Distribuerat 3d-ljud i virtuella miljöer – realistisk ljudsättning av avatarer

PONTUS NYMAN

KTH Numerisk analys
och datalogi

Examensarbete
Stockholm, Sverige 2005

TRITA-NA-E05111

Numerisk analys och datalogi
KTH
100 44 Stockholm

Department of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, Sweden

Distribuerat 3d-ljud i virtuella miljöer – realistisk ljudsättning av avatarer

PONTUS NYMAN

TRITA-NA-E05111

Examensarbete i datalogi om 20 poäng
vid Programmet för teknisk fysik,
Kungliga Tekniska Högskolan år 2005
Handledare på Nada var Sten-Olof Hellström
Examinator var Yngve Sundblad

Sammanfattning

Traditionella VR-system har ofta ett enkelt och bristfälligt stöd för uppspelning av ljud. Det kan röra sig om ett system för uppspelning av små ljudklipp som lagras på en server. Vill man ha möjlighet att använda distribuerat ljud, så kallade ljudflöden, får man istället använda ett fristående konferenssystem. Detta fungerar väl som ett kommunikationssystem. Problemet är att konferenssystemet inte har någon koppling till VR-systemet och avatarernas positioner. Någon geometrisk koppling till ljudet i form av riktnings- och avståndssimulering är därför inte möjlig. Dessutom är de flesta konferenssystem designade för låga bitflöden. Ljudkvaliteten är därför ofta undermålig.

Istället för att ljudet bara ska tillhandahålla en enkel kommunikation borde det användas för att simulera rumsligt ljud och på så sätt öka upplevelsen. Ett första steg mot detta är att simulera riktning och avstånd med hjälp av virtuella positioner. Ett andra steg är att med hjälp av geometrin även simulera rumsakustik. På så sätt hjälper ljudet till att förhöja VR-upplevelsen och dessutom öppnar man upp för ny funktionalitet i systemet.

I det här examensarbetet har jag undersökt möjligheten och även tagit fram en prototyp för ett 3d-ljudsystem för VR-applikationer. Ett sådant system är i praktiken ett multikanalskonferenssystem där varje ljudflöde, varje kanal, har en virtuell position kopplad till VR-systemet. Med hjälp av dessa ljudflöden, deras virtuella positioner och en multikanalsljudanläggning kan en klient simulera avatarens ljudupplevelse. Hög ljudkvalitet och låg latens har varit viktiga kriterier.

Prototypen består av en server och en klient. Servern håller reda på uppkopplade klienter och deras avatarer och distribuerar relevanta ljudflöden. Den mest komplicerade komponenten i systemet är dock klienten. Den sänder och tar emot ljudflöden, antingen via servern eller direkt mot en annan klient. För att diverse nätverksbegränsningar, som fördröjning och paketförluster, inte ska orsaka för stora problem använder klienten ett flertal olika tekniker, till exempel kompression och olika typer av maskering av paketförluster. Både server och klient har testats och utvärderats och båda har visat sig fungera mycket bra.

Distributed 3d-sound in virtual environments

realistic sound simulation for avatars

Abstract

The traditional VR research has focused mainly on graphics and interactivity. Sound has often been neglected or at most very simple with few features. The approaches have varied from simple server based sample clips to monophonic conference systems with low bitrate codecs for distributed speech. The sound quality of these systems is often poor. External conference systems usually don't have a connection to the VR scene graph which makes it impossible to make use of the positions of the avatars. At the best this gives a working communication system.

Instead of just using distributed speech as a communication system it should be used to increase the virtual experience. This is how graphics have been used since the beginning. The first step to do this with distributed sound is to use virtual positions to spatialize sound sources at each avatar. This will simulate sound direction and distance. The next step is to use geometrical data from the scene graph to simulate acoustics as well. Using sound and sound streams this way is more than just having a communication system. This will enhance the VR experience and also add new functionality for interactivity.

In this master thesis I have examined the possibility to use distributed sound to enhance VR experience. I have for this purpose developed a prototype for adding distributed 3d-sound to VR applications. The system is basically a distributed multichannel conference system with 3d positions linked to each channel. The system is also prepared for acoustic simulation. High sound quality and low latency have been important design criteria.

The prototype consists of a server and a client. The server is quite simple. It monitors connected clients and their respective avatars and distribute sound streams. The client is far more complicated. Basically it sends and receives sound streams, either thru a server or by connecting another client directly. To deal with the effects of network delay and packet loss the client uses a number of more or less advanced methods, for example compression and loss recovery techniques. Both the server and the client have been tested and evaluated and have shown to work very well.

Förord

Den här rapporten är resultatet av ett examensarbete som utfördes på NADA, institutionen för numerisk analys och datalogi, på KTH under hösten 2003 och vintern 2004/2005. Arbetet gjordes på uppdrag av Tools-studion på Interaktiva Institutet i Umeå (http://www.interactiveinstitute.se). Min handledare där var Peter Lundén. Handledare på NADA var Sten-Olof Hellström och examinator var Yngve Sundblad. Examensarbetet var en del av kultursimulatorprojektet.

Uppdraget bestod i att utveckla ett ljudsystem av server-klient-typ för distribution av ljudflöden. Tidigt förstod jag att examensarbetet handlade om såväl en generalisering som en specialisering av konferenssystem och andra typer av VoIP[1]. En generalisering i meningen att klienten hanterar varje ljudkälla separat, ett multikanalsystem istället för ett monosystem. En specialisering i meningen att servern utför en mer reglerad distribution av ljudflöden, istället för att ”alla skickar till alla” som i traditionella konferenssystem. Denna insikt medförde att mycket tid har ägnats åt att studera och implementera lämpliga tekniker från VoIP-forskningen.

Det har varit en mycket lärorik period och trots att projektet inte genomfördes som planerat blev prototypen över förväntan lyckad. Det finns fortfarande en del att göra och förbättra (se avsnitt 8.3), men prototypen är idag helt funktionsduglig.

[1] Voice over IP

Innehåll

Figurer

Kapitel 1

Inledning

Med denna introduktionsdel har jag för avsikt att beskriva bakgrunden till examensarbetets problem samt att försöka klarlägga varför det är intressant. Jag kommer att introducera det centrala begreppet 3d-ljud och även kort förklara begreppet avatar. Slutligen tänker jag kortfattat berätta om det projekt som detta arbete var en del av.

1.1 Bakgrund

Traditionellt sett inom VR[1] har ljudupplevelsen allt som oftast hamnat i skuggan av bildupplevelsen. Målet med bilden har ända sedan VR-teknikens och datorgrafikens begynnelse oftast varit att ge användaren en så realistisk synupplevelse som möjligt. Ljudet har till största delen handlat om att tillhandahålla en fungerande kommunikation. En realistisk ljudåtergivning med 3d-ljud och simulerad rumsakustik har inte prioriterats.

Anledningarna till denna bortträngda position kan tänkas vara flera. För det första så kräver en realistisk ljudåtergivning en del hårdvara som åtminstone inte tidigare varit speciellt spridd eller tillgänglig. Ljudkort till datorer och multikanalsljudanläggningar för att nämna två exempel. Idag har alla nya datorer stöd för ljud och hemmabio har blivit ett begrepp. Dessutom så finns det metoder för att åstadkomma 3d-ljud för vanliga hörlurar.

Den viktigaste anledningen har dock inte med teknik att göra. Jag tror att ljudets roll för att förhöja realismen och närvarokänslan i en VR-miljö har varit, och fortfarande är, underskattad. Synupplevelsen är direkt och tar stor plats, medan ljudupplevelsen till sin natur är mer subtil och svårfångad.

1.1.1 Vilka system finns idag?

Det finns idag ett stort antal konferenssystem och olika typer av VoIP-applikationer. Några exempel är Vat, RAT (Robust Audio Tool), NeVoT, Microsoft NetMeeting,

[1] Virtual Reality

Linphone, GnomeMeeting och Skype. En del av dessa är slutna, andra har öppen källkod. Problemet med alla dessa är att de inte är avsedda att användas i ett VR-sammanhang, de har inget stöd för 3d-ljud och i allmänhet är ljudkvaliteten bristfällig.

RAT, som nämns ovan, har visserligen ett slags 3d-ljudsystem som lokaliserar ut de olika konferensdeltagarnas röster i rummet, men detta system går inte att styra eller att ansluta till någon VR-motor. RAT använder denna funktionalitet enbart i syfte att öka hörförståelsen då flera deltagare pratar samtidigt.

Efter noggranna efterforskningar har jag inte funnit ett enda system med de krav och den funktionalitet som min uppdragsgivare söker.

1.2 3d-ljud

Den enklaste formen av ljudåtergivning är den monofona. Detta innebär att ljudet endast återges i en kanal. För en lyssnare betyder det att ljudet tycks komma från en enda punkt i rummet. Stereoljud återges med hjälp av två kanaler. Detta gör det möjligt att placera den virtuella ljudkällan längs en linje i rummet. Tredimensionellt ljud, eller 3d-ljud, är den naturliga utvidgningen av ovanstående. I idealfallet ska lyssnaren kunna uppleva att ljudet kommer från vilken riktning, och vilken punkt, i rummet som helst. I praktiken har dock de flesta 3d-ljudsystem begränsningar. Till exempel brukar man prioritera det horisontella planet eftersom hörselsinnets förmåga att lokalisera ljud i höjdled är dålig. För att åstadkomma 3d-ljud kan man använda antingen ett multikanalssystem med många högtalare eller ett par vanliga stereohörlurar. Dessa två kräver helt olika typer av metoder.

Använder man ett multikanalssystem handlar det om att hålla isär olika ljudkällor i skilda kanaler och sedan ge varje högtalare en egen unik mixning. Hur ljuden ska mixas för varje högtalare beror på avatarens orientering, ljudens virtuella position och högtalarplaceringen. Det finns många olika typer av högtalarplacering. De kommersiella surroundsystem som idag används i såväl biografer som i hemmet är en enklare form av 3d-ljud. De använder oftast sex kanaler: två bak, tre fram och en subbas. Högtalarna är oftast placerade i ett plan och kan därför inte placera ljudet i höjdled. Det ljudsystem som jag använder för mitt examensarbete består av åtta högtalare med en högtalare i varje hörn av rummet.

För att åstadkomma 3d-ljud med hörlurar använder man ”binaural reproduktion” av ljudet. Genom att simulera de effekter som hjärnan utnyttjar för att lokalisera ljudkällor i rummet uppnår man på så sätt 3d-ljud med bara två kanaler. I första hand handlar det om att simulera den nivåskillnad och tidsdifferens som öronen upplever när de fångar upp ljudet från en ljudkälla. Dessa variationer beror på den lilla avståndsskillnad till ljudkällan för respektive öra som uppstår. Det går också att modellera de reflektioner och den filtrering som kroppen och framför allt huvudet och öronen ger. För detta används något som kallas HRTF[2]. En mer detaljerad genomgång av detta finns i [1].

[2]Head-Related Transfer Function

1.3 Vad är en avatar?

Eftersom jag ibland använder mig av avatarbegreppet kommer jag här att definiera vad jag menar med det. Ordet avatar kommer ursprungligen från hinduism och betecknar där en gud förkroppsligad i en människas eller ett djurs gestalt. Under 90-talet populariserades begreppet av science fiction-författaren Neal Stephenson och det kom att betyda ungefär "en inkarnation i Virtual Reality". Idag används ordet i många olika datorsammanhang och begreppet är därför något urvattnat. Ofta syftar man på de bilder och figurer i sig som markerar en användare eller dennes sinnesstämning i chattprogram och forum på Internet.

I VR-sammanhang är en avatar en representation av en användare i den virtuella världen. Det är inte bara den visuella representationen som åsyftas, utan hela användarens representation. För mig är det förstås viktigt att även rösten eller ljudet inkluderas.

1.4 Kultursimulatorprojektet

Kultursimulatorn (går även under den mystiska benämningen I3MS) är ett projekt av Tools-studion på Interaktiva Institutet i Umeå. Projektet handlar om att skapa en internetbaserad distribuerad och interaktiv miljö där människor kan träffas och lära sig om varandras kulturer. Den tekniska utrustningen till projektet består av en tvåväggs VR-kub med ett tillhörande 3d-ljudsystem. Projektet består av två delar där första delen är avklarad. Den gick av stapeln våren 2003 i St Petersburg. Till stadens 300-års jubileum sattes ett slags rollspel i gränslandet mellan datorspel och teater upp. Titeln var "The Key to Saint Petersburg" och handlade om hur stadens nycklar som stals av svenskarna återlämnades.

Nästa del av projektet sker i Barcelona 2004 på Universal Forum of Cultures. Till dess ska systemet förbättras. Min uppgift är att förbättra ljudsystemet som idag består av ett monofont konferenssystem. Avsikten med examensarbetet är dock inte att bara förbättra ljudsystemet i just det här projektet, utan att bygga ett så generellt system som möjligt.

För (aningen) mer information om kultursimulatorprojektet hänvisar jag till hemsidan, http://www.tii.se/tools/projects/i3ms/.

Kapitel 2

Problemdefinition

Här definierar jag själva problemet och försöker också klarlägga de olika delproblemen. Jag beskriver sedan vad jag uppfattar som målet med arbetet. Sist diskuterar jag kriterier och avgränsningar.

2.1 Precisering av problemet

Problemet för examensarbetet består i att designa, implementera, testa och utvärdera ett fristående realtidssystem för distribuerat 3d-ljud i VR-miljö. Ljudsystemet ska mynna ut i en prototyp för vidare utveckling. Det går att dela upp examensarbetet och prototypen i flera mindre delproblem.

2.1.1 Distribution av ljudflöden

Systemet ska distribuera avatarers och andra ljudkällors ljudflöden (ljudeffekter, tal, etc) över nätverk. Varje avatar prenumererar på de ljudflöden, de kanaler, som den kan höra. För att distributionen ska fungera bra krävs det ett väl genomtänkt protokoll. Ju fler avatarer som är närvarande i VR-miljön, desto mer data kommer att belasta nätverket. Att distribuera varje avatars ljudflöde till alla andra blir snabbt ohållbart. Nätverkstrafiken måste därför reduceras för att så många avatarer som möjligt ska kunna närvara. Detta är ett problem av kombinatorisk karaktär.

Datorkommunikation över nätverk garanterar inte att all data kommer fram till mottagaren. Därför behövs det ett system som på något sätt maskerar sådan borttappad data. Metoder för detta kommer jag att lägga stor vikt vid eftersom det är en fundamental del av ljudkvaliteten i systemet.

Att åstadkomma en väl fungerande distribution är själva kärnan i ljudsystemet och examensarbetets huvuduppgift. I kapitel 3 går jag igenom den för problemet relevanta nätverksteorin och diskuterar de protokoll som används i prototypen. Kapitel 4 tar upp olika metoder för att hantera nätverksbegränsningar som paketförluster och latens.

2.1.2 Kompression av ljudflöden

När man väl har lyckats reducera antalet distributionskedjor i så stor utsträckning som möjligt går det att minska nätverkstrafiken ytterligare genom att utnyttja kompression. Det finns en mängd olika algoritmer, eller kompressionsformat, för detta ändamål. Denna del av projektet går ut på att utvärdera, jämföra och använda dessa för att reducera datamängden i varje ljudflöde.

I kapitel 5 diskuterar jag olika typer av kompression och tar upp de mest kända formaten av varje sort.

2.1.3 Realistisk ljudsättning av avatarer

När en avatar tar emot de ljudflöden som den prenumererar på ska den spela upp dessa i ett 3d-ljudsystem för högtalare. Själva ljudmixen för 3d-ljudet sker i det fristående systemet Snd3d. Där kan man lätt ställa in den aktuella högtalaruppsättningen vilket gör det totala systemet mycket hårdvaruportabelt. Denna del av problemet handlar om att integrera prototypen med Snd3d.

