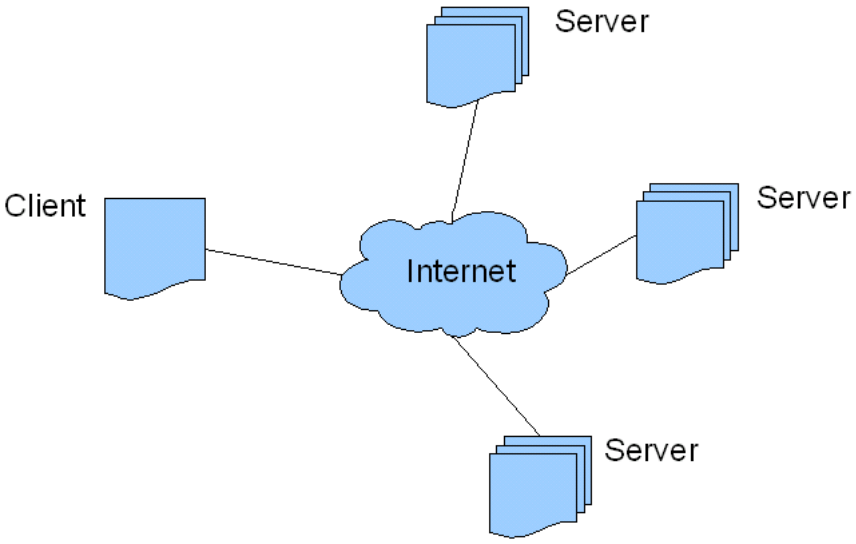
Datanet
Obligatorisk opgave 4:
Redundant datalagring over netværk

René Hardi Hansen
Michael Falcke Nilou
Anders Bjerg Pedersen
Hold 1

18. oktober 2007



1 Indledning

Denne opgave er stillet som en godkendelsesopgave i kurset Datanet i efteråret 2007 ved DIKU. For at kunne læse og forstå indholdet af denne opgavebesvarelse, forudsætter vi at man har kendskab til programmering og netværk svarende til 1. år på datalogi samt en forståelse af netværk svarende til kurset Datanet. Opgaven går ud på, at implementere et system i programmeringssproget C, C++ eller Java, der kan bruges til at lagre filer med redundans. Filerne lægges ind i systemet ved at en klient kontakter en server, der så lagrer filen i et netværk af servere på følgende måde: Først deles filen i 2 dele. Så beregnes en bitvis Xor-sum på parret. Disse datablokke distribueres så blandt et netværk af servere, som, hver især, efterfølgende skal kunne forespørges, finde og samle en fil som returneres til klienten, når denne anmoder om at modtage en fil som den tidligere har modtaget en kvittering på at den har lagt op. Systemet af servere skal selv holde rede på hvor de manglende dele af filer befinder sig. Systemet skal være robust overfor tab af forbindelse til en vilkårlig server der har elementer af en given fil. Filen skal alligevel kunne gendannes og leveres til klienten. Opgaven er en netværksopgave, og fokus er derfor specielt på netværksdelen af opgaven. Det er oplyst, at klient-server kommunikationen skal implementeres via via TCP-forbindelse, og at server-server kommunikation skal foregå via UDP.

2 Analyse

I dette afsnit af rapporten beskriver vi væsentlige beslutninger vi har truffet, samt udfordringer i opgaven, som vi har fundet det væsentligt at dedikere særlig opmærksomhed.

2.1 Valg af programmeringssprog

Vi har valgt at implementere programmet i Java. Java er et programmeringssprog som vi er godt kendt med fra vores studium. I lærebogen: Computer Networking, A top-Down Approach, Kurose & Ross, kan der i kapitel 2.7 og 2.8 findes konkrete eksempler på, hvordan en socketprogrammeringsopgave kan løses i praksis.

