, and analysis and design of information retrieval and visualisation techniques. One of our guiding principles is to design systems around commercial off−the−shelf (COTS) hardware and software, best in class algorithms and custom software, using leading industry standards for common, flexible and expandable infrastructure. The JRTP library is used to fulfil the requirements for one of our systems to transport audio information via the Real−time Transport Protocol to a Java Media Framework (JMF) based player. After examining numerous RTP products, both commercial and open source, the JRTP library was selected for use; it provided the best interface for the functionality we desired, and easily met our performance criteria. The JRTP product has so far proven to be bug free, and we are very satisfied with it."*

- Dario Maggiorini (dario@dsi.unimi.it) sent me an extended abstract of the project he is participating in, called 'A Testbed Environment for the Performance Evaluation of Modular Network Architectures'. The group of people working on the project are developing a framework to test and develop protocols. This framework is in user space (instead of kernel space) to make working on protocols easier. Currently, the framework is being used to test QoS supporting protocols, of which RTP is used to achieve QoS in terms of jitter. It is JRTPLIB that provides the necessary RTP functionality. Preliminary results indicate that after a stream of packets has been shaped using information provided by RTP, the resulting jitter is negligible.
- The following reply is from Henry Lau (hlau@nuera.com):

  *"Nuera is developing an API function library to support enhanced telephony services over a SIP−based network. The library will be available on Windows NT and UNIX platforms. The JRTPLIB is used to provide RTP media to and from the SIP gateways. This includes playing voice prompts, recording voice and DTMF digit detection. The packets received by JRTPLIB are processed by a call controller object which in turn generates appropriate events for the higher layer telephony application."*

  From Nuera's World Wide Web site (www.nuera.com) I obtained the following information: Nuera Communications develops, manufactures and sells carrier–class IP telephony solutions that enable communications service providers to quickly migrate to converged networks and to effectively compete in the rapidly changing global communications service market.
- Simon Bicskey (simon@c–lab.de) told me that the library is being used in a project called 'PathFinder'. The C–Lab PathFinder is a small vehicle which can be controlled by people through a World Wide Web page. The user then receives a video stream originating from the camera on the vehicle. Simon sent me the following information:

  *"The project where JRTPLIB is used in is one of the part−projects of a large scale embedded system project, called PathFinder. This particular part−project is like an experiment in order to improve the quality of the video stream. It's working after the following scenario: there is a server with a service developed by Siemens and C−Lab in another project which can among others record and playback video streams on request. It can also receive on−the−fly generated streams and handle them as any other already stored stream. This software, called SmartPump, relies on RTSP, RTP, RSVP and others, and has a cache facility too. RSVP is being supported through a separate router with the necessary capabilities. In a computer there is a MPEG−Encoder card which generates the stream that will be sent to the server through RTP*

*using a RSVP−reserved bandwidth. JRTPLIB is used to handle the entire RTP/RTCP communication with the server and to send the data packets. The SSRC generated by the library is used to identify the data stream in the RTSP session. If we manage to achieve better visual performance (less flicker in the received stream) than the currently used architecture, then the results will be incorporated into the PathFinder project, in the way that the MPEG card will receive the output of the robot's camera."*

## 8.4 Summary

To be able to easily use RTP in several applications, I wrote a library which provides RTP functionality. The library is called JRTPLIB, which stands for "Jori's RTP Library". It is written in C++ using an object−oriented approach.

The library makes it easier to send and receive RTP packets. The user can select any number of destinations for the data. Multicasting can be used for efficient distribution of the data. The RTCP functionality is completely handled internally.

The structure of the library is very modular, which makes the source code easy to understand and easily extensible. The Application Programming Interface (API) is relatively simple to use. The use of standard socket functions, makes the library usable on a wide variety of platforms. Several techniques are used to make the library as fast as possible, which is desirable for real−time applications.

After using the library for a while, I decided to make it available on the World Wide Web. This has helped me to improve the support for several platforms.

# Chapter 9: A VoIP framework

After the creation of JRTPLIB, I started designing a VoIP framework, which is described in this chapter. I designed this framework because I wanted to be able to easily create several VoIP test applications. By separating the VoIP components from the rest of the application, this task is much simpler. Furthermore, I wanted to create it in such a way that would allow testing of different components, for example different compression techniques.

This work led to the C++ framework described in this chapter. First, I will present the general structure of this framework. Afterwards, its implementation is discussed.

## 9.1 Framework layout

The structure of the framework is depicted in figure 9.1. The VoiceCall class is the one which connects several components. As you can see, these components are pretty much the ones described in this thesis.

The SampleInput and SampleOutput classes are the ones responsible for grabbing and reconstruction of voice signals.