Mer om detta i kapitel 7 där jag diskuterar själva implementationen av systemet.

2.1.4 Problem vid in- och uppspelning

Vid inspelning finns en del problem att ta hand om. Om en avatar är tyst och ljudflödet bara innehåller brus eller låga bakgrundsljud ska detta inte skickas vidare över nätverket. Detta för att minska nätverkstrafiken mellan avatarerna. Det behövs därför någon metod för att avgöra om användaren är tyst eller inte, alltså att skilja till exempel lågt tal från brus.

Vid uppspelning krävs någon form av buffertar, dels för att få ett jämnt dataflöde till ljudkortsdrivrutinen och dels för att ge distributionssystemet tid att sortera oordnade paket och hantera borttappade paket. Inga eller för små buffertar ger ett ”klickande” ljud då drivrutinen ofta blir utan data. Stora buffertar ger visserligen bättre ljudkvalitet, men försämrad interaktivitet i systemet på grund av hög latens.

I kapitel 7 diskuteras detta mer i detalj.

2.2 Målet med examensarbetet

Som jag ser det finns det två mål med examensarbetet. Det ena är att komplettera befintliga VR-system med realistisk ljudsättning av avatarer för att ”förhöja upplevelsen”. Jag är medveten om att detta är ett löst och otydligt begrepp. Vad som menas med upplevelser i VR och hur man mäter och jämför sådana är något som jag inte tänker ta upp. Jag nöjer mig med att löst definiera en ökad upplevelse inom VR som en större sinnlig och koncentrationsupptagande process.

Det andra målet är att ge möjlighet till utökad funktionalitet hos VR-applikationer. Ett målande exempel på hur ny funktionalitet uppstår är om vi tänker oss en konferens i VR-miljö. I det här systemet skulle man kan luta sig nära sin bordsgranne och viska något som bara den personen kommer att höra i sitt vänstra

öra. Detta samtidigt som en annan talare håller ett föredrag som alla närvarande kan höra. I befintliga konferenssystem är endast en monofon ljudkanal verksam. Ett sådant viskningsscenario är då inte möjligt att genomföra.

2.3 Kriterier för prototypen

Det finns några grundläggande kriterier, eller krav, som prototypen bör uppfylla. Dessa kriterier ska genomlysa hela arbetet med prototypen. Vid de olika val av lösningsmetoder som hela tiden dyker upp ska den metod som på bästa sätt uppfyller kriterierna väljas. Anledningen till att ställa upp dessa kriterier är att hela tiden arbeta mot rätt mål.

2.3.1 Hög ljudkvalitet

De flesta konferenssystem, traditionella och befintliga, är konstruerade för låg bandbredd. Därför använder de oftast äldre kompressionstekniker för låga bitflöden (<30 kbps[1]). Vid så låga bitflöden blir ljudkvaliteten dålig.

Jag har istället valt att utnyttja modernare och mer högkvalitativa format. Dessa är visserligen mer bandbreddskrävande, men numera är bredband så pass spritt att det är möjligt att skicka ljud med höga bitflöden (>64 kbps). I det projekt där prototypen i första hand ska användas är dessutom tillgången på bandbredd inget problem.

2.3.2 Låg latens

Latensen är den tid det tar från det att ljud genereras hos sändaren (till exempel genom att tal spelas in med en mikrofon) till det att ljudet når fram till mottagarens hörselsensorer (till exempel genom att höra tal från högtalarna). En distribuerad och interaktiv realtidsapplikation kräver låg latens, mindre än 250 millisekunder [6]. Blir latensen större leder detta till svårigheter i kommunikationen. I ett konferenssammanhang yttrar sig detta genom att deltagarna omväxlande pratar i munnen på varandra och tyst väntar på att någon annan ska tala.

2.3.3 Generellt och utbyggbart

Genom att klokt strukturera upp programkoden och separera olika avgränsade delar blir möjligheten till att återanvända kod större. Ett modulärt tänkande vid programmeringen gör det lättare att byta ut eller lägga till olika typer av funktionalitet, till exempel olika kompressionsformat.

2.3.4 Plattformsoberoende programmeringskod

Med plattformsoberoende programmeringskod menas i första hand att systemet ska fungera väl på de tre stora operativsystemen: Microsoft Windows, Mac OS X och

[1] kilo bit per second

Linux. Kompilerar koden på dessa tre är chansen stor att programmet fungerar på en mängd andra UNIX-system. Detta ställer krav på vilka externa kodbibliotek som är lämpliga för uppgiften.

2.4 Avgränsningar

På grund av ämnets omfattning och examensarbetets ganska snäva tidsram så beslöt jag tidigt att inte låta prototypen få bli allt för stor. Målet var att ha en fungerande prototyp med den mest grundläggande funktionaliteten.

Ett intressant och viktigt problem som därför inte tas upp i rapporten är hantering av återkoppling. I ett högtalarbaserat system blir det lätt rundgång då ljudet från uppspelningen går in i mikrofonen. Detta kan göra det svårt att uppfatta vad som sägs över nätverket. Det är ett komplicerat problem som det forskats mycket kring. Den mesta forskning har dock handlat om en- och tvåkanaliga system. För ett multikanalsystem som i mitt fall blir det ett ännu mer komplicerat problem.

Akustiksimulering vid ljudsättning av avatarer är ett mycket intressant forskningsområde som även det är allt för stort och omfattande för att jag ska kunna ta upp det under examensarbetets ramar.

Kapitel 3

Nätverksarkitektur och protokoll

I det här kapitlet kommer jag kortfattat att förklara de aspekter av nätverksarkitektur och nätverksteori som är relevanta för att distribuera ljudflöden för direktuppspelning. För en mer grundläggande genomgång av olika sorters nätverk och Internet hänvisar jag till annan litteratur, till exempel [4] och [13]. Jag börjar med en mycket kortfattad sammanfattning av IP och Internet.

3.1 IP och Internet

Det vi idag kallar för Internet är egentligen en stor mängd olika sorters nätverk. Dessa är sammankopplade i ett enda virtuellt nätverkssystem där hård- och mjukvara tillsammans ger illusionen av ett enhetligt nätverk. Den viktigaste hårdvaran för detta är routern som kopplar samman olika nätverk med varandra. Grunden i mjukvaran är protokollet IP[1]. All kommunikation bygger på att datorer skickar och tar emot små datapaket, Internet är alltså vad man kan kalla för ett paketväxlingssystem. IP har som huvuduppgift att möjliggöra en ”best effort” leverans av paket mellan datorer i det virtuella nätverket.

Med begreppet ”best effort” menar man att protokollet på bästa sätt försöker leverera paket mellan sändare och mottagare. Förbindelsen är dock inte helt pålitlig. Det finns en sannolikhet att paket anländer i oordning, att det kommer två identiska paket eller att paket förloras helt. Det beror inte på någon brist i protokollet utan är en medveten och fundamental del av systemet. Man vill nämligen att nätverket ska ha plats för så mycket trafik som möjligt. Skulle protokollet då garantera helt pålitliga paketleveranser skulle detta medföra en påtaglig minskning av den totala bandbredden. Genom att tillåta en viss osäkerhet i överföringen har man nått en kompromiss. Som vi ska se går det att uppnå pålitliga överföringar på applikationsnivå i alla fall.

För att åstadkomma denna kommunikation använder IP ett eget hårdvaruoberoende adresseringssystem med IP-adresser. Varje dator får en egen unik adress. När två datorer på skilda fysiska nätverk kommunicerar går trafiken genom en eller flera

[1] Internet Protocol

routrar. I routrarna finns tabeller som uppdateras automatiskt med information om nästa hopp, nästa router, för att paketet ska nå sin destination.

3.2 Transportprotokoll

IP omger sig med en hel familj av olika protokoll för olika syften. Bland dessa finns transportprotokollen. Deras uppgift är, helt enligt namnet, att transportera paket över det kommunikationsnät som IP spänner upp. Det finns ett flertal olika, men de två vanligaste som man brukar tala om är TCP och UDP.

3.2.1 TCP

TCP står för Transmission Control Protocol. Det är ett transportprotokoll som erbjuder pålitliga dataöverföringar. TCP garanterar att paket kommer fram, att de kommer i rätt ordning och att identiska eller skadade paket plockas bort. För att åstadkomma detta använder TCP ett system med återsändningar av förlorade paket. Sändare och mottagare håller reda på vad som kommit fram med hjälp av en korrespondens av speciella bekräftelsepaket, populärt kallade ”ackar”. När sändaren skickar ett paket startas en timer. Om ingen ack har kommit tillbaka från mottagaren när timern går ut ses paketet som förlorat och en återsändning sker. För att utnyttja bandbredden mer effektivt ser TCP till att sändaren skickar flera paket i följd innan den väntar på en ack.

Det finns även ett slags system med flödeskontroll. TCP håller reda på hur stora paketförlusterna är genom hålla räkning på antalet icke returnerade ackar. När dessa blir för stora minskar protokollet utflödet av paket. Detta utnyttjas även åt andra hållet, det vill säga paketutflödet ökar upp till en viss gräns då antalet förlorade paket och överföringshastighet nått en kompromissnivå. Eftersom de flesta förlorade paket beror på trafikstockningar i routrar fungerar denna teknik bra. Det bästa sättet att undvika sådana stockningar är att adaptivt justera dataflödet.

TCP är det mest pålitliga sättet att transportera data över Internet, men det finns trots allt en liten sannolikhet för mottagaren att erhålla korrupt data. Ett paket som skickas med TCP innehåller en 16-bitars checksumma. Stämmer inte denna checksumma hos mottagaren skickas ingen ack och återsändning sker. En rimlig uppskattning är att ungefär ett paket av 5000 är korrupt. Eftersom checksumman är 16 bitar så kommer i medeltal ett korrupt paket av 65536 felaktigt att klassas som korrekt. Detta ger att ungefär ett paket av 300 miljoner som skickas med TCP kommer att vara korrupt trots att mottagren accepterar det. Det har argumenterats för att den nuvarande checksumman är för kort och bör förlängas [9].

Att TCP är det populäraste och mest använda transportprotokollet är inte så konstigt eftersom de flesta applikationer prioriterar pålitliga dataöverföringar. Däremot är TCP inte speciellt lämpat för att transportera realtidsdata. Anledningarna till detta är flera. För det första så kan återsändningar, väntetider och sent ankomna paket leda till oacceptabla fördröjningar i nätverkstrafiken. För det andra så ger flödeskontrollen ett ojämnt dataflöde som lämnar luckor i realtidstrafik. För det

tredje så kräver protokollets komplicerade beteende ett ganska stort pakethuvud (24 bytes används av TCP) i varje paket. Denna extra information leder till ineffektiv utnyttjande av bandbredden.

Trots alla dessa problem så är det inte ovanligt att applikationer för konferens och VoIP använder TCP för att transportera ljudflöden.

3.2.2 UDP

UDP, eller User Datagram Protocol, erbjuder inte något av den funktionalitet vi hittar i TCP. UDP är egentligen bara ett gränssnitt mellan IP och applikationslagret och utnyttjar enbart den enkla överföringsmekanism som finns inbyggd i IP. Det som UDP ger som inte IP stöder är möjligheten att samtidigt använda flera separata överföringskanaler med hjälp av så kallade portar. Dessutom innehåller även UDP en checksumma på 16 bitar. De problem som listades för TCP finns inte. Andra fördröjningar än de som IP står för existerar inte eftersom protokollet inte använder återsändningar eller flödeskontroll. UDP använder bara 4 bytes i pakethuvudet.

3.2.3 Unicast och multicast

Det finns två olika sätt att utnyttja UDP. Man kan säga att protokollet omfattar två paradigmer. Det som skiljer dessa är sättet att se på sändare och mottagare.

Det vanligaste sättet att använda UDP är att, precis som med TCP, skicka data mellan en sändare och en mottagare. Finns det flera mottagare får sändaren helt enkelt skicka samma paket flera gånger. Detta kallas för att skicka data över UDP med unicast.

Ett annat sätt att överföra data på är att en sändare skickar data till en grupp med mottagare. Detta kallas multicast. Skickar man data med UDP i multicastläge finns det ett godtyckligt antal mottagare. IP ger stöd för detta. Det finns speciella IP-adresser avsedda för multicasting. Tyvärr så är endast en liten del av den 32-bitars adressrymd som IP (version 4) förfogar över avsedd för detta. En mottagare gör ett aktivt val som abonnent av en viss multicastadress.

Med hjälp av multicasting kan en sändare med liten bandbredd skicka dataflöden till ett stort antal mottagare utan att belasta processor och det lokala nätverket alltför mycket. Den egentliga distributionen från multicastadress till enskilda datorer sker i routrar. Det finns dock en stor begränsning. Multicast är fortfarande en ganska ny teknik. För att allt ska fungera måste alla routrar mellan sändare och mottagare ha stöd för multicasting. Så är ännu inte fallet. Därför finns Mbone.

MBone står för IP Multicast Backbone och var från början ett forskningsprojekt för att testa applikationer med multicaststöd. Det bestod av en mängd nätverk som stödde multicast genom tunnling[2]. Sedan dess har det vuxit stort och idag brukar MBone benämnas som det nätverk i nätverket som stöder multicast på Internet. Det består både av routrar med riktigt stöd för multicast och av tunnlade lösningar.

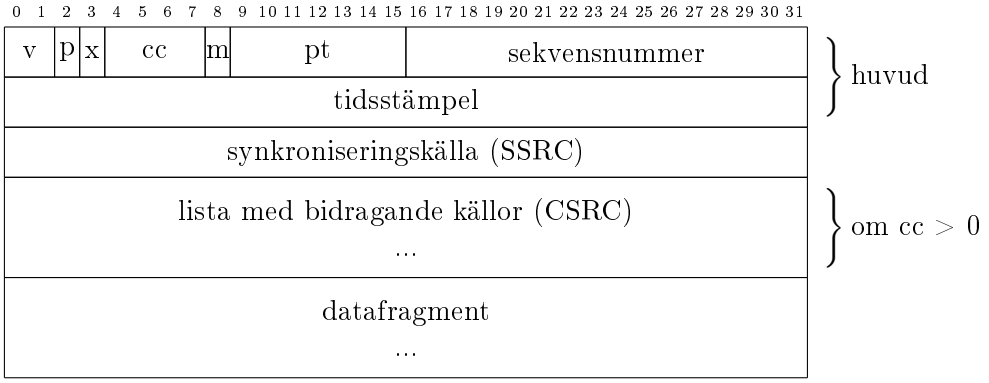
[2]Tunnling innebär i det här fallet att två routrar med multicaststöd kommunicerar via en router utan detta stöd genom att kapsla in multicast IP-paket i vanliga unicast IP-paket. Den mottagande routern med multicaststöd packar upp paketet till sitt ursprungliga skick.

3.3 Ett protokoll för realtidsdata

Jag har argumenterat för att TCP inte är anpassat för att skicka realtidsdata. Nu går det även att konstatera att UDP ensamt inte heller räcker till. Att skicka realtidsdata med UDP är helt enkelt för opålitligt. Sändaren kan inte veta om de olika paketen i dataflödet kommer fram i ordning, eller om de ens kommer fram överhuvudtaget. Det finns heller inte möjlighet att ta reda på hur stor del av överföringen som går förlorad. En del av den funktionaliteten som TCP erbjuder vore att föredra.

3.3.1 Dataöverföring med RTP

För att tillgodose de krav som finns på överföring av realtidsdata har RTP [12], Real-time Transport Protocol, utvecklats. RTP är ett högnivåtransportprotokoll som ligger mellan nätverkets transportprotokoll och applikationen. Protokollet är designat för att vara oberoende av såväl nätverksprotokoll som transportprotokoll, men oftast används det tillsammans med IP och UDP. I praktiken är RTP ett pakethuvud och ett kontrollsystem för övervakning av nätverkstrafik. Protokollet i sig erbjuder inte någon funktionalitet för pålitliga överföringar[3].



