Delt lyd over IP netværk

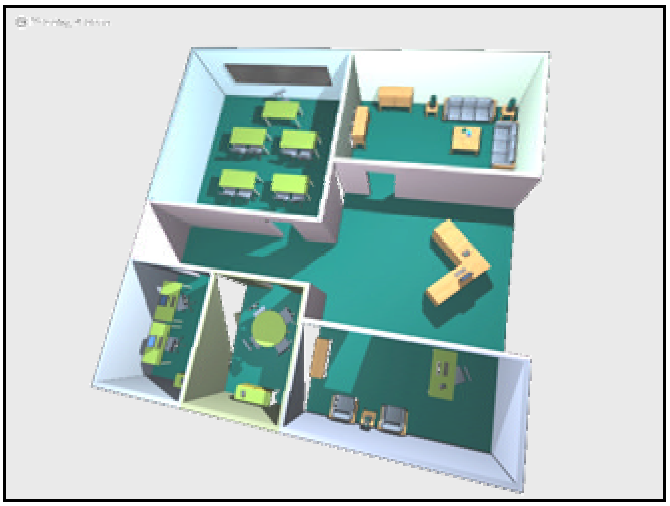
Borch, Ole; Mathiassen, J.S.; Møller, H.J.

Publication date:
2004

Document Version
Også kaldet Forlagets PDF

Virtual College™
LYD

Appendix 3: Delt lyd udviklingsrapport



Projekt MIKS under DDN

Ole E. M. Borch, Aalborg Universitet
Jesper Skelmose Mathiassen, Aalborg Universitet
Hans Jacob Møller, Aalborg Universitet

ID: DDN.MIKS.Main001.Appendix003
10. februar 2004

Indholdfortegnelse:

Introduktion

Samarbejdende aktiviteter over netværk leder naturligt tanken hen på anvendelse af multitast som kommunikationsmetode. Princippet går ud på at alle klienter lytter efter datapakker på nettet indeholdende en defineret IP-multicastadresse. Når blot en eller flere klienter sender på samme adresse, vil alle modtage samme data. Der er altså ingen server involveret – kun datanettet. Problemet med multicast, er at der anvendes UDP som transportprotokol (ofte lukket da UDP er connection-less og derfor virker som 'reklamepostkasse') og at multicast typisk ikke forwardes af rutere. Derfor ønskes en application der benytter UDP med mindst mulige porte. Problemet er om der også skal anvendes en fast server eller denne centrale facilitet skal ligge i alle klienter hvor et abitreringssystem skal afgøre hvem der skal være server. Andre variationer findes også. Her vælges en client/server løsning.

1 Foranalyse

I dette kapitel vil rellevante systemer blive analyseret. Desuden vil der være en beskrivelse af forskellige protokoller der kan have rellevans.

1.1 Protokoller

1.1.1 TCP (Transport Control Protocol) [RFC793]

TCP er den gængse transportprotokol på internettet. Den er robust, og generelt et godt valg til overførsel af data.
Den har dog en del komplikationer i forbindelse med realtids audio, bl.a. kan dens congestion control, hvor transmissionraten midlertidigt reduceres, skabe huller i datastrømmen. Dette er brugbart for almindelig streaming media da den gennemsnitlige transmissions rate er mere konsistent, og tillader mange forbindelser at skalere sig mod hinnanden.
TCP bygger også på et data aknowledge system, som sikre at tabte pakker bliver retransmiteret. I forbindelse med audio kan det betyde at en sample ankommer efter den skulle være afspillet, og er derfor spildt båndbredde, da den er forældet.

1.1.2 HTTP (HyperTekst Transfer Protocol) [RFC1945]

HTTP bygges ovenpå en TCP forbindelse, og fungere efter besked/svar princippet.
HTTP bruges hyppigt i forbindelse med netradio, hvor en stabil strøm af data, der kan modtages gennem proxy servere og firewalls er ønskeligt. Der bruges ofte en meget stor lokal buffer til at kompensere for TCP protokolens slowstart, og congestion kontrol, ofte 10-15 sekunder.

1.1.3 UDP (Universal Datagram Protocol) [RFC768]

UDP bruges flittigt i Voice over IP sammenhænge pga. det lave delay og meget høje udnyttelsesgrad. UDP er dog blokeret i visse firewall, men flere applikationer kompensere for dette ved at bygge sig op så det stille så lille et krav til hullet i firewallen som muligt. Ofte (se Team Speak og Pico Phone) ved at kun bruge én enkelt port, som kan ændres hvis tilpasning er nødvendigt. Client/Server modellen er også brugt, hvor man kan bruge en server mellem to firewalls, så indgående forbindelser ikke er et krav.

1.1.4 RTP (Real Time Protocol) [RFC1889]

RTP er en transport protokol brugt til lyd- og billedemedia, transmiteret over UDP, og giver faciliteter til f.eks. synkronisering af lyd og billede, samt codec specificering.
RTP laver dog et vist overhead, og skal kombineres med RTCP (Real Time Control Protocol) for at fungere optimalt.

1.1.5 SIP (Session Initiation Protocol) [RFC2543]

SIP eller Session initiate protokol, er en protokol standard beregnet til at brugere kan kalde op, og udveksle forskellige informationer før selve opkaldet. F.eks. om transportprotokol og codec.

SIP kan fungere ovenpå TCP eller UDP, og kan udveksle informationer om firewalls og evt. Proxy servere i brug, og dermed hjælpe programmer med at etablere den bedst mulige forbindelse inden selve forbindelsen, og dermed opkaldet, bliver oprettet.
Et muligt scenarie kunne f.eks. være at finde ud af hvilke, om nogen, af klienterne den kan modtage UDP pakker, eller og en alternativ transport løsning, f.eks. via TCP, skal søges.

1.2 Codecs

Da systemet implementers ved hjælp af en klient-server struktur, er det fornunftigt at implementere en komprimering af lyden inden den skal sendes over netværket. Da systemets primære krav er lav latens og høj lyd kvalitet skal disse tages i betragtning, ved et valg af codecs. Desuden er det vigtigt at codecs ikke belaster computeren det skal bruges på for meget, da der også skal tages højde for andre krævende del systemer som for eksempel streaming video.

1.2.1 GSM [TRIT]

Tritonus er en implementation af Java Sound API, der tilføjer funktionalitet til den allerede eksisterende API. Derudover stilles der en del ”plugins” til rådighed, som ikke kræver andet en Java.Sound for at fungere. Blandt disse plugins findes forskellige implementationer af lydcodecs, f.eks GSM med en ~ 1:10 komprimering hvor Java.Sound’s oprindelige muligheder for lyd komprimering begrænser sig til my- og a-law codecs der giver en 1:2 komprimering. Der findes en god API skrevet i javadoc og projektet er open source, under GNU Library General Public License.
Tritonuses GSM codec kræver minimum 320 bytes PCM signed 8kHz 16-bit lyd som så bliver komprimeret til 33 byte. Dette vil sige at framestørrelsen som minimum skal være :
Ikke Komprimeret Datarate = 16bit * 1channel * 8000Hz = 16byte/ms = 128kbit/s.
Komprimeret Datarate = 128kbit/s * 33byte/320byte = 13.2kbit/s
320 bytes 16 bit PCM lyd svare til 160/8000 sekunder = 20 ms
Det ses at en GSM komprimering vil medføre en minimums framesize på 20 ms.
Desuden er der en forsinkelse i selve GSM codec pga. beregningstid, dette må dog forventes at kunne kodes og dekodes på <20ms. da GSM er beregnet på streaming lyd, dvs. worst case for GSM codec vil være 60ms tilføjet delay fra en person taler til en anden person hører det talte.
Dette codec ville nemt kunne implementeres da det har klare interfaces. der er en klasse med to metoder,en encode og en decode, der skal bruges.
Der er på nuværende tidspunkt ikke lavet en færdig test implementation for at teste delay i GSM. Men der findes programmer der benytter sig af det på streaming lyd over internettet et eksempel er eConf.

1.2.2 Speex [SPX]

Speex er et Open Source codec til brug ved tale compression. Speex er ikke noget letvægts codec, men levere god kvalitet og lavt delay, selv ved lave bitrates. Ifølge deres side kan det komprimere hårdere end GSM, ved samme audible kvalitet.
Speex findes i en nemt tilgænglig Java version, der ved prototype implementation viste sig nemt anvendelig.

1.3 Test af codecs

Der er blevet testet 2 forskellige implementationer af codecs i java. **Speex** og **Tritornus GSM** .

Testen er foretaget på en 16 sekunders lydssample [1]hvor en mand taler i opløsningen : 8000kHz, 16bit og er i ukomprimeret PCM[2].
Ud fra kriterierne til systemet er følgene kvalitetsmål blevet målt :

1. **Lydkvalitet** En subjektiv vurdering af lydkvaliteten, ud fra en skala fra 0 til 6 hvor :
   0: Ingen lydkvalitet Totalt Uforståeligt.
   1: Dårlig lydkvalitet. Svært uforståeligt.
   2: Rimelig lydkvalitet. Enkelte dele kan være uforståelig, men helheden forstås.
   3: Tilfredsstillende lydkvalitet. Kun sjældent er lyden uforståelig. Nogen støj og skratten er tilladt.
   4: God Lydkvalitet. Lyden er fuld forståelig. Dog opleves nogen støj eller skratten.
   5: God Lydkvalitet. Lyden er fuld forståelig. Nærmest ingen støj eller skratten.
   6: Skarp Lydkvalitet. "Ren" tale. Ingen støj eller skratten.
2. **Bitrate** Antal bits pr sekund lyden er efter kompresion.
3. **CPU belastning** Hvormeget cpuen på test computeren er belastet i gennemsnit under enkodning og dekodning.
4. **Delay** Tilføjet latens i millisekunder.

Testprogram i psodo kode :

1. *byte[] <- sound*
2. *startmillis = systemtime*
3. *compressed[] = encode <- byte[0+i,319+i]*
4. *compressedtime = compressedtime + (systemtime - startmillis)*
5. *startmillis = systemtime*
6. *decompressed[] = decode <- compressed[]*
7. *decompressedtime = decompressedtime + (systemtime - startmillis)*
8. *i = i+320*
9. *play decompressed[]*
10. *if length of byte[] >= i :*
11. *goto 2*
12. *else :*
13. *screen <- (compressiontime*320)/length of byte[]*
14. *screen <- (decompressiontime*320)/length of byte[]*
15. *exit*

Testen gav følgende resultater for de respective codecs :[3]

[1] Filen der blev brugt hedder mand16b.wav
[2] Pulse Code Modulation, standart lyd der sendes/modtages fra lydkortet.

Codec :	**GSM**	**Speex VBR**[4]	**Speex 11.2k**	**Speex 6.0k**	**Speex 2.4k**
Lydkvalitet :	5	5	5	4	3
Bitrate :	13.2kbs	6-24.8kbs,15.2kbs	11.2kbs	6kbs	2.4kbs
Belastning :	~10%	~26%	~20%	~14%	~10%
Delay :	<<1ms	10ms	<<1ms	<<1ms	<<1ms

Lydkvaliteten var god i de tre første codecs, og fint brugbar i Speex 6.0k dog med en smule metallisk klang. Speex2.4k tilføjede en del metallisk klang til signalet der dog stadigt var hørebart.