2.2 Protokollerne TCP og UDP

Clint-server kommunikationen foregår over TCP protokollen. Med TCP skabes der efter en handshake, en forbindelse mellem en client-proces og en server-proces. Processerne udveksler data indtil clienten har afleveret filen og modtaget en kvittering i form af en ticket fra serveren. Herefter lukkes forbindelsen. Server-server kommunikationen foregår via UDP. UDP er en forbindelsesløs protokol der blot skubber pakker ud til modtageren. Vi er derfor nødt til at sikre os i implementationen af programmet, at der etableres forbindelse mellem servere. Idet vores opgave udelukkende består i at aflevere en datafil fra server til server ved hjælp af UDP, har vi valgt ikke at implementere særlig sikkerhed for at de data vi sender, faktisk når korrekt frem til den modtagende server. Hvis man skal sikre at data når frem og er korrekt, så vil det være nødvendigt at implementere et TCP-lignende system over UDP, med alt hvad det vil indebære af nummererede pakker, handshakes , cheksummer og retransmissioner. Vi sender filen til en server der svarer

på en broadcast, først sender vi information om hvor stor en fil vi sender og dernæst sender vi hele filen. På modtagersiden modtages filen og når den indikerede filstørrelse er modtaget sendes ok tilbage til afsenderen.

2.3 Tab af forbindelse: client-server, TCP

2.3.1 Upload

I opgaven er det foreslået, at vi håndterer servere via fast indkodede IP-adresser. Denne løsning har givet os nogle problemer med at finde ud af, hvilke servere der er tilgængelige på et givent tidspunkt. Vi har derfor besluttet at løse dette ved at lade de tilgængelige servere svare tilbage på en broadcast "request file upload". Herved opnår vi at clienten umiddelbart kan læse i svaret, hvilke servere der er tilgængelige, samt i hvilken rækkefølge de svarer. Herefter kan clienten oprette en TCP-forbindelse til den første ledige server, og påbegynde sin upload. Hvis TCP-forbindelsen tabes kan det håndteres af client-programmet, der kan fange en "Socket Exception". Brugeren kan da blot prøve at forbinde igen, hvorefter programmet kan vælge at forbinde til den server der svarer først på broadcast'en. Vi vil ikke implementere denne undtagelse i vores version af programmet, men det kan umiddelbart gøres.

2.3.2 Download

For download kan vi broadcaste: "request file download" en UDP-pakke med nummeret på den datafil der ønskes hentet fra en server. De servere der har den ønskede datafil kan så svare tilbage på denne broadcast. Dernæst oprettes (som under 2.2.1 Upload) en TCP-forbindelse, til den første server der svarer tilbage. Tilsvarende som ovenfor, hvis TCP-forbindelsen tabes inden filen er modtaget.

2.4 Tab af forbindelse: server-server, UDP

2.4.1 Upload

Vi vil lave den samme type broadcast som nævnt ovenfor (under 2.2.1 Upload), mellem servere. En server der broadcast'er en "request fileupload" vil ligeledes modtage svar fra servere der er klar til at modtage en datafil. Serveren vælger herefter blot de 2 første servere der svarer tilbage, som modtagere af datafilen. Da UDP ikke etablerer en reel forbindelse, er der her tale om at beslutte hvordan vi håndterer en situation, hvor en server der modtager en datafil, ikke giver accept på at datafilen er modtaget. I en sådan situation lader vi en timeout (TTL i TCP-protokollen) samt 3 gentagne forsøg på upload, afgøre hvornår vi lader programmet udsende en ny broadcast og prøve at uploade datafilen til den server der nu svarer først.

2.4.2 Download

Når en server skal finde de manglende datafildele kan den blot som client-programmet broadcaste: "request file download" i en UDP-pakke med nummeret på den datafil der skal samles til clienten. Serveren der samler, henter dernæst data fra de 2 servere der har gemt datafilen. Hvis der ikke er forbindelse til begge servere, hentes indholdet fra

den server der svarer. Datafilen til clienten kan da gendannes på basis af denne datafils indhold og sendes til clienten. Hvis en forbindelse til en server mistes under processen, så vælger serveren blot den tilbageværende serverforbindelse og genskaber datafilen. Hvis serveren mister forbindelsen til begge servere kan vi ikke genskabe den oprindelige datafil, så må vi returnere en fejlmeddelelse til clienten. (Vi må håbe at den pigen der genrne vil finde sin datafil har en god backup ;o))

