CHALMERS |  GÖTEBORGS UNIVERSITET

Remote directory sharing
- RAccess

Kandidatarbete inom Data- och informationsteknik

ALEXANDER BERGERSEN
JOHAN GRÖNBERG
ERIC MEADOWS-JÖNSSON
EVELINA ORRLÖV

Institutionen för Data- och informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

GÖTEBORGS UNIVERSITET

Göteborg, Sverige 2012

Kandidatarbete/rapport nr 2012:35

Sammanfattning

Det finns idag inget enkelt sätt att dela mappar och filer mellan datorer över Internet utan att behöva spara dessa på flera ställen. Detta är ett problem då man ibland inte vill ha all sin information spridd över flera källor eller inte har nog med utrymme för att spara samma data på flera ställen.

Problemet i fråga blir att koppla ihop datorer över Internet. Då datorer kan flyttas runt mellan olika nätverk och därför ha olika IP-adresser krävs en lösning som håller reda på detta. En användbar lösning ska inte kräva att användarna ska veta om denna information. Eftersom projektet är över Internet så krävs det att det är väldefinierat hur kommunikation utförs.

Projektet delades in i tre komponenter, en klient, en server och en master-server, som kunde utvecklas sida vid sida av varandra då den enda interaktionen mellan modulerna är nätverkskommunikationen. När projektet var i slutfasen kunde alla modulerna sättas ihop för att få en fungerande prototyp.

Prototypen har utvecklats för Windows med hjälp av C# och .NET Framework 4. Projektets resultat är lyckat då en prototyp har framställt som presenteras senare i rapporten som innehåller nästan all önskad funktionalitet.

Abstract

Today there is no easy way to share files and folders between computers without having to save them in multiple locations. That is a problem due to the fact that it is not always desirable to spread information across multiple platforms or when there is not enough space to save multiple instances of the information.

The problem to solve is to connect computers over the Internet. Due to the fact that computers connected to the Internet can be moved around and be given different IP-addresses, there is a solution needed to keep track of this. A usable solution should not require its users to know this information. Since the project is over the Internet, network communication has to be well specified.

The project was split into three components, a client, a server and a master-server, that could be developed simultaneously as the only interaction between modules is the network communication. When the project was in the finishing stages all the modules could be put together to form a working prototype.

The prototype has been developed for Windows using C# and the .NET Framework 4. The project results were successful, a prototype that is discussed later in the report was developed that fulfilled almost all of the demands in the specification.

Ordlista

AES - Advanced Encryption Standard, en symmetrisk krypteringsalgoritm som använder sessionsnycklar för att kryptera kommunikation mellan två parter.

API - Application Programming Interface. Ett gränssnitt mot funktionalitet programmerare använder sig av för att skapa programvara.

Boiler-plate kod - Repetitiv kod som förekommer på flera ställen med ingen eller lite modifikation.

Drivrutin - Ett program som specificerar hur datorn interagerar med ett "fysiskt medium" och kan utföra funktioner som traditionella program inte har tillåtelse att göra, till exempel att kommunicera med hårdvara.

GUI - Graphical User Interface. Grafiskt användargränssnitt.

Integrationsserver - En server som automatiskt kompilerar kod och kör tester.

I/O - Input Output. Läsning och skrivning mot andra enheter som inte är minne eller processor. Till exempel hårddisk, nätverk eller skrivare.

Kernel - Kärnan av ett operativsystem.

Kernel-space & User-space - Program som körs i user-space har restriktioner på sig och skyddas av operativsystemet för att buggar i programmet inte skall kunna skada andra program som körs på datorn. Alla traditionella program på datorn körs i user-space, så som Office och webbläsare. Program som körs i kernel-space har färre eller inga restriktioner på sig och kan därför utföra mer saker, till exempel att kommunicera med hårdvara, men buggar i dessa program kan påverka stabiliteten på datorn. Program som körs i kernel-space är kerneln själv och drivrutiner.

Maskinkod - Assemblerkod som är skriven i processorns språk.

Migration - En process för att flytta data ifrån ett lagringmedium eller utrymme till ett annat.

Molnet - Teknik och tjänster på Internet som är stora och skalbara. Kan nås av användare via webbläsare eller program som använder sig av webbtjänster. Exempel är Dropbox och Gmail.

MVC - Model-View-Controller, ett populärt designmönster för applikationer med grafiska gränssnitt.

NAT - Network Address Translation. En teknik för att översätta IP-adresser som används på lokala nätverk till en IP-adress som används på Internet.

NAT-traversal - En teknik för att ansluta till datorer på Internet som befinner sig bakom ett NAT-filter.

NIST - National Institure of Standards and Technology, ett amerikansk institure som ansvarar för standarder i USA.

Kodportning - När kod konverteras från en platform till en annan eller konverteras till ett annat språk.

Protokoll - Ett strikt system för kommunikation med meddelanden och regler som flera parter har kommit överens om.

Ramverk - Ett mjukvaruramverk är ett bibliotek som kan kalla på användarens kod.

Rainbow table - En förberäknad tabell som användas för att få ut källan från en hashad sträng. Används oftast vid lösenordsknäckning.

Repository - Förvaringsställe för kod. Används i samband med versionshantering.

RSA - Krypteringsalgoritm döpt efter Ron Rivest, Adi Shamir, Leonard Adleman, en asymmetrisk algorithm som använder sig av publika och privata nycklar, används för att förhandla om sessionsnycklar till symmetriska algoritmer.

Salt - Slumpmässigt genererad sträng som används på en annan sträng när denne körs igenom en hashfunktion för att få en tredje sträng som inte kan återskapa originalet.

SMB - Server Message Block.

SSH - Secure Shell. Ett protokoll för att skapa en säker anslutning till andra datorer.

Token - Ett stycke information

Transaction – Används i denna text i form av databas transaktioner, dessa är en arbetsenhet där återhämtning ifrån fel kan ske och är när något fel uppstår i någon form

Virtuell maskin - En process-virtuell maskin kör ett program i ett system som är plattformsoberoende och låter programmeraren skriva program i ett högre-nivå språk.

VPN - Virtual Private Network. Ett protokoll för att skapa en tunnel mellan två punkter så att de upplever att de är anslutna till samma nätverk.

WDK - Windows Driver Kit. En grupp verktyg för skapa för drivrutiner för Windows.

Innehållsförteckning

1 Inledning

Det finns lösningar inbyggda i de flesta operativsystem för att enkelt dela ut mappar över lokala nätverk till andra användare som ska ha tillgång till dem. Det finns däremot inga liknande system för att dela ut mappar över Internet utan att använda sig av någon tunnel så som VPN eller SSH.

Att kunna dela ut mappar över Internet är användbart vid många olika tillfällen, till exempel om information finns i filer på en dator hemma som det behövs åtkomst till ifrån kontoret. Om presentationen glömdes att föras över och ligger kvar på en annan dator så hämtas den från den gemensamt delade mappen. Det är även användbart för en grupp människor som vill dela filer med varandra utan behov av att spara dem på flera ställen. Exempel på sådana grupper är personer i ett företag eller som gör ett grupparbete på en skola.

Det är ett problem idag att kunna dela filer snabbt och enkelt mellan flera datorer över Internet utan att sätta upp en tunnel eller utan behov av extra lagringsutrymme. Eftersom det inte redan existerar en lösning så finns det plats på marknaden för ett enkelt system där en eller flera användare lätt kan dela hela mappstrukturer, hårddiskar eller enkla filer mellan flera datorer. Sammanfattat är det problematiskt att dela ut mappar ifrån en dator till andra datorer, klienter, där man vill att datan ska kunna tas emot, skicka data till klienter vid behov och endast vid behov, samt att visa denna data på ett sätt som simulerar lokala filer på klientdatorn.

Det finns idag inga alternativ som löser ovan specificerade problem. Det finns alternativ som använder sig av en tredje part för lagring av all information, så som tjänster i molnet. Men det är inte alltid man vill att en tredje part ska ha hand om lagringen av ens data. Dessutom kostar alla alternativ pengar om man vill lagra mer än en liten mängd data. Det finns också alternativ som tillåter synkronisering av mappar mellan flera datorer men detta medför duplicering av data då den kommer vara lagrad på flera ställen och detta är inte alltid lämpligt eller ens möjligt.

1.1 Problemformulering

Problemställningen nämndes i föregående delkapitel. Det handlar om att en person ska kunna dela mappar eller hela hårddiskar mellan alla eller några av sina datorer. Det grundläggande i detta problem är att delningen ska ta plats över Internet istället för ett lokalt nätverk, vilket komplicerar mycket då datorer på Internet kan ha dynamiska IP-adresser och för att många datorer idag faktiskt är bärbara och flyttas runt mellan olika nätverk. Ett fungerande program måste dock kunna borste från detta och kunna användas var användarna än befinner sig. För att uppnå detta krävs det en dedikerad server som håller reda på vad som delas ut vart. En mapp ska kunna delas ut ifrån en dator, denna dator är servern i kommunikationen. Vilken annan dator som ska helst ska kunna användas för att ansluta till mappen givet att den kör programvaran och kan ansluta sig till servern.

Datorn som ansluter till en mapp är klienten i kommunikationen.
För att hindra obehörig åtkomst till privat information behövs ett användarkonto-system. Användarkonton behöver hanteras av en tredje part som servern och klienten

båda kan lita på eftersom användaruppgifter inte ska hanteras på okända datorer. Detta kan också hanteras av den tidigare nämnda dedikerade servern.

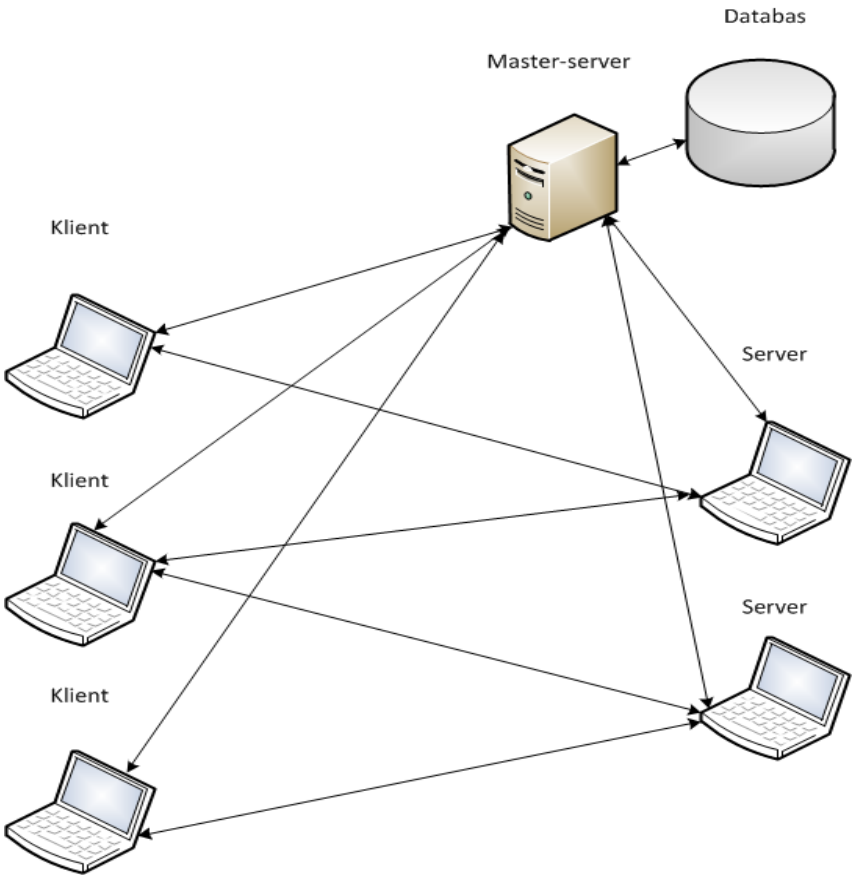
Problemet som beskrivs ovan för med sig många olika problemområden. För att det ska vara möjligt att ta emot mappar och visa dem på en klientdator så behövs en filsystemsdrivrutin vilken gör det möjligt att interagera med datorn och säga åt den hur informationen som tas emot ifrån servern ska visas. Vid alla distribuerade system är ett problemområde hur systemet hanterar samtida användare, speciellt här då samtida användare av filer kommer behöva hanteras.