Belastningen var klart laves ved GSM codeces i forhold til de tre andre codec af samme lydkvalitet (minimum 10% lavere) , men havde en smule højere bitrate en Speex11.2k.

Det ses af testen at Delay er minimal ved kompression / Dekompression, untagen for Speex VBR hviket yderligere tilføjer 10 ms delay i kompressionen af signalet. Der skal dog også tages højde for minimum rammestørrelsen for alle codecsene er 20 ms hvilket i sidste ende vil tilføje 40 ms delay.

Der blev desuden lavet test med en 16000Hz version af samme lydsample, Lydkvaliteten blev ikke væsentligt forbedret, og for alle Speex codecs var der stadigt tale om minimum 20 ms rammer, og belastningen steg betragteligt. GSM derimod kan nøjes med 10 ms pakker ved denne hvilket halvere latens som følge af minimum pakkestørrelse, desuden var der ingen mærkbar belastningsforøgelse. Denne fordobling af frekvensen vil dog også medføre en fordobling af bitraten for alle codecs.

Et max båndbreddeforbrug for serveren vil kunne udregnes ved hjælp af følgende formel :

Maxbåndbreddeforbrug = n*bitrate.

Dette er gældende for både upload og download, givet at serveren mixer signalerne.

Ved eksempeltvist 10 brugere der bruger GSM og alle 10 taler vil der være tale om 132kbs både op og ned. Det gælder at uploadraten forbliver stabilt på maxbåndbreddeforbrug ved et givent antal brugere så længe der tales, hvorimod download vokser lineært med antal brugere der taler.

---

[3] Testen blev foretaget på en Pentium 4 1.8 Ghz med 768 MB Ram.

[4] VBR = Variable BitRate, De tre værdier I bitrate angiver minimum, maksimum og gennemsnit.

1.4 Audio konference systemer

1.4.1 Team Speak [TSHP]

Teamspeak er et audio baseret conference værktøj, minded på computerspillere på Internettet. Det giver mulighed for at flere brugere (spillere) kan tale sammen via Voice over IP, og spillere kan indele sig i grupper og undergrupper (f.eks. spil, og hold).
Teamspeak er pga. sit tilhørsforhold til online spils verdenen, produceret med forskellige parametre:
Lavt delay. Da Teamspeak skal andvendes i realtid skal kommunikation kunne foregå ubesværet og uden forsinkelser.
Lavt båndbreddeforbrug. Ved online spil kan en belastet forbindelse foringe spillet, og dermed mindske chancerne i forhold til med/modspillere.
Firewallkompatibilitet. Da spil allerede bruger UDP kan Team Speak det også, men benytter en simpel protokol, der nemt kan benyttes i forbindelse med NAT[5].
Teamspeak er Freeware men desværre closed source, således kan det ikke bruges direkte til andet end sit formål, men indeholder mange ideer, som den simplicficerede applications protokol der går igen i andre projekter.

1.4.1.1 Test af Teamspeak

Opsætning:
1 Team Speak klient på en 350Mhz maskine med Windows 2000.
1 Fjernserver(Team Speak Public server. Formodet lokalitet i Bayern Tyskland)
1 Nærserver. Privat, placeret via MAN[6] ca. 2.5 km væk.
Udførsel:
Klienten blev koblet op til hver server, og lydkvaliteten/latens tiden
blev forsøgt målt og vurderet.
Codec Speex blev brugt i begge tilfælde. Dette Codec er normeret til ca. 30ms.
delay bidrag, og var påtvunget på fjernserveren.
Resultat:
Delay kunne ikke bestemmes præcist til nogen af serverne, men skønnes <1 sekund til fjernserveren, og mindre endnu til nærserveren (estimeret til ca. 60-70ms, på basis af sammenligning med andre programmer).
Lydkvaliteten var god. På fjernserveren kunne der høres en smule knitren, men på nærserveren var det nærnaturlig tale, meget lig telefon.

[5] Network Address Translation. Et system der gør det muligt for flere hosts at dele same ip adresse.
[6] Metropolitan Area Network. Et netværk med flere routere, der ofte spænder flere kilometre.

1.4.2 PicoPhone [PICO]

PicoPhone er et internettelefoni værktøj, der er opbygget til at kunne efterligne funktionaliteten i det almindelige telefonsystem.
PicoPhone er bygget op til at skulle være simpelt at bruge for brugeren, og arbejde med samme kvalitet og brugervenlighed som det almindelige telefonsystem. Det er bygget til at bruge GSM codec, som også kendes i telefonverdenen, men er udbygget til at supportere andre. Lavt delay er vigtigt for PicoPhone, og er forsøgt holdt lavt, samtidig med at brugeren hele tiden kan se en opmåling af transmissionstiden i sit vindue.
PicoPhone benytter en hjemmelavet UDP baseret protokol til at sørge for lavt delay og NAT (firewall) gennemtrængning.
PicoPhone er closed source, men har nogle ideer der kan genbruges. Blandt andet en meget nemt styring af softwareforstærkning.

1.4.2.1 Test af PicoPhone

Opsætning:
2 Maskiner med PicoPhone. De to maskiner var forbundet via LAN, pga. manglende muligheder for større afstande.
Udførsel:
Den ene maskine ringede op til den anden, og latens og kvalitet blev observeret.
Resultat:
PicoPhone meldte selv latenstider på 70-100ms med GSM codec, Dette kunne dog ikke verficeres præcist. Samtalen var i fin kvalitet, og var fri for jitter og andre atrefakter.

1.4.3 Speak Freely [SPKF]

Speak Freely er et Open Source projekt minded på flexibelt internet telefoni. Speakfreely benytter RTP over UDP til kommunikation, og har faciliteter til Jitter korektion og pakketab.
Speak Freely fokusere på person til person samtale med høj lydkvalitet og standardiserede kommunikationsveje, hvilket gør programmet kompatibelt med andre programmer, eksempelvis microsoft netmeeting.
Kildekoden til Speakfreely er frit tilgænglig, men er skrevet i C, og er ganske omfattene.

1.4.3.1 Test af Speak Freely

Opsætning:
GSM codec blev brugt. Jitter compensation slået fra.
En echo server i schweiz (dvs. på WAN) blev benyttet med ping statistik: E = 75 ms.
Udførsel:
Da echo serveren gemmer på lyden i 10 sekunder før den bliver sendt tilbage, blev tiden fra transmission til reception målt ved hjælp af et stopur, og fratrukket 10 sekunder. Dette blev gjort 3 gange og der blev målt gennemsnit.
Resultat:
Total latenstid lå på knapt 1 sek. Men der bør regnes med en hvis unøjagtighed i målingerne.
Lydkvaliteten var tydelig, men med nogen skratten.

1.4.4 eConf [ECNF]

eConf er et produkt der kan bruges til undervisning over Internettet, det er skrevet i java og er under GPL[7] licens. Ideen ved eConf er at en lærer optager en ”session” og derefter kan studerende gå ind på en hjemmeside og følge denne optagelse. Da selve sessionen ikke er realtid foregår der heller ingen tovejs realtids kommunikation. Til lyddelen kan benyttes GSM komprimering der er baseret på tritonus´ plugin.

[7] GNU Public Licence.

2 Analyse

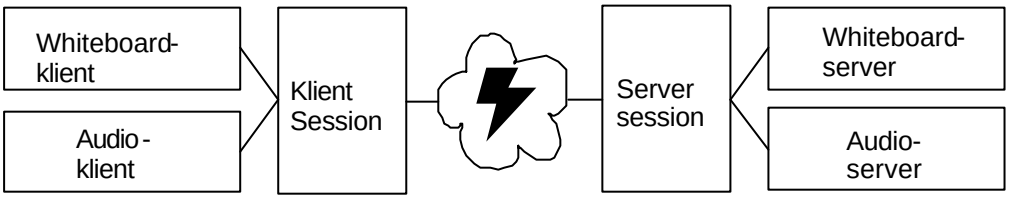
I analysen vil delproblemer blive identificeret, og ideer til system opbygning vil blive beskrevet. Denne analyse vil lede til en problemformulering.

2.1 Analyse af anvendelsesområde

I analysen af anvendelsesområdet analyseres de aktører de enkelte delproblemer skal interagere med. For at give et større overblik, vil opdelingen af problemstillingen blive klarlagt.

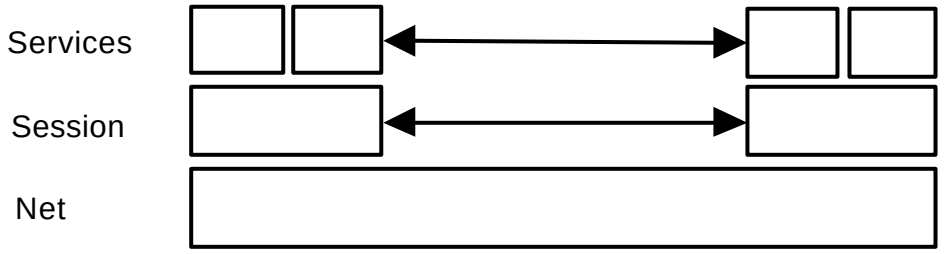
2.1.1 Problemopdeling

Da det er afgørende at både whiteboard-data og audio-data kan sendes over netværket samtidigt uden at blokere for hinanden, anses kommunikationskanalen som et fælles problem, således data-traffikken kan overvåges og reguleres. Denne fælles kommunikationskanal mellem klient-problemerne og server-problemerne består af to delproblemer, klient session og server session, på henholdsvis klientsiden og serversiden. Dette er visualiseret i Figur 1.



Figur 1

Denne arkitektur kan også opfattes som en protokolstak, hvor applikationerne yderst i arkitekturen ligger øverst på stakken. De underliggende lag i protokolstakken er for disse applikationer transparente. Dette er visualiseret på Figur 2.



Figur 2

2.1.2 Idenfikation af aktøre

De enkelte delproblemers anvendelsesområde har tilsammen følgende aktøre.

2.1.2.1 Netværk

Netaktøren representerer det netværk, som forbinder de enkelte computere med hinanden.

Det er en vigtig aktør på grund af netværkets nondeterministiske natur og fastlægelsen af de protokoller der skal anvendes for at kunne kommunikere over netværket.

2.1.2.2 Bruger

Bruger aktøren er den aktør som interagerer med systemet udefra. Brugeren stiller i dette projekt krav om funktionalitet og grafik. Brugeren er på samme tid producer og consumer.

2.1.2.3 Mikrofon

Mikrofonen er producer for audiodelen. Problemstillingerne omkring mikrofonen består i at kunne processecere de data der genereres, inden nye data er klar.

2.1.2.4 Højtaler

Højtaleren er consumer for systemet. Højtaleraktøren er vigtig da den rejser problemstillinger med hensyn til at afspille lyd kontinuert og i en acceptabel kvalitet.

