Examensarbete

HÖGSKOLAN I SKÖVDE

ENTITETSHANTERING I P2P-NÄTVERK MED VORONOI-TOPOLOGI

ENTITY MANAGEMENT IN P2P NETWORKS WITH VORONOI TOPOLOGY

Examensarbete inom huvudområdet Datalogi
Grundnivå 30 högskolepoäng
Vårtermin 2016

Henrik Smedberg

Handledare: Sanny Syberfeldt
Examinator: Henrik Gustavsson

Sammanfattning

Detta arbete undersöker hur skalbarheten i ett peer-to-peer-nätverk (P2P) byggt med voronoi-based overlay network (VON) som topologi, påverkas av entitetshantering och felhantering, när det används i nätverkslösnigen till ett realtidsspel med många spelare. Under arbetet skapades en testplattform som används under ett experiment för att utvärdera huruvida aspekter såsom ansvarsuppdelning och nodkraschhantering påverkar antalet meddelanden som behöver skickas och därigenom skalbarheten i nätverket. Experimentet undersöker flera fall, med olika mycket betoning på entitetshantering och felhantering och resultaten visar att nätverket behåller sin skalbarhet och att totala antalet meddelanden som skickas håller sig mestadels opåverkad mellan fallen, trots hanteringen.

Nyckelord: Nätverk, voronoi, entitetshantering, P2P.

Innehållsförteckning

1 Introduktion

Distribuerade system har många användningsområden, varav ett är flerspelarnätverksspel (engelska: massive multiplayer online games) eller MMOG. Ett av de största MMOG genom tiderna är World of Warcraft (Blizzard Entertainment, 2004), som tillåter allt från 5000 - 10000 samtidiga spelare i en och samma spelvärld. För att kunna tillåta många samtidiga spelare är det nödvändigt att nätverket bygger på en skalbar lösning.

Den vanligaste nätverksarkitekturen är klient-server, som fungerar på så vis att en server ansvarar för hela nätverket och vet om alla i det, medan klienter är de som bygger upp nätverket, men vet bara om servern. Servern är den som berättar för alla klienter om vad som händer och ansvarar för hela nätverket. Denna vanliga arkitektur, är inte skalbar för växande nätverk, då all belastning ligger på samma centrala punkt. En alternativ arkitektur är peer-to-peer (P2P), där ingen nod i nätverket är central, utan alla noder har samma roll och behandlar varandra lika. P2P är en skalbar arkitektur då all belastning delas upp över alla noder i nätverket.

VON (engelska: Voronoi-based Overlay Network) är en topologi för P2P-nätverk. Topologin efterliknar voronoidiagram och låter enkelt sajter gå med i, lämna och röra sig runt i ett nätverk. Många topologier finns för uppbyggnaden av själva nätverket, men få lösningar finns som beskriver hur entitetshantering bör implementeras i ett P2P MMOG.

En fördel med klient-server över P2P är att servern hanterar alla entiteter centralt och inga svårigheter dyker upp då en spelare lämnar spelet. I en P2P-lösning är det upp till alla noder att tillsammans ansvara för alla entiteter, vilket skapar problem när en nod lämnar nätverket eller kraschar, då den lämnar entiteterna den ansvarade för ägarlösa och det är nödvändigt att välja ut en ny ägare bland de noder som fortfarande behöver känna till entiteten.

Under detta arbete har en testplattform skapats för att kunna utvärdera hur ökad betoning på entitetshantering och felhantering påverkar skalbarheten i ett VON-nätverk. Testplattformen består av ett enkelt spel som innehåller spelare, fiender och fiendeskapare, fler spelare kan enkelt gå med i spelet och under vissa förhållanden kan även nodkrascher simuleras. Största vikten i testplattformen ligger i entitetshanteringen, det vill säga att sköta uppdelningen av ägarskap för entiteter över noderna i nätverket.

Under ett experiment testades flera olika fall i testplattformen med olika betoning på entitetshantering och felhantering. Fallen jämfördes med ett basfall utan någon hantering för att utvärdera ifall antalet meddelanden som behöver skickas påverkar skalbarheten i nätverket. Resultaten visar att alla fall som testades var skalbara jämfört med basfallet, samt att totala antalet meddelanden som skickas inte påverkas mycket, trots entitets- och felhantering.

2 Bakgrund

I detta kapitel presenteras bakgrunden till områden, tekniker och relevanta problem till problemformuleringen. Först presenteras en bakgrund till nätverksspel, samt arkitekturer för uppbyggnad av nätverk, sedan följer en introduktion till några problem som rör området, samt algoritmer och lösningar till de problemen.

2.1 Nätverkspel

Det finns många typer av nätverksspel. I vissa är antalet spelare väldigt litet, som i till exempel det tävlingsinriktade spelet StarCraft II (Blizzard Entertainment, 2010), där det är allra vanligast att man spelar en mot en, i andra kan flera tusen spelare samtidigt ta del av samma spelvärld som i MMOG-spelet World of Warcraft (Blizzard Entertainment, 2004). Oberoende av skala på nätverket, så behöver alla nätverkspel en god nätverkslösning för att upprätthålla spelbarhet och illusionen av att spela tillsammans.

2.2 Klient-server

Enligt Coulouris, Dollimore, Kindberg och Blair (2012) innebär klient-server-arkitekturen (KS) att noder är uppdelade i klienter och servrar, där en klient endast kan kommunicera med en server, medan en server kommunicerar med många klienter. Vanligast med KS är att det endast finns en server som alla klienter kommunicerar med, men en hierarkisk uppbyggnad med flera servrar går också att skapa, där en server till klienter även agerar klient till en annan server.

Klient-server är den absolut vanligaste arkitekturen, både till nätverksspel och till andra distribuerade system, då arkitekturen både är enkel att implementera, samt enkel att administrera. KS har dock många nackdelar, som till exempel att om en server som agerar värd i ett nätverksspel skulle krascha, så skulle hela spelet behöva avbrytas. KS är inte heller skalbart. Då varje klient ökar belastningen för servern så finns det en maxgräns för hur många spelare som kan spela tillsammans.

2.3 Peer-to-peer

Coulouris m.fl. (2012) beskriver arkitekturen peer-to-peer (P2P) som att de inblandade processerna har liknande roller och agerar som jämlikar, istället för att hierarkiskt skilja mellan varandra. Varje nod kommunicerar via samma gränssnitt till alla andra noder. Alltså finns ingen hierarki mellan noderna och det finns ingen server som har ett centralt ansvar.

P2P är en mer ovanlig arkitektur, då den är betydligt svårare att implementera, men P2P är en mer skalbar arkitektur än KS, då det för varje ny nod som är delaktig i nätverket finns en till nod att fördela arbete till. P2P för dock med sig problem som inte finns i KS. Om en klient lämnar nätverket i en KS-lösning så är det mycket enklare att hantera än om en nod skulle lämna en P2P-lösning. P2P har också mycket lägre säkerhet än KS (Lua, Crowcroft och Pias, 2005) vilket leder till att det är mycket enklare att fuska i ett P2P-spel än i ett spel som använder KS.

2.3.1 Distribuerad hashtabell

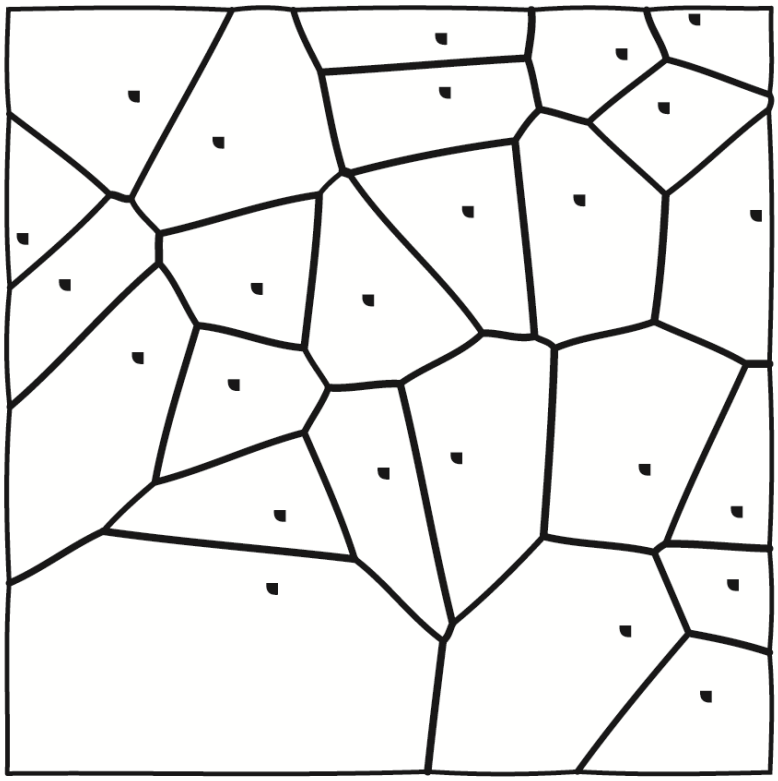
Distribuerade hash-tabeller (DHT) är en topologi för informationshantering i P2P-nätverk. Lua m.fl. (2005) förklarar DHT som att information placeras deterministiskt i nätverket, hos noder som har identifierare som stämmer överrens med informationens unika nyckelvärde. Det går sedan att slå upp information i nätverket, genom att söka på en informationsnyckel, tills noden hos vilken information finns har hittats.

DHT fungerar som ett uppslagsverk för information och dess mest spridda användningsområden är för peer-to-peer-fildelning, där det genom system som BitTorrent (BitTorrent, Inc., 2004), går att sprida och dela filer, skalbart över internet, utan att påverkas av centrala servrars bandbredd eller krascher. DHT är dock inte så användbara för MMOG med realtidskrav, då det kan ta lång tid att slå upp information, vilket skulle orsaka fördröjningar i spelet.

2.3.2 Voronoi-based overlay network

Voronoi-based overlay network (VON) är en topologi för P2P-nätverk som liknar voronoidiagram, Figur 1 visar hur ett voronoidiagram kan se ut. Ett voronoidiagram är uppbyggt av ett visst antal punkter och lika många celler. Linjerna som bildar gränsen för en cell går alltid mittemellan två punkter, så att ingen cell saknar en punkt eller har fler än en punkt. Hu, Chen och Chen (2006) beskriver VON på följande vis: var nod i nätverket representeras av en punkt, eller sajt i voronoidiagrammet och var sajt har också en intressearea (engelska: area of interest), eller AOI, i formen av en cirkel med sajtens position i voronoidiagrammet som centrum. En nod vet om alla andra sajter inom dess AOI, men vet också alltid om alla sina grannar, även om grannsajten befinner sig utanför nodens AOI. Hu, m.fl. (2006) förklarar även procedurerna för att gå med i, förflytta sig i och lämna ett VON-nätverk.

När en sajt förflyttar sig så uppdateras nodens, samt alla dess grannoders topologier för att se till att alla noder delar samma bild av nätverket. En nod vet inte om mer än vad som ryms i dess sajts AOI, så i större nätverk så vet sannolikt en nod inte om alla andra noder i nätverket, utan bara den delmängd sajter som befinner sig närmast. Därför är VON väldigt skalbart, då det i teorin går att skapa oändligt stora nätverk. VON är också väldigt väl anpassade för användning i MMOG av samma anledning. Till skillnad från DHT har en nod dock inget sätt att få reda på information om någonting som ligger utanför dess AOI.



Figur 1 Exempel på ett voronoidiagram

2.4 Hybrid

Målet med peer-to-peer-hybrid-lösningar är att få fördelarna med P2P, medan nackdelarna jämnas ut av fördelarna med KS, genom att både ha klassiska P2P-noder och centrala servrar i samma nätverk, men endast använda servrarna som backuper eller som medlare. Jämfört med en ren P2P-lösning, skulle en hybridlösning till ett MMOG tillåta mer administrering och enklare fuskhantering.

Almashor, Khalil, Tari och Zomaya (2013) föreslår en hybridlösning där topologin liknar voronoidiagram i 3D, där tredje dimensionen används för att representera nodernas nätverkskapacitet och bandredd och noder är uppdelade i svaga noder, supernoder, servrar, etc. där starkare noder kan användas som medlare och som ägare över entiteter. Vilket gör det möjligt att använda några starkare servrar i nätverket och låta dessa vandra runt och hjälpa till med belastningen hos vanliga spelare, vid högbelastade områden i spelet.