Ett annat delproblem är att i ett användbart program bör inte sättet information visas på hindra användaren från att använda alla filer på samma sätt som användaren skulle göra på datorn där filerna faktiskt ligger lokalt. Detta betyder att en användare till exempel ska kunna lyssna på musik, editera bilder eller läsa textdokument och gå igenom alla filerna som om de vore lokala. Eftersom information delas över Internet så behövs också informationsöverföringen vara krypterad så att ingen utomstående person kan få tag i information som inte är menad för dem.

Till det givna problemet finns det en kravspecifikation som är uppställd, den kan återfinnas i Appendix A.

1.1.1 Systembeskrivning

För att lösa ovan specificerat problem behöver systemet som skapas delas upp i tre delar så som kan ses i figur 1.1; klient, server och master-server (den dedikerade servern nämnd tidigare). Master-servern används endast vid registrering och inloggning av användare och för att hålla reda på utdelade mappar. Servern och klienten är bara program som kan köras på användares datorer. Servern delar endast ut mappar och svarar och utför kommandon skickade av anslutna klienter. Klienten visar filsystemet på klientdatorn, alltså mappstrukturen av den utdelade mappen på servern, så att användaren och program på användarens dator kan använda filsystemet som om det var lokaliserat på klientdatorn.



Figur 1.1 Denna figur visar de tre olikadelar problemet delas in i

1.2 Liknande lösningar

Mycket av inspirationen till projektet kommer från Windows inbyggda fildelning. Den har till stora delar den funktionalitet som önskas. Den visar filerna som om de befinner sig lokalt på datorn och de strömmas - det är ingen synkronisering som sker. Men en del konceptuella skillnader i dess protokoll gör att den inte lämpar sig som lösning för detta specifika problem. Protokollet, SMB, bygger på NETBios [1]som är designat för lokala nätverk och kommer inte fungera över Internet utan hjälp av en så kallad tunnel [2]som simulerar ett lokalt nätverk, så som VPN [3].

Dropbox är ett program där en användare kan lägga till filer denne vill kunna komma åt från andra platser i en speciell mapp. Dropbox sparar sedan undan dessa filer på sin hemsida. Endast de personer användaren gett tillåtelse till kan komma åt filerna. Filerna synkroniseras till alla datorer knutna till användarens konto. Dropbox lösning har både fördelar och nackdelar. Eftersom filerna synkroniseras till alla datorer så kan man jobba med dem även om man inte har en Internetanslutning. Däremot dupliceras data till alla datorer som används så det är ingen bra lösning för stora mängder data. Dropbox kan inte heller använda operativsystemets inbyggda skrivskydd, så två användare kan jobba med samma fil och råka skriva över varandras ändringa [4]. Med vår lösning finns

operativsystemets inbyggda skrivskydd kvar så att endast en person i taget kan göra skrivändringar medan flera däremot kan läsa.

1.3 Syfte

Syftet med projektet är att ge en lösning på problemet som finns beskrivet i problemformuleringen, som är att dela filer och mappar över Internet mellan en användares datorer så länge denne är inloggad på rätt användarkonto. En given lösning ska också följa kravspecifikationen som är given i Appendix A efter bästa möjliga förmåga inom de givna ramarna för ett kandidatarbete.

Den givna lösningen ska enligt problemformuleringen även vara lättanvänd, vilket hädanefter definieras som att en van datoranvändare snabbt ska kunna lära sig programmet utifrån att använda det och inte genom att behöva läsa extra dokumentation.

1.4 Avgränsningar

Eftersom detta projekt utförs under en strikt tidsram finns det avgränsningar som inte kommer att hanteras i detta projekt. Dessa avgränsningar är inte vitala för problemställningen och anses inte möjliga att genomföra inom ramen för detta projekt. Om funktionaliteten som önskas i lösningen finns i ett redan utvecklat mjukvarubibliotek används detta istället för att egenutveckla funktionaliteten för att det ska kunna fokuseras på andra delar av lösningen.

Det är heller inte möjligt inom den givna tidsramen att implementera funktionalitet så att den som delar ut mappar i lösningen inte ska vara tvungen att öppna en port på sin router och på så sätt tillåta trafik att komma åt programmet. Det finns inte tid att utveckla så kallad NAT-traversal-teknik vilket skulle lösa detta problem. Att det inte fanns tid till det är den största anledningen till att detta inte implementerades. En annan anledning är att det inte heller finns någon standard för hur NAT-traversal utvecklas, de bra implementationer som finns i utspridda peer-to-peer-program är hemliga så det är svårt att utveckla på ett bra sätt på egen hand.

Det kommer inte heller utvecklas en egen drivrutin för filsystemet utan en lösning som redan finns för att kommunicera med operativsystemet kommer att användas. Implementation av krypteringen var också en sak som inte kunde utföras under den givna tidsramen.

2 Utvecklingsmetod

När projektet startades uppfördes en kravspecifikation, se Appendix A, för att göra gällande vad som skulle utföras och vad som krävs av en slutprodukt som ska lösa det givna problemet. Utifrån denna kravspecifikation delades sedan projektet upp i delproblem. En utvecklingsmetod togs fram för att få ett gemensamt sätt att arbeta på.

2.1 Delproblem

I projektets början utvärderades problemet och delades in i flera delproblem som kan lösas var för sig. Utifrån kravspecifikationen delades problemet in i sex olika delar.

2.1.1 Applikationslagerprotokoll

Applikationslagret är ett av de sju lagren i OSI-modellen [5]och används så som framgår av namnet av applikationer och program. För att programmet ska kunna fungera över Internet krävs ett fungerade protokoll som specificerar all kommunikation inom programmet. Ett protokoll behövs för att ha ett standardiserat format på datan som används för att kommunicera med, så att alla instanser av programmet kan vara med i kommunikationen. Detta krävdes också för att utvecklingen skulle kunna ske modulariserat då ingen information behövs om hur den andra modulen är implementerad utan endast om den exakta strukturen på datan som mottas ifrån andra moduler.

2.1.2 Säkerhet

När känslig information skickas över Internet och sparas i databaser så krävs säkerhet på flera olika områden. Det behövs säkerhet i databasen så att även om någon utomstående får tag i databasen så kan denne inte få ut någon användbar information. Vidare behövs kryptering och bra autentisering och auktorisering så att ingen utomstående kan läsa meddelanden som skickas och inte heller ta över en del i protokollet som till exempel i en masquerade attack [6].

2.1.3 Klient

Det behövs en klient som är den del av programmet där en användare tar emot mappar samt väljer vilka mappar som ska tas emot eller sluta tas emot. Klientdelen ska på så sätt vara den del av programmet som initierar så gott som all kommunikation enligt klient-server-modellen [7].

2.1.4 Server

Servern är den andra delen i klient-server-lösningen och är den del man delar ut mapparna ifrån. Eftersom det är specificerat att programmet inte ska spara mer information än nödvändigt så kommer servern också att ansvara för att strömma datan till klienten allt eftersom den är begärd. Servern ska till skillnad från klienten vara mestadels passiv. Den enda gång servern är aktiv är när den skickar information till master-servern.

2.1.5 Master-Server

Eftersom det är specificerat att lösningen ska fungera över Internet så krävs det att programmet inte är beroende av att IP-adresser förblir desamma som de var när mappen började delas ut. För att lösa detta problem skapas en master-server där man registrerar ett användarnamn och under detta användarnamn registreras vilken mapp som delas ut

på vilken IP-adress. Servern kommer att skicka ett hjärtslag till master-servern varje timme för att visa att den fortfarande är aktiv. Dessutom skickas hjärtslag varje gång information om mappar och/eller IP-adresser behöver ändras.

2.1.6 Nätverksdel

Eftersom programmet utvecklas modulariserat och alla de tre modulerna klient, server och master-server är nätverksberoende så bestämdes att det var för projektets bästa att ha en gemensam nätverksdel för att hantera all kommunikation. Detta för att inte behöva implementera tre olika versioner av samma funktionalitet samt för att endast en implementation ska behöva feltestas.

2.2 Versionshantering

Versionshantering är att hela tiden uppdatera existerande kodbas genom att lägga till förändringar. På detta sätt kan man enkelt se vem som har gjort vad och när det är gjort. Det är också också lätt att gå tillbaka en eller flera versioner genom att ta bort de senaste förändringarna om det upptäcks buggar eller andra felaktigheter.

För att hantera projektets kodbas används en teknik som hetet Git, den utvecklades av Linus Torvalds år 2005 och är ett versionshanteringssystem har visat sig tillförlitligt och används flitigt inom industrin av många stora projekt så som Linux [8]och Ruby on Rails [9]. Dessutom har vissa medlemmar i projektet föregående erfarenhet av Git vilket förenklar arbetet. Git kräver inte heller en aktiv Internetanslutning då all data även sparas lokalt på datorn så att ändringar kan göras lokalt för att sedan läggas till i projektet när uppkoppling finns att tillgå, vilket är bra då alla medlemmar i projektet inte alltid har tillgång till Internet.

För att kunna synkronisera versionshanteringen mellan alla olika datorer så användes Github [10] som ett privat repositorium som enbart är tillgängligt för projektmedlemmarna för att lägga upp ny kod och meddelanden till varandra angående möjliga problem.

2.3 Dokumenthantering

Detta projekt har en ganska stor dokumentbas med olika dokument så som protokollspecifikation, kravspecifikationer, argumentation om olika programdelar, hur saker bör göras, med mera. För att hålla dessa dokument synkroniserade mellan de olika datorerna används en mapp på googledocs [11]som skapats med inställningarna att endast medlemmarna inom projektet kan editera dokumenten. På så sätt är dokumenten sparade på ett ställe och alla som läser kan snabbt se uppdateringar. Det finns också stöd för uppdateringar av flera personer i realtid vilket är ovärderligt för idéstorming.

2.4 Testning

Under hela projektets gång skrevs integrationstester som testar att olika delar av programmet fungerar i samarbete med varandra. Detta är för att programmeraren ska kunna känna sig säker på att denne inte förstör saker eller introducerar nya buggar under kodskrivandet. Varje gång kodbasen uppdateras i repositoriet så bygger en integrationsserver automatiskt all kod för att se till att allt går att kompilera och kör alla tester för projektet. Integrationsservern som användes var TeamCity [12], skapad av JetBrains, som är gratis för små projekt. Denna process kallas kontinuerlig integration där man kontinuerligt utför kvalitetskontroll och används mycket inom branschen. Alla

i projektet kan ta del av resultaten och kan direkt åtgärda oförutsedda problem så att produkten håller sig så buggfri och stabil som möjligt.

3 Implementering

3.1 Plattformsval

Problemspecifikationen angav ingen specifik plattform de olika delarna i systemet skulle köras på. I början av projektet diskuterades då fördelar och nackdelar med olika plattformar som finns idag. Linux har en stor fördel i den mycket testade och välanvända filsystemsdrivrutinen Fuse [13] men nackdelar i att det inom projektet saknas tillräcklig kompetens inom Linux samt att den har en mindre användarbas. Slutligen valdes Windows som utvecklingsplattform då alla medlemmar i projektet har störst erfarenhet av Windows samt att det fanns lätt tillgång till druvrutiner för filsystem och att det inte skulle få plats inom ramarna för detta projekt att utveckla för multipla plattformar.