In the framework, the intervals for both capturing the voice signal and playback are assumed to be the same. The SamplingTimer class is responsible for indicating when this interval has elapsed.

With VoIP in virtual environments, there can be several persons speaking at the same time. The VoiceMixer component is responsible for mixing these signals together. This component gives the SampleOutput class a new block of speech data when an interval has elapsed, so it must make sure that everything is in the correct order.

The VoiceCompressor and VoiceDecompressor classes compress and decompress a block of speech data respectively.

For VoIP in virtual environments, the sender of speech data has to include some information about the position of the sender. This is done in the Location3D class. The actual 3D effects are added at the receiver side by the class Transform3D.

Somehow, the speech data will have to be transmitted and received. These tasks are the responsibility of the VoiceTransmitter class.

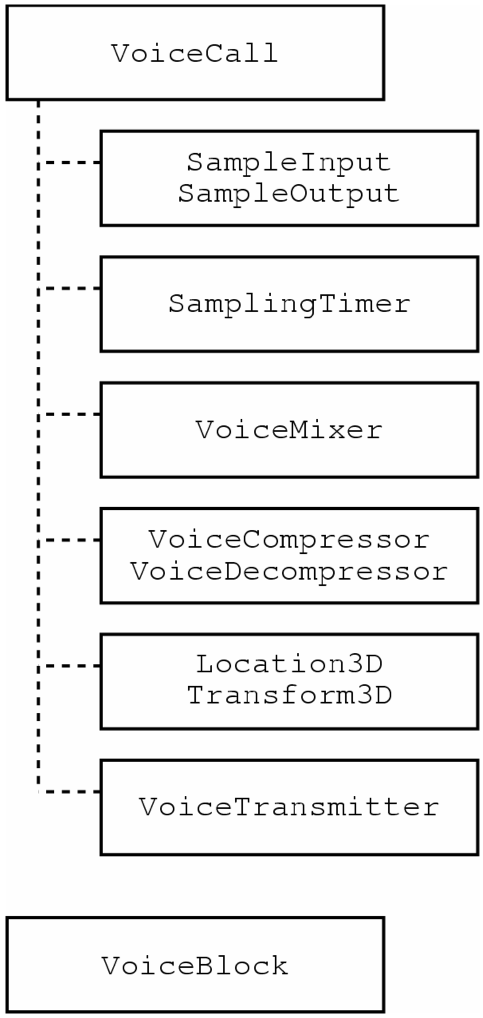


Figure 9.1 – VoIP framework layout

The VoiceBlock class is used as a container for voice data. An instance of this class is passed between the previously listed components to make VoIP possible. This class has member functions indicating properties of the data, for example the sampling rate, so that each component can process the data correctly.

In this basic framework, only the VoiceCall and VoiceBlock are really implemented, the other classes are abstract. Using inheritance, this allows us to try several different techniques.

For example, we can easily try several compression schemes by implementing them in classes inherited from VoiceCompressor and VoiceDecompressor.

## 9.2 Implementation

In this section I will cover the implementation of the framework. First, I will explain how the VoiceCall class uses the other components to make VoIP possible. Then, the components themselves are discussed.

### *9.2.1 Main VoiceCall routine*

The VoiceCall class contains several members to set up the different components. When this is done, the application only has to call the 'Step' member function to make VoIP possible. The outline of this function is depicted in figure 9.2 below.

```
VoiceCall::Step()
{
    // Check if the sampling interval has passed
    if (samplingtimer->HasTimeOut())
    {
        // Get a sampled block and start sampling again
        sampleinput->GetSampleBlock(&inputblock);
        sampleinput->StartSampling();

        // Get a block from the mixer and play it
        mixer->GetSampleBlock(&outputblock);
        sampleoutput->Play(&outputblock);

        // Restart the timer
        samplingtimer->RestartTimer();

        // Prepare block for transmission and transmit it
        if (add3Dinfo)
            location3d->Add3DInfo(&inputblock);
        if (compressblock)
            compressor->Compress(&inputblock);
        transmitter->SendBlock(&inputblock);

        // Adjust the current sample offset
        mixer->GetSampleOffset(&sampleoffset);
        transmitter->SetSampleOffset(sampleoffset);

        // Poll for incoming data
        transmitter->Poll();

        // Add input from the connection to the mixer
        for (each voice source)
        {
            while (this source has data available)
            {
                // Get some data and process it
                transmitter->GetSampleBlock(&block);
                if (decompressblock)
                    decompressor->Decompress(&block);
                if (add3Deffects)
                    transform3d->Create3DEffect(&block);

                // Send processed data to the mixer
                mixer->AddBlock(&block);
            }
        }
    }
}
```

Figure 9.2 – Main VoiceCall routine (pseudocode)

In principle, this function should be called continuously to assure that the necessary actions are taken at the exact point when the sampling (and playback) interval has elapsed. However, one could implement the abstract classes in such a way that the application only calls the ‘Step’ function when necessary. To give an example, you could let an implementation of the SamplingTimer send a signal to the application when an interval has elapsed. In turn, the application can then call the ‘Step’ function.