2.5 Feldetektering

I ett P2P-nätverk är det nödvändigt att veta vilka noder som finns kvar i nätverket. När en nod kontrollerat lämnar nätverket, kan den tydligt berätta det för alla resterande så att de kan uppdatera topologin och i fortsättningen utgå ifrån att den noden inte finns kvar. Men

vid okontrollerade frånkopplingar, som till exempel vid krascher, är det fortfarande nödvändigt att nätverket vet om att noden inte finns kvar.

2.5.1 Gossip

Gossip är ett vanligt protokoll för feldetektering i distribuerade system. Det finns många olika implementationer av gossip-protokollet, men det kan generellt förklaras som att noder berättar för sina närmaste grannar att de finns kvar i nätverket. Grannarna skickar sedan det vidare till sina grannar och så vidare. Varianter inkluderar till exempel att endast en granne finns som gossip-mottagare för varje nod, eller att en granne slumpmässigt väljs ut som mottagare (Jelasity, Montresor och Babaoglu, 2005). När ett gossip-meddelande inte kommit fram i tid, så kan noden antas ha kraschat.

2.6 Leader election

Coulouris m.fl. (2012) beskriver ett "ledarval" som en algoritm som väljer ut en unik process för att ha en viss roll. Det skulle till exempel kunna vara vid en nodkrasch i ett nätverk, då det också är nödvändigt att utnämna en efterträdare till den kraschade noden. Det finns många algoritmer för ledarval, men två av de mer vanligt förekommande kallas "ring-based election" och "bully".

2.6.1 Ring-based

Enlig Coulouris m.fl. (2012) som refererar till Chang och Roberts (1979), fungerar en ringbaserad ledarvalsalgoritm på så vis att noderna i valet ställer upp sig själva i en ring, där var nod har en granne framåt och en granne bakåt, varje nod har också ett ID-nummer och målet med algoritmen är att identifiera den nod som har högst ID. Algoritmen fungerar på så vis att noden som startade valet börjar med att skicka ett meddelande innehållande dess ID till nästa granne, som jämför meddelandet med sitt eget ID och ifall det är större så skickas meddelandet direkt vidare till nästa granne, men om det är mindre så byts det ut till att innehålla nodens egna ID och det nya meddelandet skickas vidare istället. Ett val är över när en nod fått tillbaka meddelandet den själv sickade, då den noden då måste ha störst ID, noden skickar även slutligen ett meddelande som säger att den blivit vald till alla i valet, så att alla vet att valet är över. Algoritmen har inget stöd för fel, så ifall en nod skulle krascha under valet, skulle valet behöva startas om med resterande noder.

2.6.2 Bully

Enligt Coulouris m.fl. (2012) som refererar till Garcia-Molina (1982), fungerar bully-algoritmen på så vis att alla noder i nätverket har ett ID-nummer och alla noder vet om varandras ID och det är noden med högst ID som skall väljas. När en nod med lägre ID upptäcker en nodkrasch, sickar noden meddelanden till alla noder den vet om med högre ID och väntar på svar, om inget svar kommer inom bestämd tidsgräns så antas att den noden också kraschat. Om en nod dock svarar, så kommer den från och med nu ta över valet. Valet är över när valledaren själv vet att den är noden med högst ID som är kvar i nätverket. Algoritmen tillåter att noder kraschar under valet, dock finns kravet att meddelanden som skickas faktiskt måste komma fram, för att det skall gå att lita på resultatet.

3 Problemformulering

Hu, m.fl. (2006) beskriver hur nätverk uppbyggda med voronoi-based overlay network (VON, se 2.3.2) fungerar, men inte hur en lösning till ett specifikt spel bör eller kan hanteras. Arbetet tar inte upp hur spelentiteter såsom fiender i nätverket bör hanteras och inte heller hur entiteterna bör hanteras om deras ägare lämnar nätverket. Om flera spelare i ett P2P-spel befinner sig nära varandra i spelvärlden och det kommer ickespelarmonster från alla håll, så är det upp till spelarna att tillsammans hålla koll på alla monster. Ansvaret kan inte läggas på en central server, som i ett nätverk byggt på en klient-server-arkitektur. Om en spelare skulle ha ansvar för alla monster så skulle den spelaren kunna få sin nätverkskapacitet överbelastad, vilket skulle orsaka konsekvenser för spelet och de övriga spelarna. Därför är det nödvändigt att dela upp ansvaret för alla monster över alla spelare i spelvärlden.

VAST (VAST Development Team, 2005 - 2016) är ett open-source-ramverk som implementerar VON för uppbyggnaden av nätverk. VAST har dock inte något inbyggt stöd för entitetshantering, vilket är nödvändigt i ett realtidsspel med till exempel fiender som går runt och attackerar spelare, eller projektiler som far genom luften. Därför är det nödvändigt att implementera en lösning till entitetsrelaterade problem, utöver de nätverks- och topologirelaterade problem som löses av existerande nätverkslösningar som VON, vid utveckling av ett spel byggt på en P2P-arkitektur.

Ricci, Genpvali och Guidi (2014) presenterar en hybridlösning där en nod äger alla entiteter inom sin voronoi-cell, medan servrar används som backuper för alla entiteter i spelet. I en sådan lösning vore det nödvändigt för en nod att ge ifrån sig ansvaret av entiteter som inte längre befinner sig i dess cell. I en hybridlösning skulle en server kunna agera medlare i det fallet, men i ett rent P2P-nätverk är lösningen mer komplicerad. En hybridlösning kan förlita sig på servrar som tar över så fort någonting kritiskt, så som överföring av entitetsansvar eller nodkrascher, händer, men i en ren P2P-lösning är det upp till alla relaterade noder att lösa situationen tillsammans och komma fram till en gemensam bästa lösning.

Problemet ligger i att VAST inte har något inbyggt stöd för entitetshantering och frågeställningen är alltså huruvida en distribuerad entitetshanteringslösning påverkar skalbarheten i ett P2P-spel byggt på VAST. Skulle entitetshantering orsaka att så många meddelanden behöver skickas att lösningen inte längre blir skalbar? Hypotesen är att antalet meddelande som behöver skickas ökar, men att komplexiteten över antalet meddelanden förblir densamma. En testplattform har utvecklats som innehåller lösningar på problem som täcker uppdelning av entiteter, samt felhantering och utvärderingar göras som baserat på entitetshanteringen, utvärderar lösningens skalbarhet i flera fall med olika mycket betoning på felhantering.

3.1 Metodbeskrivning

I detta kapitel presenteras den testplattform som utvecklats under arbetet och relevanta problem och lösningar tas upp i stycke 3.1.1. Hur testplattformen och arbetet utvärderas tas upp i stycke 3.1.2. I stycke 3.1.3 så beskrivs avgränsningar som gjorts under utvecklingen av testplattformen och slutligen tas en diskussion om utvärderingsmetoden upp i stycke 3.1.4.

3.1.1 Utveckling

Under utvecklingen har en testplattform skapats som tillåter flera spelare att befinna sig i samma spelvärld, tillsammans med fiender. En P2P-nätverkslösning som hanterar entitetsuppdelning och nodkrascher har implementeras tillsammans med lösningar på relaterade problem, såsom ledarval.

Testplattformen består av ett enklare spel i 2D, vari spelare kan gå runt i världen, samt attackera fiender. Fienderna själva har en enklare AI för att bestämma vilken spelare de kommer röra sig mot. För att generera fiender i spelet finns även en fiendeskaparentitet, denna fiendeskapare har ett beteende som beroende på tid skapar nya fiender inom ett område. Varje ny fiende som skapas kommer bli tilldelad till en nod så att den blir delaktig i nätverksspelet.

Nätverkslösningen implementerades med open-source-programvaran VAST (VAST Development Team, 2005 - 2016), som implementerar VON så som det presenteras av Hu m.fl. (2006). VON har i teorin stöd för nätverk med oändligt många noder, dock begränsar testplattformen antalet noder till 16 i utgångsfallet då det anses vara mer än tillräckligt för att demonstrera lösningen till övertygande grad, då en nod i genomsnitt inte har fler än sex grannar (Aurenhammer, 1991) och 16 samtidiga spelare är ett vanligt antal till en raid i MMOG-spelet Star Wars: The Old Republic (BioWare, 2011). 16 borde också gå att utan problem samtidigt simulera lokalt på en och samma dator. 16 samtidiga spelare ger ett realistiskt exempel på ett scenario som skulle kunna uppstå i ett kommersiellt MMOG-spel.

Nodkrascher är ett väldigt allvarligt problem i P2P-nätverk och första steget till att lösa det, är att identifiera att en nodkrasch har uppstått. VAST har inbyggd feldetektering som använder gossip-principen (se 2.5.1) för en nod att meddela sina grannar att den fortfarande finns i nätverket. Testplattformen förlitar sig på denna implementering för att identifiera nodkrascher.

Entitetsägarbyten sker i tre fall: när en ny entitet skapas i nätverket, när en rörlig entitet lämnar sin nuvarande ägares voronoi-cell och när en nod lämnar nätverket, både planerat, eller oplanerat via en krasch. I alla dessa fall är det nödvändigt att snabbt välja ut en ny ägare, så att spelet kan återupptas så snabbt som möjligt utan att stå still för någon spelare. Testplattformen fokuserar främst på att hantera nodkrascher, men lösningar är naturligtvis även nödvändiga till de andra fallen.

I fallet med rörliga entiteter och när en nod lämnar nätverket kontrollerat, om antagandet görs att ingen nodkrasch kommer ske, vore det tillräckligt att den tidigare ägaren utser nästa ägare med en princip som kan liknas vid och hädanefter kommer kallas *stafettöverföring*. Men en sådan lösning skulle inte vara tillräcklig om möjligheten finns att den nya utvalda ägaren under tiden har kraschat, om ingen felhantering finns på plats. För fallet med en nodkrasch används en lösning som implementerar en bully-algoritm (se 2.6.2), då den tillåter att flera noder kraschat samtidigt. Bully-algoritmen är inte särskilt skalbar, dock har en cell i ett voronoidiagram genomsnittligen endast sex grannar (Aurenhammer, 1991), så algoritmen kommer i de flesta fall ändå gå ganska fort. Bully-algoritmen väljs framför ring-based (se 2.6.1) då den tillåter nodkrascher under körning, vilket ring-based inte gör.

3.1.2 Utvärdering

Arbetet utvärderas via ett experiment (Wohlin, Runeson, Höst, Ohlsson, Regnell och Wesslén, 2012) som mäter huruvida entitetshantering och felhantering kan påverka

skalbarheten i nätverket. Variabeln som undersöks är antalet meddelanden som behöver skickas, under olika fall. Under experimentet undersöks flera fall, som utgångspunkt används ett basfall utan felhantering och ägarutbyte, där ingen nod kommer lämna nätverket och ingen entitet kommer att byta ägare. Detta basfall ger en tydlig bild över hur många meddelanden som genomsnittligt skickas till och från varje nod i nätverket då entiteter endast uppdateras.

Första fallet utöver basfallet använder endast stafettöverföring, alltså kommer entiteter att röra på sig, vilket kommer få dem att byta ägare. Jämförelser med basfallet svarar på hur mycket mer trafik stafettöverföringen tillför. Det andra fallet använder, utöver stafettöverföring även simulerade nodkrascher som hanteras med bully-algoritmen (se 2.6.2), för att svara på hur mycket mer trafik som krävs vid felhantering. Det tredje fallet använder bully-algoritmen för alla ägarbyten, för att simulera ett "värsta möjliga fall" i ett väldigt ostabilt nätverk. Och ett Det fjärde fallet bygger på det tredje fallet med ökande antal spelare. Fjärde fallet använder alltså även det bully-algoritmen för alla ägarbyten, men antalet spelare kommer att öka under körningen. Under alla testfall är det nödvändigt att alla entiteter rör sig och skapas deterministiskt, så att inget resultat förvrängs av annorlunda rörelsemönster eller andra inkonsekvenser.