3.2 Språkval

När programmeringsspråk skulle väljas fanns det ett par egenskap som tycktes fördelaktiga. Eftersom Windows hade valts som plattform behövde språket vara lättanvänt på Windows. Syntax skulle vara likt något som projektmedlemmarna tidigare har erfarenhet med. Det skulle vara enkelt att skapa grafiska användargränssnitt och det var ett plus om det finns en stor community och massa hjälp-resurser att tillgå då detta gör det lättare att undvika att köra fast i utvecklingen. Till en början studerades C och C++ eftersom de drivrutiner som granskats alla hade API:n till C och för prestandan i språk som kompileras direkt till maskinkod. Men eftersom de flesta gruppmedlemmar hade lite erfarenhet av C/C++ och för att det tar längre tid och är svårare att skriva kod i än med språk som körs i en virtuell maskin så valdes inte något av de språken. Tillslut beslutades det att C# skulle användas eftersom språket överlag och syntaxen är lik Java vilket alla gruppmedlemmar har använt förut. I Visual Studio, som är programmet man skriver C#-kod i, finns inbyggt stöd för att snabbt och enkelt skapa grafiska användargränssnitt och det finns också en stor och aktiv community runt språket. Dessutom finns det bra resurser, så som böcker och information på Internet, för hjälp med koddesign och API.

3.3 Bibliotek

Programmerare använder sig ofta av bibliotek i skapandet av program. Dels för att det är bra att använda sig av saker som redan är skapade. Dels för att stora, kända bibliotek har många användare och därför fler ögon i letandet efter buggar vilket gör biblioteket mer stabilt och ofta bättre. Det låter även programmeraren koncentrerar sig på kärnfunktionaliteten i sin produkt, det som gör den unik. För att hantera alla bibliotek (förutom Dokan) användes NuGet [14]som är ett verktyg i Visual Studio som förenklar installationen och hanteradet av externa bibliotek. Programmen i systemet använde sig av följande bibliotek för att skapa produkten:

3.3.1 Dokan

För att förenkla utvecklandet av ett eget filsystem användes drivrutinen och biblioteket Dokan som förenklar skapandet av ett filsystem genom att omsluta WDK i ett skal som är enklare att använda och låter användaren skriva drivrutinen i user-space. Det innebär att en programmerare själv inte behöver sätta sig in i WDK och lära sig all dess funktionalitetet utan kan använda ett bibliotek som faciliterar skapandet av ett filsystem. Detta troddes kunna hjälpa eftersom drivrutiner anses vara svåra och komplexa att skapa på grund av bredden av WDK och att drivrutiner körs i kernel-space. Dokan valdes

eftersom det inte fanns tid att utveckla en egen drivrutin och det finns inga alternativ till Dokan som inte kostar pengar [15].

3.3.2 NUnit

NUnit användes för testning av systemet. NUnit är ett ramverk som man med hjälp av enkelt och strukturerat kan skriva tester till programmet. Inkluderat är även verktyg för att köra testerna och få ut resultat från dem. Resultaten visar exakt vilka test som fallerade och var och på vilka punkter de gjorde det. Detta underlättar programmeringen eftersom det blir känt direkt vilken kod som inte fungerar enligt testen och denna kan modifieras tills testet inte längre misslyckas. NUnit började som en port av Javas JUnit [16] till .NET men skrevs om i version 2 till att använda sig av .NET:s starka sidor. NUnit används av många stora projekt, är aktivt utvecklat och är allmänt känt som stabilt och väl beprövat. Det fanns också tidigare erfarenhet internt av användandet av detta bibliotek [17].

3.3.3 Ninject

Ninject är ett ramverk för Inversion of Control och dependency injection [18] (beskrivs i Designmönster nedan, 3.4.2) som gör att man kan associera olika gränssnitt till implementationer av dem på ett enkelt och smidigt sätt. Programmeraren kan även, bland annat, bestämma livslängden på objekten som skapas och hur och var objekten ska skapas. Ninject förenklar dependency injection och tar bort mycket av så kallad boiler-plate-kod som programmeraren annars skulle behöva skriva. Biblioteket skapades för att skaparna såg fördelarna med Inversion of Control men tyckte att alla existerande lösningar krävde för mycket konfiguration och var för komplexa. Ninject skapades därför med filosofin att vara enkelt att använda och inte kräva onödig konfiguration. Biblioteket valdes för att dess filosofi överensstämde med den inom projektet.

3.3.4 Protobuf-net

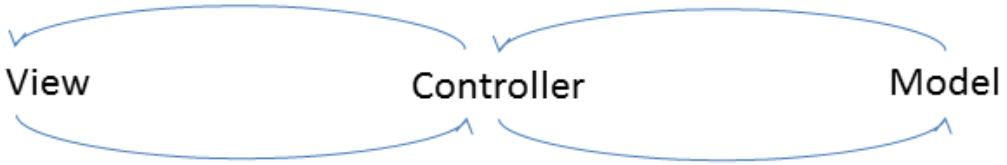
Protobuf-net är en .NET-implementation av Protocol buffers [19] som är ett sätt att koda data för överföring. Protocol buffers skapades av Google som behövde ett effektivt och snabbt dataformat som även är utbyggbart. Det används även mycket av Google internt. I projektet användes det i nätverksprotokollet för kodning av all data som ska skickas mellan de olika parterna i nätverket. Protobuf-net är skapat för att vara enkelt för .NET-programmerare att använda och man behöver bara annotera klasserna och dess data som ska användas och Protobuf-net kan automatiskt koda om dem för överföring. Eftersom Protobuf-net visade sig vara så effektivt och enkelt att använda så designades hela protokollet runt Protocol buffers. Protobuf-net valdes eftersom det behövdes ett snabbt format för att koda binär data och för att det förenklar för programmeraren om det är en enkel integration med .NET [20].

3.4 Designmönster

I programmering uppkommer ofta mönster då problemen som ska lösas ofta är mycket likartade eller kan abstraheras till något likartat. Istället för att då lösa problemet om och om igen finns det enkla mönster som man kan tillämpa i sina lösningar för att få enkla och strukturerade lösningar som sedan också kan förstås av andra människor. Eftersom detta är ett så omfattande mjukvaruprojekt används flera olika designmönster för att få struktur på programmet.

3.4.1 Passive view

Passive view är en av många variationer på det klassiska designmönstret MVC som berör hur ett användargränssnitt bör interagera med ett program. [21]. Denna vidareutveckling adresserar mest de existerande problem som finns med att testa applikationer som har stor interaktion genom användargränssnitt. Dessa problem löses genom att flytta ut nästan all logik från användargränsnitt till en annan klass som kallas Controller som motsvarar Controller i MVC. Till skillnad från MVC så är aldrig viewn aktiv då all logik är flyttad till Controllern. Detta betyder att View och Model aldrig kommer att kommunicera direkt som de gör i MVC. Istället är det ett "linjärt" system där View pratar med Controller och Controller pratar med Model så som kan ses i figur 3.1. [22]. Här nedan visas på ett förenklat sätt hur passive view används i projektet.



Figur 3.1 Kommunikationsflödet i Passive View mönstret

Att all logik är koncentrerad gör att testning kan fokuseras på ett område (Controller) utan risk för några problem med det grafiska användargränssnittet. Detta är anledningen till att just detta designmönster valdes.

3.4.2 Inversion of Control

Inversion of Control [23]används för att minska kopplingen mellan olika objekt i en kodbas. Istället för att kopplingen bestäms av en kompilator, vilket gör den statisk, finns det specifik kod som bestämmer kopplingen under programmets körning. Det gör att kopplingen blir mer dynamisk. Oftast önskas svag kopplingen mellan objekt så att koden blir så lite beroende av andra delar av koden som möjligt. Svagt kopplad kod är positiv i flera aspekter. Delar av koden kan bytas ut eller ändras utan att andra delar av koden behöver ändras eller endast behöver ändras minimalt. Det gör det även lättare att flytta kod till andra kodbaser och börja använda nya bibliotek i existerande kod. Eftersom objekt endast känner till gränssnitten till andra objekt så kan implementeringen av gränssnitten ändras. Andra fördelar är att man kan byta ut delar av koden under testning och därmed isolera exakt de objekt man vill testa så att en sådan liten del av koden som möjligt testas vilket gör det lättare att hitta och rätta till buggar.

För att åstadkomma Inversion of Control används dependency injection som är ett sätt för ett objekt att få de objekt den beror på injicerade i sig. I detta fall valdes att göra

injicering i objektets klasskonstruktor. När ett objekt behöver skapas dynamiskt, oftast när mer än ett objekt behövs, används factory pattern [24]. För att implementera dessa designmönster användes Ninject som omnämnt tidigare.

3.5 GUI

För att få ett användarvänligt program så bestämdes det att ett grafiskt användargränssnitt skulle användas för att kommunicera med användarna genom. Det bestämdes att projektet ska ha fyra olika vyer för att separera olika funktionaliteter så gott det går och därmed göra programmet lättare att använda. Dessa fyra olika vyer är ett loginfönster, ett registreringsfönster, ett klientfönster samt ett serverfönster.

I loginfönstret kan en användare logga in med ett redan registrerat användarkonto. Registreringsfönstret ska tillåta att en användare registrerar ett nytt användarkonto. Klientfönstret ska lista mappar som en användare kan ta del av samt ge denne möjlighet att ansluta till dessa. Slutligen ska serverfönstret ge användaren möjlighet att dela ut samt sluta dela ut specifika mappar.

3.6 Komponenter

Systemet delades in i tre olika komponenter. Dessa är master-server, server och klient. Det är master-servern som sköter autentisering av användare samt lagring av information om vilka mappar som finns på vilka datorer, servern delar ut mappar från sin dator och som kommunicerar med den klient som vill ta emot informationen och klient tar emot mappar som den väljer ifrån en eller flera serverar och monterar dessa lokalt på sin dator.

3.6.1 Master-server

Master-servern är den komponent som till funktionaliteten har minst roll i programmet, det jobb den utför är dock vitalt för att resten av programmet ska fungera. Master-servern existerar för att autentisera användare samt för att koppla ihop auktoriserade klienter med servrar så att dessa kan starta sitt informationsutbyte.

Eftersom alla dessa uppgifter kräver information ifrån andra parter så har master-servern en helt passiv roll i programflödet. Detta betyder att den aldrig initierar kommunikation med någon annan utan lyssnar på en bestämd port efter meddelanden och att den sedan utför olika uppgifter beroende på vad den andra parten begär.

3.6.1.1 Databas

Master-servern ska hålla reda på information om användare så som användarnamn och lösenord, vilka IP-adresser användare använder för att dela ut mappar samt vilka mappar som delas ut från vilken IP-adress.

Eftersom detta kan bli en stor mängd information som måste bibehållas även vid en omstart eller programkrash behövs en databas för lagring. I början av projektet valdes databasen SQlite då det är en liten och lättviktig databas som inte kräver någon konfiguration eller server för att fungera. Denna databas valdes för att den är enkel samt att det med denna lösning gick snabbt att få igång arbetet och få en fungerande databas [25].

Senare i projektet upptäcktes dock ett problem med SQlite och detta var en avsaknad av transactions vilket behövdes vid uppdateringar av databasen då mycket information uppdateras på en gång och det är viktigt att all data i en transaktion kommer med som den ska. Misslyckas en operation ska hela transaktionen kunna backas tillbaka. Även en avsaknad av migrations, vilket behövs i projektet vid testning samt för att distribuera uppdateringar av databasen till alla utvecklare, spelade en roll i beslutet att välja bort SQlite. För att få tillgång till denna funktionalitet valdes Microsofts SQL Server då den har all funktionalitet som sökes samt en stor support- och hjälpcommunity.

3.6.1.2 Autentisering

Användare autentiserar sig mot master-servern för att auktoriseras för att logga in på klienten/servern och kunna få tillgång till programmets funktionalitet. Autentisering sker genom att ett användarnamn och ett lösenord skickas till master-servern för att sedan kollas upp i databasen. Då lösenord är känslig information så körs dessa igenom en 256 bitars hashfunktion tillsammans med ett 256 bitars salt innan de läggs in i databasen.