Figur 3.1. RTP-huvudet.

Ett RTP-paket innehåller ett huvud, se figur 3.1, med ett antal olika fält, både obligatoriska och valfria. Därefter följer själva datadelen. Denna innehåller ett fragment av realtidsdata, till exempel 20 millisekunder av tal. För att ytterligare belysa protokollets funktionalitet kan det vara informativt att gå igenom de mer intressanta fälten i pakethuvudet. För en mer detaljerad beskrivning hänvisar jag till specifikationen [12].

[3] I engelsk litteratur används begreppet Quality of Service.

- version (v): 2 bitar
  Detta fält talar om vilken version av RTP som paketet innehåller. Den nuvarande versionen är 2. Version 0 och 1 syftar på tidiga prototyper av protokollet.

- extension (x): 1 bit
  Denna bit markerar om huvudet är förlängt.

- datatyp (pt): 7 bitar
  Detta fält talar om vilket format, vilken typ av data, som följer efter det fasta huvudet. Är det ett ljudflöde som transporteras via RTP talar detta fält till exempel om hur ljudet är kodat.

- sekvensnummer: 16 bitar
  Sekvensnumret ökar med ett för varje paket som skickas. Med hjälp av sekvensnumret kan mottagaren upptäcka om paket har förlorats, om samma paket kommit flera gånger eller om paketen anländer i oordning. Mer om mekanismerna bakom detta längre ned.

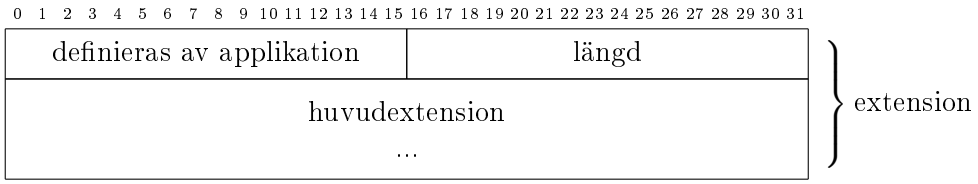
- tidsstämpel: 32 bitar
  Tidsstämpeln anger tidsangivelsen av den första oktetten i datadelen. Tiden mäts i valfritt format, men det krävs dock att den ökar monotont och linjärt och att den på något sätt är kopplad till datainnehållet. Handlar det om ljudflöden är samplingsklockan en lämplig tidsinstans. Stämpeln ökar med ett för varje dataenhet, samplingar när det gäller ljud, oavsett om data skickas eller av någon anledning slängs bort. Detta fält används hos mottagaren för att beräkna en jitterparameter (se avsnitt 3.4.2) och kan också användas för synkronisering av olika slag, till exempel läppsynk vid animering.

- synkroniseringskälla (ssrc): 32 bitar
  Detta fält identifierar avsändaren. Varje källa har ett eget unikt nummer.

- datafragment: upp till storleken av MTU[4]
  Här följer sedan ett fragment av data ur flödet. För ljudapplikationer vanligtvis 20 millisekunder komprimerat ljud.

Vill man lägga till funktionalitet som inte redan finns representerad i pakethuvudet kan man efter det fasta huvudet sätta in en huvudextension, se figur 3.2. Meningen är att man bara ska använda detta för funktionalitet som är oberoende av dataformatet. All annan extra information ska skickas tillsammans med datainnehållet.

[4]MTU står för Maximum Transmission Unit och är den största datamängd som kan transporteras mellan två noder utan att fragmentering sker. Detta värde kan variera för olika nätverk, men 1500 bytes har blivit ett standardvärde för Internet.



Figur 3.2. En RTP extension.

3.3.2 Kontroll av dataflöden med RTCP

RTCP, RTP Control Protocol, är en utvidgning av RTP i form av ett eget protokoll. RTCP har flera olika funktioner. Den primära är att ge applikationen gensvar på överföringskvalitet mellan sändare och mottagare. Detta sker genom ett utbyte av speciella rapporteringspaket. Aktiva sändare skickar sändarrapporter, medan de källor som inte sänt på ett tag skickar mottagarrapporter. Alla deltagare i RTP-sessionen skickar rapporter till alla de andra. För att detta inte ska kräva alldeles för mycket bandbredd så måste rapporteringsfrekvensen styras av deltagarantalet för den pågående sessionen. RTCP måste därför själv hålla reda på deltagarantalet.

Den respons som rapporterna ger kan användas till en rad olika funktioner. Till exempel är de användbara för att adaptivt kontrollera kompressionsgraden hos dataflödet. I en konferenssituation där tal överförs kan man då ställa ned kvaliteten på det överförda talet om onormalt mycket paket skulle gå förlorade. På så sätt minskar storleken på varje paket och trafikstockningar hos mellanliggande routrar löses förhoppningsvis upp. Mer om detta i avsnitt 3.4.1 om paketdynamik och förluster.

Rapporteringspaketen används också för att synkronisera tidsstämplar med en referensklocka. Har man flera olika typer av trafik, till exempel både ljud- och videoflöden, som har olika tidsskalor för sina tidsstämplar krävs det att dessa då och då synkroniseras. Annars kan den ena tidsskalan halka efter den andra och det kan bli en situation när till exempel ljud och bild hamnar mer och mer ur synk.

RTCP innehåller även stöd för en typ av rapporter som innehåller olika beskrivningar av en deltagare. Dessa rapporter kallas SDES, Source description, och kan innehålla bland annat namn, email, telefonnummer och allmänna noteringar. Det finns även en speciell SDES-typ som kallas CNAME. Denna ska vara unik för varje deltagare och kan till exempel bestå av IP-adressen. SDES används för att lättare kunna identifiera olika deltagare i en RTP-session.

3.4 Paketdynamik och överföringsproblem

Paketväxlande nätverk har flera fördelar. De gör det möjligt att på ett enkelt sätt hantera kommunikation mellan olika typer av nätverk. De möjliggör också en flexibel och dynamisk fördelning av bandbredd. Paketens dynamiska natur ger dock upphov till en del problem och begränsningar.

Mycket forskning har bedrivits om paketförluster, fördröjning och paketdynamik under de senaste femton åren. En hel del av denna sammanfattas i [2], [3] och [9]. Man har studerat nätverksbeteende experimentellt med hjälp av simulering genom att följa testpaket och man har även försökt bygga matematiska modeller av paketväxlande nätverk. Dessa modeller har varit ganska grova, men ändå kunnat förutsäga testresultat någorlunda kvalitativt.

Problemet med Internet är att det hela tiden växer och förändras, det blir mindre och mindre homogent. Det förändras kraftigt både över lång tid, med nya protokoll och användningsområden, och på kort sikt över olika tider på dygnet på grund av tidsskillnader mellan länder. Internet är ett gigantiskt dynamiskt system som blir allt svårare att teoretiskt modellera. Även experimentella undersökningar är problematiska. Oftast görs statistiska undersökningar mellan vissa noder på nätet och med en viss typ av testpaket, till exempel små UDP-paket. En sådan undersökning kan dock inte sägas representera ”riktig” nätverkstrafik, utan säger egentligen bara att ”den här typen av trafik, mellan dessa noder, den här tiden på dygnet, den här dagen ser ut så här”. Det går dock att skönja vissa tendenser i trafiken och naturligtvis är det dessa som tester av den här typen försöker klarlägga. Det är viktigt att ha detta i åtanke när man läser siffror om hur ”bra” nätet fungerar, hur stor del av paketen som går förlorade etc.

3.4.1 Paketförluster

Paketförluster är en av de mest påfallande begränsningarna med paketväxlande nätverk. Anledningen till att paket ibland går förlorade är följande: Mellan en sändare och en mottagare finns ett flertal routrar. I varje router finns en buffert av FIFO-typ[5]. Om en router utsätts för hög pakettrafik kan det hända att denna buffert blir full. Inkommande paket till en sådan router slängs helt enkelt bort och går förlorade. Ett sätt att minimera paketförlusterna vore förstås att göra bufferten större, men detta har en baksida. En större buffert gör att fördröjningen vid pakettransporten ökar och detta är inte önskvärt.

De flesta mottagare upplever i medel paketförluster på 2-5% [10]. Dessa uppträder oftast som enskilda förluster, men till och från försvinner flera på varandra följande paket. I medel är dock antalet i rad förlorade paket litet, särskilt vid låg eller normal nätverkstrafik. Ibland uppträder förluster på ett stort antal i rad följande paket, mer än femtio paket i rad är inte otänkbart. Sådana förlustströmmar sker främst vid mycket hög nätverksintensitet.

För en konversation med tal över nätverk går paketförluster på upp till 5% att tolerera, därefter börjar det bli svårt att göra sig förstådd. Med hjälp av olika tekniker för att dölja paketförluster (sådana diskuteras utförligt i kapitel 4) går det att höja denna toleransnivå upp till 15% [10]. Denna siffra gäller för små talfragment (4-40ms).

[5] First in first out, en kö alltså.

3.4.2 Nätverksfördröjning

Nätverksfördröjningen[6] är den tid det tar för ett paket att transporteras från sändare till mottagare. Vägen där emellan går via ett antal olika noder, eller routrar. Den totala fördröjningen utgörs alltså av summan av de fördröjningar som uppkommer för varje sådan delsträcka. Tittar man närmare på vad denna delfördröjning består av så hittar man två komponenter.

Den ena komponenten är en konstant fördröjning som är den tid det tar för själva utbredningen genom noden och länken till nästa nod. Denna fördröjning utgörs av den fysikaliska transporten av signalen och dessutom den tid det för routern att behandla ett paket. Routern hämtar destinationsadressen från pakethuvudet, söker med hjälp av denna i routingtabellen för att hitta adressen till nästa "hopp", flyttar paketet till utporten och skickar det vidare. Denna tid anger i princip den minimala nätverksfördröjningen för den aktuella rutten.

Den andra komponenten är en variabel fördröjning som kommer från den tid som paketet befinner sig i nodkön. Det är denna variabla komponent av nätverksfördröjningen som är den intressanta komponenten för nätverksapplikationer. Det är den som går att påverka och det är den som talar om nätverkets befintliga tillstånd. Denna variabla nätverksfördröjning anges ofta som ett värde som kallas för jitter. Jitter brukar definieras som standardavvikelsen av differensen av sändningstiden mellan sändare och mottagare. Detta ger oss alltså ett mått på routervägens högst tillfälliga tillstånd när det gäller buffertköer och därmed också trafikintensiteten.

3.4.3 Övriga problem

Som det nämndes tidigare så kan paket anlända i oordning till mottagaren. Anledningen till detta är att paket kan ta olika vägar mellan sändare och mottagare. IP är konstruerat för att vara dynamiskt och stabilt. Om en nod i det virtuella nätverket försvinner ska detta inte påverka nätverket i stort. Istället tar pakettrafiken en annan väg. Det går inte att anta att två på varandra följande paket tar samma väg. Olika vägar ger också olika nätverksfördröjningar. Därför kan ett paket passera ett tidigare skickat paket genom att ta en annan väg med lägre fördröjning.

Det finns även en viss sannolikhet att mottagaren får två identiska paket. Detta beror på fel i routrar.

Dessa problem uppträder endast då UDP används (ur applikationens synvinkel vill säga). Genom att inkludera ett sekvensnummer i pakethuvudet och räkna upp detta för varje paket som skickas så är det lätt att identifiera dem. På så sätt kan mottagaren slänga bort dubbletter och sortera oordnade paket i en lokal buffert. Som vi har sett använder RTP sådana sekvensnummer.

[6] I engelskspråkig litteratur benämns denna som end-to-end delay.

Kapitel 4

Hantering av nätverksbegränsningar

I det här kapitlet ska jag beskriva olika metoder för att hantera och begränsa effekterna av olika nätverksbegränsningar. I huvudsak handlar det om paketförluster och latens och vad man kan göra för att minimera effekten av dessa. Först diskuterar jag vad sändaren av data kan göra för att hjälpa mottagaren att hantera paketförluster, därefter vad mottagaren kan göra själv. Sedan följer ett avsnitt om olika sätt att allokera resurser i nätverk för att ge bättre överföringar. Slutligen diskuteras latens och vad man kan göra för att minimera denna.

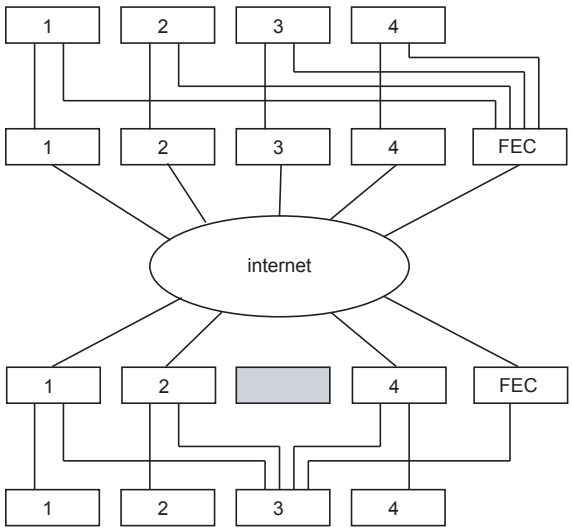
4.1 Sändarbaserad hantering av paketförluster

Genom att preparera den data som ska skickas över nätverk eller skicka med olika typer av redundant information kan sändaren hjälpa till att maskera paketförluster hos mottagaren. Det finns många olika sätt att göra detta på. Jag tänker gå igenom några av dessa.

4.1.1 Återsändning

Återsändning av paket är kanske det enklaste och den första sändarbaserade åtgärden man kommer att tänka på. Metoden går helt enkelt ut på att sändaren vid behov skickar om hela paket. Det går att göra på flera sätt. Till exempel kan sändaren hålla reda på överföringskvaliteten med hjälp av ett protokoll som RTP och skicka återsändningar oftare när paketförlusterna är stora och mer sällan när de är små. Ett annat sätt är att utnyttja ett mer TCP-baserat system där sändare och mottagare skickar ackar och använder timers för att på så sätt bara skicka återsändningar när det behövs. Det är förstås möjligt att använda TCP som överföringsprotokoll om man vill ha så pålitliga överföringar som möjligt.

Om man inte håller på med interaktiva realtidssystem är återsändningar, förmodligen med hjälp av TCP, den bästa lösningen som löser alla problem med paketförluster. Men annars är detta knappast en användbar teknik i någon form. Återsändningar tillför en alldeles för stor fördröjning för att de ska vara användbara i en interaktiv miljö.



Figur 4.1. Principen för felkorrigerande kodning.

4.1.2 Felkorrigerande koder

En teknik som härstammar från datakommunikationens barndom är felkorrigerande koder[1]. Då tillämpade man detta på bitflöden för att upptäcka och korrigera fel på enskilda bitar. När det gäller dataflöden över nätverk finns det idag många olika algoritmer för att använda sådan felkorrigering på hela paket.

Felkorrigerande kodning fungerar genom att man använder flera dataenheter, bitar eller som i detta fall paket, och genererar en ny dataenhet, en slags feldata. En felkorrigerande kod appliceras bitvis, alltså den i:te biten i felkoden genereras från den i:te biten i varje ingående dataenhet. Hur många dataenheter som ingår i en felkod beror på den algoritm som används. Om man ska skicka n stycken dataenheter och en kodning baseras på k dataenheter så kommer man att behöva skicka n - k extra dataenheter med felkoder.