3.1.3 Avgränsningar

Mauve m.fl. (2004) tar upp Consistency och Correctness, termer som används för att jämföra olika spelares speltillstånd vid samma tidpunkt i samma spelvärld, något som testplattformen inte lägger fokus på att uppfylla. Simuleringen körs dessutom bara lokalt på en och samma dator utan att simulera verklighetstrogna nätverksförhållanden, vilket leder till att problem som packet loss och nätverksfördröjningar inte tas hänsyn till, då i och med att inga meddelanden faktiskt behöver skickas till en annan dator över internet utan direkt till samma maskin som skickade dem, de kommer komma fram på försumbar tid.

Det har heller inte lagts någon fokus på att upprätthålla någon spelkänsla eller över huvud taget göra ett intressant spel till testplattformen. I och med att fördröjningar inte behandlas är det heller inte nödvändigt att lösa problemet med entiteter som inte får uppdateringar under tiden ägarbyte utreds, så en teknik som client-side prediction (Bemir, 2001), som simulerar uppdateringar av entiteter ifall inget uppdateringsmeddelande skulle komma fram på förväntad tid, har inte heller implementeras.

VON (se 2.3.2) har i teorin stöd för oändligt många noder, dock är ett allvarligt problem för topologin "crowding" eller när väldigt många sajter samlas nära samma punkt då detta orsakar ökad nätverkstrafik, då så många fler sajters AOI överlappar. Crowding är ett problem som inte tas upp till testplattformen, men som i ett kommersiellt MMOG byggt på VON vore absolut nödvändigt att hantera.

3.1.4 Metoddiskussion

Syftet med arbetet är att svara på huruvida entitetshantering och felhantering påverkar skalbarheten, i ett i övrigt skalbart nätverk byggt på voronoi-based overlay network (VON, se 2.3.2). Av den anledningen är det nödvändigt att testplattformen tillåter många noder och entiteter att finnas i samma spelvärld, så att en tydlig bild av ett kommersiellt spel kan testas. Det är också nödvändigt att experimentet utförs med samma deterministiska beteende för alla entiteter, för att minska risken för opålitliga mätresultat på grund av inkonsekventa rörelsemönster.

Arbetet utvärderas via ett experiment, då andra metoder, som användarstudier eller fallstudier är mindre lämpade för denna typ av arbete (Wohlin, m.fl., 2012). Det intressanta resultatet i arbetet är antalet meddelanden som behöver skickas. En användarstudie vore mer aktuell ifall det som skulle mätas var en spelares uppfattning om någonting i testplattformen, snarare än kvantitativa mätningar. En fallstudie passar inte heller då det som undersöks är kvantitativa resultat från olika testfall och inte en undersökning av testplattformen, eller området i sig.

De mest intressanta fallen under experimentet är fall 3 och fall 4, då bully-algoritmen är mer komplex än stafettöverföringen och mer drastiskt sätter skalbarheten i testplattformen på prov. Hypotesen är att fall 1 och fall 2 är skalbara då totala antalet meddelande som skickas förväntas öka, men att komplexiteten förblir densamma. Resultaten från fall 3 och fall 4 är svårare att förutspå.

Experimentet sker genom att flera processer, på en och samma dator, kör testplattformen och skickar meddelanden till varandra. Inga meddelanden skickas alltså över internet, då intressant mätdata endast består av antalet meddelanden som sickas. Faktorer som packet-loss eller nätverksfördröjningar är inte intressanta för experimentet. Dock skulle mer verklighetstrogna förhållanden påverka mätresultaten, i och med att fler meddelanden skulle behöva skickas för att garantera att all nödvändig information om entiteterna har kommit fram.

Under utvärderingen har endast de meddelanden som rör entitetshantering och felhantering tagits hänsyn till. Meddelanden som rör upprätthållandet av nätverket och topologin räknas alltså inte med i utvärderingen. Detta är för att nätverksdelen i arbetet implementeras med VAST (VAST Development Team, 2005 - 2016) och det är entitetshanteringens skalbarhet som utvärderas, inte hur effektivt VAST implementerar upprätthållandet av ett VON-nätverk. Detta också för att en nätverkslösning är nödvändig oavsett entitetshantering och olika implementationer kan skilja i effektivitet.

Tillhörande området finns även andra intressanta aspekter att undersöka. Ricci m.fl. (2014) undersöker huruvida storleken på en nods intressearea påverkar belastningen mellan noder och servrar i en hybridlösning baserad på VON. Arbetet utvärderar antalet ägarbyten som måste ske, mellan både nod till nod, samt nod till server, med hopp om att inte överbelasta varken noder eller servrar med fler entiteter än nödvändigt, med varierande storlek på noders intressearea när spelare och entiteter rör sig med varierande höga hastigheter.

4 Implementation

I detta kapitel presenteras testplattformen som har använts till experimentet. Först i kapitlet presenteras innehållet i och implementationen av testplattformen, de olika typerna av entiteter samt deras beteende beskrivs. Senare i kapitlet tas implementationen av djupare tekniska funktioner och algoritmer upp. Sist i kapitlet presenteras en pilotstudie.

4.1 Testplattform

Det grafiska i testplattformen implementerades med SDL2.0 (SDL Community, 1998 - 2016) ett open-source-bibliotek som kan användas för multimediahantering, så som utritning i både 2d och 3d. SDL2.0 valdes till rendering på grund av dess enkla gränssnitt, men då grafik i sig inte är intressant för experimentet, hade det inte spelat någon roll hur det implementerades, eller ifall det implementerades någon grafik alls.

Nätverkslösningen till testplattformen har implementerats med open-source-ramverket VAST (VAST Development Team, 2005 - 2016), som tillåter skapandet och upprätthållandet av ett VON-nätverk (Voronoi-based Overlay Network, Hu, m.fl. 2006, se 2.3.2). Till varje nod i nätverket finns en process och instans av testplattformen och alla processer körs på en och samma dator.

Varje entitet i testplattformen har ett unikt id-nummer, bestående av ett positivt 64-bitars heltal som är en kombination av ett slumptal och nätverksporten som används av den lokala noden. En port är endast kopplad till en nod och i och med det stora antalet värden som ryms inom 64-bitar, så antas varje id-nummer som genereras vara unikt. Id-nummergenereringen har inspirerats av hur VAST genererar unika nod-id-nummer och fungerar på ett snarlikt vis.

VAST genererar id-nummer beroende på nodens ip-adress, dess nätverksport och en id-grupp och en räknare som tillhör den id-gruppen och slår ihop alla värden med logiska skiftningar, samt bitvis "eller" (Patterson och Hennessy. 2013). Testplattformen genererar id-nummer med samma logiska operationer, men genom att istället utgå från ett slumptal, som slumpas fram första gången ett nytt id-nummer skall genereras och sedan ökas med ett för varje nytt id-nummer, samt nätverksporten som används av den lokala noden. Programkoden för hur id-nummer genereras i testplattformen finns i Figur 2.

```
Vast::id_t GetRandomID(Vast::Node *node)
{
    static Vast::id_t UID = (Vast::id_t)rand() << 32;
    return ((Vast::id_t)node->addr.publicIP.port << 16) |
        (UID << 14) | UID++;
}
```

Figur 2 Programkod för id-nummer generering, Vast::id_t är ett positivt 64-bitars heltal.

Varje spelare representerar en nods sajt i topologin, så en spelares position i spelvärlden blir alltså sajtens position i topologin. Ifall en ny händelse, så som en förflyttning, eller attack, inträffar, så skickas ett uppdateringsmeddelande till alla andra sajter inom en sajts intressearea (AOI, Hu, m.fl. 2006, se 2.3.2). Ett spelaruppdateringsmeddelande innehåller

spelarens tillstånd, samt dess position. En spelare kan befinna sig i ett av tre tillstånd: "attacking" när spelaren utför en attack, "moving" när spelaren rör sig mot ett mål och "idle" när spelaren står still utan att attackera. Hur spelartillstånden representeras grafiskt visas i Figur 3, där spelare 1 befinner sig i "idle" och är vit, spelare 2 befinner sig i "moving" och är grön och spelare 3 befinner sig i "attacking" och är blå med en blå ring runt sig. Varje entitet uppdateras 60 gånger i sekunden och ett uppdateringsmeddelande skickas sist i en spelares uppdateringsfunktion ifall en händelse har inträffat under uppdateringen. Varje spelare har även ett unikt id-nummer, samt ett namn, som meddelas nyupptäckta grannar via en handskakningsprocedur.

Handskakningsproceduren som implementerats till testplattformen fungerar på så vis att, när listan över noder i nätverket som kan fås från VAST innehåller en ny nod, så skickas ett handskakningsmeddelande som innehåller den lokala spelarens namn, id-nummer, samt en boolean som avgör ifall det är ett nytt meddelande, eller ett svar som skickas. När ett handskakningsmeddelande tas emot, så sätts den spelarens namn samt id-nummer. Ifall det var den första handskakningen så skickas ett svarsmeddelande. Jämfört med handskakningsproceduren som används i TLS (Dierks och Allen, 1999) är handskakningsproceduren i testplattformen betydligt mycket enklare.

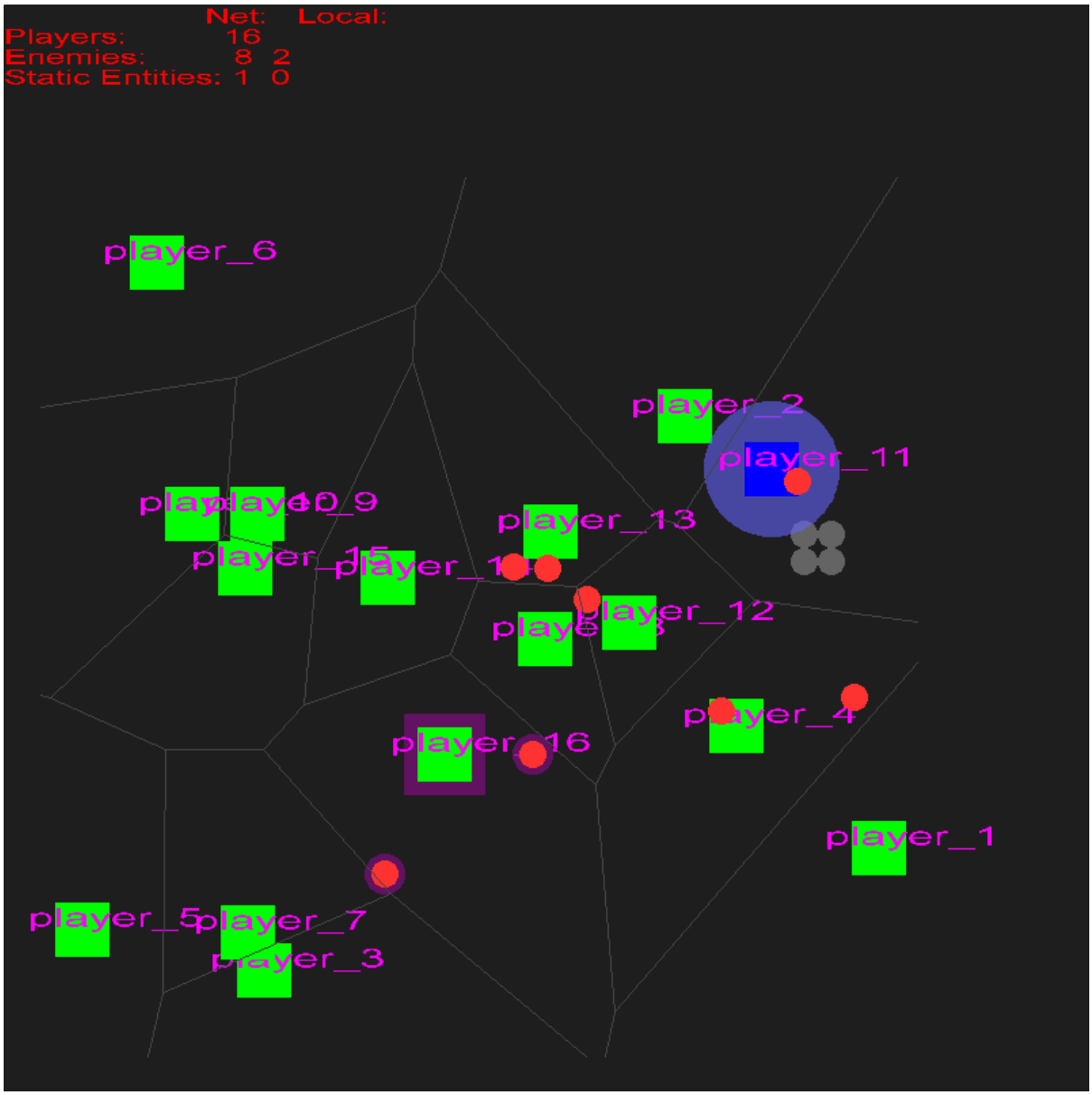
Figur 3 En skärmdump som illustrerar hur spelares tillstånd representeras grafiskt.