The pseudocode should be quite clear, but it may be necessary to explain what the sample offset is for. When voice data is received, it is put in a VoiceBlock instance. This will also hold information about when this data will have to be played, expressed in terms of samples. This timing information is used by the mixer to insert this data at the correct position in its output stream. When the mixer passes a block of data to the playback routine, the sample number of the first sample in its queue has changed. It is this value that is passed to the transmission component.

The current sample offset is needed each time a participant joins in. The timing information in the packets of this participant only provides information about when the data should be played, relative to the start of the participant’s transmission. To provide the correct timing information for the mixer, the current sample offset has to be added to the timing information in the packet.

There is another use for the current sample offset. After having calculated the timing information for a block of voice data, the transmitter can easily see if it is still possible to play this data: if the sample offset of the block is smaller than the current sample offset, it is useless to process the block any further since its playback time has already passed. The transmitter can then simply discard the data.

### 9.2.2 Grabbing and reconstruction

The grabbing and reconstruction routines are rather straightforward. They simply use the operating system’s capabilities to either record or playback speech data. I have made such routines for both Linux and MS−Windows platforms.

On both platforms, I have implemented SampleInput and SamplingTimer in one class, using multiple inheritance. This way, I could use the recording interval as timing information.

### 9.2.3 Mixing

At the end of each sampling interval, the mixer sends a block of data to the playback routine. For this reason, I have implemented the mixer using a linked list of such blocks. Initially, these blocks contain only silence.

Each time the mixer receives a block, it adds the data to the blocks in the list. Because of the principle of superposition mentioned in chapter three, the data simply has to be added to the data which is already present.

### 9.2.4 Compression schemes

Because of the structure of the framework, it was easy to try several compression schemes. The first scheme I implemented was the simple silence suppression technique which I described in chapter three. This silence suppression scheme was also used in the other compression methods which I implemented:

- a simple delta modulation scheme,
- a DPCM technique, and
- the wavelet coding method described in chapter four.

To my own opinion, the simple DPCM technique produced the most acceptable results. The delta modulation method simply could not reproduce the signal accurately enough. The wavelet coding technique could do this, but to achieve the same quality as the DPCM technique, the required bandwidth was comparable to that of DPCM.

I have also tested a Linear Predictive Coder, using a library which provided the LPC coding and decoding functions. The resulting communication quality was very good, but the speech signal did sound somewhat synthetic.

To allow somebody to choose a compression technique at run–time, I created a wrapper class for these compression modules. If all users in a session use that wrapper class as the compression and decompression module, each user can select the compression scheme he prefers.

### 9.2.5 Localisation effects

At the sender side, the sender's position was added to the data. This information was then used by a receiver, together with its own position, to create a localised effect. The technique used was described in chapter seven.

### 9.2.6 Transmission

The transmission module was implemented using RTP to send the data. The RTP functionality was provided by JRTPLIB, which made the implementation of this module both easier and faster. No additional work was done to provide QoS.

Because no QoS guarantees can be given, this module tries to stay synchronised as good as possible. When too many consecutive packets from a certain source have to be discarded, the module resynchronises with that source.

Also, as was mentioned in chapter three, the amount of buffering is determined using jitter information provided by RTCP packets. Buffering is simply done by adding some value to the timing information of a block with speech data. This will cause the mixer to insert the block further on in its linked list and this, in turn, will cause the data to be played back a bit later.

The module also filters out any duplicate packets. When the same data is passed to the mixer several times, that piece of the signal will sound much louder because of the principle of superposition. Such a discontinuity is very disturbing when VoIP is used in 3D environments.

## 9.3 Encountered problems

The only[17] real problem I encountered was with the original design of the framework. In the first version, there was not one single 'Step' routine. Instead, there was one routine which covered the input and one for the output. The timing had to be done by the SampleInput and SampleOutput modules. While this allowed the recording and playback intervals to be different, it was not possible to get these intervals low enough. The problem seemed to occur because the timing information provided by playback was not accurate enough.

I then started rewriting the framework. The timing function was put in a separate component and the routines for input and output were combined in one single 'Step' function. This is the structure which I still use today and I am now able to get the recording and playback interval as small as I want: even a one millisecond interval is no problem.