2.5 Xor-sum

En Xor-sum er 0 hvis de to bit er ens, og 1 hvis de er forskellige. f.eks. er 1001 Xor 1100 = 0101. For at lave en Xor-sum deles den modtagne fil op i 2 dele. Hvis antallet af bytes i den modtagne datafil er lige, kan Xor-summen umiddelbart beregnes bit for bit. Hvis antallet af bytes i datafilen er ulige, kan vi dele filen 1 to, ved at lave en heltalsdivision med 2. Resten indikerer at datafilen er ulige, og vi deler filen i 2 dele, hvor den ene del har størrelsen (datafil/2) og den anden har størrelsen (datafillængde/2)+1. Hvis man opfatter dette som sædvanlige højrestillede binære tal, findes f.eks. fgl. 2 eksempler på Xor-summer : 1)1001 Xor 2)11100 = 0)10101 eller 1)1001 Xor 2)01100 = 0)00101, hvor 0),1) og 2) er servere der indeholder hhv. Xor-sum, 0) og data, 1) og 2). Hvis vi forestiller os, at vi mister et vilkårligt binært tal i en af disse summer, og dernæst skal gendanne det oprindelige, er der mulighed for at den venstre bit i det gendannede mønster er en bit der ikke var i de oprindelige data. Da vi ikke har de oprindelige data, er vi nødt til at bevare sikker information der viser om datafilen har været af lige eller ulige længde. Hvis datafilen oprindeligt har været af ulige længde, og de mistede data fandtes på server 1), da må vi i gendannelsen se bort fra den venstre bit fra server 1 i den gendannede datafil.

2.6 Datastruktur for datafil-system

Datafil-systemet skal, for hver gemt datafil, indeholde halvdelen af den gemte filstørrelse (første eller sidste halvdel) eller den tilsvarende Xor-sum. Da det skal være muligt at finde de 2 andre servere der indeholder oplysninger om datafilen, skal der for hver fil være en referance til de 2 øvrige servere der har oplysninger om en aktuel fil. Der er som nævnt i afsnittet om Xor-sum, behov for at vide om den oprindelige fil er lige eller ulige af størrelse. Vi har brug for at registrere oplysning om hvilken type data den aktuelle server indeholder (første, sidste eller Xor-sum) og endelig skal der være en ticket, som identificerer den oprindelige datafil entydigt overfor både client- og server-programmerne.

2.7 Væsentlige problemer i forbindelse med overførsel, lagring og lokalisering af filer i netværket

Problemerne med lagring af datafilerne i netværket er nu reduceret til blot at være et spørgsmål om at skrive data fra hukommelsen ned serverens disk. Lokaliseringen er blevet løst ved at broadcaste ticket nummeret. Tilbage står så selve overførslen af datafilen. Mellem clienten og serveren overføres filen over en TCP-forbindelse, som sikrer at serveren modtager hele filen uden fejl. Server til server kommunikationen sker derimod med UDP-datagrammer. Med denne måde at overføre datafilen på, er der ingen sikkerhed for at datafilen kommer frem i samme stand som den er modtaget på den første server. Vi kunne implementere en egen version af TCP over UDP, men det opfatter vi som ud

over opgavens formål og derfor vil vi acceptere at der som resultat af denne opgave, er mulighed for at datafilen der returneres til clienten, faktisk ikke nødvendigvis er intakt. Det kan dog sagtens lade sig gøre at udvide programmet med håndtering af fejl således at det kan sikres at den datafil der afleveres til clienten er identisk med den der oprindeligt er modtaget.

2.8 Protokoller til klient-server forbindelser

2.8.1 Upload

Transmit UDP Broadcast "requst file upload".
Receive UDP "ack".
Herefter etableres en TCP-forbindelse og upload påbegyndes.
Receive ticket eller fejl (-1).

2.8.2 Download

Transmit UDP Broadcast "request file download" ticket.
Recieve UDP "ack".
Herefter etableres en TCP-forbindelse og download påbegyndes.
Receive datafil eller fejl (-1).

2.9 Protokoller til server-server forbindelser

2.9.1 Upload

Transmit UDP Broadcast "requst file upload".
Receive UDP Portnummer.
Kommunikation over UDP indtil OK modtages eller til timeout.

2.9.2 Download

Transmit UDP Broadcast "request file download" ticket.
Recieve UDP Portnummer.
Kommunikation over UDP indtil filen er modtaget eller til timeout.