Om det var en klient som autentiserades skickar master-servern tillbaka en token som är specifik för den klienten.

3.6.1.3 Auktorisering

För att en klient ska kunna få tillgång till utdelade mappar krävs det ett andra steg utöver autentiseringen ovan. Det går till så att när en klient vill ansluta sig till en server skickar klienten med den token den fick av master-servern. Servern vidarebefordrar sedan denna token till master-servern för att kontrollera att den är giltig för denna klient.

3.6.2 Server

Servern i projektet är inte en regelrätt server, det laddas inte upp filer och mappar till någon icke-lokal server som sedan kan laddas ner igen om klienten är auktoriserad. Servern visar endast vilka mappar på datorn som delas ut och kan vid behov strömma filerna dessa innehåller via Internet till en icke lokal, auktoriserad användare. Mapparna och dess filer behöver inte flyttas från serverns dator till användarens för att vara åtkomliga.

När det gäller kommunikation är servern i princip passiv, den lyssnar efter meddelanden från klienten och reagerar sedan på dem. Den enda gången servern är den som initierar kommunikation är när den skickar hjärtslag till master-servern, vilket sker en gång i timmen så länge servern är igång. Hjärtslagen berättar för master-servern att servern fortfarande är i gång och skickar också med information om vilka mappar som delas ut av servern.

Servern tillåter inte vilken användare som helst att komma åt mapparna den delar ut, utan när en användare försöker koppla upp sig mot servern kollas användarens åtkomsträttigheter, ett användarnamn och en token som användaren fått vid inloggning, med master-servern som sedan svarar på om användaren i fråga har den rätta auktorisationen för att komma åt serverns mappar.

3.6.3 Klient

Klientdelen i programmet är den som används för att koppla upp användarens dator mot mapparna servern delar ut.

Klienten är den aktiva delen när det gäller kommunikationen, det är den som initierar alla konversationer när det gäller dess kommunikation med de andra delarna av programmet. En användare väljer sedan vilken eller vilka mappar denne vill ta del av samt hur dessa mappar ska visas på datorn. Dessa mappar monteras som lokala filsystem på klientens dator. Klienten håller reda på vilka servrar den tar emot vilka mappar ifrån så att den sedan vid behov kan be den servern strömma över den information som behövs.

Klienten kan skapa nya filer och mappar i den delade mappen som då faktiskt skapas på serverns dator. Den kan också göra ändringar eller läsa filer. När en mapp inte behövs längre så kan användaren avmontera den och på så sätt stänga anslutningen mot servern och ta bort filsystemet från sin egen dator.

Vid avstängning av en klient stängs alla anslutningar ner och alla mappar avmonteras från användarens dator så att inga misstag ska inträffa om en användare glömmer att stänga alla anslutningar före programmet stängs av.

3.6.3.1 Filsystem

Filsystemet är en del av klienten och används för att låta användare och program på klientdatorn komma åt och kunna arbeta med serverns utdelade mappar och filer. Filsystemet kommer att visas som en disk i utforskaren.

I Windows måste filsystem implementeras som en drivrutin i kernel-space. Det är svårt och tar lång tid att skapa en drivrutin, speciellt om man vill få den stabil och säker.

Därför gjordes en avgränsning och det valdes att använda en färdig drivrutin där man kan implementera funktionalitet i user-space. Detta sparade mycket tid som gjorde att det kunde läggas mer tid på andra delar av produkten. Projektet använder Dokan som drivrutin. Dokan har en drivrutin som körs i kernel-space och gör callbacks till projektets kod som körs i user-space när operativsystemet kallar på drivrutinen.

3.7 Programstruktur

Eftersom programmet skrivs i Visual Studio så används Visual Studios projektstruktur. I toppen av strukturen finns en lösning, under lösningen finns flera projekt, varje projekt är ett program eller ett bibliotek, och under projekten finns mappar och källkodsfiler.

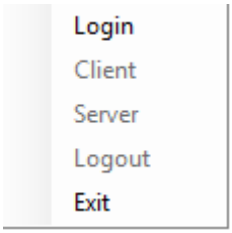
Beslut gjordes att dela upp lösningen i fyra projekt. “Raccess” som innehåller servern och klienten, “Raccess.Master” som innehåller master-servern, “Raccess.Common” som innehåller kod gemensam för klienten/servern och master-servern så som nätverkskod och annan infrastrukturkod och till sist “Raccess.IntegrationTests” som innehåller alla tester.

4 Funktionalitet

4.1 Gentemot användaren

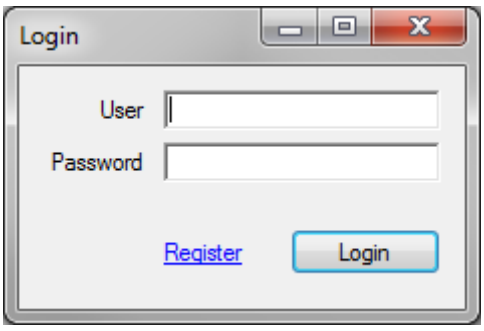
Under projektet tillverkades en prototyp av programmet. Den implementerar det mesta av den funktionalitet som står uppställd i kravspecifikationen. Kapitel 5.1 går igenom detaljerna om vilka delar ur kravspecifikationen som är uppfyllda och vilka som inte är det.
Trots att prototypen inte är den slutliga produkten, ser den näst intill ut som den färdiga produkten väntas göra. I detta avsnitt undersöks det grafiska användargränssnittet för att se hur väl specifikationen implementerades. Därefter följer en redogörelse för precis hur prototypen arbetar när den är i bruk.

När prototypen startas så dyker det upp en ikon i aktivitetsfältet på skärmen. När användaren högerklicka på ikonen dyker det upp en lista bredvid ikonen på de fönster som finns att öppna, denna meny kan ses i figur 4.1. Det finns också en exit-knapp. Innan användaren har loggat in så är det endast länken till loginfönstret som är klickbar. Länkarna till de andra fönstrerna blir klickbara först när användaren är inloggad.



Figur 4.1 Figuren visar menyn om en användare högerklickar på ikonen i aktivitetsfältet.

Loginfönstret som kan ses i figur 4.2 har endast två funktionaliteter. Dels funktionen att logga in på systemet genom att skriva in användarnamn och lösenord, men användaren kan också komma till registreringsfönstret genom att trycka på länken “Register”. Vid ett försök att logga in med en oregistrerad användare kommer det upp ett felmeddelande. Vid en lyckad inloggning stängs loginfönstret och det dyker upp en så kallad “toast notification” vid ikonen i aktivitetsfältet som informerar användaren att denne har blivit inloggad och att de andra delarna av programmet är tillgängliga från aktivitetsfältet.

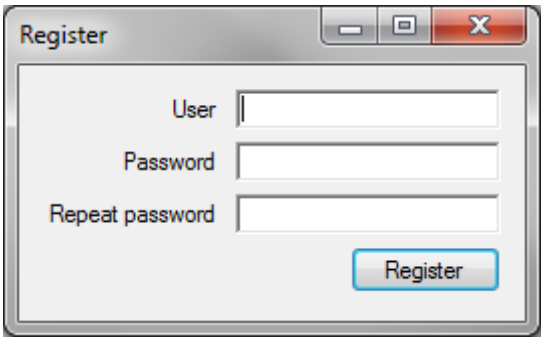


Figur 4.2 Visar login loginfönstret

Registreringsfönstret kan ses i figur 4.3 och är även det simpelt i sitt upplägg. Det består endast av tre textfält, ett för användarnamn, ett för lösenord och det sista för att upprepa

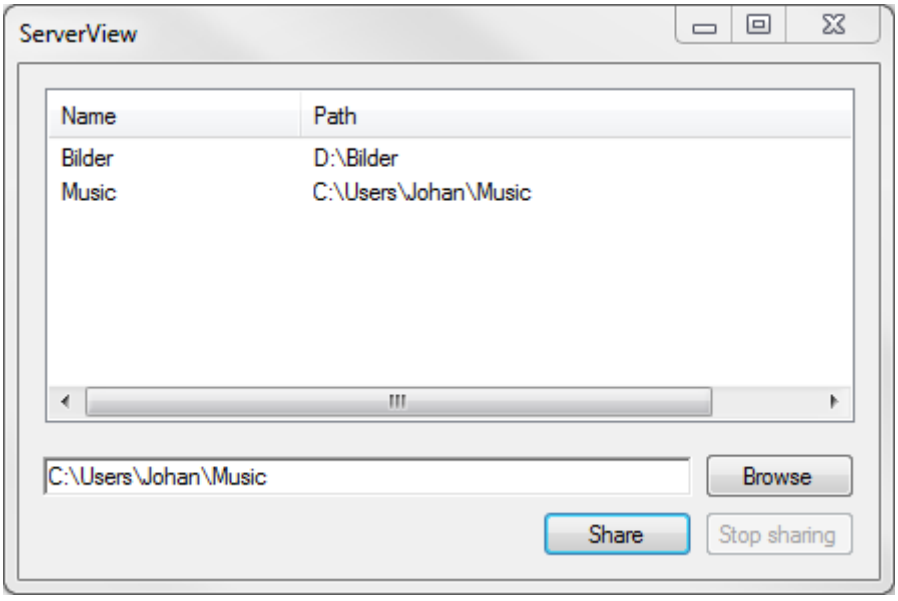
Figur 4.2 Figuren visar inloggingsfönstret

lösenordet. Ifall lösenordet inte skrivs in lika i det andra lösenordsfältet så meddelas detta till användaren genom att textfältet blir rött.



Figur 4.3 Visar registreringsfönstret

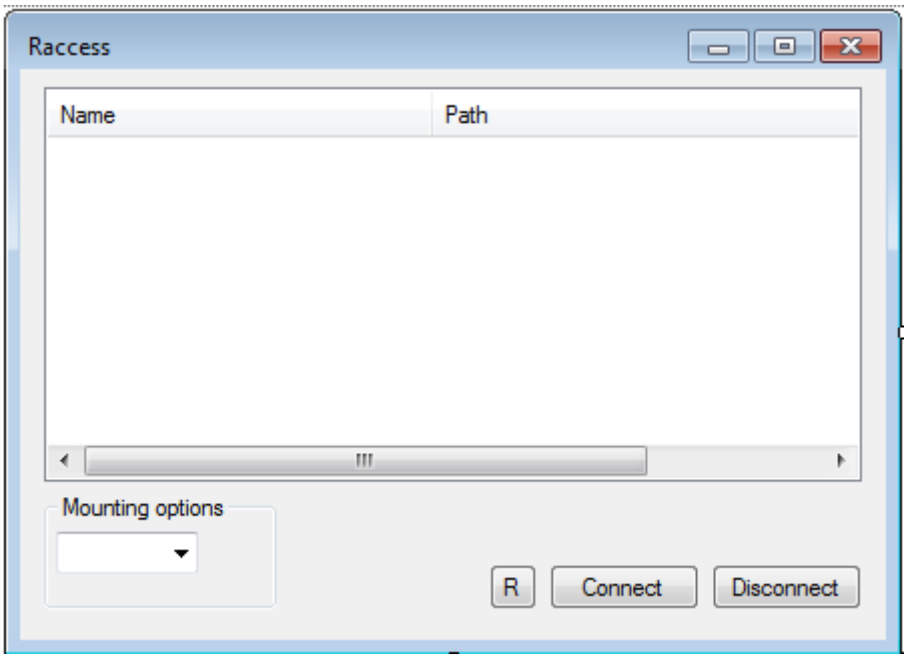
När användaren är inloggad kan denne välja att öppna serverfönstret eller klientfönstret. Serverfönstret, vilket återfinns i figur 4.4, har funktionalitet för att dela mappar från den egna datorn, och klientfönstret som återfinns i figur 4.5 har funktionalitet för att koppla sin dator till de mappar som delas med användarens konto.