2.1.2.5 Admin

Admin aktøren stiller krav om at kunne administere systemets tilstand. Problemstillingen som opstår på grund af admin er aflæsning af de tilstande systemet har været i og mulighed for at påvirke systemet på en udvidet måde. Admin kan både være producer og consumer.

2.1.2.6 Klientsession

Aktøren klientsession er klientens adgang til netværket og stiller krav til kommunikationen. Klientsession er en consumer.

2.1.2.7 Serversession

Aktøren servicesession fungerer som adgang til netværket for serverservicene.

2.1.2.8 Klientservice

Klientservice er en aktør i forhold til klientsession. Den stiller krav til dataoverførsel over netværket. Klientservice skal kunne sende data til en server.

2.1.2.9 Serverservice

Serverservice er en aktør i forhold til særversession. Serverservice stiller på samme måde som klientservice krav til dataoverførsel over netværket. Endvidere stiller serverservice krav om at kunne sende data ud til flere klienter.

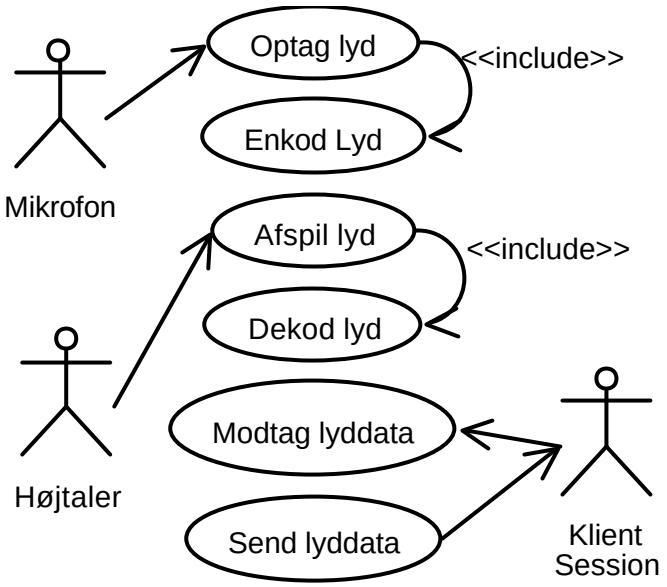
2.2 Klient

2.2.1 Audio

2.2.1.1 Usecases

Et usecase diagram er lavet for lyd applikationen, her illustreres de forventede

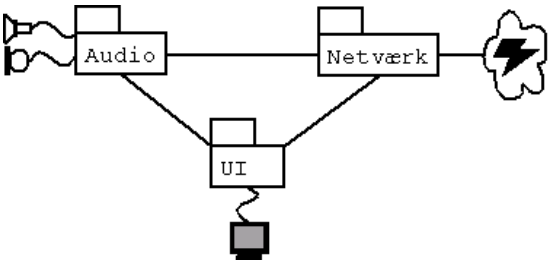
funktionaliteter systemet skal stille til rådighed for aktørene dette er vist i Figur 2-4.



Figur 2-3

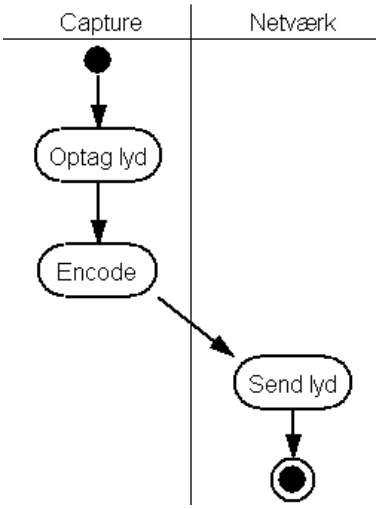
2.2.1.2 Delproblemer

Ud fra usecase diagrammet kan der opstilles en model for audiosystemet der består af en række delproblemer som vist i Figur 2-4. Disse er Audio systemet der håndtere afspilning og optagelse af lyd, netværks systemet der håndtere trafik til og fra netværket og UI som er brugerens administrative kommunikationsvej for brugeren til de to andre del systemer. Ud fra disse Delsystemer og aktivitets diagrammet i Figur 2-3. Ud fra dette konstrueres der swimlanes der viser hvilke funktioner de forskellige delsystemer varetager.



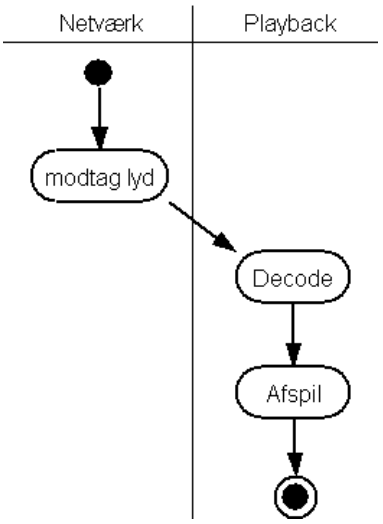
Figur 2-4

2.2.1.3 Swimlanes



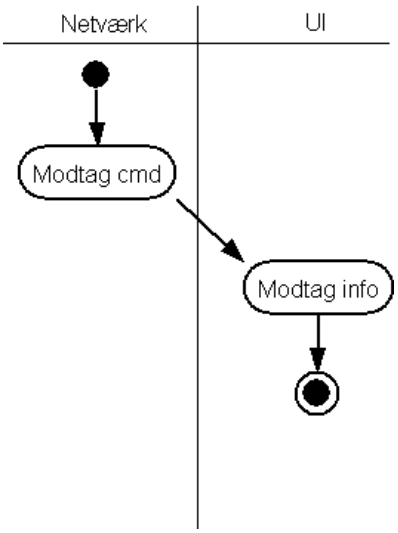
Figur 2-5

På Figur 2-5 ses et aktivitets diagram for optagerdelen, capture, og netværksdelen. Pilene beskriver flowet igennem systemet og viser hvordan lyden kommer ind i systemet går igennem en række aktiviteter for tilsidst at blive sendt ud på netværket.



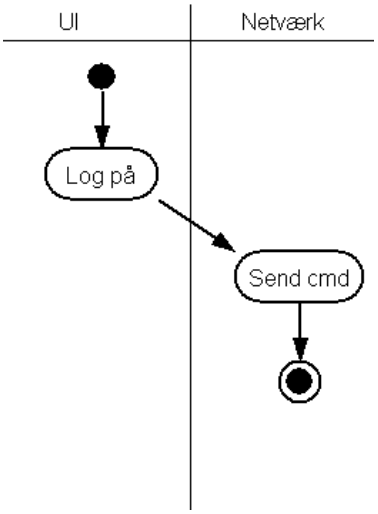
Figur 2-6

Figur 2-6 viser hvordan lyden modtages fra netværket for at blive dekodet og afspillet af playback subsystemet.



Figur 2-7

Figur 2-7 er en generel beskrivelse af hvordan en administrativ kommando modtages over netværket.



Figur 2-8

På Figur 2-8 ses hvordan en kommando sendes til netværket, her illustreret ved et log på.

2.2.1.4 Funktions tabel

Aktivitet	**Funktion**	**Beskrivelse**
Log på	Log på systemet.	Logger på serveren, ved hjællp af netværket, og initialisere kommunikationen
Log af	Log af systemet.	Sender en farvel kommando

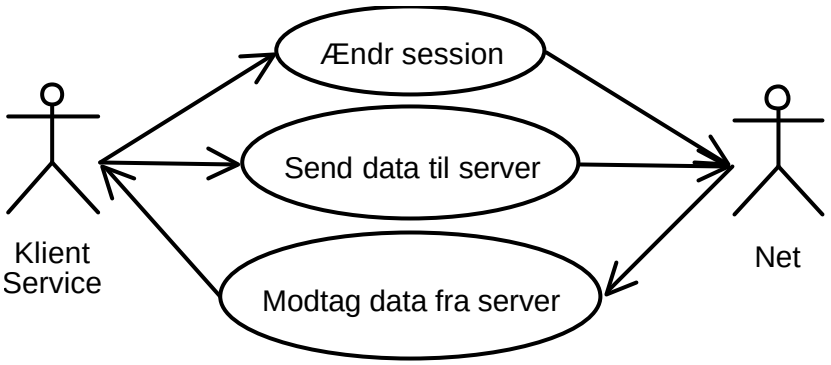
		til serveren, og afbryder dermed kommunikationen
Skift Gruppe	Skift gruppe.	Skifter den kanal på serveren som klienten sender og modtager data til/fra.
Send information	Send information.	Sender information til serveren omkring klienten.
Modtag information	Modtag information.	Modtager information sendt fra serveren.
Optag lyd	Optag lyd.	Påbegynder lyd optagelse fra mikrofon.
	Stop optagelse	Stopper lydoptagelsen.
Afspil lyd	Start afspil	Starter afspilningen af lydsamples i bufferen.
	Slut afspil	Aflutter afspilningen.
Enkod lyd	Enkod	Enkod en eller flere samples med et defineret codec, feks GSM eller Speex.
Dekod lyd	Dekod	Dekod en eller flere samples der er enkodet.
Send lyddata	Send til netværk	Sender netværkspakker der indeholder lyddata til netværket.
Modtag lyddata	Modtag fra netværk	Modtager pakker indeholdende lyddata fra netværket.
Send besked	Send til netværk	Sender pakker til netværket der indeholder administrative data
Modtag besked	Modtag fra netværk	Modtager pakker fra netværket der indeholder administrative data

Tabel 2-1

2.2.2 Klientsession

På klientsiden existere der to aktøre, nemlig netværket og den service der bruger sessionmanageren.

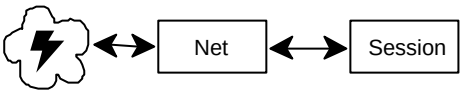
2.2.2.1 Usecase



Figur 2-9

2.2.2.2 Delproblemer

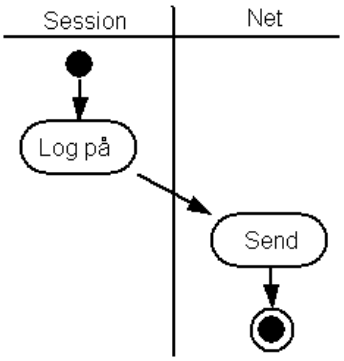
Fra de usecases vist i Figur 2-9, kan der konstateres to delsystemer. Net, til at håndtere protokol og data transmission, og session der vedligholder information omkring brugeren og information modtaget fra serveren. Et diagram ses i Figur 2-10.



Figur 2-10

2.2.2.3 Swimlanes

Swimlanes for klient session er meget simple, da dens primære funktion er at være en slags proxy mellem services og netværks aktørene. Figur 2-11 viser et eksempel på dette, brugeren af klienten ønsker at logge på, og dette sendes til netværket. Når der modtages fra netværks aktøren sendes dataen vidre til den ønskede service.