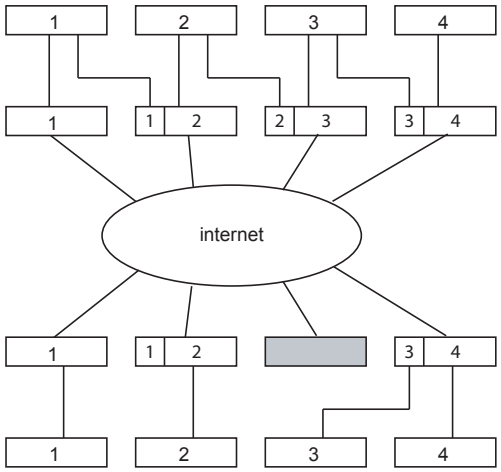
Om en dataenhet sedan försvinner så kan en felkod användas för att återskapa den del som gått förlorad. Förloppet åskådliggörs i figur 4.1.

Två vanligt förekommande algoritmer i nätverkssammanhang är paritetskodning och Reed-Solomon kodning. Paritetskodning använder sig av en enkel exklusiv-eller (XOR) operation som appliceras på grupper av paket. En begränsning är att det bara går att återskapa ett förlorat paket per sådan paritetsgrupp. Det finns ett flertal olika typer av paritetskodning.

Reed-Solomon kodning är en mer avancerad algoritm som bygger på egenskaper hos polynom för vissa nummerbaser.

En fördel med felkorrigerande koder är att de är oberoende av sin data. De går att tillämpa på alla typer av dataflöden, såväl ljud- som videoflöden. Till nackdelarna

[1] I engelsk litteratur kallas det forward error correction (FEC).



Figur 4.2. Principen för redundant återsändning.

hör att kodningen introducerar en viss extra fördröjning, hur stor denna blir och om den utgör ett större problem beror på vilken algoritm man använder. Det allvarligaste är dock att felkodningen tillför ganska mycket ny data. När det gäller införandet av reparationsdata och annan redundant data ska man ha i åtanke att denna kommer att öka dataflödet till routrarnas buffertar. Detta kommer i sin tur att leda till ökade paketförluster. En teknik som baseras på införandet av redundant information motverkar alltså i viss utsträckning sitt eget syfte. Med andra ord är det viktigt att minimera mängden redundant reparationsdata, för upp till en viss gräns gör denna trots allt nytta.

Felkorrigerande koder ställer krav på bandbredden. De är också ganska känsliga för större paketförluster, och då främst förlustströmmar då många paket i rad går förlorade. I en sådan situation är den korrigerande koden överflödig och till ingen nytta.

4.1.3 Redundant återsändning

En variant av återsändning är att skicka ett och samma ljudfragment med flera olika paket, tillsammans med andra ljudfragment. Det går till så att man först skickar en primär version av ett ljudfragment. Detta skickas på vanligt sätt över nätverket. Därefter i ett senare paket skickas en sekundär version av ljudfragmentet i samma paket som ett nytt primärt fragment. Förloppet åskådliggörs i figur 4.2.

Det redundanta, sekundära fragmentet kodas vanligtvis med mycket lägre ljudkvalitet än det primära. Anledningen till detta är dels att ett paket inte rymmer hur mycket data som helst och dels att man inte vill föra in allt för mycket ny data av samma anledning som i fallet med felkorrigerande koder.

Genom att utnyttja information från mottagaren, till exempel med hjälp av RTCP, går det att anpassa mängden redundant data till den aktuella mängden

paketförluster. Det finns också anledning att styra avståndet mellan primära och sekundära ljudfragment [6]. Skickas de i två på varandra följande paket är tekniken känslig för paketförluster. Detta beror på att om paket n går förlorat så är sannolikheten att paket n + 1 också går förlorat något större än medelsannolikheten för paketförlust. Genom att separera primära och sekundära fragment mer blir metoden bättre på att hantera paketförluster, dock på bekostnad av en något större fördröjning eftersom mottagaren måste ha en längre buffert för att vid behov kunna nyttja det sekundära fragmentet. Mottagaren har dock redan en viss buffertlängd för att kunna sortera paket som anländer i oordning. Det krävs förmodligen inte något mer utöver denna.

Fördelen framför andra med denna teknik är att den inte leder till några större fördröjningar. Fördröjningen ges av det maximala avståndet mellan primära och sekundära fragment.

4.1.4 Interfoliering

Interfoliering innebär att man använder små ljudfragment, mindre än ett normalt fragment som vanligtvis brukar vara 20 millisekunder. Dessa subfragment ordnas därefter på ett sådant sätt att intilliggande subfragment separeras för att kunna skickas i olika paket. Till exempel kan man använda subfragment med längden 5 millisekunder och lagra 4 stycken i varje paket. I det första paketet lagrar man då, som visas i figur 4.3, subfragment 1, 5, 9 och 13. I det följande paketet hamnar subfragment 2, 6, 10 och 14, och så vidare. När paketen når mottagaren ordnas subfragmenten i nummerordning.

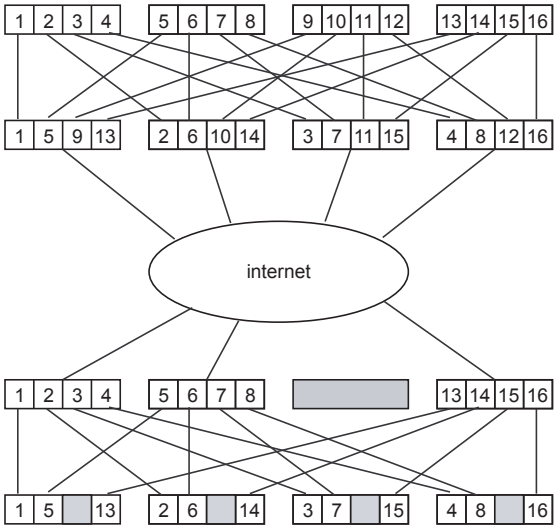
Om ett paket går förlorat försvinner visserligen lika mycket data som om varje paket innehöll ett enda normalt 20 millisekunders fragment. Skillnaden är att istället för ett stort hål blir det fyra mindre. Genom att sprida ut förlusten blir det lättare att maskera effekterna av det borttappade paketet. Ett stort hål i ljudflödet kan vara av stor skada om det är tal som transporteras. När hålet närmar sig längden av ett fonem[2] (5-100 millisekunder) försämras förståelsen av tal drastiskt [10] och dessutom fungerar mottagarbaserade metoder för paketförluster (vi ska snart titta närmare på dessa) bättre för små luckor i ljudflödet.

Interfoliering har den fördelen jämfört med tidigare beskrivna metoder att det inte krävs någon extra data alls. Däremot blir paketen fördröjda hos sändaren eftersom dessa innehåller subfragment från en lång tidssekvens, sändaren blir tvungen att hålla en något längre buffert än vid till exempel återsändning för att kunna ordna om fragmenten.

4.2 Mottagarbaserad hantering av paketförluster

Om inte sändaren och mottagaren använder TCP eller någon TCP-baserad kommunikation så kommer det att uppstå paketförluster. Som vi har sett kan dessa på olika

[2]Ett fonem är talets minsta beståndsdel.



Figur 4.3. Principen för interfoliering.

sätt minimeras och döljas genom olika sändarbaserade tekniker, men det kommer ändå att uppstå hål i dataflödet på grund av paketförluster. Dessa hål kan på olika sätt döljas genom att ersätta det förlorade fragmentet med någon slags ersättande data. Det finns en mängd olika sätt att producera sådan ersättningsdata.

Till skillnad från de sändarbaserade teknikerna som i de flesta fall var oberoende av sin data, så är samtliga av de metoder jag här ska ta upp enbart applicerbara på ljudflöden.

4.2.1 Enkla substitutioner

Den enklaste ersättningsmetoden för mottagaren är att fylla hålet från det förlorade paketet med tystnad. Detta fungerar dock bara vid mycket låga paketförluster (<2%) och små hål (<4ms). Metoden kan möjligen appliceras med någorlunda bra resultat tillsammans med interfoliering, men egentligen finns det ingen anledning att använda detta.

En teknik som visat sig fungera förvånansvärt väl är att fylla hålet med brus. Hjärnan har en förmåga att reparera ofullständiga ljudupptagningar av framför allt tal. Denna egenskap fungerar vid brussubstitution, men inte vid utfyllnad med tystnad [10].

Dessa enkla substitutioner har den fördelen att de är lätta att implementera och att de inte introducerar någon nämnvärd fördröjning. Däremot ger de inget vidare resultat annat än vid mycket goda nätverksförhållanden. Båda metoderna är ganska spridda och används i allt för stor utsträckning än idag.

4.2.2 Repetition

Om ett eller flera fragment i rad går förlorade kan man ersätta dessa luckor i ljudflödet med kopior av fragmentet innan förlusten. Detta kallas för paketrepetition. Metoden har visat sig vara klart bättre än de enklare substitutionerna i fråga om ljudkvalitet. I mobiltelefoner för GSM-nätet används denna metod. Vid dålig mottagning kan man ibland höra paketrepetitionerna ganska väl.

Om flera fragment i rad förloras så blir den subjektiva ljudkvaliteten ännu bättre om man gradvis tonar ned ljudstyrkan hos de repeterade enheterna. Även detta utnyttjar GSM. GSM repeterar de första 20 millisekunderna vid paketförlust med samma amplitud och tonar sedan ned de följande repeterade signalerna till amplituden 0 över 320 millisekunder. Detta system har visat sig vara en bra kompromiss mellan de enkla substitutionerna och mer komplexa metoder.

4.2.3 Interpolationsbaserad och regenerationsbaserad substitution

Det finns ett flertal olika metoder som går ut på att utnyttja paketen före och efter en paketförlust. Genom att interpolera dessa på olika sätt kan man återskapa mer originallik och fogfri ersättningsdata. Detta kräver betydligt mer datorresurser än de ovan beskrivna metoderna.

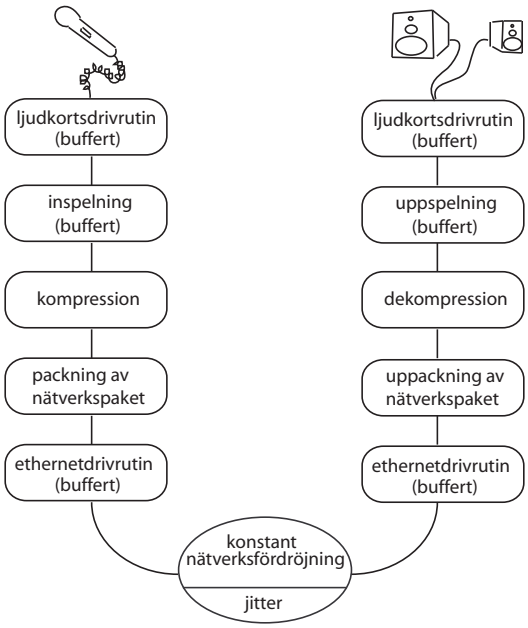
Regenerationsbaserad substitution använder tillståndsdata och parametrar från en kompressionsalgoritm för att återskapa ett borttappat ljudfragment. I en del kompressionsformat finns sådana metoder inbyggda. De är ofta beräkningsintensiva och tar mycket resurser från processorn.

4.3 Resursallokering i nätverk

Ovanstående metoder bygger på att applikationen är medveten om nätverket och dess begränsningar. Effekten av dessa behandlas lokalt hos sändare och mottagare. Ett helt annat sätt att skapa pålitliga överföringar är att allokera resurser längs nätverksrutten mellan sändare och mottagare. Ett flertal protokoll har föreslagits för detta syfte. Det mest utvecklade och spridda protokollet för IP är RSVP[3] [15]. RSVP ger möjlighet att begära lämpliga resurser av nätverket. En del applikationer kräver pålitliga överföringar med få paketförluster, men har inga krav när det gäller fördröjningar. Detta gäller till exempel traditionella filöverföringar med FTP och andra liknande protokoll. För andra applikationer är det precis tvärtom, de har stora krav på snabba och jämna dataflöden, till exempel ljud- och videoflöden.

Resursallokeringen är mottagarbaserad. Sändaren talar om för mottagaren att den är redo att sända data varpå mottagaren allokerar resurser hos mellanliggande noder. Protokollet är därför skalbart och dynamiskt. Från enkla unicastsändningar till stora multicastgrupper. Vid multicasting behöver inte en mottagare resursallokera hela vägen till källan för multicastingträdet, den behöver bara ta sig till en redan reserverad gren av trädet.

[3] Resource Reservation Protocol



Figur 4.4. Latensens fördröjningskedja.

Viktigt att komma ihåg är att RSVP inte är något routingprotokoll. Istället utnyttjar protokollet den underliggande routningen, den som IP erbjuder. RSVP ligger på samma nivå som transportprotokollen i protokollstacken. När rutten genom nätverket ändras på grund av förändringar i topologin, ändras resursallokeringarna adaptivt.

För att denna resursallokering ska fungera krävs det att routrar har inbyggt stöd för RSVP. Mellanliggande noder, eller routrar, som inte stöder protokollet tunnlar RSVP-trafiken som består av vanliga paket skickade över UDP. Att allokera resurser för delar av en rutt är dock inte optimalt, och kan till och med vara helt bortkastat. Detta är den stora nackdelen med denna metod.

4.4 Minimering av latensen

Vi har sett att nätverket själv ger upphov till en fördröjning. Utöver denna innehåller ett distribuerat realtidssystem en rad andra fördröjningar. Alla dessa utgör en kedja, se figur 4.4, och de bildar tillsammans latensen i systemet.

För att kunna minimera latensen är det en bra början att lokalisera alla större fördröjningar i systemet. Förutom nätverksfördröjningen utgörs dessa av bland annat en rad olika bufferter och processering av data. En del av dem kan bara påverkas marginellt eller inte alls. Här nedan tänker jag diskutera de fördröjningar som mer än marginellt kan minskas och metoder för detta.

4.4.1 Ljudkortsdrivrutinen och fördröjningar i operativsystemet

Operativsystem är inte optimala i sin nuvarande form för att hantera realtidsapplikationer [7]. Operativsystemet i en vanlig dator för hemmabruk kan bara hantera en process i taget, därför används en processväxlare för att flera processer ska kunna exekveras "samtidigt". En process får tillgång till CPU:n ett kort ögonblick och sedan går programkontrollen över till nästa process medan de andra processerna vilar. Detta går så pass fort att för användaren ser det ut som att flera processer körs samtidigt. Parallellbearbetning[4] kallas det för. Problemet är att processväxlingen sker asynkront och man kan inte veta hur länge en process förblir vilande.

För en realtidsapplikation med ljudflöden innebär detta vissa problem. Bufferten i operativsystemets ljudkortsdrivrutin måste hela tiden matas med ljuddata. Blir denna buffert tom uppstår avbrott i uppspelningen. Frekventa korta avbrott av den typen leder till en klar försämring av ljudkvaliteten och förståelsen av tal. Operativsystemets processväxlingssystem gör att denna buffert inte matas under hela processens inaktiva period. Det är därför viktigt att buffertstorleken i ljudkortsdrivrutinen är tillräckligt stor för att det ska finnas nog mycket data att spela upp tills det att applikationen återigen får kontrollen och kan mata in ny data. Problemet med detta är fördröjningen. Det går att göra buffertstorleken så pass stor att det aldrig blir något avbrott i uppspelningen, men ju större buffert desto större blir fördröjningen i ljudkortsdrivrutinen.

Ett sätt att hantera detta är att gå direkt på operativsystemet. I Linux är det ganska vanligt att öka frekvensen med vilken processväxlingen sker. Detta är visserligen ganska effektivt, men det är ingen bra lösning på problemet. För det första så krävs det god förståelse av Linuxkärnan och att man kompilerar om denna (eller snarare "patchar" med speciella realtidsmoduler). Dessutom finns det ingen motsvarande lösning i andra operativsystem, till exempel i Windows.