Figur 4.4 Visar serverfönstret

I serverfönstret kan man bläddra fram en viss mapp som ska delas ut. För att dela en mapp så markerar användaren denna och trycker sedan på knappen “Share”. På samma sätt slutar mapparna att delas genom att en mapp som delas markeras och sedan trycks knappen “Stop sharing” ner.

Klientfönstret är upplagt på liknande vis. När det öppnas syns de mappar som går att ansluta till i det stora fältet. Användaren markerar den mapp denne vill ansluta till. Därefter trycker användaren på “Connect”. På samma sätt slutar användaren att strömma innehållet i en mapp genom att markera mappen och trycka på “Disconnect”. När användaren strömmar mappar så kommer mapparna visas på användarens dator som lokala mappar. Användaren får därför välja var respektive mappen ska placeras. Det är möjligt att visa de strömmade mapparna både som hårddiskar eller som mappar i valfria kataloger på datorn.



Figur 4.5 Visar klientfönstret

4.2 Systemfunktionalitet

4.2.1 Säkerhet

Krypteringen i systemet består av både ett publikt nyckelsystem, RSA, och ett som använder sig av sessionsnycklar, AES. Ett publikt nyckelsystem fungerar genom att man har en publik och en privat nyckel. Den publika nyckeln kan vem som helst känna till och den används för att kryptera meddelanden till personen med den motsvarande privata nyckeln, då det bara är den personen som kan dekryptera och läsa meddelandet. Bara personen som skapat nyckelparet bör känna till den privata nyckeln [26].

System där sessionsnycklar används fungerar på så sätt att när en part vill kommunicera med en annan skapar de en sessionsnyckel som de på något sätt (specificeras aldrig hur) ger till den andra parten de vill kommunicera med, utan att någon annan tar del av den. Sedan kan de två med sessionsnyckeln kryptera och skicka meddelanden som bara den andra kan dekryptera och läsa. En sessionsnyckel bör ej återanvändas [27].

Krypteringen i detta system fungerar på samma sätt mellan servern/klienten - master-servern och klienten - servern. När master-servern eller en server startar genererar de ett RSA-nyckelpar bestående av en privat och en publik nyckel. Den publika nyckeln används av de som vill kommunicera med servern/master-servern för att kryptera sagd

kommunikation medan den privata används av servern/master-servern för att dekryptera. När en enhet vill kommunicera med servern/master-servern skickar den ett meddelande som ber om den publika nyckeln. Servern/master-servern svarar med att skicka den nyckeln till enheten ifråga som då genererar en AES-sessionsnyckel som krypteras via RSA med den publika nyckeln och skickar sessionsnyckeln tillbaka till servern/master-servern. Efter att sessionsnyckeln är mottagen krypteras all data som skickas mellan de två inblandade enheterna med sessionsnyckeln genom AES.

Detta system kan jämföras med SSL som fungerar på liknande sätt [28].

4.2.2 Registrering

Registrering av nya användarkonton görs av en klient och kräver en inmatning av användarnamn och lösenord. Lösenord måste dessutom utifrån standard anges två gånger så att användaren inte av misstag har matat in fel lösenord första gången. Det kontrolleras lokalt i klienten att båda lösenorden är lika samt att de består av minst ett tecken som inte är ett mellanslag. Om den inmatade informationen inte går igenom dessa kontroller rödmarkeras det fält där felinmatningen har skett så att användaren förstår vad som är fel, om det däremot lyckas så skickas en förfrågan om registrering till master-servern.

Master-servern tar emot förfrågan om registrering och måste då evaluera om den mottagna informationen går att registrera en användare med. Detta görs genom att kontrollera användarnamnet mot databasen, om det redan finns en användare med detta namn skickas ett felmeddelande tillbaka till den klienten som skickade förfrågan. Om det inte finns någon användare vid detta namn så skapas ett så kallat salt som sedan konkateneras med lösenordet för att köras igenom en hashfunktion för att skapa ett hash av lösenordet så att lösenordet inte sparas som vanligt text. Detta är också för att det ska vara så gott som omöjligt att ta reda på originallösenordet även om någon skulle få tillgång till databasen. Till detta används hashfunktionen SHA-1 då den är en av de fem hashfunktioner som NIST definierar som säkra för användning [29]. Eftersom lösenorden i vissa fall är små och för att samma användare inte ska tilldelas samma hash även fast de har samma lösenord skapas även ett salt för varje användare som konkateneras med lösenordern före det körs igenom hashfunktionen. Detta skyddar även mot attacker med rainbow tables då varje hash också beror på en slumpmässig sträng och inte bara på lösenordet [30].

Detta lösenord läggs sedan in i databasen tillsammans med det salt som genererades för användaren och sedan skickas ett meddelande tillbaka till klienten om att en användare skapats.

4.2.3 Inloggning

Inloggning utförs enligt specifikation på både klienter och servrar för att användare ska få tillgång till funktionaliteten dessa erbjuder. För att logga in krävs det att användaren anger ett namn och ett lösenord som består av minst ett tecken som inte är ett mellanslag. Detta kontrolleras lokalt av klienten före någon data skickas över nätverket. När dessa krav uppfylls så skickas ett meddelande, på det sätt som beskrivs i 4.2.5 nedan, till master-servern.

Master-servern tar emot meddelandet och ser att det är ett auktoriseringsförsök och kollar således om det finns en användare med det givna användarnamnet i databasen. Om detta inte existerar så skickas ett felmeddelande tillbaka. Om det däremot finns så tas det salt som har sparats för användaren fram och sedan återskapas samma förlopp som beskrivs ovan (i 4.2.2) och kontrolleras mot det som har sparats i databasen. Om användaren loggas in felfritt så skapas en token som också läggs in i databasen. Detta är en slumpmässig 32 byte lång bas64-sträng. Detta betyder att varje tecken kommer att vara ett av 64 möjliga samt att det kommer att vara 44 tecken långt. Det kommer att vara 1/ (44^64) risk att någon lyckas återskapa just denna token. Den token som skapats sätts sedan in i databasen tillsammans med en tidsstämpel för när den skapades, denna är sedan giltig i en minut från att den blivit skapad. Detta för att det ska vara så gott som omöjligt för någon att skapa en egen token inom tidsramen då den skapade token är aktiv.

Om det är en klient som loggar in kommer sedan denna token skickas tillbaka till klienten för senare användning. Om det däremot är en server som loggar in så behöver denna token inte skickas tillbaka då en server inte har någon användning av denna information.

4.2.4 Utdelning och prenumeration på mappar

Servern har en lista med de mappar den delar ut. Den skickar denna lista en gång i timmen till master-servern som i sin databas har alla listor med utdelade mappar från de olika servrarna som är uppkopplade mot sagd master-server. När master-servern tar emot meddelande från en server med serverns lista kollar master-servern den mottagna listan mot sin databas och uppdaterar den lokala listan i databasen om det behövs. Servern skickar också en uppdatering om det läggs till nya mappar till dess lista med utdelningar. Klienter får ta del av listorna med utdelade mappar från servrarna de är auktoriserade att koppla upp sig mot via master-servern. Det sker genom att när master-servern uppdaterar sin databas skickar den de relevanta listorna till klienten som då uppdaterar sin lokala lista.

När klienten vill prenumerera på en mapp skickar den ett meddelande till servern, som äger mappen, med en prenumerationsförfrågan och mappens namn. När servern tar emot meddelandet kollar den först upp att den faktiskt delar ut mappen. Om den gör det svarar den med ett “Ok” och mappen börjar strömmas till klienten och kan nu används på klienten som vilken lokal mapp som helst. Om mappen faktiskt inte delas ut skickar servern tillbaka ett meddelande till klienten som berättar detta.

4.2.5 Nätverkskommunikation

All kommunikation mellan parterna, klient, server och master-server, sker enligt protokollet specificerat i Appendix B. Kommunikationen är uppdelad i meddelanden som är den enhet som skickas mellan parterna. Varje meddelande består av ett kommando, ett identifikationsnummer, och möjligen data som är specificerat av ett kontrakt och längden på nämnda data. Kommandot är titeln på meddelandet, det säger vad meddelandet handlar om. Identifikationsnumret används för att identifiera specifika meddelanden, det är nödvändigt om en part skickar flera meddelanden i en följd och vill skilja på svaren.

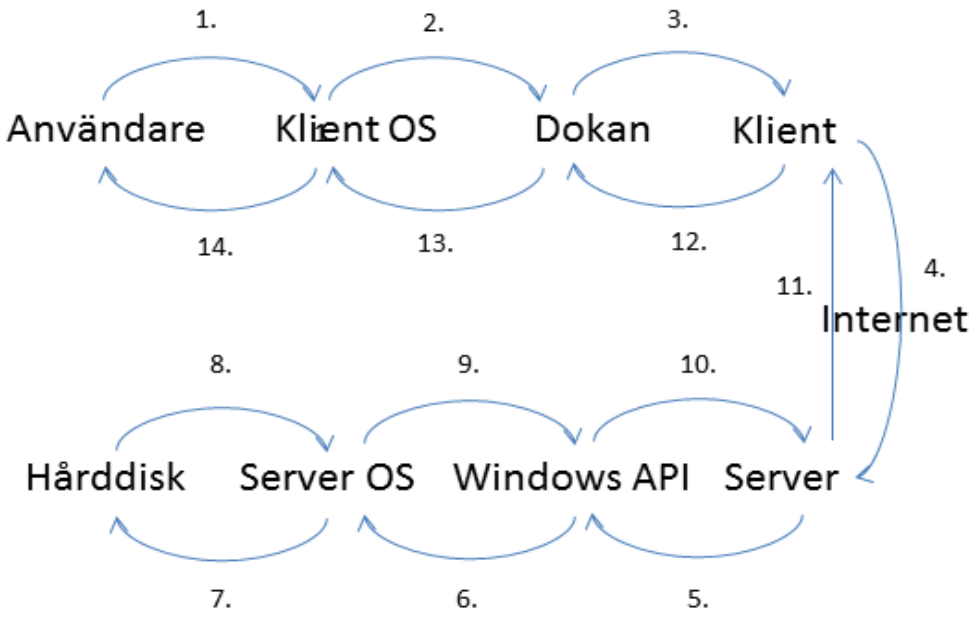
Datan i meddelandet skickas binärt och följer direkt efter textdelen av meddelandet. Datan kodas enligt Protocol buffers-specifikationen och görs av Protobuf-net, omnämnt tidigare. Datan är specificerad av kontrakt vilket är klasser som annoteras med hjälp av Protobuf-net-biblioteket. Det gör att datan alltid är typsäker och fel hittas tidigt i utvecklingsfasen.

4.2.6 Filsystem

För att visa och låta användaren interagera med serverns utdelade mappar och filer används ett filsystem. Ett filsystem är ett sätt att organisera data och bestämma operationer som kan utföras på objekten i filsystem, till exempel skriva till och uppdatera filer, kopiera, flytta och byta namn på mappar och filer. Beslut gjordes att använda sig av Dokan som drivrutinen för filsystemet så att en egen drivrutin inte behövde skrivas eftersom det är en stor och komplex uppgift. Dokan låter en utvecklare skriva filsystemet i user-space och fördelarna med det har beskrivits i tidigare kapitel, 3.3.1.

För varje filsystemsoperation gör drivrutinen en callback till projektets kod. En callback är en referens till en funktion som gör att mjukvara kan kalla på den funktionen vid ett senare tillfälle. För varje callback skickas information om hur filsystemsoperationen ska utföras från klienten till servern. Servern utför operationen lokalt på sitt filsystem och skickar tillbaka resultatet till klienten som i sin tur svarar Dokan med resultatet.