Figur 2-11

2.2.2.4 Funktionstabel

Aktivitet	Funktion	Beskrivelse
Log på	Log på	Håndterer brugerens ønske om at logge på en server.

Log af	Log af	Giver serveren besked om at brugeren logger af.
Skift gruppe	Skift gruppe	Orientere serveren om at bruger nu tilhøre en anden gruppe.
Setup	Setup	Servicen sender information om sig selv og sin opsætning til serveren.
Send information	Send information	Bruges af servicen til at sende beskeder, f.eks. log på.
Modtag information	Modtag information	Beskeder fra serveren sendes vidre til servicen.
Send data til server	Send data	Sender data, f.eks. lydpakker, til serveren.
Modtag data fra server	Modtag data	Modtager data fra serveren. F.eks. lydpakker.

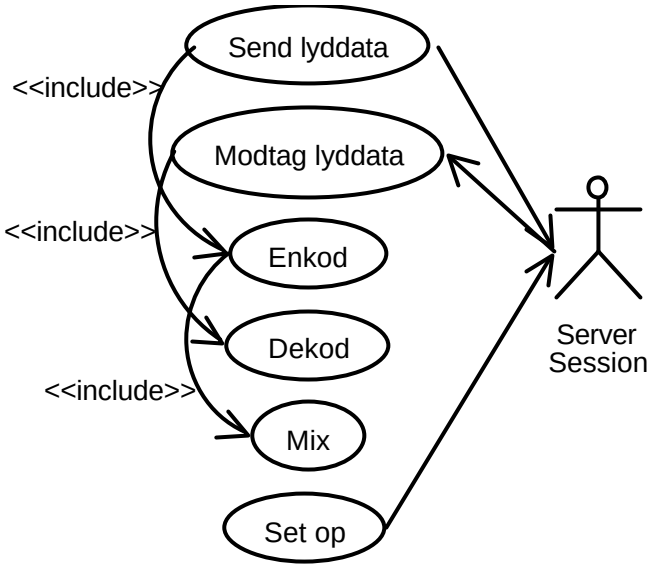
2.3 Server

2.3.1 Audio

2.3.1.1 Usecases

Den eneste aktør i serveren er netværket, der primært bidrager med klientens kommandoer og lyddata. Yderligere skal serveren kunne opfylde funktioner som at enkode og dekode, samt mixe lyd.

Usecasediagrammet kan ses i Figur 2-12.

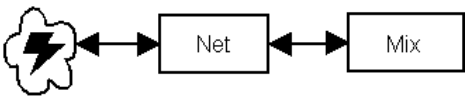


Figur 2-12

2.3.1.2 Delproblemer

Udfra usecasediagrammet i Figur 2-12 konstateres det at systemet har to delproblemer. Net, som er ansvarlig for at afsende og modtage trafik, samt vedligholde information vedrørende klienterne, og Mix, som er ansvarlig for behandling af lyddata, en- og dekodning, samt mixing af signaler.

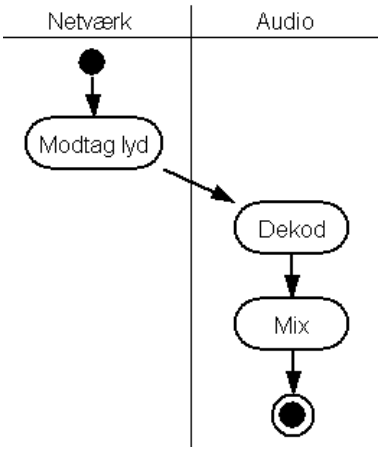
Et diagram over delsystemerne er vist i Figur 2-13.



Figur 2-13

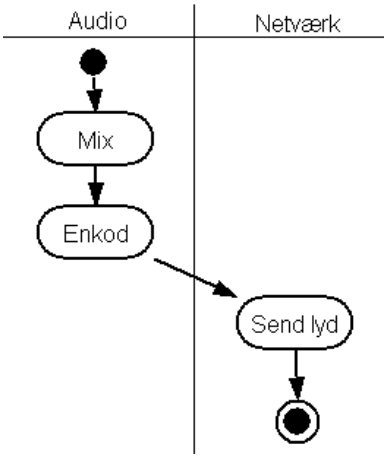
2.3.1.3 Swimlanes

Swimlanes for de forskellige usercases kan ses i de følgende figure. For funktionerne Log på, Log af, Skift kanal og Send/modtag information er disse undladt, da de er enkeltstående funktioner.



Figur 2-14

Figur 2-14 viser hvordan netværket modtager lyd, og sender dette videre til audio systemet som dekoder og derefter mixer signalet med andre samtigdige signaler.



Figur 2-15

Figur 2-15 viser her hvordan mixeren sender det mixede signal til enkoderen, der derefter udsender det komprimerede lydsignal på netværket, og dermed til serveren.

2.3.1.4 Funktionstabel

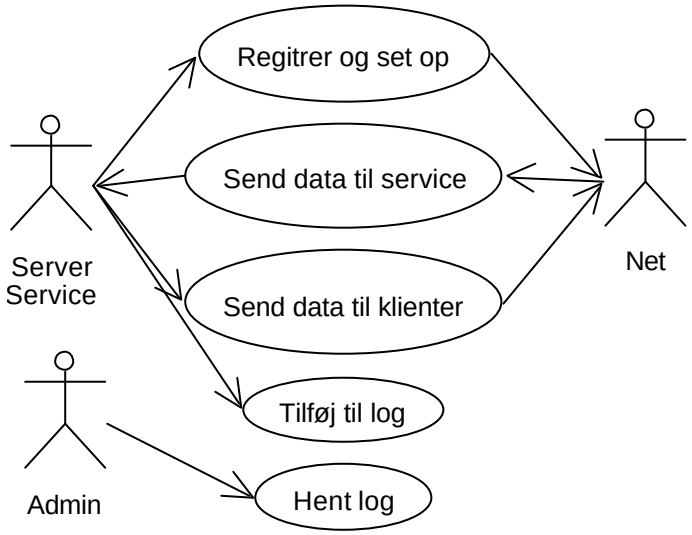
Her beskrives funktionaliteten på serveren. Flere af funktionerne er spejlinger af klientens behov, og usecases på klientsiden.

Aktivitet	Funktion	Beskrivelse
Log af	Log af	Varetager klienter der logger af.
Log på	Log på	Varetager klienter der logger på.
Skift gruppe	Skift gruppe	Regitrere gruppeskift hos klienter
Send information	Send information	Sender information til

		klienter
Modtag information	Modtag information	Modtager information fra klienter
Send lyd	Send lyd	Sender lyd til klienter, i enkodet form.
Modtag lyd	Modtag lyd	Modtager enkodet lyd fra en klient.
Enkod	Enkod lyd	Enkoder lyd til transmission
Dekod	dekod lyd	Dekoder lyd.
Mix	Indsæt lyd Mix lyd	Mixeren kan tage flere signaler og mixe dem sammen til en enkelt lydkilde.

2.3.2 Serversession

2.3.2.1 Usecase



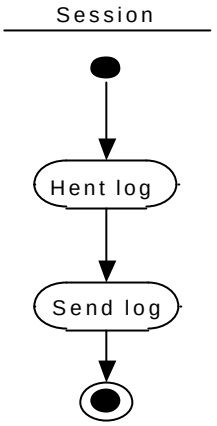
Figur 2-16

Usecases for serversiden af session delen kan ses i Figur 2-16. De tre aktøre i systemet er servicen, som er det undersystem der benytter session klassen, klienten, som er data fra session klienten modtaget via netværket. Admin er en udvidelse af klient der kan hente en logfil online, og dermed udføre en slags fjern-diagnostisering eller overvågning.

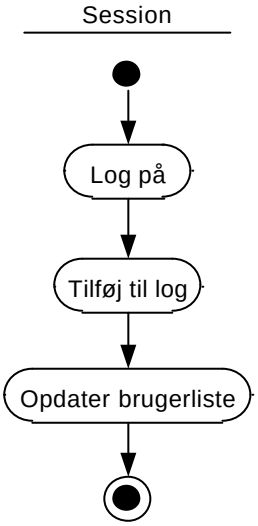
2.3.2.2 Delsystemer

Session manager for serveren er meget lig det for klienten og kan deles op i same delsystemer som Figur 2-10.

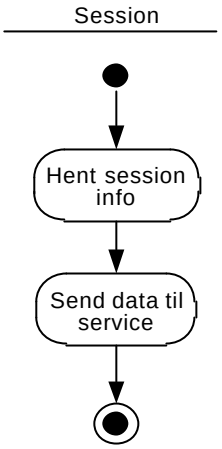
2.3.2.3 Swimlanes



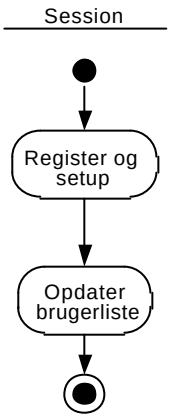
Figur 2-17



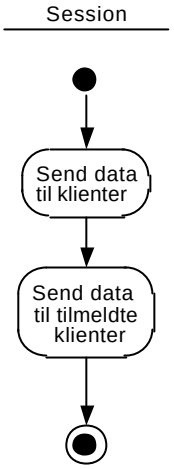
Figur 2-18



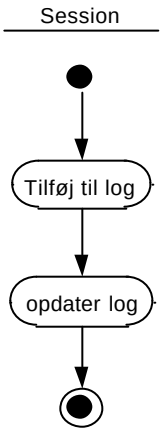
Figur 2-19



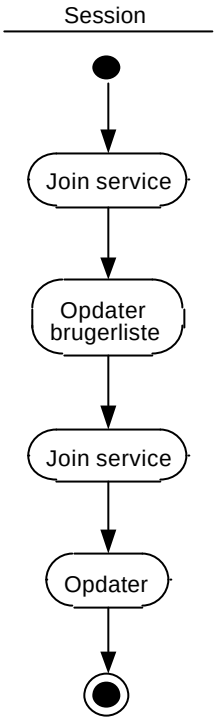
Figur 2-20



Figur 2-21



Figur 2-22



Figur 2-23

2.3.2.4 Funktionstabel

Aktivitet	Funktion	Beskrivelse
Log på	Log på	Varetager klienter der logger på.
Log af	Log af	Varetager klienter der logger af.
Skift gruppe	Skift gruppe	Regitrere gruppeskift hos klienter.
Service kontrol	Join service	Varetager en given klients tilmelding af en service.
	Leave service	Varetager en given klients afmelding af en service.
	Registrer og opsæt	Varetager at en serverservice registrere sig i session. Varetager samtidig at service specifikke ønsker

		opsætning.
Data kommunikation	Send data til service Send data til klienter	Sender data til en given service. Sender data til et antal klienter.
Administration	Tilføj til log Hent log	Tilføjer en besked i loggen Henter hele loggen.

3 Problemformulering

Der ønskes udviklet et lyd konference system med lave latenstider med samtidigt høj lydkvalitet. Systemet skal laves ved hjælp af en klient/server arkitektur og implementeres i java. På grund af firewall kompatiblitet skal der benyttes mindst mulige netværks porte. Dette er rellevant da flere *services* skal kunne tilkobles systemet.