I [7] beskrivs en metod för att på applikationsnivå uppnå en bra balans mellan kontinuerlig uppspelning och låg fördröjning i ljudkortsdrivrutinen. Den bygger på att man uppskattar CPU-belastningen genom att läsa in all data som hunnit lagras i ljudkortets inspelningsbuffert. Datamängden ger ett exakt värde på tiden sedan förra inläsningen, och således också för tiden som processen varit inaktiv. Dessa värden lagras sedan av applikationen. Genom att utnyttja denna statistik är det möjligt att adaptivt styra buffertstorleken i ljudkortsdrivrutinen. Denna metod har visat sig ge såväl färre avbrott i uppspelningen som lägre fördröjning.

4.4.2 Metoder för adaptiva buffertlängder vid uppspelning

Vid direktuppspelning av ljudflöden krävs det en uppspelningsbuffert på applikationsnivå. Den främsta anledningen till detta är, som tidigare nämndes, att paket kan anlända i oordning och med skiftande fördröjning över nätverket. En uppspelningsbuffert jämnar ut fördröjningsjitter vid uppspelning och applikationen hinner

[4] På engelska kallas det för multitasking.

då även att sortera paketen i rätt ordning. Ett paket som inte kommer i tid för uppspelning räknas som förlorat. Anländer det sedan ändå slängs det bort. Genom att göra denna buffert större kan antalet försenade paket minimeras. Detta sker dock, som vi tidigare sett, på bekostnad av fördröjning. Ju större buffert, desto större fördröjning.

Återigen har vi en balans att upprätthålla. Detta kan göras antingen genom att sätta en fast buffertstorlek som inte orsakar allt för stora paketbortfall och fördröjningar eller så kan man använda en adaptiv uppspelningsmekanism med varierande buffertstorlek.

I [8, 11] beskrivs olika algoritmer för att uppskatta nätverksfördröjningen och sedan utifrån denna adaptivt justera buffertlängden. Distributionen för nätverksfördröjningen är inte känd och kan variera relativt mycket på kort tid. Till exempel uppträder ibland ”spikar” då en plötslig stor ökning av fördröjningen över nätverket äger rum. Denna följs av en rad paket som kommer fram ungefär samtidigt. Enklare adaptionsalgoritmer ser inte detta. Används sådana kan paketförlusterna vid spikar därför bli onödigt stora. Både [8] och [11] beskriver dock även mer avancerade algoritmer som adapterar snabbt nog för att justera buffertlängden för spikar.

4.4.3 Fördröjning vid kompression och dekompression

Det är nödvändigt med kompression för att minska belastningen på nätverket. Kompression och dekompression av ljud är en komplicerad process som ger ett signifikant bidrag till latensen i systemet. Här måste man därför göra ett lämpligt val av kompressionsformat. Fördröjningen är dock inte den enda parametern att titta på när det gäller detta val. Till exempel så kan en snabb algoritm komprimera sämre vilket kommer att leda till större paketförluster. Mer om olika kompressionsformat i nästa kapitel.

Kapitel 5

Ljudkompression

I detta kapitel ska jag diskutera ljudkompression, framför allt beskriva några av de vanligaste och mest spridda kompressionsformaten. Jag tänker inte gå in på några tekniska eller matematiska detaljer, det är ett alldeles för stort område. Istället kommer jag försöka beskriva de olika formatens karaktäristiska egenskaper och bedöma de utifrån de kriterier som är relevanta för examensarbetet. Jag har tittat på lite olika aspekter: Karaktäristiska egenskaper för formatet i fråga, status när det gäller patent och licensfrågor och dessutom tillgängligheten av kodbibliotek för olika plattformar.

En enda monosignal med cd-kvalitet (16 bitar och 44.1 kHz) ger ett bitflöde på ungefär 700 kbps. Ljud kräver med andra ord en hel del bandbredd. Därför behövs ofta kompression. Det finns många olika kompressionsformat. En del är generella, andra är specialiserade för till exempel bild, video eller ljud. Man brukar kategorisera alla dessa i två stora huvudgrupper: förlustbaserade[1] och förlustfria kompressionstekniker.

Att testa och jämföra olika kompressionsformat är otroligt svårt. Olika tester ger olika resultat beroende på vad man jämför och hur man genomför testet. Intressanta egenskaper att titta på är kompressionsgrad, ljudkvalitet, fördröjning och hur mycket datorresurser som används vid kodning och avkodning. Kompressionsgrad och ljudkvalitet är svåra att jämföra mellan format eftersom dessa två är så nära sammanlänkade. Ett optimalt test av kompressionsgraden skulle kräva att man höll ljudkvaliteten konstant mellan formaten och vice versa. Lyssningstester av ljudkvalitet är också så pass subjektiva att sådana är ytterst svåra att genomföra. Det krävs teknik av hög kvalitet och personer med mycket god hörsel.

Med detta sagt vill jag poängtera att de kommentarer kring just kompressionsgrad och ljudkvalitet som figurerar här nedan ska ses som ”dominerande åsikter”. När det gäller de förlustfria formaten där ljudkvaliteten är densamma både före och efter kompression är det något lättare att jämföra de olika formaten. Därför har jag gjort en mindre test av dessa format.

När det gäller patent och licensfrågor ska man ha i åtanke att dessa än så länge

[1] På engelska kallas det för ”lossy”.

gäller USA i första hand. I Sverige finns idag inga mjukvarupatent som omfattar nedanstående kompressionsformat och algoritmer. Ny lagstiftning angående mjukvarupatent är på gång inom EU så vad som sker i framtiden är än så länge ganska oklart. Jag anser ändå att patent och licensfrågan är viktig eftersom det kan avgöra i vilka delar av världen som applikationen är gångbar. Kommentarerna angående patent i nedanstående text gäller i första hand för USA, men kan alltså komma att gälla även i Europa.

5.1 Förlustbaserad kompression

Genom att utnyttja kunskaper om det mänskliga örats begränsningar och ta bort mycket av de mer subtila (jag undviker att kalla det för ohörbara) detaljerna i ljudet går det att uppnå mycket god kompression. Detta utnyttjar förlustbaserade kompressionsalgoritmer. Ju högre kompressionsgrad, ju mer detaljer i ljudet man väljer att filtrera bort, desto färre antal bitar per sekund går det åt för att koda signalen.

Det går att styra kvaliteten genom att välja vilket bitflöde man vill ha. Oftast kan man välja detta på olika sätt. Det enklaste sättet är CBR, Constant Bitrate, där man sätter ett fast bitflöde för signalen. Sedan finns det också ABR, Average Bitrate, och VBR, Variable Bitrate. Den förra tillåter bitflödet att fluktuera något för att anpassa sig till signalen, men lovar att ge ett visst medelbitflöde. Den senare däremot väljer kontinuerligt det mest lämpliga bitflödet. Är detaljrikedomen i ljudet stor används mycket data. Är detaljrikedomen istället liten, till exempel då få frekvenser används och vid tysta signaler, används lite data. För ljudflöden över nätverk är VBR ej lämplig att använda på grund av det ojämna dataflödet.

Här nedan följer en kort redogörelse för några av de största och mest spridda förlustbaserade kompressionsformaten.

5.1.1 MPEG-1 Audio Layer 3

MPEG-1 Audio Layer 3, populärt kallad mp3, är idag det största och mest spridda formatet för högkvalitativt ljud. Det utvecklades i slutet av 1980-talet av Fraunhofer IIS och stödjer mono och stereo i frekvenser upp till och med 48 kHz. Vid det här laget har formatet åldrats ganska mycket och kan inte längre konkurrera med nyare och modernare format när det gäller ljudkvalitet per bit.

MPEG-1 Audio Layer 3 är visserligen en öppen standard och vem som helst kan skriva en egen implementation. Men standarden innehåller en samling patent på ett flertal algoritmer, och det krävs därför en licens för att få använda mp3 i en applikation. Licensiering sköts av ett företag som heter Thomson. Köper man en licens så tillhandahåller Fraunhofer IIS kodbibliotek för ett flertal plattformar, bland dessa finns Windows, Mac OS och Linux. Det finns också en stor mängd tredjepartsimplementationer, men det krävs alltså en licens även för att få använda dessa eftersom det är algoritmerna man betalar för, inte implementationen.

5.1.2 mp3PRO

mp3PRO är en vidareutveckling av mp3-formatet utvecklad av Coding Technologies. Det är fullt bakåtkompatibelt med mp3. Den största förbättringen ligger i kodningen av låga bitflöden (<64kbps). Där är mp3PRO idag oöverträffad när det gäller ljudkvalitet per bit. Liksom mp3 är mp3PRO patenterad och även för denna tillhandahåller Thomson licenser. Det finns kodbibliotek för alla de stora plattformarna.

5.1.3 MPEG-2 AAC

MPEG-2 AAC (Advanced Audio Coding) är baserad på mp3, men inte bakåtkompatibel. Tekniken utvecklades av Fraunhofer IIS tillsammans med bland andra Dolby Laboratories, AT&T och Sony. Tanken var att använda mp3 som bas för att utveckla ett nytt format bättre anpassat för att användas till videoströmmar. Formatet innehåller stöd för multikanalsljud och är anpassat för att ge god ljudkvalitet redan vid låga bitflöden.

AAC är omgärdat av en mängd olika patent registrerade av ett flertal stora företag. Fraunhofer IIS tillhandhåller kodbibliotek (för såväl kodning som avkodning) till de tre stora plattformarna Windows, Mac OS och Linux.

5.1.4 MPEG-4 AAC

MPEG-4 AAC är, föga överraskande, en vidareutveckling av MPEG-2 AAC. De viktigaste förändringarna är i första hand förbättrad ljudkvalitet vid låga bitflöden och bättre skalbarhet vid direktuppspelning av ljud över nätverk. Formatet används flitigt av framför allt Apple i QuickTime 6 och i iTunes 4. Formatet ger klart bättre kompression och ljudkvalitet per bit än mp3. När det gäller patent och kodbibliotek gäller samma sak som för MPEG-2 AAC.

5.1.5 Windows Media Audio

Windows Media Audio är en stor komponent i Microsoft Windows som innehåller flera olika format. För det första så finns Windows Media Audio standardformatet. Detta kan ses som Microsofts konkurrent till mp3. Den kan koda och avkoda mono och stereosignaler i upp till och med 48 kHz.

Windows Media Audio Professional är till för mycket högkvalitativt ljud och kan hantera 24 bitars upplösning med en samplingsfrekvens på 96kHz i stereo, 5.1 multikanalsljud eller 7.1 multikanalsljud. Denna kan ses som en konkurrent till MPEG-2 och MPEG-4 AAC.

Windows Media Audio Voice är ett format för låga bitflöden (<20kpbs). Det kan användas i två lägen, ett för musik och ett för tal.

Detta är alltså Microsofts egen kompressionsstandard för ljud och är en del av Windows. Windows Media Audio omgärdas inte av några patent, men är dock ett helt slutet format. En utvecklare som vill använda WMA måste därför använda

mjukvara från Microsoft för att kunna koda och avkoda ljud. Därför krävs det indirekt en licens, en Windowslicens. Windows Media Audio stöds fullt ut enbart av Microsoft Windows.

5.1.6 Real Audio

Real Audio utvecklades av Real Networks och är, precis som Windows Media Audio, ett slutet format som kräver licens. Det finns för ett flertal plattformar, bland andra Windows och Linux. Real Audio skiljer, precis som Windows Media Audio, på tal och musik.

5.1.7 Musepack

Musepack, nämns ibland även som MPEG+, är ett format specialgjort för höga bitflöden (>160kbps). Det utvecklades av Andre Buschmann och Frank Klemm och är baserad på MPEG-1 Audio Layer 2. Numera är Musepack ett helt öppet format, open source och patentfritt. För höga bitflöden visar ett flertal olika tester att det har oöverträffad ljudkvalitet. Vid låga bitflöden är det inte alls lika bra och det finns många bättre alternativ.

Formatet använder enbart VBR vilket gör att det inte är riktigt lämpligt att använda för direktuppspelning över nätverk.

5.1.8 Ogg Vorbis

Ogg Vorbis, eller bara Vorbis, är ett helt öppet format, open source och patentfritt. Vorbis är gjord för att användas i mono eller stereo, med alla möjliga bitflöden. Kodbibliotek för kodning och avkodning av ljudflöden finns för både Windows och Linux. Dokumentationen för detta är dock ganska bristfällig. Ljudkvaliteten är mycket god och kan konkurrera med den man hittar hos mer spridda och patenterade format.

5.1.9 Speex

Speex är till för kompression av tal. Formatet är open source och patentfritt och är utvecklat som ett alternativ till traditionella och ofta slutna talkodningsformat. Där äldre algoritmer enbart har varit för extremt små bitflöden, oftast mindre än 10 kbps, har Speex en större spännvidd. Den kan användas i tre olika frekvensband: 8, 16 och 32 kHz. Beroende på vilket av dessa frekvensband som används och på diverse inställningar kan man få bitflöden mellan 2 och 44 kbps. Det finns kodbibliotek för både Windows och Linux som är både enkla och väldokumenterade.

Speex är mer komplicerat än många äldre talkodningsformat och ger en något längre fördröjning än dessa. Dessutom är det betydligt mer resurskrävande. Användning av hög ljudkvalitet för 16 och 32 kHz banden kan inte rekommenderas för realtidsapplikationer där mycket datorresurser krävs till andra delar också, till exempel rendering av grafik i ett VR-system.

5.1.10 Övriga äldre talkodningsformat

Det finns en stor uppsättning av äldre talkodningsformat som utvecklats sedan 60-talet och framåt. Dessa har ofta ganska luddiga patentuppgifter och det finns inte heller så ofta någon form av ”officiella” kodbibliotek. Formaten utvecklades ofta för telekommunikation och är således av låg kvalitet. Jag beskriver några av dessa kortfattat här.

G.711 är den internationella standarden för hur telefonsamtal kodas över en 64kbps-kanal. Den utvecklades redan på 60-talet och finns i två varianter. En amerikansk som kallas μ-law, och en europeisk som kallas A-law. Båda dessa arbetar i 8 kHz.

Under 80-talet standardiserades en annan talkodningsteknik som kallas g.721 ADPCM. Denna har ett dataflöde på 32 kbps och nästan lika bra ljudkvalitet som g.711. Senare uppgraderingar av detta format, g.726 och g.727, klarade av flera olika dataflöden. 16, 24, 32 och 40 kbps.

Sedan 1992 finns också g.728 som är en talkodning som har låg inbyggd fördröjning. Den ger dataflödet 16 kbps och arbetar i 8 kHz.

I mobiltelefonnätet används GSM som står för Global System for Mobile communications. Den arbetar i 8 kHz och ger dataflödet 13 kbps. Ljudkvaliteten ligger strax under g.728. Den stora fördelen med GSM är att den är mycket snabb och lätt kan köras i realtid på ganska enkla processorer.

5.2 Förlustfri kompression

Har man riktigt höga krav på ljudkvaliteten är förlustfri kompression ett alternativ. Baksidan är att man inte uppnår lika god kompression som med de förlustbaserade formaten. Det krävs därför stor bandbredd för att kunna använda dessa format vid direktuppspelning över nätverk.

5.2.1 FLAC

FLAC står för Free Lossless Audio Codec och är ett helt fritt förlustfritt format. Det stödjer både mono- och stereosignaler i upp till och med 96 kHz. Flera olika bitlängder stöds, från 8 bitar ända till 24 bitar per sampling. Det finns kodbibliotek för kodning och avkodning till Windows, Mac OS, Linux och till ett flertal andra operativsystem. Kodbiblioteket är relativt väldokumenterat. FLAC är också väl anpassat för direktuppspelning över nätverk.

5.2.2 Monkeys Audio

Monkeys Audio är ett förlustfritt kompressionsformat som utvecklades av Matthew T. Ashland. Det finns stöd för mono och stereo i 8, 16 och 24 bitar. Monkeys Audio är visserligen open source och patentfri, men det råder lite oklarheter om vad licensen egentligen säger. Den är helt öppen för privat bruk, men ska man använda den i ett kommersiellt sammanhang så krävs ett skriftligt tillstånd från upphovsmakaren. Vad

detta egentligen innebär är höljt i dunkel. Än så länge finns den bara för Windows, men enligt den officiella hemsidan så är versioner för såväl Linux som Mac OS på väg. Även här råder vissa oklarheter huruvida projektet fortfarande är aktivt.