I figur 4.6 kan man se ett typiskt flöde för en filsystemsoperation, till exempel en filkopiering, startad av en användare på klientdatorn. Användaren informerar operativsystemet (1) om att en kopiering ska utföras, genom att till exempel dra en fil till en annan mapp. Operativsystemet ser att operationen utfördes i ett filsystem som körs av Dokans drivrutin och skickar vidare operationen till den (2). Dokan kallar i sin tur på klientens kod genom (3) en registrerad callback. Operationen kodas om till data enligt ett Protocol Buffers-kontrakt för att skickas över Internet till servern (4). Servern tar emot datan och kodar av den enligt samma kontrakt och kallar på operativsystemet genom Windows API (5) för att låta operativsystemet utföra operationen (6), kanske mot en hårddisk som är hanterad av ett annat filsystem (7). Operativsystemet svarar på serverns anrop via Windows API (8-10) med resultatet av operationen. Servern kodar resultatet till data, som tidigare för att skicka den till klienten (11). Klienten kodar av datan och skickar vidare resultatet till Dokan (12) som svarar operativsystemet (13) och visar resultatet för användaren (14).



Figur 4.6 Projektets kod består av klienten och servern samt kommunikationen över Internet och gränssnittet till Dokan och Windows API.

5 Resultat och Verifiering

5.1 Prototypens kravuppfyllnad

För att utvärdera hur lyckad prototypen blev jämfördes den mot kravspecifikationen (finns att läsa i sin helhet i Appendix A). I den är all funktionalitet som projektet hade som mål att uppnå listad. En prototyp som uppfyller alla krav i specifikationen kan ses som den produkt som var målet för projektet. Huruvida produkten även i verkligheten är vad som önskades kommer fram efter att en analys av användartester har gjorts. Detta diskuteras i nästa kapitel.

Merparten av villkoren i kravspecifikationen har uppfyllts. Det finns undantag då det var känt att projektet var mycket tidsbegränsat,. Precis som kravet om leveransvillkor specificerar: "Helst ska produkten levereras som en färdig version men en fungerande prototyp är också godtagbart". Det skulle därför kunna anses som att kravspecifikationen trots allt uppfylldes, då en fungerande prototyp blev färdigställd.

Mer detaljerat är det som saknas för att ha en färdig produkt:

Bland de funktionella kraven, vad programmet är kapabelt att göra, är all den funktionaliteten som begärs av kraven implementerad. Vidare är den feltolerans som har implementerats inte tillräcklig för det behov som produkten har på grund av att den drivrutin som används för filsystem kan medföra stora problem om programprocessen skulle dö utan att avmontera filsystemet, bland annat vid en datorkrash.

När det gäller icke-funktionella krav, så finns det ett prestandaproblem som strider mot kravspecifikationens krav om kapacitet. I kravspecifikationen går det att läsa: "Systemet bör vara tillräckligt snabbt för att inte begränsas av datorkapacitet utan av I/O så som nätverk eller hårddisk". På grund av att projektets kod skrevs i C# som körs på .NETs virtuella maskin krävs det marshalling när vi kommunicerar med Dokans C-bibliotek. Marshalling är konverteringen av data mellan Cs och .NETs datastrukturer. Det är även viss prestandaförlust i vanliga funktionsanrop eftersom koden som körs förflyttar sig mellan .NET och C. Ett annat prestandaproblem är när en mapp med många filer öppnas på klienten. Servern måste skicka information om varenda fil i mappen vilket kan ta lång tid och gör att applikationen som öppnar mappen ser ut att låsa sig medan den väntar på servern att svara. En lösning på detta vore att inte vänta på att all filinformation skickats utan skicka stycken av filer.

I alla övriga aspekter uppfylls kravspecifikationen.

5.2 Användartester

Som det berördes vid i föregående stycke så bestämdes det att det skulle utföra användartester av prototypen, för att få en bild om hur väl prototypen fungerar i praktiken. Även en produkt som uppfyller alla kravspecifikationer kan brista i användarvänlighet, utan att det är något som kommer fram när endast utvecklarna provkör produkten. De personer som själva har jobbat med en produkt från första början är alldeles för införstådda i alla delar av produkten för att kunna göra en subjektiv bedömning av den.

Prototypen är utvecklad med tanken att personer med datorvana ska kunna använda den utan mer hjälpmedel än en README-fil som ingår i programfilerna. Trots detta tilläts en blandad grupp personer testa prototypen, både personer med datorvana, och personer utan lika mycket datorvana. Detta för att få mer perspektiv på prototypen.

Användartesterna gick till så att testpersonerna fick programmet skickat till sig. Efter instruktioner fick personen sedan registrera sig, logga in och så vidare. Först efter att testpersonen använt programmet efter våra instruktioner fick de lov att på egen hand utforska det och själv upptäcka de olika möjligheterna med programmet. Efter att testpersonen fått testa på programmet får denne möjlighet att svara på en enkät som satts ihop, om hur testpersonen upplevt prototypen. Frågorna till enkäten samt de instruktionerna som gavs till användarna finns att återfinna i Appendix C. Resultaten av testerna var i stora drag positiva. Det kommer att diskuteras mer ingående i nästa kapitel.

6 Diskussion

6.1 Prototyp

De krav ur kravspecifikationen som inte uppfylls handlar om att systemet inte klarar av att kringgå NAT-filter utan måste förlita sig på att användare har öppnat portar i sin router för att det ska fungera samt att programmets prestanda hindras av källor andra än I/O.

Som det finns beskrivet ovan i kapitel 5.1 begränsas programmets hastighet av kommunikationen med vald drivrutin. Detta kunde ha förhindrats genom att utveckla en egen drivrutin. Det fanns dock inte tillräckligt med tid för det i detta projekt. En annan lösning är att utveckla egen kod i C som sedan kommer att agera mellanhand mellan projektets C#-kod och drivrutinen. Detta skulle gå snabbare då vi skulle ha total kontroll över hur ofta kommunikation mellan C- och C#-kod måste inträffa. Detta har inte heller kunnat utföras i projektet då det inte har funnits nog med tid för att läsa på om C och skriva den nödvändiga koden.

6.2 Användartester

Användartesterna som utförts har givit bra respons för möjliga förbättringar av vissa delar av programmet samt gett en fingervisning om vilka delar som är tillräckligt bra. Testerna har också lett till att en del buggar upptäckts som inte blev upptäckta i den interna testningen.

Responsen om de olika grafiska användargränssnitt som finns har varit varierad men ofta bra. Styckena nedan går igenom den mesta konstruktiva kritiken som har mottagits. Login- och registrerningsfönstret har inte nog med felmeddelanden för olika fel utan har ett och samma felmeddelande för alla fel som kan inträffa. Detta fel är en kvarleva från ett tidigt stadium i projektet och har missats, det bör uppdateras så att bra felmeddelanden kan visas som hjälper användarna. En användare tyckte också att man inte ska behöva logga ut för att skapa en ny användare, men det var inget som ändrades för det beslutades att utloggning ska krävas vid registrering då vi anser att det inte finns någon anledning att registrera en ny användare samtidigt som man är inloggad på en annan användare.

När man markerar en mapp som man har monterat i klientfönstret så bör det i listan där man väljer olika hårddisketiketter visas den mapp där mappen faktiskt är monterad. Bra feedback som när tid ges kommer att implementeras i programmet då detta nu är intuitivt, när något är markerat ska all information som är associerat med det valet visas. Ingen av testarna hade dock några andra problem med att använda klient-användargränssnittet och att ansluta till eller från en mapp och tyckte det var intuitivt vilket är ett bra betyg.
När användarna själv delade ut mappar och tog emot dem tyckte den stora majoriteten att det var intuitivt. Det enda som skulle kunna ändras är att det finns information inkluderat i programmet om att användaren själv måste öppna portar i servern för att få detta att fungera. Det är svårt att kontrollera genom kod att en port faktiskt är öppen då det måste vara någon utomstående som testar att koppla upp sig mot datorn. Denna information kommer istället att inkluderas i en README-fil som inkluderas med programmet.
Över lag så har användartesterna varit bra och lärorika då det har framkommit vilka

idéer som fungerar bra och vilka som inte är helt fulländade samt för att kunna upptäcka och fixa buggar som inte tidigare har hittats.

6.3 Implementationsval

6.3.1 Plattformsval

Projektet utvecklades för Windows. Då det inte ansåg finnas tid för att utveckla till flera plattformar, fick en väljas över de andra. Så vad finns det att säga om Windows?
Den stora fördelen med Windows är den utbredda användningen av det. Detta innebär att det finns en stor potentiell användarbas för produkten som utvecklats. I arbetet har detta också betytt att det varit mycket enkelt att hitta många testpersoner som enkelt har kunnat testa programmet. En annan fördel är att det finns det en uppsjö av hjälpmedel såsom bibliotek och drivrutiner som är anpassade till Windows format.

En nackdel med att utveckla för Windows, och över huvud taget med att utveckla för endast en plattsformstyp, är att arbetet blir begränsat av de ramar som just den plattformen har. För att ta ett konkret exempel så hade det varit möjligt att utnyttja Fuse som drivrutin för att sätta upp filsystemet på klienten om Linux valdes som plattform och Fuse hade kunnat vara ett bättre val som filsystemsdrivrutin. Det hade också behövt läggas ner tid på att lära sig Linux för att använda sig av det vilket hade komplicerat och förlängt utvecklingsfasen ytterligare utan uppenbara fördelar.

Val av plattform är en komplicerad fråga. Vilken plattform som än väljs, på bekostnad av någon annan, gör att man är tvungen att välja bort vissa egenskaper och vissa användare. Det ultimata hade varit att utveckla till alla tillgängliga plattformar, men detta leder i sin tur till mer arbete. Utifrån de förutsättningar och visioner som fanns i början av projektet så anses trots allt valet av plattform ha varit ett korrekt sådant.

6.3.2 Kryptering

Vi valde att kryptera genom en blandning av RSA (publik nyckelkryptering) och AES (sessionsnyckelkryptering). Att använda en blandning av ett publikt nyckelsystem och ett sessionsnyckelsystem är en krypteringsstandard [28].

Först används en så kallad asymmetrisk algoritm för att skapa en privat och en publik nyckel så att både klienter och servrar kan kommunicera med master-servern utan behov av att master-servern skapar fler nycklar. Alla parter kan kryptera meddelanden med den publika nyckeln men bara master-servern kan dekryptera dessa meddelandena, då det endast är master-servern som har tillgång till den privata nyckeln. En nackdel med den asymmetriska krypteringen är dock att den är långsamt vilket löses genom att använda RSA för att komma överens om en symmetrisk nyckel mellan parterna och sedan använda symmetrisk kryptering för all vidare kommunikation. Denna symmetriska nyckel kallas en sessionsnyckel och är unik för varje session. För sessionsnycklar används AES.

6.3.3 Bibliotek

I allmänhet har det bara varit till nytta att använda sig av diverse hjälpbibliotek i arbetet. Undantaget Dokan, som visserligen har varit till stor hjälp men har genererat en del komplikationer, som vi kommer tillbaka till i stycke 6.3.5 nedan. Att ta hjälp av existerande ramverk som utför precis den funktion projektet behöver sparar en massa

tid. Det är förstås möjligt att implementera all kod själv, ifall utvecklarna vill att all kod ska vara egenutvecklad. Men det kan, beroende på vad som krävs, ta mycket resurser. Det är resurser som kan läggas på att utveckla annan funktionalitet som är mer värt att lägga tid på. Ett konkret exempel går att ta från det projekt som dokumenteras i denna rapport. Utan att utnyttja de bibliotek som användes så hade det tidsmässigt inte varit möjligt att hinna med att ens på börja implementeringen av krypteringen.

6.3.4 Språkval

De andra språken som övervägdes innan C# valdes var C och Java. C hade tillfört bättre prestanda till projektet då det som diskuteras i nuläget kräver kommunikation mellan C#- och C-kod som orsakar prestandabesvär. Dock hade det behövts inlärning av C. Att man dessutom själv måste hålla reda på minneshanteringen i koden vilket lätt kan leda till minnesläckor och andra sidoeffekter var också det en nackdel, vilket ledde till att C valdes bort som språk. C# valdes över Java då det är ett modernare språk som också föredrogs då det är bättre anpassat för plattformen som valts att utveckla för.