Fiender uppdateras bara av den lokala noden som äger dem och deras beteende går ut på att jaga spelare. Ett uppdateringsmeddelande för en fiende innehåller fiendens tillstånd, dess unika id-nummer, samt dess position och skickas efter att fienden har uppdaterats. Ett ägarbytesmeddelande för en fiende innehåller utöver allting i ett vanligt uppdateringsmeddelande, även vilken spelare fienden jagar enligt dess beteende. Detta behöver inte skickas i ett vanligt uppdateringsmeddelande då det bara är intressant för den nod som skall uppdatera fienden. Fiender representeras grafiskt i testplattformen som en röd cirkel, vilket illustreras i Figur 4.

Figur 4 En skärmdump som illustrerar en fiende som jagar en spelare.

Utöver spelare och fiender finns även en statisk fiendeskaparentitet i testplattformen. Denna entitet rör sig aldrig och finns alltid för alla spelare som en del av spelvärden, dock är det fortfarande bara en nod som skall ha ansvar för den. Fiendeskaparen har som beteende att den skapar en ny fiende efter en viss tid. Inga uppdateringsmeddelanden skickas, utan fiendeskaparen uppdateras endast hos den lokala noden som ansvarar för den, tills dess att den skall byta ägare, då ett ägarbytesmeddelande skickas, som innehåller fiendeskaparens unika id-nummer, samt tidräknaren för att veta när nästa fiende skall skapas.

Fiendeskaparentiteten representeras grafiskt som fyra grå cirklar tillsammans. Detta går att se i Figur 5 som illustrerar en skärmdump av hela spelfönstret för testplattformen. Figur 5 visar även hur testplattformen ser ut under körning, med många fiender som jagar många spelare, samt hur lokala entiteter identifieras med en lila bakgrund (se spelare 16). Linjerna i voronoi-topologin för nätverket, samt information om entiteterna i testplattformen ritas också ut.



Figur 5 En skärmdump från testplattformen.

4.1.1 Beteende

Då experimentet består av flera testfall, så är det viktigt att alla entiteter beter och rör sig deterministiskt, så att alla testfall bygger på samma grund och inget oväntat mätresultat kommer från ickekonsekventa beteenden. Av denna anledning genereras pseudoslumptal för alla entiteters beteenden.

Spelarnas beteende är väldigt enkelt, var tredje sekund väljs en ny slumppunkt ut, vilken spelaren rör sig mot med konstant hastighet. Om en fiende befinner sig inom en viss radie från en spelare, så kommer spelaren att attackera. Spelarattacker är dock endast estetiska och har ingen egentlig påverkan, fiender tar ingen faktisk skada. Ett annat spelartillstånd skickas i spelarens uppdateringsmeddelande och den byter färg, men en attack har ingen annan påverkan i testmiljön än så. Spelarens beteende är som det är på grund av att den ursprungliga tanken var att implementera en nätverkslösning som var händelsedriven. Alltså att för varje ny händelse, såsom en attack eller en ny förflyttning, skicka ett händelsemeddelande till nätverket. Implementeringen av en händelsedriven lösning visade sig dock vara mer komplicerad, så valet gjordes att istället gå över till att alltid skicka entiteters hela tillstånd, då detta var enklare att implementera. Att spelare kan attackera är kvar från den händelsedrivna lösningen i vilken det hade påverkat antalet skickade meddelanden men bidrar inte i nuläget till mer än att liva upp visualiseringen av simuleringen.

En fiendes beteende går ut på att jaga den spelare som var närmast fienden när fienden skapades. Detta skall fortgå så länge den spelaren finns kvar i simulationen. Alltså måste vilken spelare som fienden jagar skickas med i ägarbytesmeddelandet för att beteendet skall upprätthållas korrekt. Ägarbytesmeddelandet innehåller den jagade spelarens id-nummer, så att beteendet kan återupptas hos den nya ägaren, då alla id-nummer är lika hos alla noder. Fiender rör sig långsammare än spelare för att garantera att de "halkar efter" så att det finns en chans att orsaka ägarbyte. Det är inte lika intressant för experimentet med fiender som är så snabba att de aldrig behöver byta ägare. Fienders beteende är också utformat för att ge mening till ägarbytesmeddelanden. Om till exempel en fiendes beteende gick ut på att alltid jaga den spelare som befann sig närmast, så skulle det inte behövas ett separat meddelande för ägarbyten då en spelare själv skulle kunna ta över ansvaret över alla fiender som befinner sig på dess voronoi-cell utan att behöva veta någon mer information än vad som fås från ett vanligt uppdateringsmeddelande. Fienders beteenden är också inspirerade av klassiska fiender såsom zombies ifrån spelet Zombies Ate My Neighbors (LucasArts, 1993).

Fiendeskaparentitetens beteende går ut på att vänta tills en viss tid har passerat och sedan skapa en ny fiende, inom ett visst avstånd från skaparen. Inledningsvis ägs den nyskapta fienden av den lokala noden och det är inte förrän under nästkommande uppdatering som rätt ägare bestäms. En nyskapad fiende behandlas alltså precis likadant som en lokal fiende från att den skapas. Att det finns en fiendeskapare alls i testplattformen är för att se till att det tillkommer fler och fler entiteter så att mängden entiteter blir en faktor i utvärderingen av experimentet. Programkoden för fiendeskaparentitetens beteende finns i Figur 6.

```
_spawnCoolDownTime -= delta;
if ((_spawnCoolDownTime <= 0.f &&
    LISTS::_netEnemies.size() < MAX_ENEMY_SPAWN_COUNT))
{
    _spawnCoolDownTime = SPAWN_COOLDOWN_TIME;
    auto newEntity = _spawnFunction();
    auto pos = _position;
    pos.x += rand() % 200 - 100;
    pos.y += rand() % 200 - 100;
    newEntity->SetPosition(pos);
}
```

Figur 6 Programkod för hur fiendeskaparen skapar en ny fiende.

4.2 Nodkrascher

En instans av testplattformen startas med två programstartparametrar. Den första är namnet för spelaren i nätverket och den andra är det slumptalsfrö som används till slumptalsgenereringen för processen. För att kunna utföra alla testfall som skall utvärderas (se 3.1.2) så är det nödvändigt att noder kan krascha. När en ny process startas, så startas även en tidräknare, som baserat på nodens nätverksport genererar en tid tills noden skall krascha. När denna räknare når noll, så avslutas processen, efter att den först skapat en ny process med samma programstartsparametrar som sig själv, med slumptalsfröet ökat med ett för att undvika att samma id-nummer genereras.

Det visade sig också vara nödvändigt att, för att simulera en nodkrasch, först lämna nätverket ordentligt. Att endast stänga processen var inte nog, då VAST väntar tills alla noder är tillbaka i nätverket innan programmet kan fortsätta uppdateras. Resultatet blir alltså att hela nätverket hänger sig om en nod kraschar utan att lämna nätverket ordentligt. För att simulera en krasch måste alltså noden lämna nätverket, innan processen stängs. Alla andra noder upptäcker sedan en nodkrasch genom att notera att antalet noder i VAST-nätverket är färre under nuvarande uppdatering, än under förra. Så ingen skillnad görs på en simulerad nodkrasch och en nod som lämnar nätverket ordentligt, allting hanteras som en krasch.

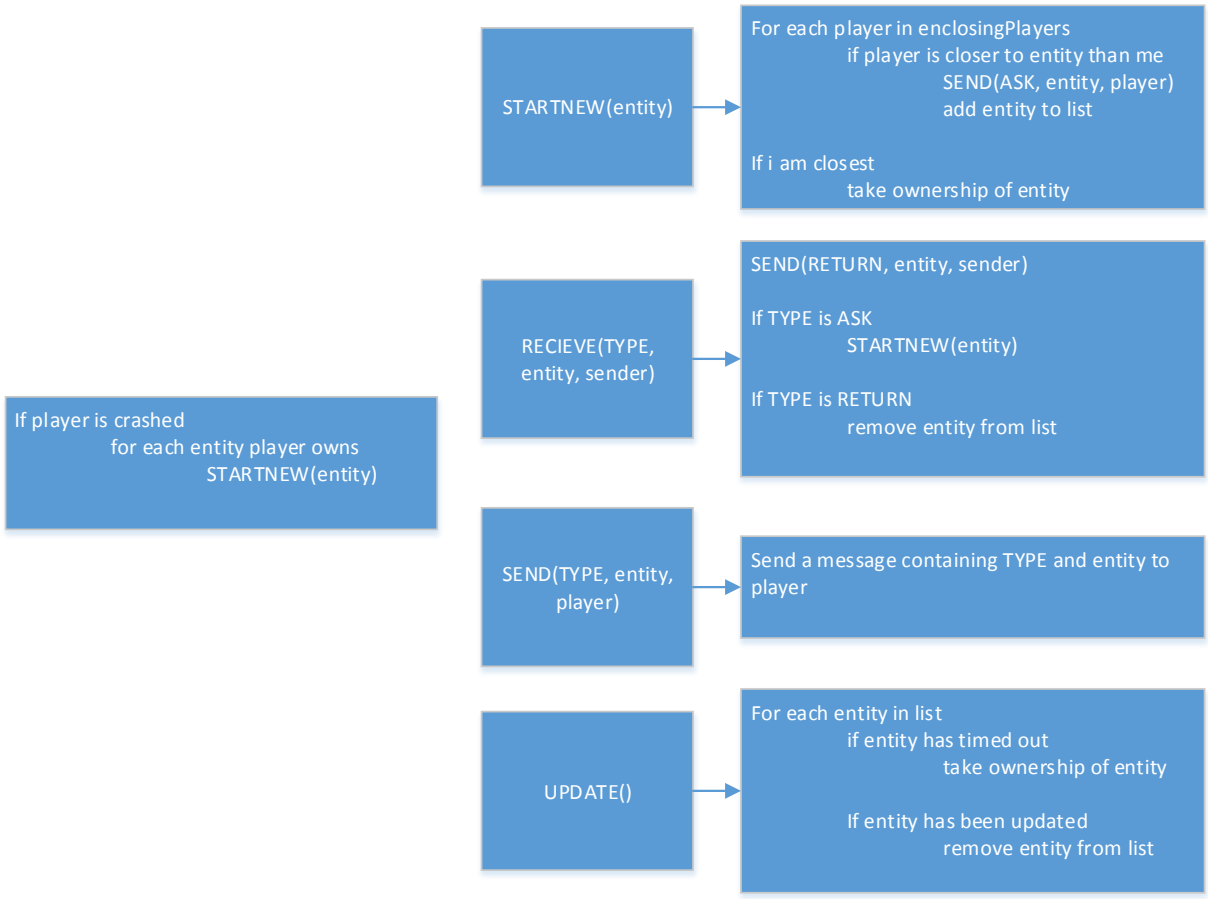
Hur VAST är uppbyggt orsakade också att endast en nod kan vara kraschad i taget. Bully-algoritmen valdes till testplattformen då den tillåter att flera noder kraschar under algoritmen, utan att den behöver startas om, till skillnad från en ringbaserad algoritm. Däremot så upptäcktes under utvecklingen, att VAST påfrestas så hårt av en lämnande nod att hela nätverket överbelastas och drabbas av inkonsekventa fördröjningar och krascher ifall fler än en nod lämnar nätverket samtidigt. Testplattformen kraschar alltså aldrig fler än en nod i taget och får aldrig möjlighet att utnyttja styrkan i bully-algoritmen.