3 Implementation

Programmet er implementeret i Sun Microsystems: Java version 1.5 (http://www.sun.com). Versionen er valgt for at have kompatibilitet med styresystemerne Linux, Microsoft Windows og MacOSX. I det følgende beskrives implementationen, som vi har frembragt den, i to adskilte pakker: en klient-pakke og en server-pakke.

3.1 Klient-pakken

3.1.1 Upload

Primærklassen i klientdelen, indeholdende main funktionen er ClientMain.
ClientMain starter en ny instans af vores grafiske brugerflade, ClientWindow, som også

indeholder det meste af funktionaliteten i klientprogrammet.
I ClientWindow har man til at begynde med en enkelt inputmulighed, navnligt et klik på knappen "Connect". Dette dette skaber et Java.awt.ActionEvent, som fanges af en instans af den indre klasse ActionHandler.
ActionHandler reagere på eventet ved at at udsende et såkaldt multicast datagram. Et multicast datagram, er et UDP datagram med den udvidede funktionalitet, at den sendes til en "multicast gruppe", fremfor blot en specifik ip adresse. Et multicast datagram modtages altså af flere modtagere ved en enkelt afsendelse.
En multicast gruppe angives som en klasse D ip adresse i intervallet 224.0.0.0 til 239.255.255.255.
Se Server afsnitet om modtagelse af filer, for yderligere forklaring på, hvordan Multicast datagrammer modtages.

3.1.2 Download

Download delen er konstrueret i stil med upload delen. I meget stor grad endda. Når en klient ønsker at hente en fil fra en server, multicaster klienten ticketnummeret på filen ud på netværket. De servere som kan finde filen, sender så et ACK retur. Klienten kan herefter oprette en TCP forbindelse til den pågældende server og bagefter anmode om filen.

3.2 Server-pakken

3.2.1 Upload

For at kunne lytte på flere porte samtidigt, er det nødvendigt at benytte en trådet server. Vores primærklasse i serverdelen er passende navngivet ServerMain. ServerMain starter tre tråde, som kører sideløbende med hovedtråden.

- MulticastListenerHandlerThread.
- TCPUploadListenerThread.
- UDPUploadListenerThread.

Navnene er stort set selvforklarende.
Vores MulticastListenerHandlerThread, står for at svare på multicast datagrammer sendt fra klientprogrammet og andre servere. I tilfælde af et datagram fra en klient, sendes blot et simpelt ACK tilbage, hvorefter serveren igen returnere til lyttetilstand på porten.
Hvis det er et multicast datagram fra en server, arbejdes her lidt mere. Der i dette tilfælde tale om, at en ny fildel ønskes sendt fra en anden server. I dette tilfælde, startes en ny tråd, UDPUploadHandlerThread, som lytter på en ny tilfældig fri port. Denne port pakkes i et datagram og sendes retur til den forespørgende server.
Denne tråd står så for at modtage den fildel der bliver tilsendt.

3.2.2 Download

Download delen er konstrueret i stil med upload delen. Til download delen kører serveren nu to nye tråde.
- TCPDownloadListenerThread. - TCPDownloadHandlerThread.

4 Afprøvning

I dette afsnit dokumenteres det, at programmets funktionalitet opfylder de krav som er beskrevet i opgaveteksten. I opgaveteksten er der eksplicit nævnt, en klient skal kunne gemme en datafil, en klient skal kunne hente en gemt datafil og en klient skal kunne hente en datafil selv om forbindelsen mistes til en server. Dette vises ved såkaldte "Use cases". Til brug ved afprøvningen, har vi fremstillet 5 filer:

1. En tom fil, 0 bytes.
2. Fil med 1 byte. (Meget lille fil)
3. Fil med 4 bytes. (Fil med lige antal bytes)
4. Fil med 5 bytes. (Fil med ulige antal bytes)
5. En stor fil.

I de følgende afsnit referer numrene her, til disse inputfiler. På denne måde kan vi dokumentere, at programmet kan håndtere de mindst mulige filer, store filer og både filer af lige og ulige bytelængde. Bytelængden har betydning som nævnt i analysen, når Xorsummen skal beregnes, og resultatet genbruges til at gendanne den gemte fil.

4.1 Klient-server upload

Forventet output

1. En ticket med et unikt filnummer.
2. En ticket med et unikt filnummer.
3. En ticket med et unikt filnummer.
4. En ticket med et unikt filnummer.
5. En ticket med et unikt filnummer.