3.1 Delmål

Sessiondelen Da flere simultane systemer skal kunne køre samtidigt og benytte den sammen netværksport, skal dette tages højde for i session. Der bør som følge af dette ikke tilføjes væsentligt latens. Desuden skal det være muligt at oprette flere forskellige grupper hvor det kun er gruppens medlemmer der kan tale sammen.

Lyddelen Lyden der kommer igennem systemet skal være sammenhængende og let forståelig. Der ønskes desuden brugen af et codec implementeret for at aflaste serveren.

4 Design og implementation

Designet blev udført med værktøj[8] der tillod at springe mellem design og implementation frit, og udviklingsprocessen blev derfor meget iterativ da design og implementering smeltede sammen til en samlet fase. Dette blev gjort for at vedligeholde en meget høj konsistens mellem kildekoden og design arkitekturen.

4.1 Deployment:

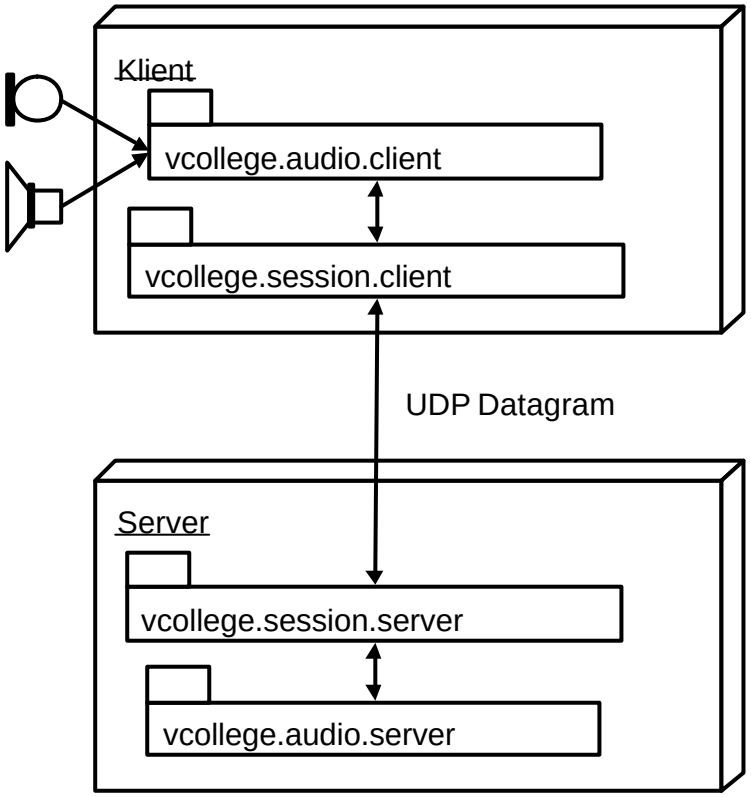


Figure 1 viser deployment af systemet, kommunikationen via. UDP mellem klient og server.

Systemet indeholder en klient og en server der kommunikere via en UDP forbindelse. Der benyttes kun UDP og kun en enkelt port til systemet, som derfor vil være forholdsvist firewall nemt at benytte med forskellige firewalls og routere. Ydermere bliver klient og server delt op i to lag der henholdvist står for netværks interfacet og audio processerings delen.

4.2 Klassebeskrivlese

I følgende afsnit vil alle klasser i systemet blive beskrevet pakke for pakke. For en fyldestgørende beskrivlese af hver enkelt metode henvises til JavaDoc.

[8] Værktøjet der blev benyttet er Together

4.2.1 Klassebeskrivelse - vcollege.audio.server.*;

I det følgende vil der beskrives et klassediagram for klasserne i denne pakke og hver enkelt klasse vil blive beskrevet.

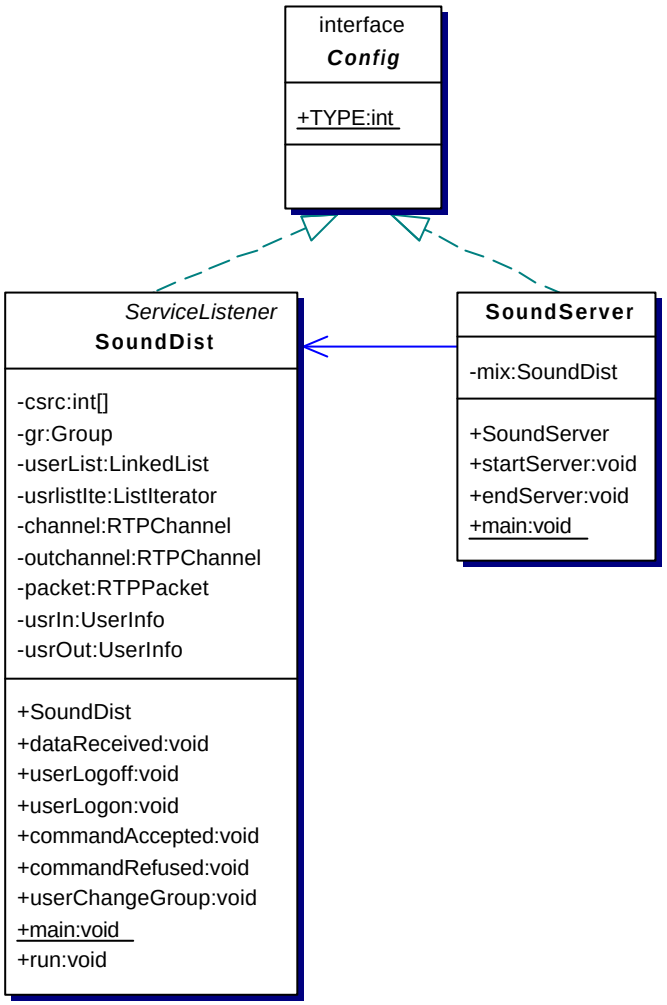


Figure 2: Klassediagram af vcollege.audio.server.*;

4.2.2 Interfacet Config

Config er et trivielt configurations interface, i dette tilfælde tager det sig af id værdien soundserveren skal have i sessionserveren, denne værdi benyttes også som type felt i RTP headeren.

4.2.3 Klassen SoundServer

SoundServer håndtere opstart af soundserver modulet. Den opretter en *SoundDist*, og hvis session.server.Session ikke er startet op startes denne også, desuden står den for at melde

sig til session.server.Session. denne klasse implementere *Config* da denne indeholder værdien ”TYPE” der benyttes i tilmeldings fasen

4.2.4 Klassen SoundDist

SoundDist står for at distribuere lyd ud til de rellevante brugere i de rellevante grupper. Oprindeligt stod denne også for at mixe lyden sammen, men dette resulterede i kraftigt forringet lydkvalitet hvorfor denne funktionalitet blev droppet. Denne klasse implementere Interfacet *ServiceListener* hvilket gør at metoden *dataReceived()* bliver kaldt hver gang en pakke af typen ”TYPE” bliver modtaget i session. Denne information bliver benyttet til at distribuere ud fra så en egentlig trådning af klassen ikke er nødvændigt. Dataene der sendes og modtages går igennem brugernes *RTPChannel.*

En udspecificeret beskrivelse af metodernes funktionalitet kan findes i javadoc for klasserne.

4.3 Klassebeskrivelse - vcollege.audio.common.;*

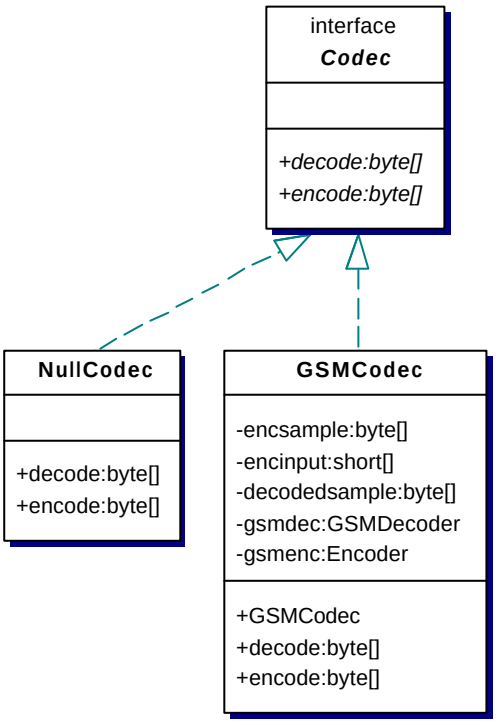


Figure 3 vcollege.audio.common.*;

Denne pakke indeholder forskellige codec implementationer der kan benyttes.

4.3.1 NullCodec

Er et dummy codec der returnere inputtet

4.3.2 GSMCodec

Er en implementation af Tritonus GSM.

4.4 Klassebeskrivelse - vcollege.session.server.;*

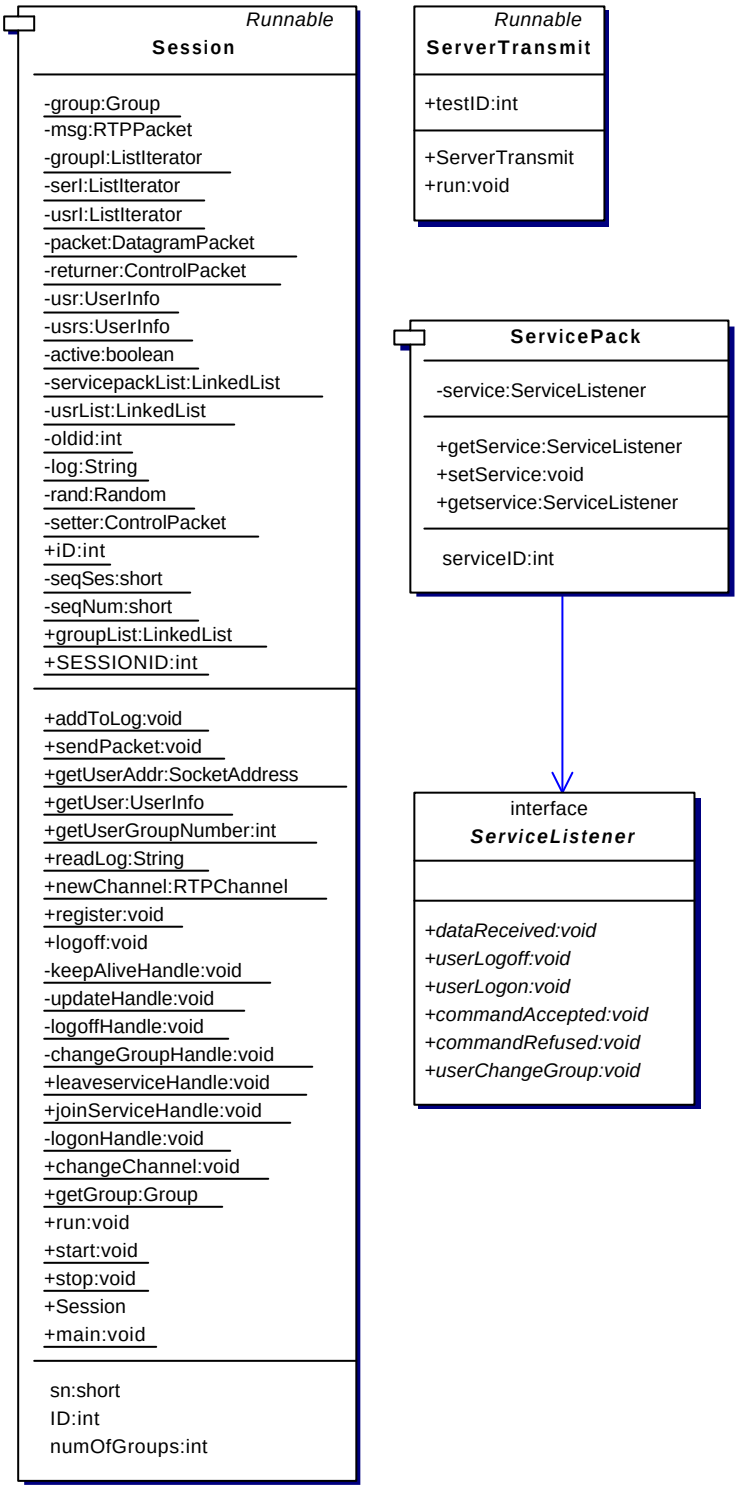


Figure 4 : Klassediagram over vcollege.session.server.*;

4.4.1 Interfacet ServiceListener

ServiceListener **e**r et interface der skal implementeres af services denne tilbyder en række metoder der kan være brugbare for servicene.