4.2.1 Ledarval

Implementationen av bully-algoritmen (se 2.6.2) som finns i testplattformen är förenklad jämfört med implementationen som beskrivs av Garcia-Molina (1982). I den typiska implementationen från Garcia-Molina (1982) så skickas periodiskt kontrollmeddelanden till andra noder, som måste skicka tillbaka ett svarsmeddelande för att berätta att de finns kvar i nätverket. Implementationen i testplattformen antar att alla noder finns kvar, tills nodlistan som kan fås från VAST har minskat i storlek. Alltså litar testplattformen på att VAST korrekt

kan identifiera nodkrascher. När en nodkrasch väl har upptäckts så börjar ett val på liknande vis som i den typiska implementationen. I Figur 7 beskrivs implementationen i testplattformen med pseudokod. Först identifieras alla entiteter som den kraschade noden var ägare till. Detta bestäms genom att för varje entitet spara vilken nod som senast skickade ett uppdateringsmeddelande om den. Ett val startas sedan för varje entitet, där den lokala noden skickar en förfrågan till alla sina omringade grannar som befinner sig närmare entiteten än vad den själv gör. Id-nummret från den typiska implementationen blir alltså avståndet mellan sajten och entiteten, men istället för att den med högst id-nummer vinner valet, så vinner i slutändan den med lägst.

Ett val forstätter genom att en nod får ett förfrågansmeddelande och direkt skickar ett svarsmeddelande tillbaka till den frågande noden, innan den sedan startar ett nytt val enligt samma procedur. Om ett val skulle startas, men inget svarsmeddelande skulle komma tillbaka, eller om noden vet om att den själv är närmast entiteten, så är valet över och noden tar själv över ansvaret för entiteten. I den typiska implementationen så avslutas ett val med att ett koordineringsmeddelande skickas till alla noder, men i testplattformens implementation, så används ett vanligt uppdateringsmeddelande till koordinering och pågående val hos en nod avslutas när väl ett uppdateringsmeddelande kommer fram för den entitet ett val pågår för.



Figur 7 Pseudokod för bully-algoritmen som implementerades i testplattformen. Algoritmen antar att "enclosingPlayers" är en lista med alla spelare som omringar den lokala spelaren och att "list" är en lista med entiteter.

4.3 Ägarbyten

Ägarbyten sker då en entitet som ägs av en viss nod, inte längre befinner sig inom dess sajts voronoi-cell. Alltså är det önskade resultatet att det alltid är den närmsta sajten som ansvarar för en viss entitet. I stycke 3.1.1 presenteras "stafettöverföring", vilket går ut på att enkelt lämna över ansvaret för entiteter till en annan nod, genom att den tidigare ägaren, som slutar uppdatera entiteten, skickar ett ägarbytesmeddelande till den nya ägaren, som tar över uppdateringen av entiteten. Denna princip används då den kräver att endast ett meddelande behöver skickas och alltså blir den enklast möjliga.

Till de testfall (se 3.1.2) där alla ägarbyten skall ske med bully-algoritmen (se 2.6.2), har bully-algoritmen som implementerades till testplattformen (se 4.2.1) anpassats för att skicka med ett ägarbytesmeddelande i förfrågningsmeddelandet så att ägarskapet kan övertas direkt när den nya ägaren har hittats. Bully-algoritmen för ägarbyten startas också genom att den första noden skickar förfrågningsmeddelanden till alla sina omringande grannar och inte bara till de som är närmare entiteten. Detta är för att det alltid endast kommer finnas en spelare som är närmare, vilket är den som skulle bli den nya ägaren, om den inte har kraschat. Dock är meningen med de aktuella testfallen att simulera ett väldigt ostabilt nätverk där noder kan vara kraschade utan att det har upptäckts lokalt. Därför skickas

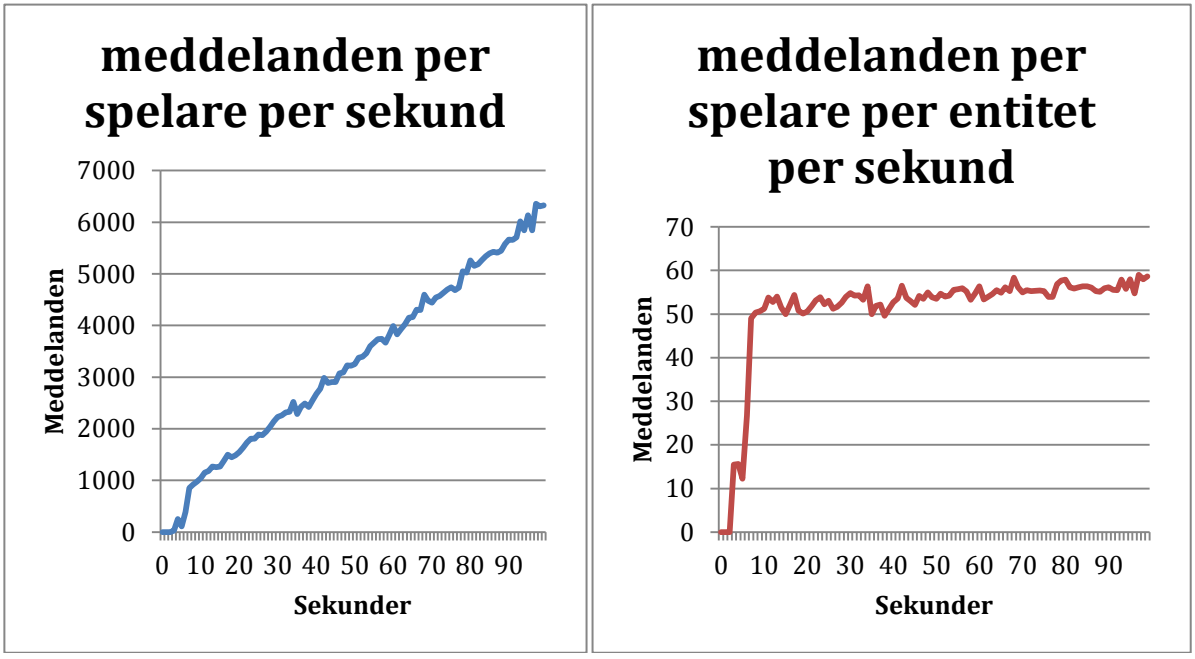
redundanta meddelanden för att försäkra att rätt ägare hittas även ifall många noder skulle ha kraschat under tiden sedan algoritmen påbörjades.

4.4 Pilotstudie

En pilotstudie har genomförts för basfallet och fall 1 (se 3.1.2). Mätningarna som gjordes undersöker antalet meddelanden som skickas per sekund per spelare, samt meddelanden per sekund per entitet i testplattformen per spelare. Testen utfördes under samma förhållanden i testplattformen. Fiendeskaparen skapade en ny fiende en gång per sekund med en gräns på att inte fler än 100 fiender skulle skapas. Det fanns totalt 16 spelare och testen pågick under 100 sekunder under vilka testplattformen uppdaterades 60 gånger per sekund. Varje fall testades endast en gång.

4.4.1 Resultat: basfall

I Figur 8 finns mätresultatet från testningen av basfallet. Det blå diagrammet visar antalet meddelanden som skickades per spelare per sekund. Antalet meddelanden ökar linjärt med antalet entiteter och medelvärdet per sekund var 3134 meddelanden för var spelare, eller 52 meddelanden per uppdatering. Det röda diagrammet visar antalet meddelanden som skickades per spelare per entitet per sekund. Medelvärdet per sekund hamnade på 50 meddelanden per spelare per entitet. Det är värt att notera att antalet växer kraftigt under de första sju sekunderna. Detta är på grund av att nya spelare går med i nätverket.

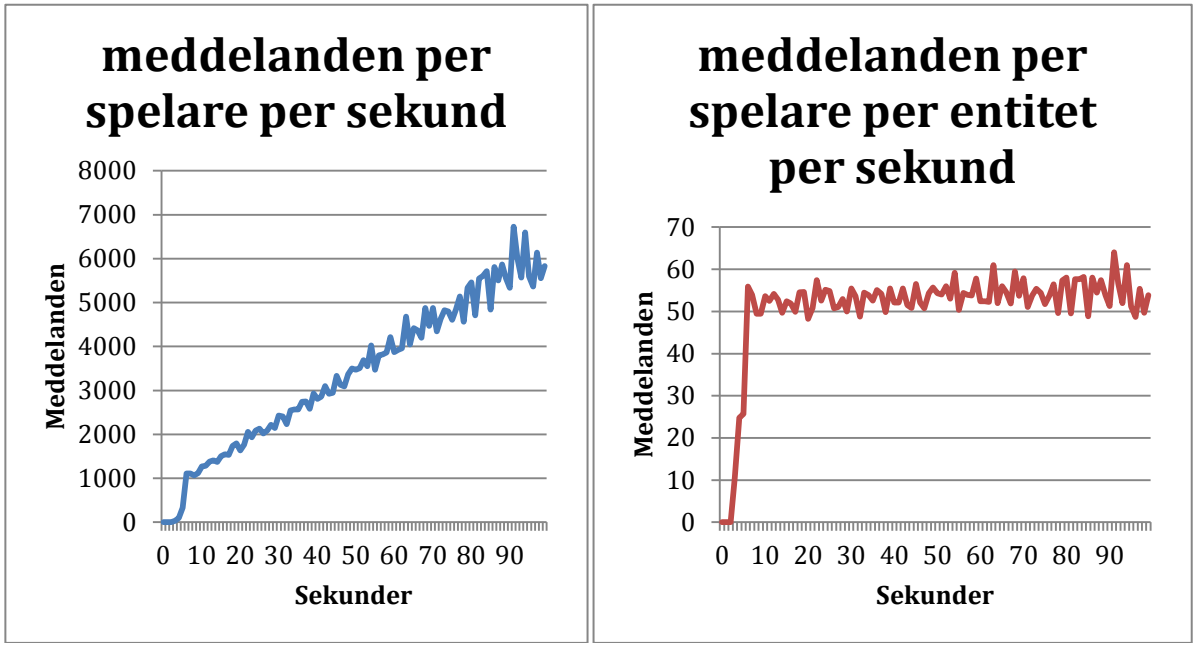


Figur 8 Mätresultat från basfallet

4.4.2 Resultat: fall 1

I Figur 9 finns mätresultatet från testningen av fall 1. Resultatet är väldigt likt resultatet från basfallet. Medelvärdet på meddelanden som skickas per spelare per sekund är 3339, eller 56 meddelanden per uppdatering och alltså bara en aning högre än basfallet, vilket var förväntat. Medelvärdet på meddelanden per spelare per entitet per sekund var 51, vilket jämfört med basfallet även det var förväntat. Fall 1 skalar mycket väl jämfört med basfallet,

då det fortfarande sker en linjär ökning av antalet meddelanden per spelare per sekund trots att antalet meddelanden är något högre.



Figur 9 Mätresultat från fall 1

5 Utvärdering

I detta kapitel presenteras och undersöks experimentet. I stycke 5.1 så presenteras testplattformen till experimentet, samt alla experimentets testfall. I stycke 5.2 så analyseras resultaten från testfallen, senare testfall jämförs med tidigare testfall för att utvärdera deras skalbarhet. I stycke 5.3 så sammanställs slutsatser från resultaten i experimentet.