6.3.5 Filsystem

Användandet av Dokan som filsystemsdrivrutin har medfört vissa problem. Dessa har redan nämnts i 5.1. Ytterligare ett problem med Dokan är att drivrutinens kod är synkron och inte asynkron. Det betyder att varje filsystemsoperation kommer att blockera en exekveringstråd vilket gör att antalet filsystemsoperationer är begränsat till antalet trådar Dokan använder sig av. Eftersom programmet är tänkt att fungera över Internet och Internet medför viss latens så är programmets prestanda begränsad vid höglatent kommunikation. Hade koden varit asynkron skulle Dokans exekveringstrådar kunna fortsätta hantera andra filsystemsoperationer medan programmet väntar på nätverkskommunikation eftersom programmets nätverkskod är asynkron. Men Dokan anses fortfarande vara ett riktigt val eftersom alternativet att skriva en egen drivrutin inte var möjligt med tidsbegränsningen i åtanke.

6.4 Avslutning

Arbetet har lett fram till en fullt fungerande prototyp, som fungerar i stora drag på det sätt som det var tänkt i den primära visionen. I arbetet på väg dit har det gjorts fördjupningar i en rad olika ämnesområden, som säkerhet, nätverkskommunikation, skriva kommunikationsprotokoll och i att skapa filsystem. Prototypen har potential att kunna vidareutvecklas, dels för att kunna uppfylla kravspecifikationen, men också för att få in fler funktioner och för att förbättra prestandan.

Litteraturförteckning

[1] R. Share, ”Just what is SMB,” okänt år. [Online]. Available: http://www.samba.org/cifs/docs/what-is-smb.html. [Använd 17 april 2012].

[2] w. Stallings, ”Transport and tunnel modes,” prentice hall, 2011, p. 644.646.

[3] K. F. kurose och R. Ross, ”Security in computer networks,” i *Computer networking a top down approach*, Boston, Pearson, 2010, pp. 760 - 766.

[4] Dropbox, ”Dropbox,” okänt år. [Online]. Available: https://www.dropbox.com/static/docs/DropboxFactSheet.pdf. [Använd 17 april 2012].

[5] K. F. Kurose och R. W. Ross, ”Protocol layers and their service models,” i *Computer Networking A top down approach*, Fifth edition red., Boston, Pearson, 2010, pp. 76-82.

[6] W. Stallings, ”Overview,” i *Cryptography and Network security*, Prentice Hall, 2011, pp. 40-43.

[7] K. F. Kurose och R. W. Ross, ”Application Layer,” i *Computer Networking a top down approach*, Boston, Pearson, 2010, p. 114.

[8] L. Torvalds, ”Github Linux,” Okänt år. [Online]. Available: https://github.com/torvalds/linux. [Använd 01 05 2012].

[9] Rails, ”Github Rails,” Okänt år. [Online]. Available: https://github.com/rails/rails. [Använd 01 05 2012].

[10] Github, ”Github,” Okänt år. [Online]. Available: www.github.com. [Använd 01 5 2012].

[11] Google, ”Google docs,” Okänt år. [Online]. Available: https://docs.google.com. [Använd 1 5 2012].

[12] JetBrains, ”TeamCity,” Okänt år. [Online]. Available: http://www.jetbrains.com/teamcity. [Använd 05 05 2012].

[13] FUSE, ”Filesystem in Userspace,” okänt år. [Online]. Available: http://fuse.sourceforge.net/. [Använd 29 05 2012].

[14] Nuget, ”Nuget,” Okänt år. [Online]. Available: nuget.org. [Använd 06 05 2012].

[15] Dokan, ”Dokan,” Okänt år. [Online]. Available: http://dokan-dev.net/en/about/. [Använd 1 5 2012].

[16] JUnit, ”JUnit,” Okänt år. [Online]. Available: www.junit.org. [Använd 14 05 2012].

[17] NUnit, ”NUnit,” Okänt år. [Online]. Available: http://www.nunit.org/. [Använd 1 5 2012].

[18] Ninject, ”Ninject,” Okänt år. [Online]. Available: http://www.ninject.org/wiki.html. [Använd 1 5 2012].

[19] Google, ”Google developers,” Okänt år. [Online]. Available: https://developers.google.com/protocol-buffers/docs/overview. [Använd 1 5 2012].

[20] Google, ”Protobuf,” okänt år. [Online]. Available: http://code.google.com/p/protobuf/. [Använd 6 5 2012].

[21] M. Fowler, ”Mode View Controler,” Okänt år. [Online]. Available: http://martinfowler.com/eaaCatalog/modelViewController.html. [Använd 1 5 2012].

[22] M. Fowler, ”Passive View,” Okänt år. [Online]. Available: http://martinfowler.com/eaaDev/PassiveScreen.html. [Använd 1 5 2012].

[23] M. Fowler, ”Inversion of Control,” Okänt år. [Online]. Available: http://martinfowler.com/bliki/InversionOfControl.html. [Använd 1 5 2012].

[24] OODesign, ”Factory Pattern,” Okänt år. [Online]. Available: http://www.oodesign.com/factory-pattern.html. [Använd 1 5 2012].

[25] SQLite, ”SQLite,” Okänt år. [Online]. Available: http://www.sqlite.org/. [Använd 1 5 2012].

[26] RSA laboratories, ”RSA Algorithm,” okänt år. [Online]. Available: http://www.rsa.com/rsalabs/node.asp?id=2146. [Använd 29 05 2012].

[27] Network Sorcery, Inc., ”AES, Advanced Encryption Standard,” [Online]. Available: http://www.networksorcery.com/enp/data/aes.htm.

[28] A. Freier, P. Karlton, Netscape Communications och P. Kocher, ”The Secure Sockets Layer (SSL) Protocol Version 3.0,” 08 2011. [Online]. Available: http://tools.ietf.org/html/rfc6101#page-5. [Använd 29 05 2012].

[29] NIST, ”FIPS PUB 180-3 Secure Hash Standard,” Information technology laboratory, Gaithersburg, MD, 2008.

[30] J. Ullrich, ”Hashing Passwords,” 2012. [Online]. Available: http://www.dshield.org/diary.html?storyid=11110. [Använd 6 5 2012].

Appendix A - Kravspecifikation

Övergripandekrav

Användare skall kunna skapa ett konto och dela ut mappar under detta konto. Ifrån samma dator eller en annan skall denne användare med samma konto kunna dela vissa eller alla av dessa mappar och montera dem som hårddiskar eller mappar. Mapparna skall sedan kunna avmonteras av klienten och kunna sluta delas ut av servern. Läsning av filer skall kunna strömmas från server till klient, skrivning av filer skall kunna strömmas av klient till server. Andra traditionella filsystemskommandon skall kunna utföras av klienten på de utdelade mapparna, så som kopiera, flytta och ta bort filer och mappar, läsa och ändra metadata.

Funktionella krav

Filsystem

- En klient skall kunna ansluta till en server och få tillgång till utdelade mappar. Se protokoll för kommunikation som servern och klienten skall stödja.
- Servern skall ge klienten full tillgång till den utdelade mappstrukturen och dess filer och endast till den utdelade delen.
- Klienten skall ge full tillgång till filsystemet för användaren och applikationer på klientdatorn så att det ska gå att använda på samma sätt som ett traditionellt filsystem på datorn.

Master-server

- Lösenord i databasen skall vara hashade och saltade för att undvika att originallösenorden kan tas fram.
- Master-servern skall tillåta klient att registrera nya användare i dess databas.
- Master-servern skall tillåta klient och server att logga in med korrekta användarnamn och lösenord.
- Master-servern skall förse klienten med en lista över serverns utdelade mappar vid förfrågan.

Klient

- Klienten skall kunna registrera nya användarkonton.
- Klienten skall kräva inloggning för att tillåta användning.
- Klienten skall möjliggöra montering av utdelade mappar som nya filssystem.
- Klienten skall möjliggöra avmontering av montera filsystem och mappar.

Server

- Servern skall med jämna mellanrum informera master-servern om sin IP-adress och sina utdelade mappar så att en klienten alltid skall kunna få information för att kunna anslutna till de utdelade mapparna.
- Servern skall kräva inloggning för att tillåta användning.
- Systemet skall kommunicera enligt protokollspecifikationen i Appendix B.
- Klienten skall autentisera sig mot servern med hjälp av master-servern så att servern aldrig behöver ta del av känslig information.

- Alla komponenter i protokollet bör kunna hantera att andra inte följer protokollet och att oförutsedda nätverksproblem inträffar.
- För att minimera nätverkstrafik skall delar av filer bara strömmas vid behov.

Icke-Funktionella krav

Användarvänlighet

- Det skall finnas olika användarinterface för de olika delarna av programmet, en för servern, en för klienten osv.
- De olika användarinterfacen skall bara innehålla de detaljer som behövs för den önskade funktionaliteten.
- Systemet bör kunna förstås vid första anblick av en person med datorvana.

Kapacitet

- Systemet bör vara tillräckligt snabbt för att inte begränsas av datorkapacitet utan av I/O så som nätverk och hårddisk.

Tillgänglighet

- Systemet skall kunna nås från Internet till alla nätverk utan NAT och alla näverk med NAT där rätt portar är öppna.
- Systemet önskas inte vara begränsat av rätt konfigurerad NAT utan kunna kringgå en NAT.

Dokumentation

- Lösningen skall icke kräva någon omfattande dokumentation utan det är tillräckligt om det finns en enkel README-fil som förklarar funktionaliteten för användare.

Leveransvillkor

- På grund av att detta är ett tidsbegränsat projekt så måste lösningen levereras senast i mitten på maj. Helst som en färdig version men en fungerande prototyp är också godtagbar.

Appendix B – Protokoll

All kommunikation sker över TCP, kommunikationen kan ske över vilken port som helst men standardport är 56123 för master-servern och 56124 för servern.

Protokollet har tre parter. En klient, en server och en master-server. Master-servern används bara för kontoverifaktion och listning av mappar.

Data serialiseras med hjälp av Protocol Buffers.

A Meddelandeformat

<message> ::= <command> <space> <id> [<space> <size>] <lf>
<command> ::= <letter> { <letter> }
<id> ::= <number> { <number> }
<size> ::= <number> { <number> }
<lf> ::= ‘\n’
<letter> ::= ‘a’ … ‘z’ | ‘A’ … ‘Z’
<number> ::= ‘0’ … ‘9’

Textdelen av meddelandet ska kodas i ASCII.

Om <size> är inkluderat i meddelandet följer binär data direkt efter <lf>, serialiserat med hjälp av Protocol Buffers och ska vara <size> bytes långt.

<id> används för att skilja meddelanden från varandra. Ett svar ska alltid svara med samma <id> som det frågande meddelandet. En part borde inte använda samma id för olika meddelanden förutom vid svar. <id> är ett signerat 64-bitars tal representerat i tvåkomplementsform utan ledande nollor. Negativa tal får ej användas.

Övrigt

Om part A upptäcker att den anslutna part B inte följt protokollet ska man svara med ProtocolError och stänga anslutningen.

Auth inom parantes betyder att man måste vara autentiserad för att skicka kommandot.

Kryptering

Varje kommunikationsinstans börjar med att parterna kommer överens om krypteringsnycklar genom de tre olika meddelanden som står nedan. För nyckeln i PUBLICKEY används RSA och sedan för SESSIONKEY så används AES som en sessionsnyckel som sedan krypteras med den publika RSA-nyckeln som har mottagits med PUBLICKEY. Efter att sessionsnyckeln har skickats så kommer all efterföljande kommunikation att krypteras med denna.