4.4.2 Klassen ServerTransmit

ServerTransmit er en tråd der startes af session og tager sig af at tømme bufferne fra *rtpchannel* og sende dem ud på netværket.

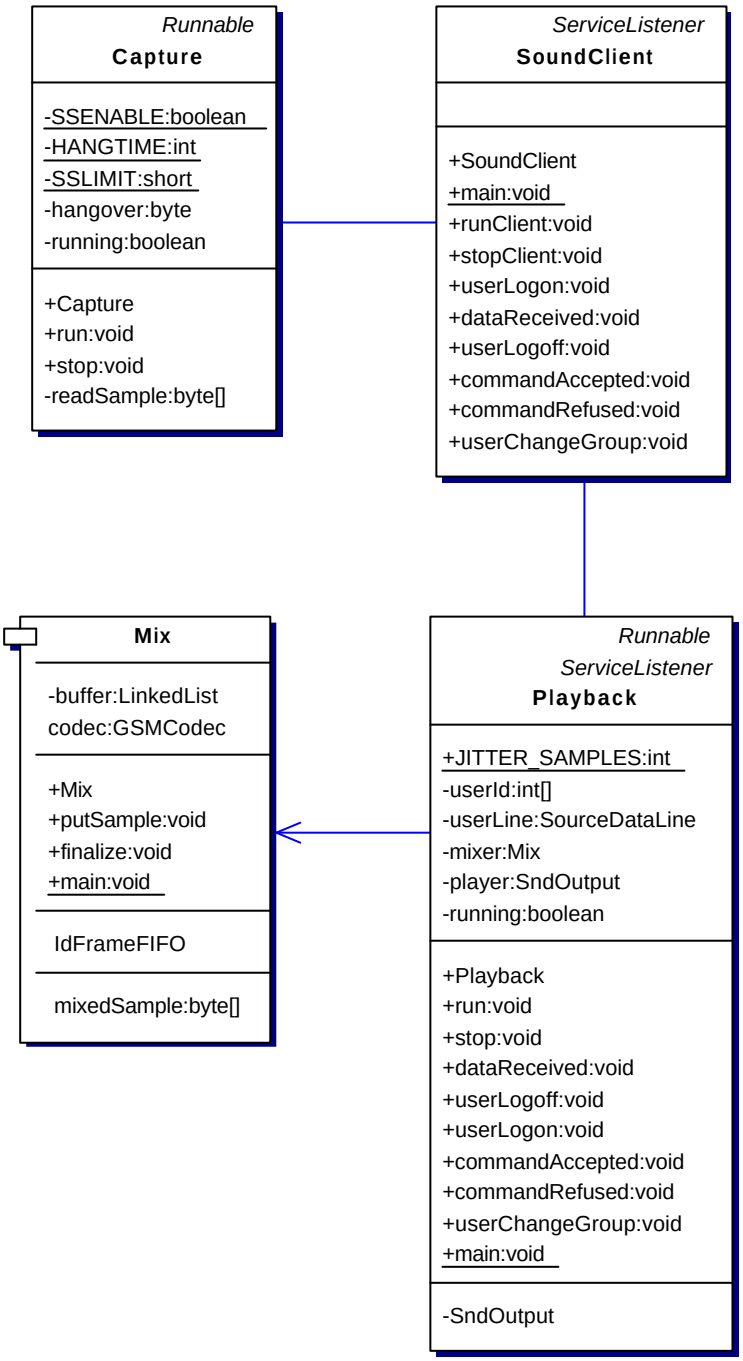
4.4.3 Klassen ServicePack

ServicePack indeholder en liste over services og stiller metoder tilrådighed der gør man kan finde en specifik service.

4.4.4 Klassen Service

Service **t**ager sig af det admininstrative arbejde i session laget, sørger for at processere inkomne pakker korrekt.

4.5 Klassebeskrivelse - vcollege.audio.client.;*



Figur 24: Klassediagram over pakken vcollege.audio.client.*;

4.5.1 Klassen Capture

Klassen Capture har til formål at stå for optagelse af lyd. Den benytter sig af javax.sound.sampled api pakken til optagelse. Metoden *readSample* skal kaldes regelmæssigt for at tømme lydbufferen.

4.5.2 Klassen Playback

Playback står for afspilning af lyd, og har indbygget en forsinkelse til jitter kompensering. Playback implementere *ServiceListener* interfacet for at kunne få *dataReceived* events. Disse bruges til forsinkelsen, som vetner på et antal af disse før den trigger afspilning.

4.5.3 Klassen Mix

Mix indeholder en software mixer, der bruges ved at kalde *putSample* med samples der skal mixes, og derefter kalde *getMixedSample* som returnere den mixede sample. Mixeren fungere efter vægtet mixing af 2 kanaler, og bruge dette til at kollapse alle samples i mixer stakken.

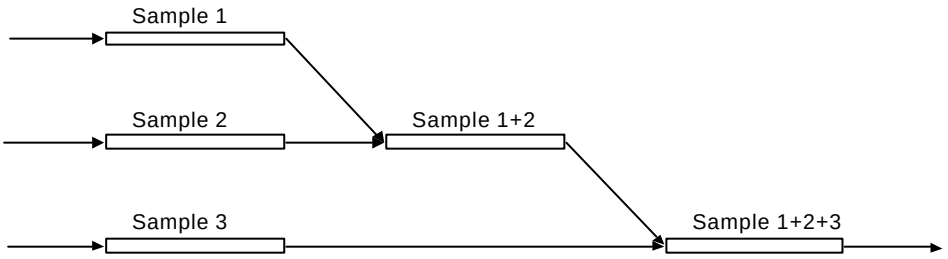


Figure 5 viser hvorledes kanaler bliver mixet parvist til det endelige mixede signal

Kanalerne mixes efterfølgende formel:

$$M(a,b) = a + b - \frac{a \cdot b}{2^{16}}$$

a og *b* er en enkelt sample i hver kanal der skal mixes, og forventes at være i 16 bits opløsning. Mixeren tager højde for at vægte den sample med mest amplitude mest, så et evt. signal uden amplitude ikke dæmper andre signaler som det mixes med.
At kollapse kanalerne parvist giver en lille fejl i forhold til ægte mixning, men den optræder først ved >2 kanaler og forstyrrelsen ved at menneskelig opfattelse af så mange signaler er væselenlig større. Desuden kræver denne metode ingen forud anelse om hvor mange signaler der skal mixes.

4.5.4 Klassen SoundClient

SoundClient klassen binder Capture og Playback sammen i en samlet enhed og udgør samtidigt en minimalistisk audio klient til test brug. Klassen kan eksekveres selvstændigt, eller bruges ved at kalde *runClient* og *stopClient* til henholdsvist at starte og stoppe klienten.

4.6 Klassebeskrivelse - vcollege.session.common.;*

Common pakkerne er lavet for at indeholde klasser der benyttes af både klient og server delen derfor kan den forekomme den lidt uoverskuelig.

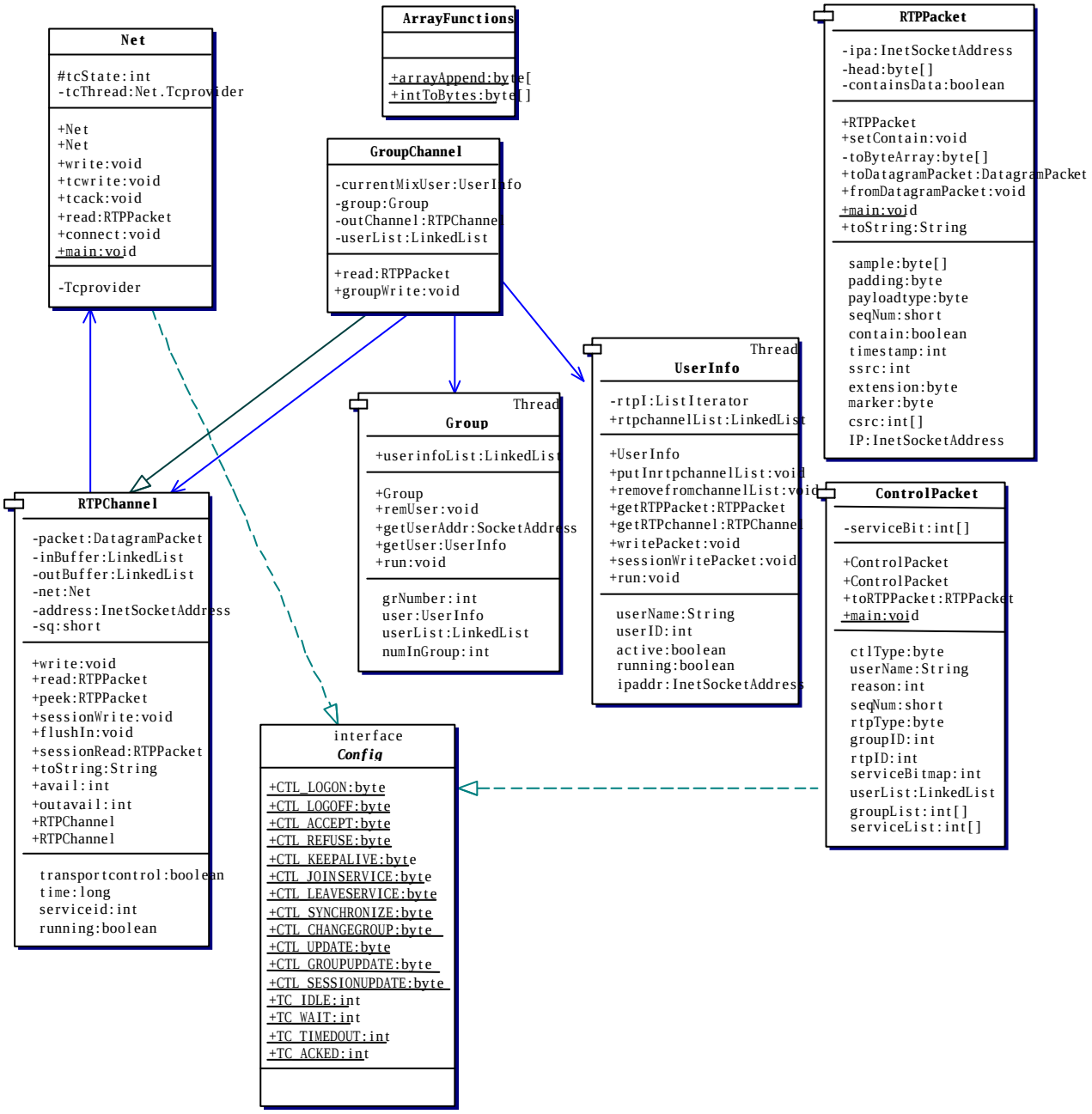


Figure 6 : Klassediagram over vcollege.session.common.*;

4.6.1 Interfacet Config

Er et trivielt configurations interface.