17 This is not entirely true. I have also spent a lot of time trying to figure out a bug which caused the input to disappear now and then. This problem was probably caused by an incorrect sound card driver. Because it is difficult to explain the exact problem and its workaround (which is quite nasty), I will not discuss it here.

## 9.4 Summary

To be able to easily create some VoIP test applications, I first developed an object–oriented VoIP framework in C++. This framework was also designed to try out several techniques to realise a specific VoIP component, for example different compression techniques. The structure of this framework reflects the components described in this thesis.

The basic framework contains a lot of abstract classes, representing VoIP components like the transmission or the playback component. Inheritance can then be used to actually implement a component. This is the mechanism which allows to try out several versions of a component.

# Chapter 10: VoIP test applications

Using the VoIP framework discussed in the previous chapter, I created several test applications. Some of them were only intended to test a particular component or tested the interaction of components by letting an application send voice data to itself.

Apart from these relatively simple test programs, I also created two more interesting applications. The first one is an Internet telephony application, the other one is a simple 3D environment. These applications are discussed in this chapter, but first there is a section that covers some issues which apply to both programs.

## 10.1 General issues

The way the VoIP related functions are handled is the same in both applications. When VoIP has to be made possible, the applications start a separate thread. In this thread, the necessary components are initialised, they are placed into a VoiceCall instance and then the Step function is called continuously. When VoIP is no longer required, the applications signals this to the thread, which then interrupts its loop and exits.

Using this continuous loop technique, it is almost certain that at the end of one sampling interval, the next one is immediately started. This is definitely the case when the system is not currently busy serving other demanding applications.

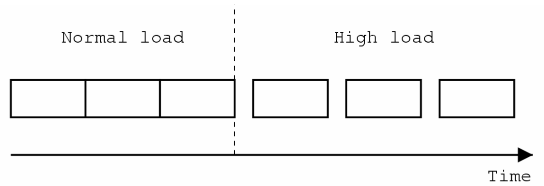


Figure 10.1 – Delay between sampling intervals

When the system is more heavily loaded, it is possible that there is a small amount of delay between the end of one interval and the start of the next. This is illustrated in figure 10.1. Since the total delay will keep building up, this will have some side effects for the communication.

For the recorded voice data, this can lead to gaps in the communication. When a packet is sent with RTP, the RTP timestamp is only increased with the amount of samples in the packet. This obviously does not take the added delay into account. Suppose that at a given time, there is a total amount $\delta$ delay caused by the high load of the system. If a packet's timestamp is T, the actual time that the packet was sampled, is $T+\delta$. The receiver will not know about $\delta$, so he will try to play the voice data in the packet at the time represented by timestamp T. But because there is a delay of at least $\delta$, it is possible that the playback time for the packet has already passed, causing a gap in the conversation.

The delay has also an effect on the playback of voice data. Because of the extra delay, the playback time of a packet will be later than it should be. This will increase the overall delay of the communication, which is very undesirable.

To solve these problems a check is performed inside the loop. It compares the actual elapsed time with the time interval represented by the sum of the sample intervals. This way it measures the extra delay caused by the load of the system. If this delay gets too large, the communication is initialised again[18].

18 In fact, the routine also checks if the difference is not too low. This is possible because of inaccuracies in the system's clock. These, in turn, will lead to incorrect measurements of the delay, so when the measured delay is sufficiently negative, the communication is also initialised again.

## 10.2 An Internet telephony application

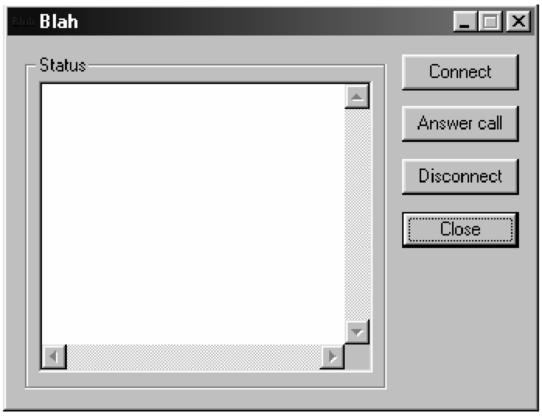


Figure 10.2 – An Internet telephony application

The first application which I made using the framework, was a simple Internet telephony application, of which only a MS–Windows version exists. The user–interface of the program is shown in figure 10.2.

When the user wants to make a call to somebody, he presses the 'Connect' button. Then, the application asks for the host to connect to and tries to establish a connection.

When there is an incoming call, this is signalled in the status window. The user can then answer the call by pressing the corresponding button.

The call is set up by a TCP connection. As long as this TCP connection exists between the two parties, they are able to talk to each other. When the TCP connection is torn down, the call is terminated. Note that the TCP connection is only a control connection and is not used to transfer the speech data. This is done with RTP, by the VoIP framework.