REQUESTPUBLICKEY

Klient -> Server,Master-Server
Server -> Master-Server
Svar: PUBLICKEY

PUBLICKEY
Server -> Klient
Master-Server -> Klient,Server
byte[] key
Svar: SESSIONKEY (Skickas krypterat med key)

SESSIONKEY
Klient -> Server,Master-Server
Server -> Master-Server
byte[] key
Efter detta meddelande förtsätter kommunikationen som planerat men exakt all kommunikation är krypterad med key vilket är en unik sessionsnyckel.

Registrering av ny användare
NEWUSER
Klient -> Master-server
string username
string password
Svar: NewUserOk, UsernameTaken
För att registrera en ny användare.

Hjärtslag
HEARTBEAT (AUTH)
Server -> Master-server
string[] directories
Svar: HeartbeatOk
Uppdaterar master-servern med serverns utdelade mappar. Ska skickas minst en gång i timmen. Informationen är giltig i 12 timmar.

Anslutningsregistrering
AUTH
Klient -> Master-server (Server -> Master-server för heartbeats, svar AuthOk)
string version
string username
string password
bool isServer
Svar: UnsupportedVersion, WrongInfo, TOKEN
Används för att autentisera en användare hos master-servern.

TOKEN
Master-server -> Klient
string token
En temporär token som klienten kan använda för att autentisera sig hos servern. En token är bara giltig upp till 1 minut efter att den har skapats.

LISTDIRS (AUTH)
Klient -> Master-server
Svar: LISTEDDIRS
Begär ut utdelade mappar.

LISTEDDIRS
Master-server -> Klient
Dictionary<string, string> directories
Svar med utdelade mappar.

USER
Klient -> Server
string username
string token
Svar: UserOk, WrongToken, UnableToAuth
Autentisering av klienten hos servern.

CHECKUSER
Server -> Master-server
string username
string token
Svar: UserOk, WrongToken
Kontroll av användare när den försöker autentisera sig hos master-servern.

SUBSCRIBE (AUTH)
Klient -> Server
string directoryName
Svar: Subscribed, UnknownDirectory
Registrering av en utdelad mapp. Krävs för att filsystemsoperationer ska kunna utföras mot den. Kan endast göras en gång per anslutning.

UNSUBSCRIBE (AUTH)
Klient -> Server
string directoryName
Svar: Unsubscribed
Avregistrering av en utdelad mapp.

Filsystem
CREATEFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
uint desiredAccess
uint shareMode
uint creationDisposition

uint flagsAndAttributes
Svar: FILESYSTEMRESPONSE

OPENDIRECTORY (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

FILESYSTEMRESPONSE
Server -> Klient
int result
long context

CLEANUP (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

CLOSEFILECALLBACK (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

FILEINFORMATION (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILEINFORMATIONRESPONSE

FILEINFORMATIONRESPONSE
Server -> Klient
int result
long context
uint fileAttributes
int creationTimeLow
int creationTimeHigh
int lastAccessTimeLow
int lastAccessTimeHigh
int lastWriteTimeLow
int lastWriteTimeHigh
uint volumeSerialNumber
uint fileSizeHigh
uint fileSizeLow
uint numberOfLinks
uint fileIndexHigh
uint fileIndexLow

FINDFILES (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FINDFILESRESPONSE

FINDFILESRESPONSE
Server -> Klient
int result
long context
List<FindData> files

READFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
uint bufferLength
long offset
Svar: READFILERESPONSE

READFILERESPONSE
Server -> Klient
int result
long context
byte[] buffer
uint bytesRead

DISKFREESPACE (AUTH)
Klient -> Server
DokanFileInfo info
Svar: DISKFREESPACERESPONSE

DISKFREESPACERESPONSE
Server -> Klient
int result
long context
ulong freeBytesAvailable
ulong totalNumberOfBytes
ulong totalNumberOfFreeBytes

GETFILESECURITY (AUTH)
Klient -> Serer
string filename
DokanFileInfo info
uint requestedInformation
uint securityDescriptorLength
bool securityDescriptorNull
Svar: GETFILESECURITYRESPONSE

GETFILESECURITYRESPONSE
Server -> Klient
int result
long context
SecurityDescriptor securityDescriptor
uint lengthNeeded

CREATEDIRECTORY (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

WRITEFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
byte[] buffer
uint numberOfBytesToWrite
long offset
Svar: WRITEFILERESPONSE

WRITEFILERESPONSE
Server -> Klient
int result
long context
uint numberOfBytesWritten

SETENDOFFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
long byteOffset
Svar: FILESYSTEMRESPONSE

FLUSHFILEBUFFERS (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

SETFILEATTRIBUTES (AUTH)
Klient -> Server
string filename
DokanFileInfo info
uint attributes
Svar: FILESYSTEMRESPONSE

SETFILETIME (AUTH)

Klient -> Server
string filename
DokanFileInfo info
int creationTimeLow
int creationTimeHigh
int lastAccessTimeLow
int lastAccessTimeHigh
int lastWriteTimeLow
int lastWriteTimeHigh
Svar: FILESYSTEMRESPONSE

DELETEFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

DELETEDIRECTORY (AUTH)
Klient -> Server
string filename
DokanFileInfo info
Svar: FILESYSTEMRESPONSE

MOVEFILE (AUTH)
Klient -> Server
string filename
DokanFileInfo info
string newFilename
int replaceIfExsisting
Svar: FILESYSTEMRESPONSE

UNMOUNT (AUTH)
Klient -> Server
DokanFileInfo info

Svar
HeartbeatOk
NewUserOk
UnsupportedVersion
WrongInfo
WrongToken
UserOk
UnknownDirectory
UsernameTaken
ProtocolError
Subscribed
Unsubscribed
AuthOk

Datatyper

DokanFileInfo

long context
ulong dokanContext
long dokanOptions
uint processId
byte isDirectory
byte deleteOnClose
byte pagingIo
byte synchronousIo
byte noCache
byte writeToEndOfFile

FindData

uint fileAttributes
int creationTimeLow
int creationTimeHigh
int lastAccessTimeLow
int lastAccessTimeHigh
int lastWriteTimeLow
int lastWriteTimeHigh
uint fileSizeLow
uint fileSizeHigh
uint reserved0
uint reserved1
string filename
string alternateFilename

SecurityDescriptor

byte revision
byte size
short control
long owner
long group
long sacl
long dacl

Appendix C - Användartest

Frågorna i användarenkäten:

- Did the login work for you? If yes, did you have any problems with the usability of the login interface?
- How well did you manage to navigate to the client after logging in? Could there be any additions that would make it feel more intuitive to get to the client?
- Did it work for you to connect to the pre-shared folder? Did you feel that there can be any additions to make connecting to a selected folder easier?
- Did you manage to add a text file in the specified directory?
- How did it work browsing through the other files in the pre-shared directory? Please provide any feedback you feel is necessary.
- How did it work to register a new user? Do you feel there are any changed that can be made to make the experience more intuitive?
- Did you manage to navigate to the server and share a directory of your choose? Could there be any additions to make this more intuitive?
- Did you manage to connect to the directory that you shared and to browse through it? If so, did it behave as you thought or was there any other functionality that you think you would have benefited from?
- Did you managed to disconnect from and stop sharing the folder or exit the program? Could there be any additions to make this more intuitive?
- Please provide any feedback that you might have about your experience with using the program over the Internet

Dessa instruktioner gavs till de användarna som testade programmet:

Stage 1 - Try to access a directory we are sharing

In the login window of RAccess please log in with the account:

Username: kalle

Password: kalle

After you are logged in please go to the client interface:

You should be able to see a directory we have already shared in the list of directories

under the heading Not Connected.

Please try to connect to this directory

After you have succesfully connected to the directory please open windows explorer and navigate to it.

You will see a directory titled "I did it". Please make a textdocument with your name or a pseudonym and put it in

this folder so we can see how many people have done it.

Also feel free to browse other documents that we have shared in this directory.

Now go back to the clientview and select the folder and press disconnect.

Congratulations, you are now done with stage 1!

Stage 2: Sharing files from your own computer to your own computer.

Yes, this is not actually usefull but will teach you a bit of the programs functionality, you might choose not to do this.

Register your own useraccount.

log in to said useraccount and go to the Server.

Select a directory of your choosing and share it

Go to the Client and refresh the list just to be safe, refresh by pressing

the tiny button with and R next to the connect button

You should now see the directory you shared in the folderlist.

Please connect to this as you did before and just look around in it.

After you feel done with this please disconnect from the directory in the Client and stop

sharing it in the server or simply press Exit to exit the program.

Stage 3:

If you feel that you wish you can also try to share files like in step 2 but over the internet.

For this to work the computer that is sharing the directory will need to allow access to the computer through

port number : 56124

After this there will be no further instructions for this step

Appendix D - Bidragsrapport

Ansvarsområden

Planering
Hela gruppen satte upp planering för projektet tillsammans under gruppmöten tidigt i projektet.

Informationsinhämtning/inläsning
Alla i grupper bekantade sig med C# sedan läste man på om det som ens ansvarsområde inom projektet handlade om. Beskrivs nedan.

Metoder
Valen inom de olika ansvarsområdena var upp till den som var ansvarig för det området. Eric i form av projektledare bestämde de val som påverkade hela utvecklingsprocessen, tex vilka kodramverk som kommer att användas.

Ansvarsområdena var följande:
Eric(ledare):

- Filsystem
- Versionshantering osv
- Nätverk

Alexander:

- Kryptering/säkerhet
- Server

Johan:

- Master-server
- GUI

Evelina:

- Klient

Genomförande
Under själva utvecklingen av produkten jobbade alla i projektet på nästan alla delarna, förutom filsystemet som Eric tog hand om själv. Han var alltså den som enskilt tillförde mest kod.

Bidrag till problemlösning, syntes och analys

På det sättet vi arbetade så löste en person de mindre problemen de stötte på under utvecklingen själva. Större och svårare problem löstes tillsammans under gruppmöten eller via Skype-konversationer.

Det blir därför omöjligt att säga hur mycket varje person faktiskt bidrog till utvecklingen av vår produkt.

Upplägget var likadant under rapport skrivningen, alla i gruppen skrev och hjälpte till på alla de olika kapitlen och delkapitlen.

Huvudansvarig författare för varje avsnitt

Råmaterialet till varje avsnitt skrevs enligt följande

- Sammanfattning/Abstract – Alexander
- Inledning inledande text – Johan
    - Problembeskrivning – Johan och Eric
        - Systembeskrivning – Eric
    - Liknande Lösningar – Alexander
    - Syfte – Evelina
    - Avgränsningar – Johan
- Arbetsmetod inledande text – Johan
    - Delproblem – Johan
    - Versionshantering – Johan
    - Dokumenthantering – Johan
    - Testning – Eric
- Implementering
    - Plattformsval – Johan
    - Språkval – Johan och Eric
    - Bibliotek – Eric
    - Designmönster inledande text – Johan
        - Passive view – Johan
        - Inversion of Control – Eric
    - GUI – Johan
    - Komponenter
        - Master-Server – Johan
        - Server – Alexander
        - Klient – Alexander
        - Filsystem – Eric
    - Programstruktur – Eric
- Funktionalitet
    - Gentemot användaren – Evelina
    - Systemfunktionalitet
        - Säkerhet – Alexander
        - Registrering – Johan
        - Inloggning – Johan

        - Utdelning och Prenumeration på mappar – Alexander
        - Nätverkskommunikation – Eric
        - Filsystem – Eric
        - Resultat – Evelina
- Diskussion
    - Prototyp – Johan
    - Användartester – Johan
    - Implementationsval
        - Plattformsval – Evelina
        - Kryptering – Alexander
        - Bibliotek – Evelina
        - Språkval – Alexander
        - Filsystem – Eric
    - Avslutning – Evelina

Efter råmaterialet till rapporten hade skrivits har hela gruppen gått igenom det i iterationer och förbättrat det tillsammans. På det här viset har alla bidragit till alla delar av rapporten.