5.2.3 Windows Media Audio Lossless

Slutligen finns också Windows Media Audio Lossless som är Microsofts förlustfria ljudformat. Detta är alltså en komponent i Windows Media Audio som jag skrev om i avsnitt 5.1.5.

5.2.4 Test av förlustfria format

Jag genomförde ett enkelt test av dessa tre förlustfria kompressionsformat. Jag använde fyra olika ljudfiler, var och en komprimerade med de olika formaten och tog tid på båda kodning och avkodning. Därefter jämförde jag de komprimerade filstorlekarna med varandra och med de okomprimerade källfilerna. Resultatet finns redovisat som tabeller i appendix. Detta test är så pass litet att man inte ska dra allt för stora slutsatser av det, men några tydliga tendenser går ändå att skåda.

Kompressionsgraden mellan de tre formaten är ganska lika. FLAC visar på en något sämre kompressionsgrad. Kompressionsgraden ligger mellan 39% och 67% av de okomprimerade källfilerna. Bäst komprimerat blir rent tal.

Det mest överraskande resultatet visar tidtagningen av kodningen och avkodningen. Här kan man konstatera att FLAC och Monkeys Audio är bra mycket snabbare än WMA Lossless. Jämför man sedan FLAC och Monkeys Audio kan man se att FLAC är avsevärt snabbare vid avkodning. Detta resultat får man vara försiktig med att tolka, men det kan vara så att FLAC har en lägre fördröjning än de två andra.

Kapitel 6

Metodval

Här följer en kortare redogörelse för vilka metoder jag har valt när det gäller de större delproblemen av arbetet. Först diskuterar jag nätverk, olika nätverkstopologier och motiverar de metoder jag valt för detta ändamål. Därefter diskuterar jag mina metodval när det gäller hantering av nätverksbegränsningar. Sist följer ett kort avsnitt om vilka kompressionsformat som jag valt att stödja och motiveringar för detta.

6.1 Val av nätverksmodell

Jag har valt att använda IP/UDP/RTP/RTCP (se kapitel 3) som grund för nätverkskommunikationen i prototypen (såväl server som klient). Det vill säga, jag använder RTP och dess fristående kontrollprotokoll RTCP över UDP överföringar på Internet. UDP ger den mest optimala lösningen när det gäller latensen i systemet. Detta på bekostnad av paketförluster och andra nätverksbegränsningar. RTP ger alla nödvändiga hjälpmedel för att hantera dessa begränsningar på bästa sätt. RTCP ger en utökad uppsättning instrument för att kunna övervaka nätverkskvaliteten och synkronisera olika RTP källor.

6.1.1 Nätverkstopologier

Tidigt i arbetet bestämde jag i samspråk med Interaktiva Institutet att använda mig av en klassisk server/klient-modell. Modellen är den att varje klient skickar sitt ljudflöde och sin virtuella position till en server. Servern skickar därefter vidare paketen med ljudfragment från ljudflödet till de klienter, eller avatarer, som den bedömer kan höra ljudflödet. All denna kommunikation sker med UDP i Unicast läge. Servern skickar därför samma paket flera gånger om fler än en avatar hör ett specifikt ljudflöde. Denna modell valdes för sin enkelhet och för att den bedömdes att fungera väl i de flesta situationer.

Det går dock att tänka sig flera andra nätverksmodeller, eller nätverkstopologier. Jag tänker här gå igenom några alternativ till ovanstående modell. De flesta av dessa bygger på multicast.

Istället för att servern skickar UDP i unicast kan den skicka ut alla paket i multicast. Då behöver den aldrig skicka samma paket mer än en gång. Detta kan fungera så att servern har en multicastadress för varje uppkopplad avatar. Adressen kan ses som den tillhörande avatarens distributionslista. När ett ljudfragment anländer i ett paket distribueras detta genom att skicka en gång till avsändarens multicastadress. För att detta ska fungera måste servern fortsätta hålla reda på virtuella positioner och fortlöpande tala om för avatarerna vilka multicastadresser som de ska lyssna på.

Istället för att låta ljudflöden gå via servern kan varje klient, eller avatar, använda en egen multicastadress och skicka sitt ljudflöde på denna. Det enda klienten behöver skicka till servern är sin virtuella position och multicastadress. Servern beräknar sin distributionstabell utifrån de virtuella positionerna och talar om för klienterna vilka multicastadresser de ska prenumerera på. Eftersom ingen realtidsdata längre går via servern kan TCP användas för all kommunikation mellan server och klient.

Det går att konstruera en ännu tunnare server om man vill. Klienterna är beroende av två servrar som det set ut nu, en ljudserver och en geometriserver. Geometriservern är någon form av scengraf som förser klienten med dess virtuella position. Det är fullt genomförbart att låta varje klient själv beräkna sin distributionslista. För detta krävs de olika avatarernas virtuella positioner som kan fås från geometriservern. Med andra ord beräknar varje klient själv vilka andra avatarer som kan höra den. Klienten behöver då inte längre skicka sin virtuella position till ljudservern utan meddelar endast förändringar i sin distributionslista. Ljudservern i sin tur talar om för de olika avatarerna vilka multicastadresser de ska börja eller sluta att lyssna till. Här har ljudservern reducerats till att endast distribuera multicastadresser.

En lösning helt utan ljudserver är inte möjlig. Dels eftersom man vill tillhandahålla filtrering av onödiga distributionskedjor och dels för att deltagarantalet ska kunna vara så dynamiskt som möjligt.

6.1.2 Paketförluster

I kapitel 4 beskrev jag en mängd olika metoder för att minimera och maskera paketförluster. När jag har valt ut metoder att implementera i prototypen bland dessa har jag tittat på några olika kriterier. I första hand har jag sorterat bort metoder som leder till för stor latens, till exempel interfoliering och återsändning av paket. Därefter har jag försökt välja metoder som är enkla att implementera, inte allt för processorkrävande och ändå relativt bra på att minska effekterna av paketförluster. I stort sett har jag följt rekommendationerna för interaktiva applikationer i [10].

Jag använder i första hand redundanta återsändningar med lågkvalitativa ljudfragment. Det är en metod som är ganska enkel att implementera och som ger ett bra resultat [6]. I andra hand, om varken det primära eller det redundanta ljudfragmentet anländer i tid, använder jag paketrepetition. Paketrepetition ger ett bra resultat även i jämförelse med mer komplexa mottagarbaserade metoder som interpolationsbaserad och regenerationsbaserad substitution [10]. Paketrepetition är dessutom enkelt att implementera.

6.1.3 Latens

Den enda aktiva metoden för att minska latensen som jag använder är adaptiva buffertlängder i uppspelningsbufferten. I övrigt försöker jag använda metoder och tekniker med liten fördröjning, till exempel ovanstående metoder för att hantera paketförluster. Även valet av kompressionsformat påverkar latensen.

6.2 Val av kompressionsformat

När det gäller vilka olika kompressionsformat som ska stödjas i prototypen har jag haft tydliga kriterier: källkoden ska vara öppen, formatet får inte innehålla några patent och det ska vara väl anpassat för ljudflöden över Internet. Det sistnämnda kriteriet innebär att formatet ska dela upp ljudflöden i korta ljudfragment som är tillräckligt små för att få plats i ett IP-paket och att dessa fragment ska komprimeras oberoende av varandra. Ingen data i flödet, inget paket, får med andra ord vara viktigare än något annat. Utöver dessa kriterier spelar ljudkvaliteten en avgörande roll. Kriterierna gör att listan över möjliga kompressionsformat blir ganska kort.

Olika format behövs för olika funktioner i ljudsystemet. Det behövs ett format för att komprimera tal och ett annat mer generellt för till exempel musik och ljudeffekter. Dessutom behövs det ett format för att komprimera de redundanta ljudfragmenten. Jag har valt att använda Speex för tal. Det finns egentligen inget annat alternativ som uppfyller de kriterier som jag har satt upp ovan.

Som generellt kompressionsformat hade jag först tänkt använda Vorbis. Men efter att ha upptäckt en del tveksamheter[1] när det gäller Vorbis i ett distribuerat sammanhang har jag istället valt FLAC. FLAC är egentligen ett mycket mer intressant format. Det är förlustfritt och har därmed optimal ljudkvalitet. Dessutom visar enkla tester att det är snabbt.

När det gäller de redundanta ljudfragmenten som bara kommer att användas sporadiskt (beroende på hur stora paketförlusterna är förstås) finns inte alls samma krav på ljudkvalitet. Här är det mest intressant att formatet ger god kompression och att det är snabbt. Därför har jag valt GSM för detta ändamål. GSM är mycket snabb och tillför endast en mycket liten fördröjning i systemet.

[1] Det verkar som om startpaketet i ett Vorbisflöde innehåller viktig tillståndsdata. Om detta paket förloras blir hela det följande dataflödet oanvändbart. Startpaketet måste alltså komma fram och kräver specialbehandling i nätverkssammanhang.

Kapitel 7

Implementation av prototypen

Examensarbetet ledde till en prototyp av en server och en klient. I detta kapitel ska jag beskriva den grundläggande arkitekturen hos de båda, diskutera och motivera de olika tillvägagångsätten. Sist diskuterar jag även de begränsningar som finns i prototypen.

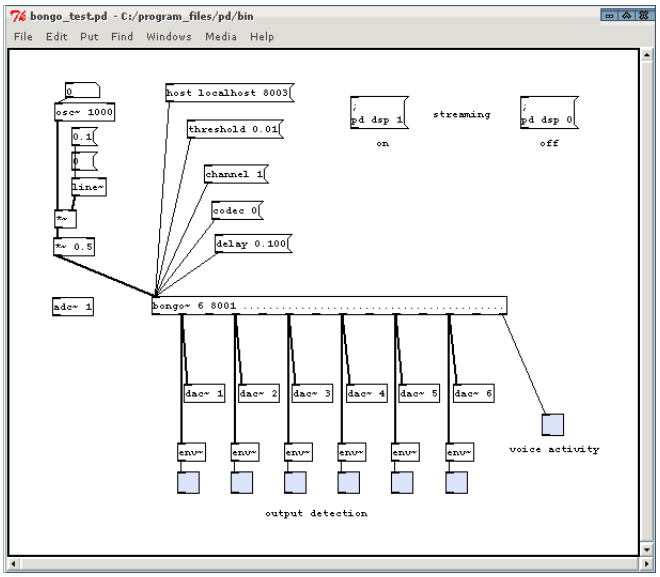
7.1 Utvecklingsmiljö

Såväl server som klient är skrivna i ANSI C. Fördelen med detta är att programmen blir mindre och snabbare än i många andra modernare högnivåspråk som till exempel Java. Dessutom är alla de ingående kodbiblioteken skrivna i C. Nackdelen med C i jämförelse med till exempel Java är att det är betydligt svårare, men helt genomförbart, att skriva portabel kod. Det krävs också mer programmerande eftersom C och dess standard kodbibliotek är ganska avskalat. Till exempel måste man skriva en hel del nätverkskod och alla datastrukturer själv.

Klienten är skriven som en modul till programmet Pd[1]. Det är en grafisk och modulär programmeringsmiljö för signalprocessering i realtid. Pd finns för Windows, Linux, IRIX och Mac OS X. Programmering sker genom att koppla samman moduler, eller abstraktioner som de kallas i Pd, till ett signalflöde. En abstraktion representerar någon form av operation, till exempel att generera en sinuston eller addition av två inkommande signaler. Klienten är i denna miljö mycket flexibel och det är därför lätt att använda den i många olika sammanhang. Till exempel är det enkelt att koppla in valfri ljudkälla till den, det är bara att göra en grafisk koppling i användargränssnittet från ljudkällan (en mikrofon, en tongenerator eller ljudsyntes, en fil, ett ljudflöde, etc) till klientens signalingång.

Figur 7.1 visar en skärmutskrift från Pd. Det som visas är en testuppsättning av klienten. En tongenerator (som skickar en sinuston på 1000Hz) är kopplad till ingången och sex stycken utgångar tar emot ljudflöden från en server.

[1] Pd (förkortning av ”pure data”) finns att ladda ned både som källkod och binärt från http://www-crca.ucsd.edu/~msp/software.html.



Figur 7.1. En skärmutskrift från Pd.

7.2 Grundläggande nätverkskommunikation

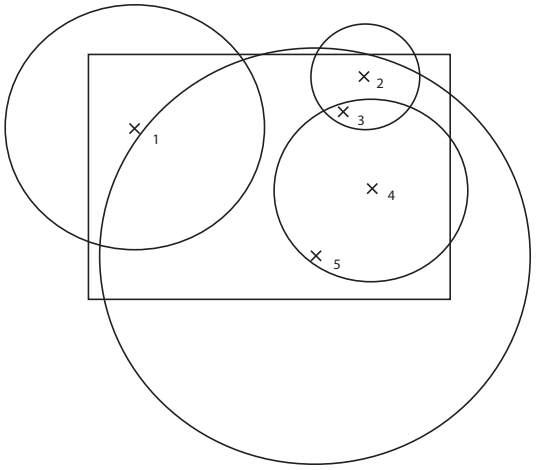
Både servern och klienten är byggda ovanpå IP (både IPv4 och IPv6 stöds) med UDP som grundläggande transportprotokoll. Utöver detta används RTP för all kommunikation mellan server och klient. Denna programkod är gemensam för server och klient.

För att ett distribuerat ljudsystem ska fungera väl räcker det inte att bara skicka ljuddata. Det krävs att servern och klienten kan kommunicera med ett något större vokabulär än vad RTP-huvudet och RTCP kan erbjuda. Därför behövs det ett applikationsprotokoll med en uppsättning olika kommandon. Exempel på sådana kommandon kan vara ”spela upp”, ”stoppa uppspelning” och ”paus”, med andra ord typiska funktioner för en musikspelare.

För detta syfte finns RTSP[2]. Det är ett kontrollprotokoll för ljud- och videoflöden, som består av en mängd olika kommandon som kan skickas mellan deltagare, eller mellan server och klient. Något förenklat kan man se RTSP som en ”fjärrkontroll” för multimediaservrar. RTSP är dock ett ganska stort och omfattande protokoll. Dessutom använder prototypservern ett slags automatiskt adaptivt prenumerationssystem av ljudflöden, något som gör att mycket av funktionaliteten i RTSP är överflödig. Av denna anledning valde jag att använda ett enklare system där man kan skicka ett fåtal nödvändiga kommandon via RTP-paket.

Som prototypen ser ut nu kan man förutom att skicka ljudflöden även skicka ändrings- eller uppdateringskommandon av tre olika parametrar: virtuell position,

[2]Real Time Streaming Protocol



Figur 7.2. Den geometriska situationen i servern.

1	2	3	4	5
	3		3	2
			5	3
				4

Tabell 7.1. Distributionstabell för befintlig geometri.

signalnivå och kanal. De två första används av servern för att filtrera bort onödiga distributionskedjor, mer om detta nedan.

7.3 Server

Servern är relativt enkel. Den skapar en distributionsmatris som hela tiden uppdateras automatiskt. Varje ljudkälla har en egen kolumn i matrisen, en lista som representerar alla avatarer som kan höra denna. Om servern tar emot ett ljudflöde från en avatar, sänder den vidare detta till alla mottagare som står med i dennas distributionslista.

I figur 7.2 illustreras den geometriska uppsättningen i servern. Kryssen markerar avatarernas virtuella positioner, cirklarna deras hörbarhetsavstånd (som ges av amplituden på ljudflödet) och kvadraten själva geometrin. Detta kan till exempel vara en fotbollsplan.