4.6.2 Klassen Net

Er sessions kommunikations del ud af til, denne tager sig af at sende og modtage pakker. Denne klasse implementere interfacet *Config*, hvor det er muligt at ændre forskellige værdier for opsætningen af transmisionsdelen.

4.6.3 Klassen ControlPacket

Er en klasse som session bruger til at konstruere og decifrere kontrolpakker med, altså pakker der ikke skal sendes vidre op til Service laget. Denne pakke implementer interfacet *Config*.

4.6.4 Klassen RTPPacket

Er som *ControlPacket* em konstruktions og decifrerings klasse denne benyttes til pakker der skal sendes vidre til servicelaget.

4.6.5 Klassen RTPChannel

RTPChannel benyttes af servicene til at sende og modtage data til og fra klienter. Derfor indeholder den en buffer med data til og fra brugerobjekter af typen *UserInfo*.

4.6.6 Klassen GroupChannel

Denne klasse har samme funktionalitet som RTPChannel, men på Gruppe niveau , den har metoder til at sende til en hel gruppe af gangen.

4.6.7 Klassen UserInfo

Dette er sessions representation af en klient indeholdende alle rellevante informationer herom. Den er gjort trådet så objekter af denne type kan nedlægge sig selv hvis en brugers forbindelse bliver afbrudt.

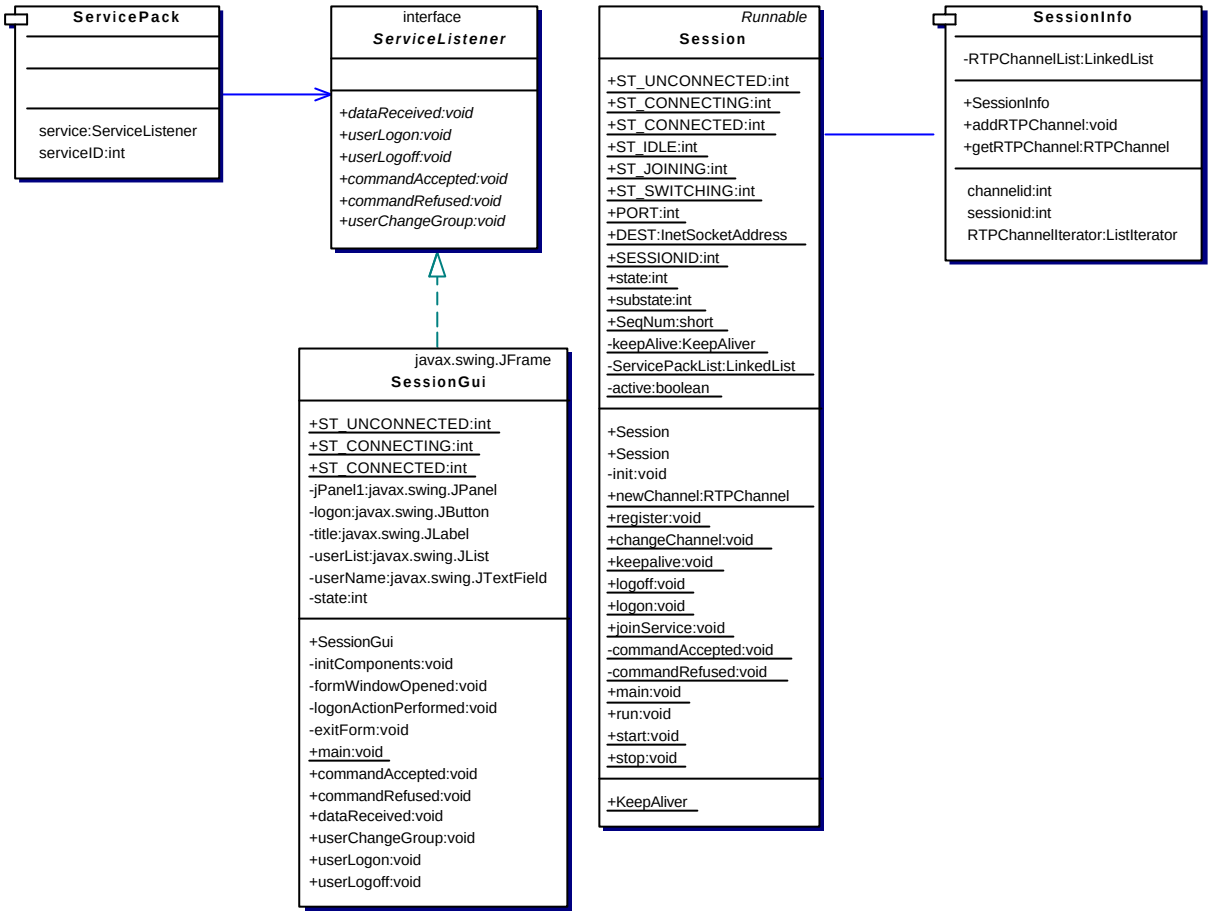
4.6.8 Klassen Group

Denne er sessio ns reprecentation af en gruppe af brugere. Den indeholder en liste af brugere og metoder til at finde og tilgå brugere. Som med *UserInfo* er denne trådet for at kunne nedlægge sig selv hvis der ikke er flere brugere.

4.6.9 Klassen ArrayFunctions

ArrayFunctions er en triviel statisk klasse der indeholder metoder til at manipulere Arrays med.

4.7 Klassebeskrivelse - vcollege.session.client.;*



4.7.1 Klassen Session

Session bruges til at vedligeholde state i systemet. Klassen er statisk , men indeholder nogle dynamiske elementer der bliver oprettet i *init* funktionen. Denne skal derfor altid kaldes før Session tages i brug, og indeholder foranstaltninger der sørger for at kald, udover det første, bliver ignoreret. Det er derfor sikkert at lade hver service kalde denne hver især inden registrering.
Session indeholder en dobbelt state machine:

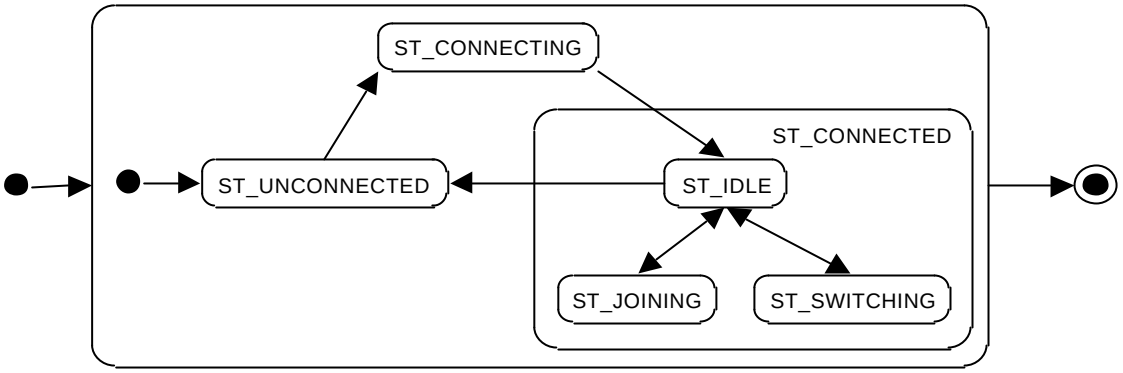


Figure 7 viser tilstandsdiagram for Session der er implementeret som *state* og *substate.*

Ingen kald er blokerende, men state kan på ethvert tidspunkt aflæses i state (ST_UNCONNECTED, ST_CONNECTING, ST_CONNECTED) og substate (ST_IDLE, ST_JOINING, ST_SWITCHING) attributerne, som er public. Disse skal dog betragtes som read-only, da ændringer i disse kan medføre en fejlfungerende Session.

4.7.2 Klassen SessionInfo

SessionInfo bruges til at vedligeholde information vedrørende klienter og grupper i systemet.

4.7.3 Klassen ServicePack

ServicePack indeholder en services' id, samt dens kanal.

4.7.4 Interfacet ServiceListener

ServiceListener er klient udgaven af det tilsvarende interface i vcollege.session.server, og skal implementeres af alle klasser der vil registreres som service i Session.

4.7.5 Klassen SessionGui

SessionGui er en prototype klient der indeholder et gui samt en anvendelse af klientsystemerne. Den kan forbinde til en Session server og sende og modtage lyd til og fra denne

5 Test

Test af systemets korrekthed er foretaget løbende igennem implementationen, flere af klasserne har en test stub, der består af en main metode der gennemføre en korrektheds test for den pågældende klasse.

Yderligere skal der testes for hvor hvad latenstiden i systemet er. Der er valgt at teste dele af systemet for delay og tilsidst en delaytest fra en mikrofon på en klient til højtaler på en anden.
Testene foretages på 100 mbit LAN netværk.

5.1 Delay i JVM

Da vi ingen indflydelse har på hvordan Java Virtual Machine (JVM) påvirker lydsignaler, fandt vi det nødvendigt at redegøre hvilke effekter og hvor meget delay der tilføjes af den vej.

Testen blev udført på én pc, Hvortil der blev tilsluttet en signal generator på lyd indgangen samt et oscilloskop på udgangen. Outputsignalet fra signal generatoren blev også sendt til oscilloskopet til sammenligning.