As was explained in the previous section, the application starts a separate thread when the VoIP part is needed. In the application, this is done as soon as the called person sends a confirmation over the TCP connection.

To my own opinion, the application worked quite well. The user–interface may not be very sophisticated, but the program allowed good quality conversation when sufficient bandwidth was present. The required bandwidth heavily depends on the compression scheme used.

## 10.3 A 3D environment

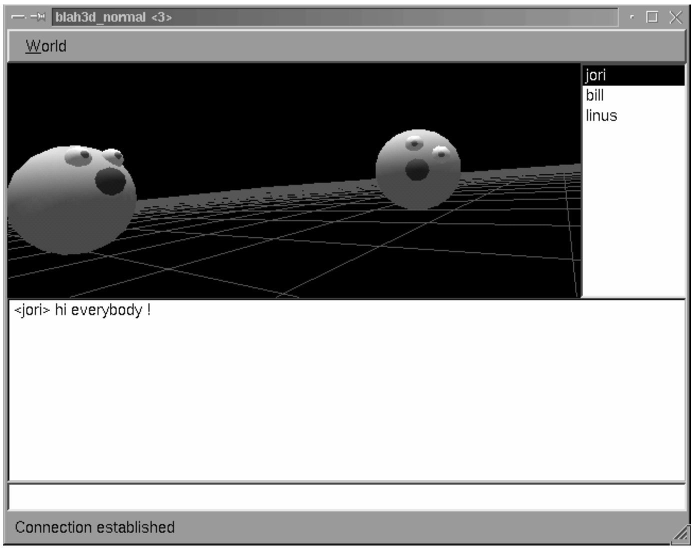


Figure 10.3 – A 3D environment

The previous application was an Internet telephony application, so there were only two persons communicating and there was no need for 3D effects. The application I created next did allow multiple participants and 3D effects.

Both a Linux and a MS–Windows version of the application exist. Figure 10.3 is a screenshot of the Linux version of the application; the MS–Windows version looks almost identical.

The user–interface contains three major parts. The bottom half of the application window is a chat interface. If the voice quality is not good enough, the participants can still communicate by sending text messages to each other. At the right hand side of the window, there is a window which shows the participants in the 3D environment. The remaining part of the application window shows this environment.

When the application is started, the user first has to join a specific 3D environment. This is done by establishing a TCP connection with a server application. Each participant in the environment will have one such connection with the server. These connections are used to transfer the text messages for the chat window and to distribute the positions of each participant. Like with the telephony application, these connections are not used for the transmission of the speech data; for that purpose RTP is used.

For the transmission of the speech data, a user can either use unicasting or multicasting. If multicasting is used, the server provides a multicast address to which the user's application should send the data. One multicast group is used for the whole environment.

To reduce the amount of processing, the application checks which other participants are within a specific range. Depending on the transmission method used – unicasting or multicasting – the program only sends speech information to participants in range, or only accepts speech data from participants in range.

The application demands a lot more processing power than the Internet telephony application. Because of this, the quality can sometimes be a bit lower, but usually I found it comparable to that of the telephony application. Like with the Internet telephony application, the required bandwidth depends a lot on the compression scheme which is used.

## 10.4 Summary

To test the VoIP framework I created several programs, among which are an Internet telephony application and a 3D environment. The VoIP part of these applications is put in a separate thread which continuously calls the VoiceCall member function 'Step'. Extra work is done to assure synchronisation between the participants.

The Internet telephony application is a relatively simple application which allows easy and good quality conversations over the Internet if enough bandwidth is available. The 3D environment application allows several persons to communicate with each other, with simple localisation effects being added to their speech signals. Both unicasting and multicasting can be selected to transmit the voice information. This application also allowed good quality communication. For both applications, the required bandwidth depends on the compression scheme which is used.

# Part IV: Conclusion

# Conclusion

In this document, I have explained how VoIP in virtual environments can be achieved and what factors play an important role. Additionally, I described my own work on this topic. Now, what are the conclusions that can be drawn from all this?

When we consider VoIP applications in general, they will probably become more widely used as time evolves. Currently, the main problem for such applications is the lack of QoS guarantees. When QoS supporting protocols like RSVP are used on a larger scale, this will certainly make VoIP more popular since people can then communicate with the quality that they desire. On LANs, where there is normally plenty of bandwidth, VoIP applications can already be used with little or no problems. However, on a larger scale, like the Internet, such QoS providing protocols will be necessary to make VoIP applications perform adequately.