Tabell 7.1 visar hur distributionstabellen, eller distributionsmatrisen, för denna situation ser ut. En kolumn i tabellen kallar jag för distributionslista. Avatar 3 saknar ett giltigt hörbarhetsavstånd och har därför en tom distributionslista. Detta kan bero på att servern ännu inte fått något uppdateringskommando för avatarens ljudnivå, eller amplitud.

Första gången som en klient skickar ett kommando till servern skapar denna en ny kolumn i distributionsmatrisen. Om en klient loggar av raderas den tillhörande avataren och dess kolumn samt alla platser där den förekommer i andra avatarers distributionslistor. Detta sker även om en avatar varit inaktiv en längre tid.

Uppdatering av distributionsmatrisen är det mest komplicerade som sker i servern. Uppdatering sker vid positionsförändring, amplitudförändring och vid kanalbyte. Mer om kanaler längre ned. Serverns upplösning av geometri och amplitud är "kvantiserad". Det krävs att förändringen av dessa är tillräckligt stor för att uppdatering ska ske. Annars skulle uppdatering ske alltför ofta och belasta servern onödigt mycket. Denna kvantiseringsupplösning kan justeras med hjälp av parametrar till servern vid start. Uppdatering sker alltid på uppdrag av en avatar genom att servern tar emot ett kommando från denna.

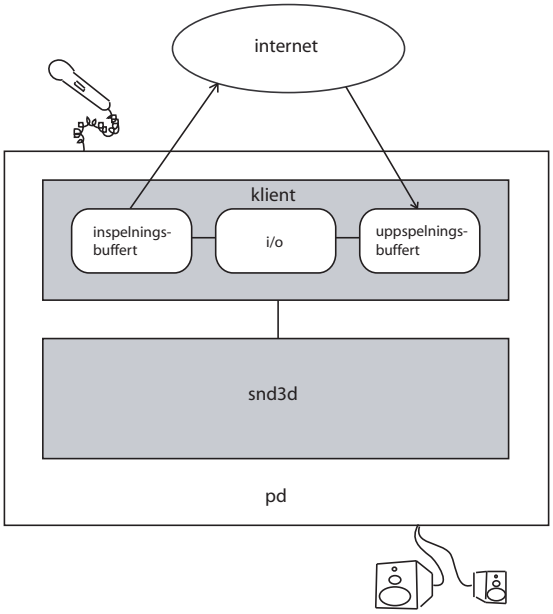
Om uppdateringen gäller amplituden finns det två fall. Det enklare fallet är att amplituden för avatarens ljudflöde minskar. Då behöver servern bara gå igenom avatarens befintliga distributionslista och kontrollera om någon inte längre hör flödet. Om så är fallet tas denna bort från listan. I det andra fallet, om amplituden ökar eller rapporteras för första gången, måste servern gå igenom alla registrerade avatarer. De nya avatarer som nu kan höra ljudflödet läggs till i distributionslistan.

Vid uppdatering av position går servern alltid igenom alla avatarer i matrisen och undersöker vilka som nu kan höra den uppdaterade avataren. Förändringarna lagras i avatarens distributionslista. Därefter går servern igenom alla andra distributionslistor och ser efter om den uppdaterade avataren finns med i var och en av dessa. Gör den det undersöker servern om denna avatar fortfarande hör hemma i den distributionslistan. Om inte tas den bort.

Distributioner i ljudsystemet är kopplade till kanaler. Av dessa finns två olika typer. Först och främst den speciella "geometriskt medvetna" kanalen 0. Den andra typen utgörs av kanal 1 och uppåt. Servern håller reda på vilken kanal som används av olika klienter.

Kanal 0 används av alla avatarer i en VR-miljö. Alla ljudflöden som hör till kanal 0 går igenom den rumsliga filtreringen hos servern. Denna kanal kan ses som standardkanalen för VR-tillämpningar. Kanal 1 och uppåt behandlas istället som i ett traditionellt konferenssystem. Det vill säga klienter som använder exempelvis kanal 5519 kan höra alla andra klienter som utnyttjar denna kanal. Ingen filtrering av ljudflöden äger rum.

Icke-geometriska kanaler (alla andra förutom 0) har flera användningsområden. I ett VR-sammanhang kan de till exempel användas för att administratörer ska kunna prata med varandra över systemet utan att alla deltagare hör denna kommunikation. Det går också att tänka sig att olika grupper av VR-deltagare kommunicerar via headset, även då kan detta kanalsystem utnyttjas. Kanalen ges av ett 32-bitars värde. Det är med andra ord ganska lätt att "gömma" sådana kommunikationskanaler i 32-bitars mängden, men det är givetvis ingen säker kommunikationskanal. För säker kommunikation behövs kryptering av ljudflöden och någon form av autenticering inom varje kanal, något som ligger långt utanför examensarbetets omfång.



Figur 7.3. Klienten, dess komponenter och miljö.

I kultursimulatorprojektet är kanalsystemet tänkt att användas för att tekniker och spelledare ska kunna kommunicera ostört.

7.4 Klient

Klienten är en betydligt mer komplicerad applikation än servern. Den består av tre relativt separata komponenter som tillsammans utgör klientens kärna. En komponent tar emot och sparar inkommande ljudflöden i buffertar. En annan sänder klientens ljudflöde till servern. Slutligen finns det en komponent som sköter in- och uppspelning från och till inkommande och utgående buffertar. Se figur 7.3 för en överskådlig bild av klienten och dess miljö. Här närmast ska jag beskriva varje komponent mer i detalj.

7.4.1 In- och uppspelning

Denna komponent (i/o i figuren) är själva huvudkomponenten och körs i en extern tråd i Pd. Den utgör gränssnittet mot Pd och har tre huvuduppgifter.

För det första hämtar den signaldata från ingången till Pd-modulen. Signaldata kan komma från en mikrofon, en linjeingång, en fil, etc. Detta beror helt på vad man kopplar till ingången i Pd:s grafiska gränssnitt. Signaldata i form av samplingar placeras sedan i en inspelningsbuffert i väntan på att kodas, packas och sändas iväg.

För det andra hämtar komponenten även signaldata från uppspelningsbufferten. Denna signal skickas ut på en tilldelad utgående kanal från Pd-modulen. Även signalens virtuella position finns tillgänglig via en utgång på Pd-modulen.

För det tredje tar komponenten hand om inkommande paket och packar upp dessa. Inkommande signaldata kodas upp av en kompressionsdekoder och samplingsfrekvensen konverteras vid behov till Pd:s samplingsfrekvens för uppspelning. Därefter kopieras ljudfragmentet till uppspelningsbufferten.

Förutom själva i/o-funktionaliteten tar denna komponent också hand om ett antal olika lokala kommandon, bland annat kommandon för byte av codec, kanal, serverinställningar och ett flertal andra. De flesta kommandon är kända och dokumenterade, men några är klassade som "hemliga" och därför odokumenterade. Dessa kommandon finns dokumenterade i klientens hjälpfil som nås genom Pd.

7.4.2 Inspelningsbuffert

Inspelningsbufferten fylls på med en sampling i taget av ovanstående komponent. När ett ljudfragment (20 millisekunder) är fyllt börjar det bearbetas. Vid behov konverteras samplingsfrekvensen för att passa det valda kompressionsformatet. Därefter komprimeras ljudet, packas ned till ett audiokommando och RTP-paket. I samma RTP-paket adderas också andra kommandon som positions-, kanal- och signalnivådata. Därefter skickas paketet till den inställda nätverksdestinationen (det kan vara en server, men också en annan klient).

Innan ett fyllt ljudfragment börjar bearbetas analyseras det för att avgöra om fragmentet innehåller tillräckligt mycket signal, ett så kallat signalaktivitetstest (eller röstaktivitetstest om det gäller tal). Om fragmentet bedöms innehålla tystnad, brus, eller bara mycket låg signalstyrka slängs det bort. Denna analys sker i två steg. Det första steget är ett så kallat peaktest. Parallellt med att fragmentet skapas undersöks signalstyrkan för varje sampling som adderas. Den största amplituden, peaken, lagras. När fragmentet är fullt jämförs peakvärdet med ett lågt satt tröskelvärde. Är värdet lägre än tröskeln bedöms fragmentet innehålla tystnad. Metoden är effektiv för att snabbt sålla bort ej inkopplade ingångar som bara genererar ett brus med mycket låg amplitud.

Om peakvärdet överstiger tröskelvärdet startar det andra steget. Steg två är ett RMS[3]-test som är en mer pålitlig metod för att få ett amplitudvärde av en signal. Peakmetoden ger ett värde baserat på bara en enda sampling medan RMS-metoden använder alla samplingar för att beräkna en slags medelamplitud. Värdet som detta ger jämförs med ett högre tröskelvärde än i första stegets peakmetod. Överstigs tröskelvärdet är saken klar, fragmentet innehåller en giltig signal.

För att inte det överförda ljudet ska klippa och låta konstigt finns det ytterligare en mekanism inblandad. Om signalaktivitetstestet är positivt krävs det därefter en viss tidslängd av konstaterad tystnad innan fragmenten börjar slängas bort och inte distribueras. I min klient är denna tid satt till 400 millisekunder som efter mycket testande visat sig vara ett bra värde.

[3]Root Mean Square

7.4.3 Uppspelningsbuffert

Uppspelningsbufferten lagrar samplingar i väntan på uppspelning. För att göra det möjligt att höra ljudet från flera källor samtidigt använder bufferten flera kanaler (inte att förväxla med de kanaler och kanalkommandon som diskuterats ovan). Varje kanal är egentligen en egen buffert och alla dessa tillsammans utgör uppspelningsbufferten. En kanalväxlare håller reda på vilka kanaler som används och av vem, samt vilka kanaler som är lediga. Till sin hjälp för detta har kanalväxlaren en avatardatabas som innehåller information om avatarers virtuella positioner etc.

Vid paketförluster försöker uppspelningsbufferten i första hand utnyttja redundanta ljudfragment från de paket som anländer hela. Om inte heller detta lyckas (på grund av paketförluster eller om klienten inte sände ett redundant ljudfragment av det förlorade paketet) uppstår hål i bufferten. För att maskera detta aktiveras paketförlusthanteringen (se nedan) som fyller hålet med olika typer av substituerad data beroende på vilken teknik som används.

Bufferten har en viss längd, eller fördröjning, för att jämna ut jitter, ge oordnade paket tid att sorteras och försenade paket en chans att hinna fram. Som det diskuterades tidigare i kapitel 3 om nätverk är det viktigt att hålla denna fördröjning på en kompromissnivå mellan de två motstridiga egenskaperna ljudkvalitet och latens. Det går att sätta buffertlängden, fördröjningen, till ett konstant värde som man sätter själv. Det går också att låta uppspelningsbufferten styra fördröjningen adaptivt genom att utnyttja jittervärdet från RTCP-rapporterna. Fördröjningen beräknas då mycket enkelt som jittervärdet (en tidslängd) multiplicerat med tre. Denna adaptiva styrning borde vara det bästa alternativet i de allra flesta fall.

7.4.4 Moduler

Utöver själva kärnan av klienten finns det ett modulärt system för komponenter som är av utbytbar och föränderlig karaktär. Varje modul förfogar över ett antal plugins som var och en implementerar en specifik teknik. Idag finns det tre sådana moduler: en för codecar, en för samplingsfrekvenskonvertering och en för mottagarbaserad paketförlusthantering.

Codec-modulen hanterar plugins för kodning och avkodning av signaldata. Den hanterar kodning och avkodning helt separat. I prototypens nuvarande form finns det sex plugins implementerade: Raw (16-bitar, 22kHz okomprimerad data), GSM, Speex 8kHz, Speex 16kHz, Speex 32kHz och FLAC[4]. GSM används främst för att koda ned redundanta ljudfragment.

Samplingsfrekvenskonvertering, eller resampling, är en ickereversibel process för att reducera eller öka tidsupplösningen av signaldata, exempelvis konvertering av ljud samplat i 44.1 kHz till 8kHz. De två plugins som finns i klienten idag är två olika tekniker för detta. Den ena använder linjär interpolering för upp- och nedsampling. Det är en mycket snabb och resurssnål teknik. Problemet är att ljudkvaliteten inte blir så bra på grund av aliaseffekter som uppfattas som knäpp i ljudet. Därför finns

[4]För Speex och FLAC används de kodbibliotek som Xiph.org tillhandahåller.

det även en plugin som använder så kallad bandbegränsad sinc interpolation[5] Det är den bästa resamplingsteknik som finns och ljudkvaliteten blir mycket bra. Nackdelen är att processorn belastas betydligt mer.

Den mottagarbaserade paketförlusthanteringen sköts även den av en modul. Det finns tre plugins: substitution med tystnad, brussubstitution och paketrepetition. De två första är implementerade mest i testsyfte, i praktiken är det alltid paketrepetition som används. Paketrepetition är, som jag tidigare nämnt, en bra kompromiss mellan ljudkvalitet och komplexitet [10].

7.4.5 Snd3d - ett system för mixning av 3d-ljud

Snd3d är en abstraktion gjord i Pd för mixning av signaler till 3d-ljud. Applikationen är skriven av Peter Lundén på Interaktiva Institutet.

Systemet tar in signaler och deras virtuella positioner som indata och gör en unik mix för varje kanal så att 3d-ljud uppnås. Eftersom mixningen kräver kunskap om högtalaruppsättning är detta fullständigt konfigurerbart. På så sätt fungerar systemet med alla tänkbara högtalaruppsättningar. Från stora multikanalssystem med stort antal högtalare till ett enkelt stereosystem. I det senare fallet blir 3d-ljudet reducerat till enkla panoreringseffekter.

Integrering mellan klienten och Snd3d är mycket enkel. Det är bara att grafiskt koppla ihop klientens utgångar för signaler och virtuella positioner till motsvarande ingångar i Snd3d.

7.5 Begränsningar i prototypen

Den största begränsningen i prototypen har med själva nätverkslösningen att göra. Systemet använder som jag tidigare beskrivit en server/klient-modell med UDP-trafik i unicast istället för multicast. Begränsningen i detta ligger hos servern. Varje avatar i VR-världen motsvaras av en klient och varje klient skickar sitt ljudflöde till servern. Servern tar emot detta ljudflöde och distribuerar det via unicast till noll, en eller ett flertal andra avatarer. Blir antalet uppkopplade avatarer för stort kommer servern helt enkelt inte orka med att distribuera all trafik. Detta händer givetvis i alla nätverksbaserade applikationer vid tillräckligt hög trafik.

För ljudsystemets närmast planerade användningsområden är den befintliga nätverkslösningen fullt tillräcklig. Vill man använda ljudsystemet i ett sammanhang med väldigt många uppkopplade klienter, hundratals eller kanske tusentals, kan det bli otillräckligt. Då kan det vara värt att utforska någon av de alternativa nätverkstopologier som tas upp i 6.1.1.

I sin nuvarande form är klienten enkeltrådig. Som jag har visat här ovan så består klienten av tre relativt separata delar, se figur 7.3. De tre delarna av klienten är nästan ett skolexempel på när trådning kan vara bra. Faktum är att prototypen i ett ganska tidigt skede var flertrådig. Jag tog bort detta då, eftersom det fanns en del

[5] Denna plugin använder kodbiblioteket Secret Rabbit Code.

problem som jag misstänkte berodde på trådningen. Nu är klienten stabil och kanske det skulle vara på sin plats att återinföra trådningen. Datorer med dualprocessorer blir allt vanligare och med trådning skulle klienten kunna bli mycket effektivare på en sådan maskin.

Systemet har ej genomgått utförliga säkerhetstester. Det kan därmed finnas allvarliga säkerhetsluckor. I första hand syftar jag på ”buffer overflow”-attacker. En ordentlig genomgång av den kod som packar upp inkommande paket hos såväl server som klient skulle behöva genomföras.