Faktisk output

1. En Java exception.
2. En Java exception.
3. En ticket med et unikt filnummer.
4. En Java exception.

5. En ticket med et unikt filnummer eller en Java exception, afhængig af om filen er lige eller ulige længde.

4.2 Klient-server download

Forventet input

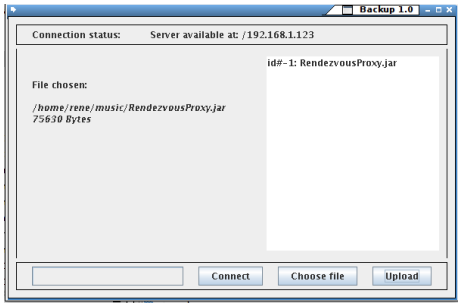
1. En ticket med et unikt filnummer til en fil med 0 bytes.
2. En ticket med et unikt filnummer til en fil med 1 byte.
3. En ticket med et unikt filnummer til en fil med 2 bytes.
4. En ticket med et unikt filnummer til en fil med 5 bytes.
5. En ticket med et unikt filnummer til en stor fil.

Forventet output

1. En tom fil, 0 bytes.
2. Fil med 1 byte. (Meget lille fil)
3. Fil med 4 bytes. (Fil med lige antal bytes)
4. Fil med 5 bytes. (Fil med ulige antal bytes)
5. En stor fil.

Faktisk output

Mangler.



Figur 1: Ticket -1

4.3 Der er kun èn server til rådighed

Forventet output, upload

Serveren skal nå en timeout og sende et ticketnummer på -1 tilbage til klienten.

Faktisk output

Forventning opfyldt. Se figur 1

4.4 Forbindelse til én vilkårlig server mistes

Forventet output, upload

Serveren skal opnå et timeout på den ene server, men uploade til den anden. Filen er lagret og et korrekt ticketnummer returneres til klienten.

Faktisk output

Ved upload af en fil med et lige antal bytes, får vi en exception på primær serveren: Exception in thread "UploadHandlerThread"java.lang.ArrayIndexOutOfBoundsException: 75639 at server.threads.TCPUploadHandlerThread.getBlock(TCPUploadHandlerThread.java:321) at server.threads.TCPUploadHandlerThread.UDPSendFile(TCPUploadHandlerThread.java:263) at server.threads.TCPUploadHandlerThread.run(TCPUploadHandlerThread.java:92)
Vores getBlock funktion indekserer altså uden for arrayet af filen under UDP forsendelsen til sekundærserver.
Ved upload af en fil med et ulige antal bytes, får vi følgende exception på primær serveren: Exception in thread "UploadHandlerThread"java.lang.ArrayIndexOutOfBoundsException: 71213 at server.threads.TCPUploadHandlerThread.splitFile(TCPUploadHandlerThread.java:157) at server.threads.TCPUploadHandlerThread.run(TCPUploadHandlerThread.java:66)
Vores splitFile funktion indekserer altså uden for arrayet af filen, når denne skal deles op.

4.5 Forbindelse til én vilkårlig server mistes

Forventet output, download

1. En tom fil, 0 bytes.
2. Fil med 1 byte. (Meget lille fil)
3. Fil med 4 bytes. (Fil med lige antal bytes)
4. Fil med 5 bytes. (Fil med ulige antal bytes)
5. En stor fil.

Faktisk output

Mangler.

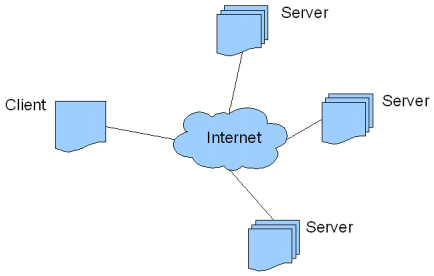
4.6 Konklusion på afprøvningen

Status på afprøvningen er nu, at programmet virker ved upload til server, forudsat filen indeholder et lige antal bytes forskelligt fra 0. Der er altså i øjeblikket en fejl, formodentlig i vores indexering ved fildeling og Xorsum beregningen. I første instans når vi så langt

som til at Xorsummen bliver indsat i den primære servers datastruktur. Programmet fejler pga. indekseringsproblemer for fil arrayet. I anden instans når vi kun til at filen skal splittes op. Her er også indekseringsproblemer, og det kræver mere detaljeret fejlsøgning at komme videre.