In the computer industry, everything evolves very rapidly. Therefore, I assume that the available bandwidth on networks will keep getting larger. This will also be helpful for the spreading of VoIP applications. When the available capacity is sufficiently large, even high quality sound will be possible, which will certainly be a stimulus for the use of VoIP programs. Furthermore, since compression techniques are still improving, such high–quality communications will be available even sooner.

Standards like H.323 and SIP make interoperability between applications of different developers possible. This way, people can choose from a variety of VoIP applications and use the ones they like the most. In turn, this will stimulate the use of VoIP. Also, since more applications will be developed, the possibilities of these applications will keep growing and improving.

The use of VoIP as a telephony alternative can save quite some costs. Since voice and data traffic can be integrated, the necessary infrastructure to provide both services is reduced. This integration will also make better use of the available bandwidth: first of all, bandwidth on a network is rarely entirely filled with data traffic. Second, classic telephone calls waste a lot of bandwidth since this bandwidth is reserved for the two parties even when someone is not speaking. Making long distance telephone calls over the Internet or another IP network will also be cheaper than using the telephone network for this purpose.

For VoIP in networked virtual environments, there are certainly a lot of possible applications. Currently, there are not many programs which provide this functionality, but I definitely believe that this will change. When better quality can be guaranteed, such applications can be an attractive alternative to chat environments like IRC. As CPU power keeps growing and more dedicated hardware becomes available, better sound localisation will improve the realism of the virtual environment. This in turn will make VoIP in virtual environments even more attractive.

Some observations can also be made about my own development. The RTP library which I developed seems to be very useful in several applications. This is indicated by the many positive reactions which I received. The library proved to be both fast and simple to use. Because of its clear structure, new features can easily be added to it.

The VoIP framework also proved to be very useful. It easily allowed me to test several components and made the VoIP part of an application portable to several platforms. This is because normally only the digitisation and reconstruction components need to be rewritten for a new platform. The framework also allowed the rather fast creation of several VoIP test programs.

Among these programs, the Internet telephony application and the 3D environment are quite useful. When enough bandwidth is available, they allow good quality conversations. These programs also made me realise that VoIP has a lot of potential for future development.

# Appendices

# Appendix A: Literature

This appendix presents a list of the books, papers, World Wide Web pages etc., which I read to learn about VoIP in networked virtual environments. First, I will list the sources to which I referred in the thesis text. Afterwards I will give a list of other sources.

## A.1 References


[1] Duda R. O., **3–D Audio for HCI**, http://www–engr.sjsu.edu/~knapp/HCIROD3D/3D_home.htm , 1997

[2] Gardner W. G., **3D Audio and Acoustic Environment Modeling**, http://headwize.com/tech/gardner_tech.htm

[3] Schulzrinne H., Rosenberg J., **A Comparison of SIP and H.323 for Internet Telephony**, http://www.cs.columbia.edu/~hgs/papers/Schu9807_Comparison.pdf

[4] Gyssens M., **Algorithms and data structures (course notes)**, 1997

[5] Mitzel D.J., Estrin D., Shenker S., Zhang L., **An architectural comparison of ST–II and RSVP**, Proceedings of Infocom 1994, ftp://catarina.usc.edu/pub/mitzel/Infocom94/infocom94.ps

[6] DataBeam Corporation, **A Primer on the H.323 Standard**, http://www.databeam.com/h323/h323primer.html, 1998

[7] Dalgic I., Fang H., **Comparison of H.323 and SIP for IP Telephony Signaling**, http://www.cs.columbia.edu/~hgs/papers/others/Dalg9909_Comparison.pdf

[8] Casner S., Jacobson V., **Compressing IP/UDP/RTP Headers for Low–Speed Serial Links**, RFC 2508, 1999

[9] Tanenbaum A. S., **Computer Networks (third edition)**, Prentice Hall, 1996

[10] Minoli D., Minoli E., **Delivering Voice over IP Networks**, Wiley Computer Publishing, 1998

[11] Wolberg G., **Digital Image Warping**, chapters 2 and 4, IEEE Computer Society Press, 1990

[12] Schnitzler J., **Digital source coding of speech signals**, http://www.ind.rwth–aachen.de/research/speech_coding.html

[13] CIP Working Group, **Experimental Internet Stream Protocol, Version 2 (ST–II)**, RFC 1190, 1990

[14] Karim A., **H.323 and Associated Protocols**, http://www.cis.ohio–state.edu/~jain/cis788–99/h323/index.html, 1999

[15] Fellbaum K., Richter J., **Human Speech Production Based on a Linear Predictive Vocoder**, http://www.kt.tu–cottbus.de/speech–analysis/course.html

[16] McElroy C., **Hybrid Coding**, http://ee.mokwon.ac.kr/~jspark/rtp/speech2/hybrid.html, 1995