Kapitel 8

Resultat och slutsatser

I det här avslutande kapitlet diskuterar jag resultatet, de prototyper som jag tagit fram, och de slutsatser som tester och användning av dessa har gett. Slutligen tar jag upp några förslag på möjliga förbättringar och utvecklingar av systemet.

8.1 Resultat

Under arbetets gång har jag tagit fram en server och en klient. Servern kallar jag Bongomon och klienten bongo~(i Pd markerar man de moduler som hanterar signaler med en tilde i namnet). Dessa två delar en gemensam kodbas med implementationer av nätverksfunktioner, RTP och diverse hjälpverktyg. Koden för allt detta är alltför omfångsrik för att tas upp i appendix. Den gemensamma kodbasen är cirka 5500 rader, klienten cirka 6000 rader och servern är ungefär 1000 rader. Man får koden enklast genom att maila mig eftersom den än så länge inte finns tillgänglig på nätet.

Ser man till de kriterier (avsnitt 2.3) som jag satte upp innan jag började implementera prototypen så har ljudsystemet fallit väl ut. Ljudkvaliteten i systemet är mycket god, endast högkvalitativ ljudkompression används och det finns även möjlighet att skicka okomprimerade flöden.

Ljudkvalitet och latens i ett distribuerat system är som två motverkande krafter. Drar man för mycket i den ena blir den andra genast lidande. Därför har jag ständigt tvingats till avvägningar och kompromisser som jag nämnt på flera ställen i rapporten. I denna kamp mellan två av kriterierna har jag prioriterat ljudkvaliteten något mer än latensen. Latensen i systemet är ändå låg, enligt min uppfattning. Detta tack vare bra nätverksprotokoll, god hantering av interna buffertar och effektiv programkod. Använder man dessutom okomprimerade ljudflöden blir latensen mycket låg. Systemet har i tester mellan Umeå och Stockholm (mer om dessa nedan) utan problem klarat latensens smärtgräns på 250 millisekunder [6].

Prototypen är generell och utbyggbar. Den är i första hand skriven för att användas till distribuerade 3d-ljudflöden men fungerar bra i många andra sammanhang, till exempel som ett traditionellt konferenssystem. Den modulbaserade kodkärnan som beskrivs i 7.4.4 gör det lätt att byta ut många funktionellt viktiga delar av

systemet, exempelvis de som har med ljudkompression att göra.

När det gäller plattformar så fungerar såväl klient som server idag för både Windows och Linux. Jag har inte provat att kompilera i Mac OS X, men det borde inte vara något större problem. Givetvis fungerar systemet även mellan olika plattformar, det vill säga klienter under olika operativsystem pratar utan några problem med varandra.

8.1.1 Testsändningar mellan Umeå och Stockholm

Jag har haft möjlighet att utföra testsändningar mellan Umeå och Stockholm. Förutom att vara ett bra avlusningsredskap har det även gett mig en bild av hur pass bra överföringar över Internet fungerar (se min diskussion kring trovärdigheten av nättester i 3.4). Detta har gett mig en något annorlunda bild än den som presenteras i [2], [3] och [9].

Tester över den aktuella rutten har gett förvånansvärt få paketförluster. Rutten har gått över vanligtvis tio routrar och paketförlusterna har bara varit bråkdelar av en procent. Vid ett test skickade jag sex parallella ljudflöden utan komprimering i flera timmar. De totala paketförlusterna kunde räknas i tiotal. Antalet paketförluster var så små att jag var tvungen att simulera förluster lokalt för att kunna testa de olika metoderna för maskering av paketförluster. Jag vill inte dra några förhastade slutsatser av detta eftersom mina tester varit enkla och inte av vetenskaplig kaliber. Däremot anser jag att man kan ana en tendens att trafik över Internet håller högre kvalitet idag än för några år sedan, vilket inte är så förvånande då tekniken ständigt förbättras.

Ett modernt test med överföringar både med TCP och UDP (unicast och multicast), simulering av olika typer av nätverkstrafik och mellan ett flertal olika noder i nätet vore onekligen mycket intressant att ta del av. Något sådant har jag dock inte kunnat hitta.

Tester mellan Umeå och Stockholm gav mig även en bild av latensen. Jag genomförde ett antal konversationer via ljudsystemet (både komprimerade och okomprimerade ljudflöden användes) och fördröjningen upplevdes aldrig som störande eller ens noterbar. Utifrån detta har jag dragit slutsatsen att latensen ligger gott och väl innanför smärtgränsen 250 millisekunder [6]. Exakta tidtagningar mellan flera olika noder på nätet hade förstås varit att föredra, men ett sådant test har ej kunnat genomföras.

8.1.2 Komprimerade eller okomprimerade ljudflöden?

Under examensarbetets gång har jag gått från att noggrant utvärdera och testa olika kompressionstekniker till att mer och mer testa okomprimerade ljudflöden. Med dagens bandbredd är det helt klart möjligt. I mina tester har jag använt ljud med 16 bitars upplösning i 22kHz för okomprimerade ljudflöden, vilket ger ett bitflöde på ungefär 350kbps. Det är inte ovanligt idag med uppkopplingar med 10Mbit vid både upp- och nedladdning. För en sådan uppkoppling ryms det många okomprimerade ljudflöden med nämnd kvalitet. Möjligheten att använda okomprimerade ljudflöden

beror även på hur mycket annan data som ska skickas över nätet parallellt med ljudet.

En annan viktig erfarenhet från utvecklingen av prototypen är att moderna högkvalitativa kompressionsformat visserligen låter relativt bra, men de är inte sällan beräkningstunga för processorn. Ofta ger de också ett betydande bidrag till latensen. Speex, som jag har testat och utvärderat mest, ger en fördröjning på minst 30 millisekunder [14]. Dessa faktorer är acceptabla i de flesta tillämpningar, men det kan vara bra att ha detta i åtanke när man ska använda ljudsystemet i någon applikation. Är det mycket som ska köras samtidigt och datorkraften är i minsta laget kan det bli för tungt att använda till exempel Speex.

8.1.3 Prototypen som del i kultursimulatorprojektet

Prototypen var tänkt som en del i den andra fasen av kultursimulatorprojektet som ägde rum i Barcelona 2004. Tyvärr kunde inte denna fas genomföras som planerat. Projektet var stort och bestod av en mängd olika delar som skulle integreras. In i det sista var situationen kaosartad då många delar var kraftigt försenade. Detta ledde till att ett ordentligt test, en samkörning av projektets alla delar, aldrig kunde genomföras som planerat. Eftersom olika komponenter var beroende av varandra, mitt ljudsystem behövde till exempel matas med virtuella positioner från VR-systemet, hade ett sådant test varit nödvändigt. Naturligtvis blev det stora problem och i slutändan genomfördes projektet i en kraftigt nedbantad version där bland annat mitt ljudsystem inte fanns med.

Av de många problem som uppstod var det allvarligaste från min sida sett att prestandan på de datorer som användes inte räckte till. Många olika komponenter skulle köras på en och samma dator, och processorn klarade inte av trycket. På grund av de uteblivna testerna var detta något som upptäcktes på plats.

Projektets kollaps har förstås påverkat mitt examensarbete på flera olika sätt. I dagens läge är ljudsystemet inte ”fullskaligt” testat med koppling till VR-system och med uppspelning av 3d-ljud. Därför har jag gått miste om viktiga användarkommentarer kring funktionaliteten i systemet. Det hade varit intressant att ta del av människors upplevelser av ljudsystemet, av 3d-ljud, i fråga om närvarokänsla och andra faktorer. Den utvärdering och de slutsatser jag kan dra av mitt examensarbete har därför en rent teknisk karaktär. Även en funktionell utvärdering med betoning på användarupplevelser hade givetvis varit önskvärd och kunnat ge rapporten en mer vetenskaplig förankring.

8.2 Slutsatser

Som jag ser det är den tekniska premissen till målet uppfylld. Bongomon, bongo~och Snd3d utgör tillsammans ett komplett och generellt system för distribution och uppspelning av 3d-ljud. Ljudsystemet har, som jag beskrev här ovan, inte testats i ett komplett VR-sammanhang. Jag har dock gjort lokala tester där jag simulerar kopplingen till ett VR-system och vet att systemet hanterar kopplingen på ett korrekt

och fungerande sätt. Ljudsystemet med alla dess komponenter är alltså testat för sig självt och fungerar väl.

Att systemet inte har testats tillsammans med en riktig VR-koppling med uppspelning i en 3d-ljudanläggning gör att målet med examensarbetet ännu inte helt är uppfyllt. Jag och Peter Lundén hoppas att inom en inte allt för avlägsen framtid kunna sätta ihop en fristående demo av hela ljudsystemet så som det var tänkt att användas i kultursimulatorprojektet. Att kunna demonstrera ett sådant scenario som jag beskrev i slutet av 2.2 vore förstås idealt.

8.3 Vidareutveckling

Även om systemet idag fungerar bra så finns det en hel del som går att göra, både i form av förbättringar och nya tillägg. Här går jag igenom några av de mer intressanta.

8.3.1 Akustiksimulering

Systemet utnyttjar idag geometrisk simulering av ljudet. Nästa steg är att även tillföra akustisk simulering som handlar om att återskapa den klangfärg som ljudet får i olika geometriska sammanhang. Klangfärgen är resultatet av framför allt reflektioner, transmissioner och diffraktion i den geometriska omgivningen. Det finns idag metoder för att göra detta i realtid, till exempel med hjälp av så kallad ”beam tracing”[5]. Metoden bygger på att man från en ljudkälla skickar ut strålknippen och följer dessa ett bestämt antal steg. Ett steg i processen utgörs av en reflektion eller transmission. Det är alltså en variant av ray tracing. Utifrån detta bygger man upp en geometrisk trädstruktur där man snabbt kan slå upp varje position i geometrin för att bygga upp ett reflektionsmönster med tillhörande filtreringar. När man väl har byggt upp denna trädstruktur går det snabbt att simulera akustiken från alla positioner i den virtuella miljön.

8.3.2 Filtrering av återkoppling

Som jag nämnde i problemdefinitionen i kapitel 2 utgör återkoppling från högtalare till mikrofon ett problem. Systemet är i sitt nuvarande skick känsligt för rundgång. Det är ett komplext problem där en hel del forskning bedrivs. Det skulle kunna vara intressant att se på de metoder som finns för mono- och stereosystem för att ta reda på om det går att generalisera något av detta och skapa en metod för ekocancellering i multikanalssystem.

8.3.3 Kryptering av nätverkstrafiken

Vill man bygga ett helt generellt system för distribuerat 3d-ljud kan man tänka sig att säker kommunikation med krypterade dataflöden och autentisering kan vara

intressant i vissa tillämpningar. För att kryptera dataflöden skulle exempelvis AES[1] kunna användas. Det är ett snabbt blockkrypto med nyckellängder på upp till 256 bitar. Det skulle vara relativt enkelt att utöka den nuvarande RTP-implementationen med stöd för AES.

8.3.4 Bättre utnyttjande av RTCP

Tack vare RTCP erbjuds systemet flera bra redskap för att övervaka kvaliteten på nätverkstrafiken. Klienten utnyttjar till exempel jittervärdet för att sätta buffertlängden adaptivt som det beskrevs i avsnitt 4.4.2. Det går att göra mer med RTCP.

I 4.4.2 beskrev jag hur man kan använda RTCP för att styra kvalitetsinställningar för ljudkompression även den adaptivt. Detta är någonting som skulle kunna testas. Jag är dock inte säker på att det är en metod att föredra då sådana kvalitetomställningar kan vara mycket störande, ett exempel på det är ljudflöden med Real Audio. När ljudet plötsligt går från klart och tydligt till dovt och burkigt kan användaren lätt tro att något fel har uppstått. I ett sådant läge övergår ljudet från att vara en funktionell och ”upplevelsehöjande” subtil faktor till att enbart vara ett störande inslag.

8.3.5 Testa multicastlösningar för servern

Det nuvarande ljudsystemet skulle inte klara av att hantera riktigt stora VR-applikationer med hundratals eller kanske tusentals närvarande avatarer. För att råda bot på detta behövs en annan nätverksmodell för servern. I avsnitt 6.1.1 går jag igenom ett flertal olika förslag på hur man skulle kunna bygga om servern.

[1] Advanced Encryption Standard

Litteraturförteckning

[1] Durand R. Begault. *3-d Sound for Virtual Reality and Multimedia*. Morgan Kaufmann, 1994.

[2] Jean-Chrysostome Bolot. End-to-end packet delay and loss behavior in the internet. *ACM SIGCOMM Computer Communication Review*, 1993.

[3] Jean-Chrysostome Bolot, Hugues Crepin och Andreas Vega Garcia. Analysis of audio packet loss in the internet. I *Network and Operating System Support for Digital Audio and Video*, ss 154–165, 1995.

[4] Douglas E. Comer. *Computer Networks and Internets with Internet Applications*. Prentice Hall, 3rd utgåvan, 2001.

[5] Thomas Funkhouser, Ingrid Carlbom, Gary Elko, Gopal Pingali, Mohan Sondhi och Jim West. A beam tracing approach to acoustic modeling for interactive virtual environments. I *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ss 21–32. ACM Press, 1998.

[6] I. Kouvelas, O. Hodson, V. Hardman och J. Crowcroft. *Redundancy control in real-time Internet audio conferencing*, 1997.

[7] Isidor Kouvelas och Vicky Hardman. Overcoming workstation scheduling problems in a real-time audio tool. ss 235–242, 1997.

[8] Sue B. Moon, Jim Kurose och Don Towsley. Packet audio playout delay adjustment: performance bounds and algorithms. *Multimedia Syst.*, 6(1):17–28, 1998.

[9] Vern Paxson. End-to-end Internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[10] C. Perkins, O. Hodson och V. Hardman. *A survey of packet-loss recovery techniques for streaming audio*, 1998.

[11] Ramachandran Ramjee, James F. Kurose, Donald F. Towsley och Henning Schulzrinne. Adaptive playout mechanisms for packetized audio applications in wide-area networks. I *INFOCOM (2)*, ss 680–688, 1994.

[12] Schulzrinne, Casner, Frederick och Jacobson. RTP: A transport protocol for real-time applications. *Draft Standard, RFC 3550*, 2003.

[13] W. Richard Stevens. *Networking APIs: Sockets and XTI*, band 1 av *UNIX Network Programming*. Prentice Hall, 2nd utgåvan, 1998.

[14] Jean-Marc Valin. *The Speex Codec Manual (version 1.0)*, 2003.

[15] Lixia Zhang, Stephen Deering och Deborah Estrin. RSVP: A new resource ReSerVation protocol. *IEEE network*, 7(5):8–?, September 1993.

Bilaga A

Tabeller från kompressionstest

format	inställningar	kodnings-tid (s)	avkodnings-tid (s)	bitflöde (kbps)
wav	—	—	—	1411
flac	medium	12	7	952
ape	normal	11	15	927
wma	none	27	24	938

Tabell A.1. Black Sabbath - Iron man (5:56)

format	inställningar	kodnings-tid (s)	avkodnings-tid (s)	bitflöde (kbps)
wav	—	—	—	1411
flac	medium	5	1	769
ape	normal	4	5	729
wma	none	11	10	736

Tabell A.2. The Beatles - In my life (2:27)

format	inställningar	kodnings-tid (s)	avkodnings-tid (s)	bitflöde (kbps)
wav	—	—	—	1411
flac	medium	22	10	594
ape	normal	23	25	562
wma	none	46	42	570

Tabell A.3. Beethoven: 9:e symfonin, andra partiet (10:26)

format	inställningar	kodnings-tid (s)	avkodnings-tid (s)	bitflöde (kbps)
wav	—	—	—	706
flac	medium	2	1	300
ape	normal	3	4	277
wma	none	11	10	278

Tabell A.4. Tal i mono (2:31)