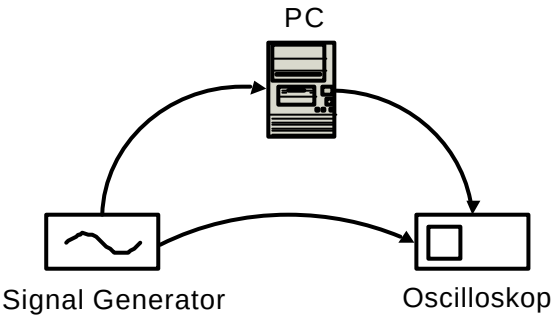


Figure 8 illustrere test opstillingen

Der blev programmeret et simpelt test program på pc'en der blot tager lyd fra input og afspiller det på output. En test forgik ved at signalgeneratoren var slukket, blev tændt og tidsrummet fra signalet fra generatoren blev vist i oscilloskopet til signalet der havde været gennem pc'en, og altså JVM'en, blev vist, blev målt.

Der blev udført 3 test:

Test	Tid (ms)
1	310
2	340
3	320

Table 1 viser resultater fra test af delay i JVM.

Dette passer med den tid lyden ligger i JVM'ens buffere, og yderligere test med forskellige størrelser af disse underbyggede denne teori.
Yderligere blev der observeret en lille frekvens forvrængning i signalet. Brug af anden software[9] viste at dette tilsyneladende ikke var et problem ved java, men må tilskrives hardware eller operativ systemets drivere.

5.2 Delay over Server

Delay over Serveren blev beregnet fra lige efter en pakke blev modtaget i Session klassen til den blev vidredistribueret af Net klassen denne testmetode gav resultater der generelt var mindre end det målbare i milli sekunder dog med enkelte outliers på 16 milli sekunder ca en gang hver 25ende pakke. Yderligere tests blev ikke foretaget da dette var nok til at konkludere at serveren ikke var en væsentlig faktor i det samlede delay.

[9] Der blev benyttet en loop test i TeamSpeak programmet.

5.3 Samlet delay fra mikrofon til højtaler

Denne test udføres for at finde det minimale delay fra en person begynder at tale til det modtages på den modsatte side. Derfor er alt jitter kompensering slået fra. Testen foretages ved at starte SessionGUI klassen op på to klienter : SpeakingClient, ListeningClient og starte en SoundServer op på en Server : SoundServer. Signalgeneratoren er koblet direkte til input på SpeakingClients lydkort og til en af inputkanalerne på Oscilloscopet. Oscilloscopets andet input er direkte fra output på ListeningClients lydkort. Dette er illustreret i Nedenstående figur.

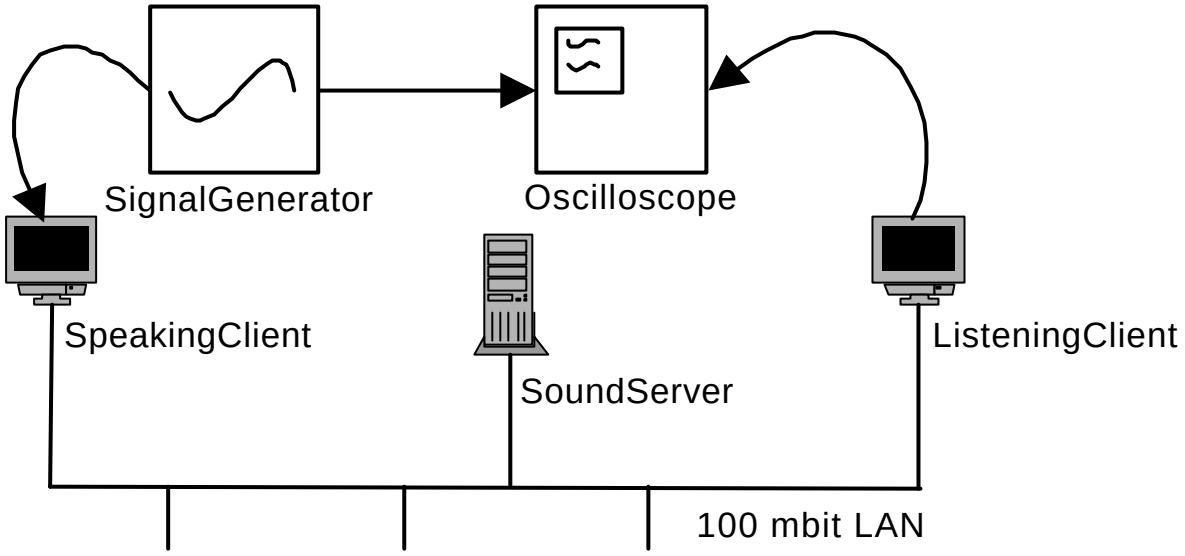


Figure 9: Testopsætning under samlet delaytest

Stikket der går fra signalgeneratoren til SpeakingClient er afbrudt fra start og tilkobles hvorefter det ses hvor hurtigt signalet er ude af lydkortet på ListeningClient, ved at sammenligne de to signaler på osciloscopet. Testen blev foretaget 3 gange.

Testnummer :	**Delay i ms :**
1.	244
2.	240
3.	210

Table 2: resultater af test

Resultatet af testen kan ses af ovenstående tabel.
Det er desuden konstateret at lyden afspilles en anelse langsommere end den optages. Dette er et fænomen i enten Operativ systemet eller hardware, da samme resultat blev

konstateret med andre lyd konference systemer (Teamspeak), dette system er ikke implementeret i java, hvorfor det kan udelukkes at det skyldes JVM.

6 Målgruppe

Da dette projekt ikke er færdigtudviklet vil det i første omgang henvende sig til udviklere med vidreudvikling for øje.

Strukturen er som før beskrevet stærkt modulær hvilket gør det nemt at tilføje og fjerne funktionalitet.

Et eksempel kunne være at tilføje video til systemet. Her skal udvikleren lave en service selv hvorefter han kan benytte *session* laget som et *middelware* så han slipper for at skulle tænke på netværksdelen.

På grund af modulariteten vil en gui *implementation* evt med information om grupper, brugere og services også kunne implementeres som en selvstændig *service.*

7 Howto

7.1 Opstart af lyd system :

Server : Serveren startes ved at starte *vcollege.audio.server.SoundServer.class* derefter er serveren startet op detaljer omkring netværksopsætning kan ændres i *vcollege.session.common.Config.java* hvorefter denne skal rekompileres.

Klient : Klienten startes ved at starte *vcollege.session.SessionGUI.class* hvorefter der trykkes logon. Ønskes serverens ip ændret skal det skrives i *vcollege.session.Session.class* i linien *public static InetSocketAddress DEST = new InetSocketAddress("192.38.48.113",PORT);* hvorefter denne klasse skal rekompileres.

7.2 Konstruere en ny service :

Da session delen af systemet fungere som et middleware kan der forholdsvist nemt tilføjes nye services.

Server: For at lave en ny server service skal man sørge for at Session delen er startet, dette gøres ved metoden *Session.start()* dette kald kan trygt ligges i *constructoren* i servicens opstartsklasse da Session kun opretter en ny hvis der ikke findes en I forvejen. Derefter registrere man sin service ved session ved metodekaldet *Session.register(TYPE, Object)* TYPE er den identifier sessionlaget benytter til at finde forskeld på services og skal være unikt, desuden benyttes den også som type felt I RTP headeren, Objektet er den service der skal registeres, i lyd serveren er dette et objekt af klassen SoundDist.

Klient : Man skal foretage same skridt som ved serveren. Derudover skal man sørge for at logge af og på dette gøres ved metoderne *Session.logon("UserName")* og *Session.logoff(). Session.newChannel(int sid, boolean tc)* benyttes til at få tildelt en ny RTPChannel i hvorfra man kan sende og modtage RTPPakker. Sid er Servicens ID og tc er hvorvidt man ønsker Sikker dataoverførsel på bekostning af performance.

8 Konklusion

Der er udviklet et system ved hjælp af en klient/server arkitektur. Systemet har vist sig at kombinere udmærket lydkvalitet med lavt delay og god firewall kompatibilitet. Systemet kan dog ikke betragtes som et færdigt produkt da flere dele af systemet endnu ikke er færdigimplementeret.

I session delen er der implementeret mulighed for flere grupper og skift mellem disse, desuden er det muligt at starte og stoppe services på klienten og serveren uden disse skal genstartes.

I lyddelen er der opnået let forståelig lyd gennem systemet ved samtidigt brug af GSM codec, desuden er der opnået en start latens så lav som 210ms. Yderligere er det fundet ud af at der er et buildup af latens et sted i operativ systemet eller i hardwaren da lyden afspilles en anelse langsommere end den optages, hvorfor silence suppress er ønskeligt.

8.1 Fremtidigt arbejde

Der bør fremover arbejdes på en bedre brugergrænseflade, og dermed udnytte de muligheder protokollen og Session giver for kontrol med brugere og fordeling af lyd. Session i sig selv mangler stadigt at blive helt stabiliseret, specielt I dens mulighed for at sende pålidelige pakker, hvor der stadigt er nogle situationer der kan skabe problemer.

9 Kilder

[RFC2543] - SIP: Session initiation protocol
http://www.ietf.org/rfc/rfc2543.txt

[RFC1889] - RTP: A transport protocol for real-time applications
http://www.ietf.org/rfc/rfc1889.txt

[RFC768] – UDP: User datagram protocol
http://www.faqs.org/rfcs/rfc793.html

[RFC791] – IP: Internet protocol
http://www.faqs.org/rfcs/rfc791.html

[RFC793] – TCP: Transmission control protocol
http://www.faqs.org/rfcs/rfc793.html

[RFC1945] HTTP: Hypertext Transfer Protocol
http://www.faqs.org/rfcs/rfc1945.html

[TSHP] - Team Speak homepage.
http://www.teamspeak.org/

[PICO] – Pico Phone homepage
http://web.tiscali.it/vitez/picophone.html

[SPKF] – Speak Freely
http://www.speakfreely.org/

[ECNF] – e-Conf homepage
http://econf.sourceforge.net/

[COCC] – Coccinella homepage
http://hem.fyristorg.com/matben/

[NETA] – Network Assistant homepage
http://www.gracebyte.com/nassi/

[MICA] – The MICA Graphics Framework homepage
http://www.swfm.com/mica.htm

[VOIP] - Techniques used in Voice over IP Systems
http://www.s3.kth.se/radio/COURSES/S3_SEMINAR_2E1380_2002/examples/S3_Andersson_Tomas_EN.pdf

[IPMS] - Protocols for Media Streaming Over IP
http://engr.oregonstate.edu/~meeuwsem/ECE499/ECE%20499%20Final%20Report.htm

[TRIT] – Tritonus homepage
http://www.tritonus.org/

[SPX] – Speex homepage
http://www.speex.org/

[DJSE] – Developing Java Servlets. James Goodwill ISBN: 0-672-31600-5

[TYJS] – Teach Yourself JavaServer Pages. Jose Annunziato et al. ISBN: 0-672-32023-1

[JSSE] – JavaServer Pages, Second Edition. Hans Bergsten ISBN: 0-596-00317-X

[PJSP] – Proffesional Java Server Programming. Danny Ayers et al. ISBN: 1-861002-77-7

[XML] – Extensible Markup Language (XML)
http://www.w3.org/XML/