5.1 Presentation av undersökning

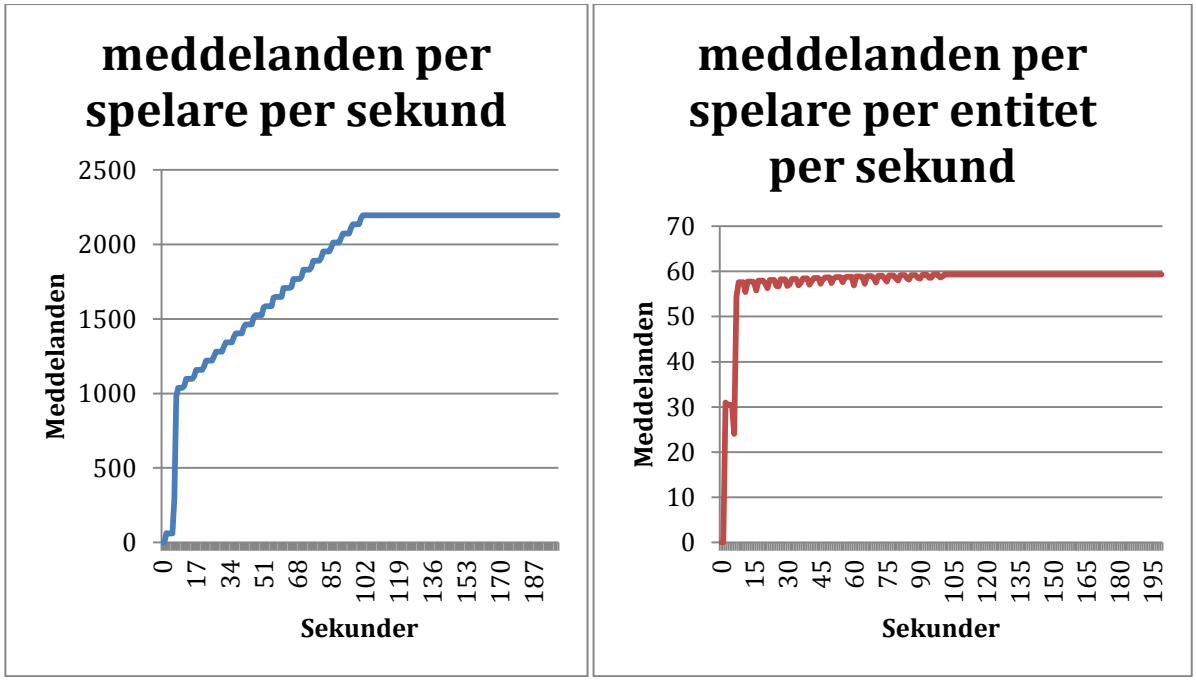
Som grund i alla testfall så börjar testet med att testplattformen har 16 spelare och en fiendeskaparentitet, det skapas sedan en ny fiende var femte sekund upp till totalt 20 fiender. Maxantalet fiender kommer alltså ha nåtts efter 100 sekunder, varefter testet fortsätter tills det pågått i totalt 200 sekunder.

Det finns totalt fem fall i undersökningen (se 3.1.2): ett basfall där det inte sker några ägarbyten eller nodkrascher, utan en och samma nod ansvarar för alla entiteter som skapas. Fall 1, där det sker ägarbyten via stafettöverföring (se 3.1.1), men inga nodkrascher. Fall 2 där ägarbyten sker med stafettöverföring och nodkrascher simuleras och hanteras via en bully-algoritm (se 2.6.2). Fall 3 där både ägarbyten och nodkrascher hanteras med en bully-algoritm. Och slutligen fall 4, som är precis som fall 3, men med flera tillkommande spelare.

5.2 Analys

5.2.1 Basfall

Detta fall agerar som en grund som de övriga fallen jämförs med. Fallet demonstrerar det minsta antalet meddelanden som alltid kommer behöva skickas under en körning. Figur 10 visar resultatet från en testning av basfallet. I det vänstra diagrammet finns antalet skickade meddelanden per spelare per sekund plottat och i det högra finns antalet meddelanden per spelare per entitet per sekund plottat. Basfallet kan jämföras med en klient-server-lösning (se 2.2) då en nod ansvarar för alla entiteter, med skillnaden att alla noder skickar meddelanden som rör sin lokala spelare till de övriga noderna direkt.

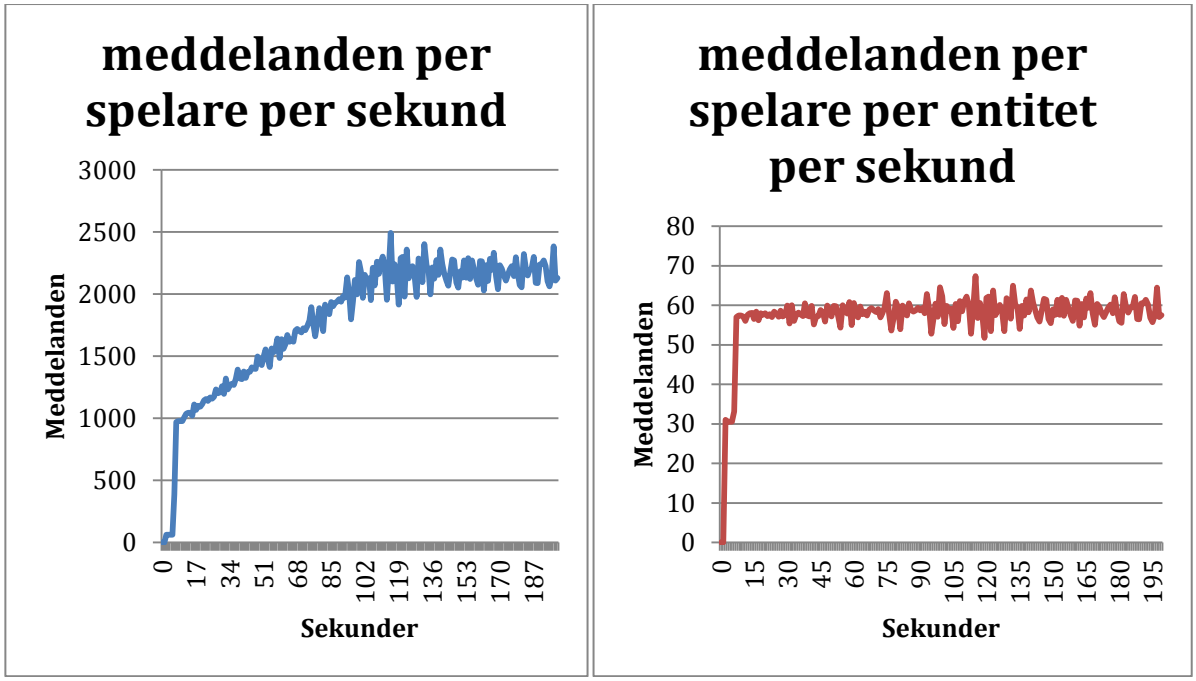


Figur 10 Mätresultat från basfallet

Resultatet blev att antalet meddelanden som skickas växer linjärt med antalet entiteter som skapas under de första 100 sekunderna, medan antalet meddelanden är konstant under den resterande tiden. Testfallet antas vara skalbart då resultatet blev att antalet meddelande växer linjärt med antalet entiteter. Testfallet demonstrerar också det minsta antalet meddelanden som skulle behöva skickas till testplattformen. Även med ägarbyten, så skulle inte fler meddelanden än i basfallet behöva skickas om inga entiteter rörde på sig. Det genomsnittliga antalet meddelanden som skickades per spelare per sekund blev 1837. Det genomsnittliga antalet meddelanden som skickades per spelare per entitet per sekund blev 57.

5.2.2 Fall 1

Detta fall skiljer sig från basfallet i att det sker ägarbyten. En sajt ansvarar endast för de entiteter som befinner sig inom dess voronoi-cell. Om en entitet skulle lämna en sajts cell, så sker ett ägarbyte via stafettöverföring. För varje ägarbyte så skickas alltså endast ett nytt meddelande. Resultatet från testet finns i Figur 11.



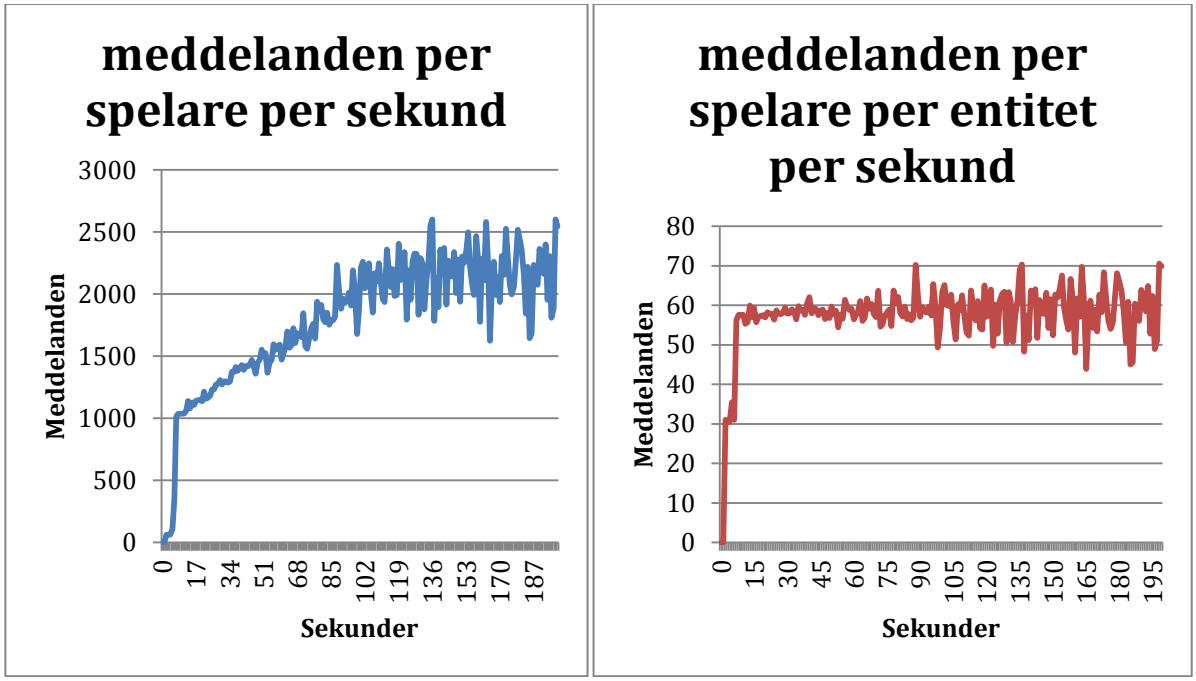
Figur 11 Mätresultat från Fall 1

Jämfört med diagrammet över resultatet från basfallet går det tydligt att se att resultatet från fall 1 följer samma mönster, det vill säga att antalet meddelanden fortfarande ökar linjärt med antalet entiteter, dock fluktuerar antalet meddelanden i viss grad och skiljer sig på vissa ställen en aning från en sekund till nästa. Det genomsnittliga antalet meddelanden som skickades per spelare per sekund uppgick till 1792, vilket är lägre än i basfallet. Anledningen till att detta är möjligt, antas vara att då processer inte är fullt samtidiga och alltså inte uppdateras exakt samtidigt, är det möjligt för ägarbytesmeddelanden som skickas från avsändare under en viss uppdatering *u(n)*, att inte komma fram hos mottagare förrän nästa uppdatering *u(n+1)*, då den inte börjar uppdateras förrän uppdateringen efter det *u(n+2)*. Och alltså inte uppdateras under en uppdatering direkt efter ägarbytet. Spikarna i resultatet (som vid sekund 115) antas bero på att många ägarbyten skedde under den sekunden, till exempel om två sajter rör sig jämte varandra med entiteter precis emellan sig som byter ägare varje uppdatering. Det genomsnittliga antalet meddelanden per spelare per entitet per sekund blev 57, alltså samma som för basfallet.

5.2.3 Fall 2

Detta fall skiljer sig från fall 1 i att det inträffar nodkrascher (se 4.2). För varje nodkrasch så startas bully-algoritmen (se 2.6.2) för alla entiteter som den noden ansvarade över för att bestämma nya ägare för dem. Efter att en nod har kraschat, återansluter den till nätverket. För varje nodkrasch så tillkommer alltså även en ny handskakning. Ägarbyten sker precis som i fall 1.

Ingen nodkrasch sker under de första tolv sekunderna, därefter planeras kontinuerligt nya nodkrascher hända inom intervall om sju sekunder. Det är dock värt att notera att nodkrascher ibland sker på noder som inte är ägare till någon entitet, speciellt tidigt i testet. Resultatet från testet visas i Figur 12.