[17] Braden R., Clark D., Shenker S., **Integrated Services in the Internet Architecture: an Overview**, RFC 1633, 1994

[18] Fenner W., **Internet Group Management Protocol, Version 2**, RFC 2236, 1997

[19] Postel J., **Internet Protocol**, RFC 791, 1981

[20] Deering S., Hinden R., **Internet Protocol, Version 6 (IPv6)**, RFC 1883, 1995

[21] Deering S., Hinden R., **Internet Protocol, Version 6 (IPv6)**, RFC 2460, 1998

[22] Delgrossi L., Berger L., **Internet Stream Protocol Version 2 (ST2) Protocol Specification − Version ST2**, RFC 1819, 1995

[23] Smith K. A., Brushteyn D., **Internet Telephony and Voice Compression**, http://www.cis.upenn.edu/~kellyann/iphone.html

[24] Howitt W., **Linear Predictive Coding (LPC)**, http://asylum.sf.ca.us/pub/u/howitt/lpc.tutorial.html, 1995

[25] Bredehoft L., **LP Vocal tract model**, http://www.netrix.com/products/present/speech/sld009.htm

[26] Heath S., **Multimedia & Communications Technology**, Focal Press, 1996

[27] Wolfe J., **Physics in speech**, http://www.phys.unsw.edu.au/PHYSICS_!/SPEECH_HELIUM/speech.html

[28] Schulzrinne H., Rao A., Lanphier R., **Real−Time Streaming Protocol (RTSP)**, RFC 2326, 1998

[29] Braden R., Zhang L., Berson S., Herzog S., Jamin S., **Resource ReSerVation Protocol (RSVP)**, RFC 2205, 1997

[30] Schulzrinne H., Casner S., Frederick R., Jacobson V., **RTP: A Transport Protocol for Real−Time Applications**, RFC 1889, 1996

[31] Handley M., Schulzrinne H., Schooler E., Rosenberg J., **SIP: Session Initiation Protocol**, RFC 2543, 1999

[32] Signals and Software Ltd., **Speech Coder Summary**, http://www.sasl.com/data_sheets/SpeechCoderSummary.htm, 1999

[33] Woodard J., **Speech Coding,** http://www−mobile.ecs.soton.ac.uk/speech_codecs/index.html

[34] Stevens W. R., **TCP/IP Illustrated − Volume 1**, Addison−Wesley, 1994

[35] Department of Electronic Engineering (City University of Hong Kong), **Theory**, http://www.image.cityu.edu.hk/~stkng/theory.html

[36] Postel J., **Transmission Control Protocol**, RFC 793, 1981

[37] Fluckinger F., **Understanding networked multimedia; applications and technology**, Prentice Hall, 1995

[38] Young H. D., **University Physics**, chapters 20 and 21, Addison−Wesley, 1992

[39] Postel J., **User Datagram Protocol**, RFC 768, 1980

[40] IBM Networking White Papers, **Voice−Data Integration in e−business**, http://www.networking.ibm.com/voice/voice−data.html

[41] Black U., **Voice over IP**, Prentice Hall, 2000

[42] Larson E., Nikola S., **Voice technologies for IP and Frame Relay Networks**, http://www.mot.com/MIMS/ISG/mnd/papers/voice_technologies_for_ip_and_frame_relay_networks.html

[43] **Voice, Video and Home Applications Overview**, http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/voice_c/vcoview.pdf

[44] Stollnitz E. J., DeRose T. D., Salesin D. H., **Wavelets for Computer Graphics; Theory and Applications**, Morgan Haufmann Publishers, Inc., 1996