5 Brugervejledning

I denne version af programpakken, forudsættes det, at klient- og Server-programmer frit kan kommunikere på netværket via TCP-pakker og UDP-datagrammer. Vær opmærksom på at en evt. firewall tillader kommunikation på de porte programmerne har behov for at bruge.



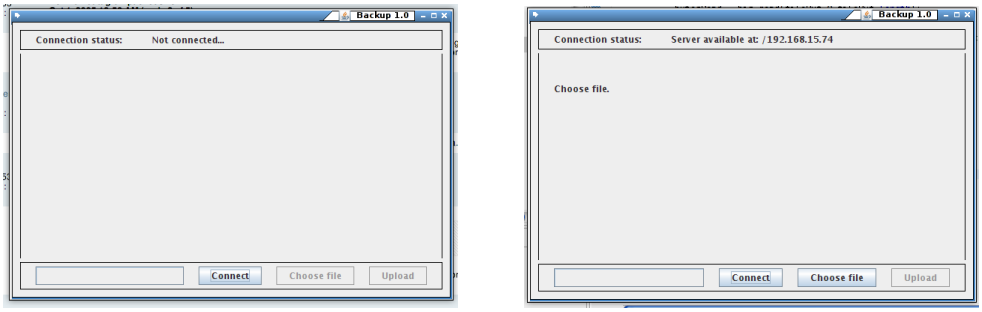
Denne pakke består af to selvstændige programmer: en klient og en server. Serveren og klient-programmet er javaprogrammer der kan bringes til at køre på en række maskiner i et internet-baseret netværk, der benytter TCP/IP protokollen. Programpakken understøtter en funktionalitet til lagring af filer med redundans. Konkret, bruges klientprogrammet til at lagre en fil på en server, der deler filen op i 3 dele. 1 del gemmes på den aktuelle server og 2 andre dele gemmes på andre servere i netværket, hvorpå serverprogrammet er aktivt. Når klient-programmet har afleveret datafilen til en server leverer serveren en kvittering der viser at filen er modtaget af serveren, og at den er lagret med redundans. Klient-programmet kan herefter kontakte en vilkårlig server i netværket (hvorpå server-programmet er aktivt), og mod aflevering af kvitteringen få udleveret den oprindelige datafil. Redundansen i systemet sikrer, at selv om én af de 3 servere som filen er fordelt ud på skulle svigte, eller forbindelsen til denne på en eller anden måde ikke kan opnås, så vil den server der kan kontaktes, automatisk genskabe datafilen og levere den til klienten. Hvis servernetværket kun består af én server klienten kan forbinde til, er det ikke muligt at genskabe datafilen. Serveren vil så aflevere en fejlmeddelelse, og brugeren/kunden må ty til at genskabe datafilen ud fra egen backup.

5.1 Start af programmer

For at få udbytte af programpakken, skal serverprogrammet bringes til at køre på 2 eller flere computere. Klientprogrammet kan nøjes med at køre på 1 computer.

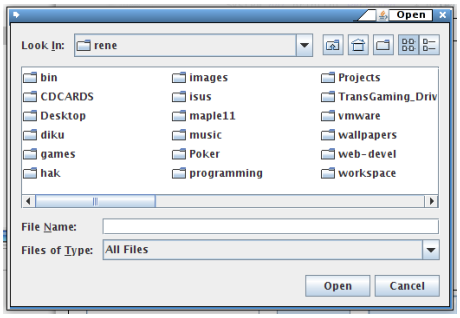
5.2 Gem en fil i netværket

Klientprogrammet skal forbinde til en server for at kunne gemme en fil med redundans via serveren.
I klientprogrammets hovedvindue vælges connect. Klientprogrammet søger dernæst selv en ledig server. Klientprogrammet laver en forbindelse til den første ledige server der svarer på forespørgslen. Når klientprogrammet har fået forbindelse til en server, (se figur 2),kan den datafil der ønskes gemt vælges: tryk på knappen "Choose file", (se figur 2), og naviger på sædvanlig måde i filsystemet, til den ønskede datafil. Vælg datafilen, (se