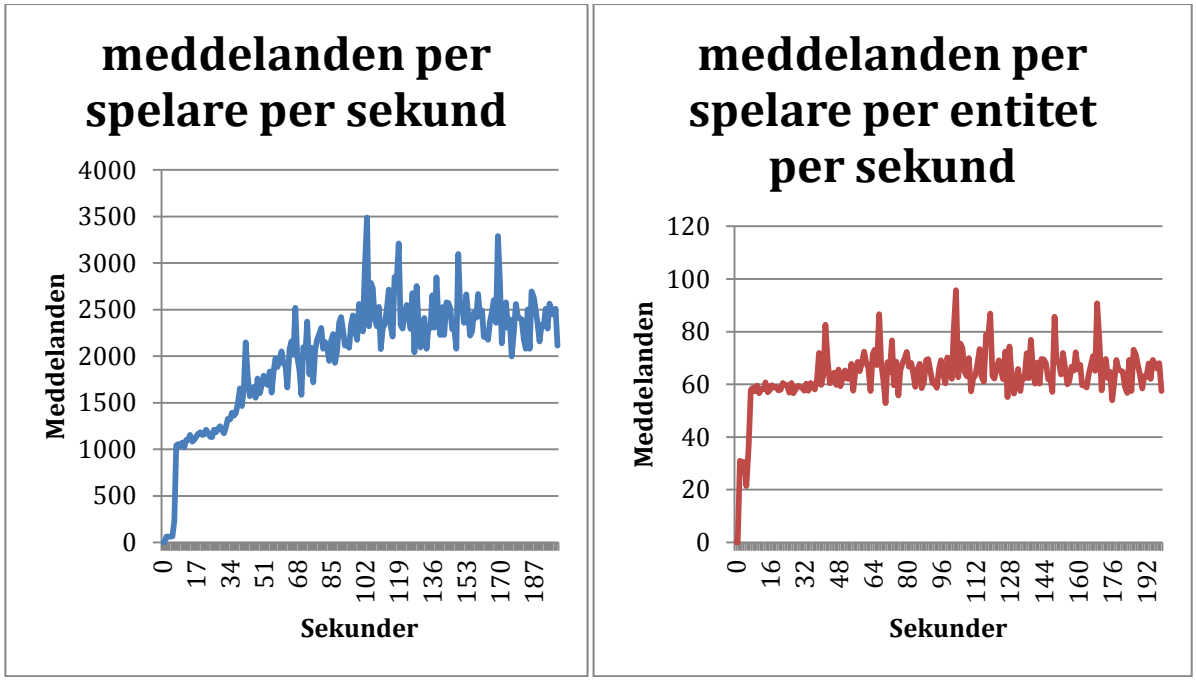


Figur 12 Mätresultat från Fall 2

Det går direkt att se att fluktuationen i antalet meddelanden ökat från fall 2, till en betydligt högre grad i alla fall efter att alla entiteter har skapats. Trots detta blev det genomsnittliga antalet meddelanden per spelare per sekund 1787, alltså till och med aningen lägre än för fall 1. Fler meddelanden behöver skickas under bully-algoritmen än under ett vanligt ägarbyte, dock tar det någon sekund innan en kraschad nod kommit tillbaka in i nätverket igen. Under den tiden så kan inga meddelanden skickas till den noden. Med det i åtanke är det inte omöjligt att det krävs färre meddelanden att slutföra bully-algoritmen, än vad som skulle behöva skickats till en kraschad nod ifall den inte kraschat, under tiden det tar för den att komma tillbaka in i nätverket. Fluktuationen har ökat, men resultatet håller sig fortfarande nära tidigare fall. Det genomsnittliga antalet meddelanden per spelare per entitet per sekund blev 57, alltså samma som för basfallet och fall 1.

5.2.4 Fall 3

I detta fall sköts, utöver endast nodkrascher, även vanliga ägarbyten av bully-algoritmen. Det kommer alltså skickas fler meddelanden vid varje ägarbyte än i tidigare testfall. Simulerade nodkrascher fungerar likadant som i fall 2. Resultatet från testfallet visas i Figur 13.

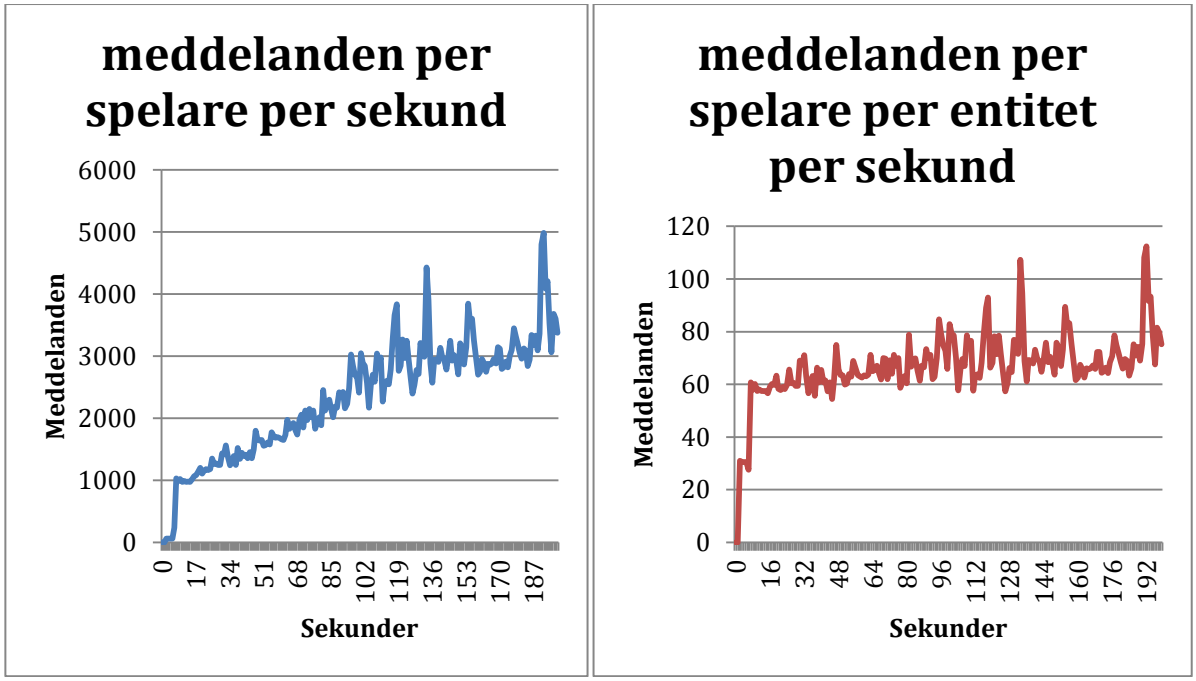


Figur 13 Mätresultat från Fall 3

Testet resulterade i att fluktueringen återigen ökat jämfört med tidigare fall, denna gång dock med betydligt mer märkbara spikar (som vid 103 sekunder). Dessa antas bero på att många ägarbyten skedde samtidigt som en nodkrasch inträffade. Det genomsnittliga antalet meddelanden som skickades per spelare per sekund uppgick till 2020 och det genomsnittliga antalet meddelanden som skickades per spelare per entitet per sekund uppgick till 63, vilket är högre än tidigare fall, dock fortfarande väldigt nära. Antalet meddelanden växer fortfarande också linjärt dock trots tillfälliga spikar. Fluktuationen håller sig på samma nivå som för fall 2.

5.2.5 Fall 4

Detta fall är precis som fall 3, med påbyggnaden att det tillkommer fler och fler noder. Den första nya noden startades 20 sekunder in i testet, därefter startades en ny nod var 25:e sekund. Under testet tillkom det alltså åtta nya noder, när testet var över fanns det alltså totalt 24 noder i nätverket. Resultatet för fall 4 finns i Figur 14.



Figur 14 Mätresultat från Fall 4

I och med att det tillkom fler noder förväntades antalet meddelanden per spelare öka per sekund under hela testet. Det inträffade många kraftiga spikar i resultatet, dock går det fortfarande att se att antalet meddelanden per sekund som skickas ökar linjärt. Det genomsnittliga antalet meddelanden som skickades per spelare per sekund uppgick till 2324 och det genomsnittliga antalet meddelanden per spelare per entitet per sekund uppgick till 67. Resultatet är fortfarande inte långt ifrån tidigare fall vilket talar för att tillkommande spelare inte har jättestor påverkan på antalet meddelanden som behöver skickas, men det för med sig att kurvan inte planar ut efter att det inte längre skapas fler fiender, som i tidigare fall.

5.3 Slutsatser

Enligt hypotesen (se stycke 3), så skulle ökade krav på entitetshantering och felhantering, medföra att fler meddelanden skulle behöva skickas, men att komplexiteten över antalet meddelanden skulle vara densamma. Detta visade sig nästan stämma, komplexiteten är densamma, alltså entitetshanteringen och felhanteringen påverkar inte den linjära ökningen av antalet meddelanden som skickas i samband med ökande antal entiteter. Dock var antalet meddelanden som skickades aningen färre i fall 1 och fall 2, jämfört med basfallet. Fall 3 visade att antalet skickade meddelanden ökar när det finns en orealistisk betoning på felhantering. I fall 4 så ökade också antalet meddelanden från fall 3, detta var dock förväntat i och med att den enda skillnaden mellan fallen är att fler spelare tillkommer i fall 4. Det intressanta i fall 4 är att ökningen av antalet meddelanden som skickades var linjär i samband med antalet spelare.

Trots att alla testfall följde samma mönster som basfallet så blev det mer och mer fluktuering i antalet meddelanden per sekund för varje följande testfall. Fluktueringen håller sig relativt

låg, dock kan tydliga spikar märkas i fall 3 och fall 4. Spikarna är förklarliga och håller sig ganska få, men de skulle eventuellt kunna orsaka överbelastningar i nätverket.

Baserat på att antalet meddelanden som skickades ökar linjärt i samband med antalet entiteter, så antas alla testfall vara skalbara jämfört med basfallet. Antalet meddelanden höll sig nära basfallet, dock ökade fluktueringen för varje följande fall. I de fall där det lades överdriven vikt på felhantering (fall 3 och fall 4) så förekom också tydliga spikar i antalet skickade meddelanden. Dessa var dock få i antal och varade aldrig i flera sekunder.

6 Avslutande diskussion

I detta kapitel finns en diskussion om arbetet, experimentet och området i allmänhet. I stycke 6.1 så sammanfattas hela arbetet, från frågeställning till experiment. I stycke 6.2 presenteras en diskussion om testplattformen, samt etiska och samhälleliga aspekter av arbetet. I stycke 6.3 presenteras en diskussion om relevant framtida arbete.

6.1 Sammanfattning

Frågeställnigen som undersökts i arbetet handlar om hur entitetshantering och felhantering påverkar skalbarheten i peer-to-peer-nätverk byggda med voronoi-topologi. Entitetshantering i detta sammanhang innebär ägaruppdelning av entiteter i en spelvärld och felhantering innebär krascher hos noderna i nätverket, inklusive återinträde i nätverket för en kraschad nod.

För att utvärdera frågeställningen så skapades ett enkelt spel, som består av spelare, fiender och en fiendeskapare, som en testplattform. Då testplattformen bygger på ett peer-to-peer-nätverk, så delas ägandet av entiteterna upp mellan alla noder i nätverket. En nod ansvarar endast för de entiteter som befinner sig i dess voronoi-cell, men då både spelare och fiender rör på sig så kommer entiteter att behöva byta ägare. Det skickas tillståndsmeddelanden om entiteterna till alla noder i nätverket och till felhanteringen används ledarvalsalgoritmen "bully" (se 4.2.1). Som nätverkslösning i testplattformen används open-source-ramverket VAST (VAST Development Team, 2005 - 2016).

Hypotesen var att ökad betoning på entitetshantering och felhantering skulle föra med sig ett ökat antal meddelanden som behöver skickas, men att komplexiteten över antalet meddelanden skulle bevaras. Experimentet utfördes under flera testfall med varierande krav på entitetshantering och felhantering. Resultaten från respektive testfall visar att komplexiteten för antalet meddelanden som skickas bevaras. Dock med ökade krav på entitets- och felhantering så sjönk, under de första två fallen antalet meddelanden som skickades jämfört med basfallet, vilket kan förklaras som att det totalt behöver skickas fler meddelanden under normala tillstånd, än under tiden ägarbyten och nodkrascher hanteras (se, 5.2.2 och 5.2.3). Alla testfall i experimentet antas vara skalbara då antalet meddelanden som skickas ökar linjär med antalet entiteter, dock leder ökad betoning på entitets- och felhantering till ökad fluktuation i antalet meddelanden.