## A.2 Other sources


- Aureal Semiconductor, **A3D Technology Background**, http://www.aureal.com/tech/download/A3D_White_Paper_doc.pdf, 1998
- Martin K. D., **A Computational Model of Spatial Hearing**, ftp://sound.media.mit.edu/pub/Papers/kdm–ms–thesis.ps.Z, 1993
- Bolot J. C., Fosse–Parisis S., **Adding Voice to Distributed Games on the Internet**
- Martin K. D., **Estimating Azimuth and Elevation from Interaural Differences**, ftp://sound.media.mit.edu/pub/Papers/kdm–mohonk95.ps.Z, 1995
- IP Multicast Initiative, **Higher Level Protocols used with IP Multicast**, http://www.ipmulticast.com/community/whitepapers/highprot.html, 1997
- Gardner B., Martin K., **HRTF Measurements of a KEMAR Dummy–Head Microphone**, ftp://sound.media.mit.edu/pub/Data/KEMAR/hrtfdoc.ps, 1994
- Hussain A., **Image Compression Using Neural Networks**, http://www.csc.umist.ac.uk/issl/research/abir/index.html, 2000
- Feng L., **Internet Telephony**, http://www.cis.ohio–state.edu/~jain/cis788–97/internet_telephony/index.htm, 1997
- Liu C., **Multimedia Over IP: RSVP, RTP, RTCP, RTSP**, http://www.cis.ohio–state.edu/~jain/cis788–97/ip_multimedia/index.htm, 1998
- Mills D., **Network Time Protocol (Version 1)**, RFC 1059, 1988
- Schulzrinne H., **RTP Profile for Audio and Video Conferences with Minimal Control**, RFC 1890, 1996
- Schulzrinne H., **Some Frequently Asked Questions about RTP**, http://www.cs.columbia.edu/~hgs/rtp/faq.html, 1999
- Atal B. S., Jayant N. S., **Speech Coding**, http://clsu.cse.ogi.edu/HLTsurvey/ch10node4.html
- Robinson T., **Speech compression techniques**, comp.speech FAQ (section Q3.1), http://svr–www.eng.cam.ac.uk/comp.speech/Section3/Q3.1.html, 1996
- **Speech Production and Acoustic–Phonetics**, http://murray.newcastle.edu.au/users/staff/speech/home_pages/tutorial_acoustic.html
- Parsiani H., **Video Compression Using Neural Networks and Fuzzy Logic**, http://mayaweb.upr.clu.edu/cecord/projects/video.html, 1996

# Appendix B: List of figures

# Appendix C: Abbreviations

| | |
|---|---|
| 3D | Three Dimensional |
| AbS | Analysis−by−Synthesis |
| ACELP | Algebraic Codebook Excited Linear Prediction |
| ADM | Adaptive Delta Modulation |
| ADPCM | Adaptive Differential Pulse Code Modulation |
| APCM | Adaptive Pulse Code Modulation |
| API | Application Programming Interface |
| ATM | Asynchronous Transfer Mode |
| BGP | Border Gateway Protocol |
| CELP | Codebook Excited Linear Prediction |
| CNAME | Canonical Name |
| CS−ACELP | Conjugate Structure Algebraic Codebook Excited Linear Prediction |
| CSRC | Contributing Source |
| DCT | Discrete Cosine Transform |
| DFT | Discrete Fourier Transform |
| DM | Delta Modulation |
| DoD | Department of Defence |
| DPCM | Differential Pulse Code Modulation |
| FTP | File Transfer Protocol |
| HRIR | Head−Related Impulse Response |
| HRTF | Head−Related Transfer Function |
| HTTP | Hypertext Transfer Protocol |
| IGMP | Internet Group Management Protocol |
| IETF | Internet Engineering Task Force |
| IID | Interaural Intensity Difference |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version four |
| IPv6 | Internet Protocol version six |
| IRC | Internet Relay Chat |
| ISDN | Integrated Services Digital Network |
| ISO | International Standards Organisation |
| ITD | Interaural Time Difference |
| ITU | International Telecommunication Union |
| JMF | Java Media Framework |
| JRTPLIB | Jori's RTP Library |
| LAN | Local Area Network |
| LD−CELP | Low Delay Codebook Excited Linear Prediction |
| LPC | Linear Predictive Coding |

| | |
|---|---|
| MCU | Multipoint Control Unit |
| MOS | Mean Opinion Score |
| MP–MLQ | Multipulse Maximum Likelihood Quantisation |
| MPE | Multipulse Excited coding |
| MTU | Maximum Transfer Unit |
| N–ISDN | Narrowband Integrated Services Digital Network |
| NTP | Network Time Protocol |
| OSI | Open Systems Interconnection |
| OSPF | Open Shortest Path First |
| PCM | Pulse Code Modulation |
| RAS | Registration, Admission and Status |
| RELP | Residual Excited Linear Prediction |
| RPE | Regular Pulse Excited coding |
| RPE–LTP | Regular Pulse Excitation – Long Term Prediction |
| RSVP | Resource Reservation Protocol |
| RTP | Real–time Transport Protocol |
| RTCP | RTP Control Protocol |
| RTSP | Real–Time Streaming Protocol |
| QoS | Quality of Service |
| SCMP | ST Control Message Protocol |
| SDES | Source Description |
| SDP | Session Description Protocol |
| SIP | Session Initiation Protocol |
| SMTP | Simple Mail Transfer Protocol |
| SSRC | Synchronisation Source |
| ST2 | Stream Protocol Version 2 |
| TCP | Transmission Control Protocol |
| TTL | Time To Live |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| VoATM | Voice over ATM |
| VoFR | Voice over Frame Relay |
| VoIP | Voice over IP |
| VSELP | Vector Sum Excited Linear Prediction |
| WAN | Wide Area Network |