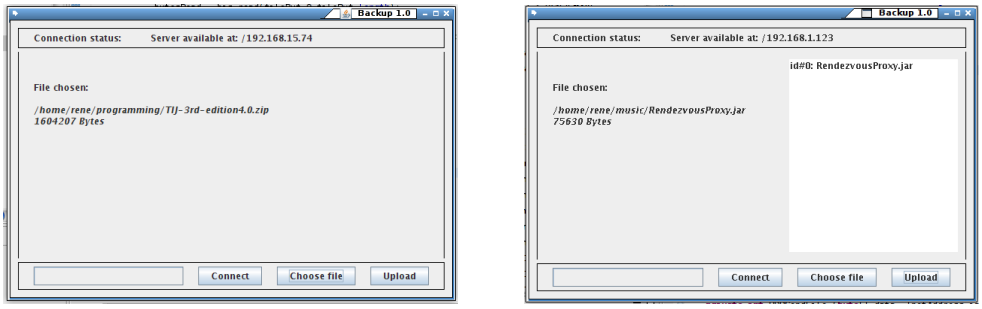
Figur 2: Connect og Choose

figur 3). Nu vises oplysninger om datafilen og dens størrelse. Herefter kan man vælge "Upload".(se figur 4)



Figur 3: Open

Nu gemmes den valgte datafil, og afhængig af størrelsen af datafilen vil der gå en tid inden klientprogrammet svarer tilbage med en ticket (billet).



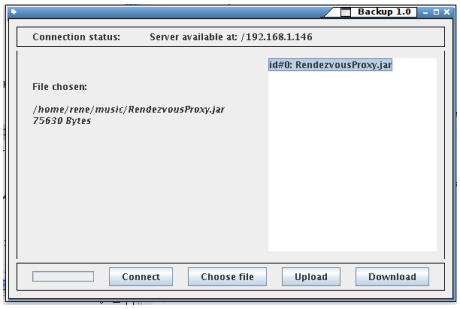
Figur 4: Upload og Ticket

Denne ticket skal bruges når det ønskes at hente en kopi af datafilen fra serveren igen.

5.3 Hent en fil i netværket

Klientprogrammet skal forbinde til en server for at kunne hente en datafil der er gemt med redundans via serveren. I klientprogrammets hovedvindue vælges connect. Klient-

programmet søger dernæst selv en ledig server. Når klientprogrammet er klar, kan den datafil der ønskes hentet, vælges.



Figur 5: Download

I klientprogrammet markerer man på sædvanlig måde Ticket for den ønskede datafil og knappen "Download"aktiveres. Klientprogrammet modtager dernæst den ønskede datafil fra serveren.

5.4 Fejltilstande og fejlmeddelelser

Der er mulighed for at der kan opstå fejl i kommunikationen mellem klientprogram og server eller mellem servere. Hvis klientprogrammet ikke har fået svar fra en server der er klar inden for 1,5 sekund, vises meddelelsen "No avaliable servers!"Hvis klientprogrammet ikke får en ticket tilbage fra serveren når datafilen er sendt afsted, får klienten meddelelsen: "File upload not available". Hvis en klienten har forbindelse til en server ikke længere har forbindelse til andre servere når en datafil skal hentes, får klienten fejlmeddelelsen: "File download not available".

6 Konklusion

Den måde vi har valgt at etablere kommunikation, datastruktur og implementation, gør, at vores program vil være meget åbent overfor skalering og udvidelse. Vi får ingen umiddelbare begrænsninger på, hvor mange klienter og servere der kan være i systemet. Endvidere medfører vores implementation, at det er nemt at teste systemet, idet vi kan køre flere servere og klienter i virtuelle maskiner på den samme computer.
Vi har haft en del vanskeligheder med at få detaljerne til at virke. Det er vores fornemmelse, at vi er på rette vej, men, der mangler en mere detaljeret test og fejlretning i programmet (J-Unit tests, f.eks.). Alt i alt, har det været en meget lærerig opgave med et interessant element af brugbarhed i implementationen. Vi har, som vanligt, erfaret, at det er meget vanskeligt at overholde præcise deadlines på spændende og muligvis store softwareprojekter.