6.2 Diskussion

I arbetet gjordes avgränsningar om att inte ta hänsyn till aspekter som nätverksfördröjningar och tappade meddelanden (engelska: packet loss) då simulationen körs genom flera noder på en och samma dator och meddelanden antas komma fram på försumbar tid. I och med detta så kunde lösningen förenklas då alla meddelanden som skickas, antas komma fram. Ägarbytesmeddelanden kunde till exempel förenklas till endast ett enda meddelande, medan det under mer realistiska förhållanden skulle krävas någon garanti på att meddelandet kommit fram, så som ett svarsmeddelande, så att entiteter inte blir ägarlösa på grund av att ägarbytesmeddelanden inte kommer fram. Det skickas alltså färre meddelanden i testplattformen än vad som skulle behövas ifall verkliga problem hade tagits hänsyn till. Resultaten från experimentet svarar inte på hur resultaten skulle se ut under mer realistiska förhållanden.

En av anledningarna till att peer-to-peer (P2P) inte är en vanligt förekommande nätverksarkitektur i kommersiella spel är för att det kan vara svårt att hentera situationer som att noder går med i eller lämnar nätverket. När nätverket är styrt från en central server så kan noder enkelt gå med och lämna nätverket utan svårighet, då noderna i sig inte bär någon last i nätverket. Men när nätverket är gemensamt styrt av all noder i det, som i P2P, så kan det eventuellt bli problematiskt när en nod som bär mycket last, oplanerat skulle krascha, då det blir upp till de andra noderna i nätverket att gemensamt hantera kraschen. För att hantera sådana situationer är lösningen i testplattformen passande, då den hanterar krascher genom en deterministisk ledarvalsalgoritm som, baserat på samma situation, ger samma resultat varje gång den körs. VON (Voronoi-based overlay network, Hu, m.fl., 2006, se 2.3.2) hanterar lämnande noder väldigt väl och kan enkelt reparera nätverket vid nodkrascher, dock ligger entitetshanteringen utanför VON-nätverket och kräver egen hantering, så att inga entiteter blir ägarlösa.

En annan anledning till att P2P inte ofta används i kommersiella spel är för att P2P-nätverk är svåra, om inte omöjliga att administrera. Detta innebär att ett spel byggt med P2P lätt kan bli utsatt för fuskande spelare, som till exempel använder tredjepartsprogram för att förfalska information om sig själv till de andra spelarna. Fusk är en aspekt som måste tas hänsyn till i kommersiella nätverksspel, då det annars skulle leda till orättvisa villkor för spelare som väljer att inte fuska. Ett exempel på en situation där en spelare skulle vilja fuska kan vara om en spelare tar skada och påstår sig ha mer hälsa än vad den egentligen har. Lösningen i testplattformen gör inget för att bekämpa fusk och fusk är ett svårlöst problem att lösa i P2P i allmänhet, GauthierDickey, Zappala, Lo och Marr (2004) presenterar dock en P2P-arkitektur som förebygger många vanliga fusktyper, samtidigt som den inte ökar belastningen i nätverket.

Testplattformen i detta arbete utvecklades med open-source-ramverket VAST (VAST Development Team, 2005 - 2016) som nätverkslösning, samt använder open-source-bibliotek SDL 2.0 (SDL Community, 1998 - 2016) till rendering. Då både VAST och SDL finns öppet tillgängliga för allmänheten, samt har ett implementeringsgränssnitt i programspråket C/C++, så borde detta arbete kunna återskapas utan vidare problem.

En slutsats som görs från resultaten (se 5.3) är att lösningen är skalbar. Detta innebär att spelutvecklare kan utgå ifrån att det finns skalbara entitets- och felhanteringslösningar att använda till nätverk som använder P2P. Som nämns ovan så finns det vissa svårhanterade situationer när P2P används. En fördel med P2P, som ligger i att det inte behövs någon central server är dock att det inte tillkommer någon kostnad för att upprätthålla nätverket. Ifall ett stort nätverk med klient-server-arkitektur (KS) skall skapas, krävs serverkraft nog att tillfredsställa det begäret. Varje server som behöver användas för med sig en viss kostnad. Så att det finns skalbara lösningar för entitets- och felhantering till P2P, talar för att P2P är ett användbart alternativ till KS, ifall det inte finns tillgångar nog att erbjuda tillräckligt med serverkraft, om en spelutvecklare ändå vill skapa spel med ett stort nätverk.

Resultaten från undersökningen antas vara skalbara då antalet meddelanden som skickades under alla testfall ökade linjärt med antalet entiteter. Dock testades endast ett scenario till varje testfall och det fanns mycket fluktuering och tydliga spikar i resultaten från de senare fallen. Rent forskningsetiskt vore det nödvändigt att testa fler scenarion för att undersöka ifall spikarna kommer ifrån slumpen i testplattformen, eller hur spikarna påverkas av andra beteenden för entiteterna. Det kan också vara intressant att undersöka djupare än antalet

meddelanden genomsnittligt för alla spelare, för att se ifall spikarna kommer från endast en eller ett par noder, eller ifall alla noder påfrestas. Djupare mätningar skulle även kunna svara på ifall spikarna kommer ifrån felaktigheter i implementationen av testplattformen.

Till nätverkslösningen i arbetet användes VAST (VAST Development Team, 2005 - 2016), ett open-source-ramverket som kan användas för att skapa ett VON-nätverk. Som tas upp i stycke 4.2, så var VAST:s implementering direkt ansvarig över vissa designbeslut som gjordes. Hur VAST är implementerat ledde till att flera noder inte kunde krascha samtidigt, vilket i sig leder till att styrkorna i bully-algoritmen inte kommer till användning, samt att ett fall med flera samtidigt kraschade noder inte kunde utvärderas.

6.3 Framtida arbete

För att i mer detalj undersöka skalbarheten i testplattformen, vore det enkelt att utvidga experimentet med flera körningar, där aspekter så som rörelsehastigheten hos både spelare och fiender ändras, samt ge entiteterna andra beteenden än endast de som finns nu, som till exempel att fiender byter vilken spelare de jagar, eller att de inte jagar spelare som många andra fiender jagar. Detta skulle kunna svara på hur andra beteenden kan påverka skalbarheten.

Lösningen i testplattformen som den är nu skulle inte pålitligt gå att använda i ett kommersiellt spel. Ifall lösningen skulle utvecklas vidare skulle de avgränsningar som gjorts under utvecklingen behöva tas hänsyn till. Det behövs felhantering ifall meddelanden om ägarbyten som skickas inte kommer fram och det skulle eventuellt behövas en garanti för att meddelanden hanteras i ordningen de skickades, inte i ordnigen de togs emot, ifall det skulle inträffa fördröjningar för meddelanden. Till exempel får inte ett uppdateringsmeddelande som försenat anländer efter ett ägarbytesmeddelande skriva över uppdateringar som skett sedan ägarbytet.

Lösningen i testplattformen är tillståndsbaserad; vid varje uppdatering skickas entiteternas tillstånd till alla noder i nätverket. Detta leder till att många fler meddelanden skickas än vad som egentligen behövs, då enkla beteenden skulle kunna förutspås på mottagarsidan. En vanlig teknik som tillåter detta är användandet av *client-side-prediction* (Bemir, 2001) tillsammans med *dead-reckoing*. Mauve m.fl (2004) beskriver *dead-reckoing* som att entiteter simuleras både hos ägare och ickeägare, men att meddelanden skickas endast då entiteternas faktiska tillstånd inte stämmer överens med dess simulerade tillstånd. På så vis simuleras entiteterna lokalt hos alla noder, medan endast en nod har det faktiska ansvaret och sköter det faktiska uppdaterandet. Detta sänker antalet meddelanden som behöver skickas, vilket hjälper skalbarheten i nätverket. Det vore enkelt att utöka testplattformen med stöd för *client-side-prediction* och *dead-reckoning*, men det skulle innebära att uppdateringsmeddelanden skulle behöva vara större, då de behöver innehålla information om saker som till exempel rörelseriktning också.

Ett mer visionärt framtida arbete vore att använda VON till, exempelvis ett nätverk med självkörande bilar, där bilarna hjälps åt att hålla reda på fotgängare vid sidan av vägen med hjälp av en entitetsuppdelning. Bilarna kan simulera fotgängarna och försöka förutspå deras beteende, som till exempel att planera att korsa vägen. Bättre förutsägelser kan fås ju mer beräkningskraft som finns att tillgå, i form av bilar som belastningen kan delas upp över.

Referenser

Almashor, M., Khalil, I., Tari, Z., Zomaya, A.Y. (2013). "Automatic and autonomous load management in peer-to-peer virtual environments". *IEEE Journal on Selected Areas in Communications*. 31 (9), 310-324.

Aurenhammer, F., (1991), "Voronoi diagrams-asurvey of a fundamenal geometric data structure", *ACM Computing Surveys (CSUR),* 23(3), 345-405.

Bernier, Y., W., (2001), "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," Tillgänglig på Internet: http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf

Bioware (2011) *Star Wars: The Old Republic* (Version: 1.0) [Datorprogram]. Lucas Arts. Tillgänglig på Internet: http://www.swtor.com/

BitTorrent, Inc. (2004). *BitTorrent* (Version 1.0) [programvara]. Tillgänglig på internet: http://www.bittorrent.com/

Blizzard Entertainment (2004) *World of Warcraft* (Version 1.0) [Datorprogram]. Blizzard Entertainment, Tillgänglig på Internet: http://eu.battle.net/wow/en/

Blizzard Entertainment (2010) *StarCraft II: Wings of Liberty* (Version 1.0) [Datorprogram]. Blizzard Entertainment, Tillgänglig på Internet: http://eu.battle.net/sc2/en/legacy-of-the-void/?-

Chang, E.G., Roberts, R. (1979). "An improved algorithm for decentralized extrema-finding in circular configurations of processors." *Communications of the ACM*, 22(5), 281–283.

Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. (2012). *Distributed systems: concepts and design*. 5. uppl., Boston: Addison-Wesley.

Dierks, T. and Allen, C. (1999). *The TLS Protocol Version 1.0*. Internet RFC 2246. Tillgänglig på internet: https://www.ietf.org/rfc/rfc2246.txt

Garcia-Molina, H. (1982). "Elections in a Distributed Computing System." *IEEE Transactions on Computers*, C-31 (1), 48-59

GauthierDickey, C., Zappala, D., Lo, V., Marr, J. (2004). "Low latency and cheat-proof event ordering for peer-to-peer games." *In Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'04)*. ACM Press, 134–139

Hu, S. Y., Chen, J. F., Chen, T. H. (2006), "VON: A Scalable Peer-to-Peer Network for Virtual Environments," *IEEE Network*, 20 (4), 22-31.

Jelasity, M., Montresor, A., Babaoglu, O. (2005), "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Comupter Systems,* 23(3), 219-252.

LucasArts (1993) *Zombies Ate My Neighbors* (Super NES) [Datorprogram]. Konami.

Lua, K. E., Crowcroft, J., Marcelo, P. (2005), "A servey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE,* 7(2), 72-93.

Mauve, M., Vogel, J., Hilt, V., Effelsberg, W. (2004), "Local-lag and Timewarp: Providing Consistency for Replicated Continuous Applications", *IEEE Transactions on Multimedia*, 6 (1), 47-57.

Patterson, D., Hennessy, J. (2013). *Computer Organization and Design: The Hardware/software Interface,* 5. uppl., San Fransisco: Morgan Kaufmann.

Ricci, L., Geovali, L., Guidi, B. (2014), "Managing Entities in MMOGs: A voronoi-based approch", *Communications in Computer and Information Science,* 456, 58-73.

SDL Community. (1998 - 2016). *SDL* (Version 2.0.4) [programvara]. Tillgänglig på internet: https://www.libsdl.org/download-2.0.php

VAST Development Team. (2005 - 2016). *VAST* (Version 0.4.6) [programvara]. Tillgänglig på internet: http://vast.sourceforge.net/

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A. (2012) Experimentation in Software Engineering. Berlin, Heidelberg : Springer Berlin Heidelberg : Imprint: Springer, 2012