Examensarbete

HÖGSKOLAN I SKÖVDE

VANLIGA PROBLEM OCH LÖSNINGAR FÖR ONLINESPEL

COMMON PROBLEMS AND SOLUTIONS FOR ONLINE GAMES

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 15 högskolepoäng
Vårtermin 2023

Carl-Vilhelm Johansson
Melvin Olsson

Handledare: Andreas Jonasson
Examinator: Mikael Johannesson

Sammanfattning

Denna litteraturanalys går igenom åtgärderna för de problem som förekommer vid skapandet av online pvp-spel. Några av de vanliga problem som kan stötas på innefattar fördröjning, paketförlust, fusk, skalbarhet, synkronisering, uppdateringsfrekvens och säkerhet. Dessa problem kan sedan negativt påverka andra aspekter av spelet, inte minst game feel. För att förhindra detta bör specifika åtgärder implementeras. Inom just online pvp-spel kan grunden till nätverkshanteringen skapas med en klient/server-uppsättning som skickar små och snabba datapaket med hjälp av nätverksprotokollet UDP. Ytterligare lösningar som “dumma klienter”, “client-side prediction”, “server reconciliation”, “interpolation”, historik och tidsstämplar i datapaketen kan användas för att förbättra spelupplevelsen. I kombination med varandra skapar dessa åtgärder en stabil nätverksinfrastruktur för ett online pvp-spel. Då vissa av dessa tekniker kan kräva en del prestanda både på klienten och servern kan framtida arbeten kring dessa lösningar fokusera på optimering.

Nyckelord: nätverkshantering, pvp-spel, onlinespel, klient, server

Innehållsförteckning

1 Introduktion

Onlinespel är väldigt populärt idag och den totala mängden pengar spenderat på onlinespel 2020 låg på 36,9 miljarder dollar (Kahraman & Kazançoğlu, 2022). Därför är det förståeligt om många spelutvecklare är intresserade av att skapa onlinespel. Det finns dock ett antal problem som spelutvecklare kan stöta på under utvecklingen. Några av de vanligaste problemen som kan uppstå är fördröjning, paketförlust, fusk, skalbarhet, synkronisering, uppdateringsfrekvens och säkerhet. Dessa problem kan sedan påverka spelupplevelsen på ett negativt sätt. För att undvika dessa problem bör vissa åtgärder tas. Eftersom dessa åtgärder skiljer sig bland spelgenrer väljs de lösningar som passar bäst specifikt för *pvp-spel*. I denna text hänvisar "pvp-spel" till onlinespel där spelaren har direkt kontroll över en karaktär i realtid (alltså inte turordningsbaserat) och tävlar mot andra spelare om att vinna. Detta är på grund av att nätverkshanteringen i just denna spelgenre har som störst påverkan på spelupplevelsen. Eftersom dessa pvp-spel innehåller realtidskontroll, simulerad rymd och polish har den också alla byggnadsblocken för game feel (Swink, 2009).

Denna litteraturanalys har syftet att redovisa vilka åtgärder som är bäst anpassade för några av de grundläggande problem som kan uppstå vid implementationen av nätverkshantering i pvp-spel. Denna metod används för att förbättra förståelsen och sammanfatta faktan som funnits sedan tidigare samt att generalisera resultaten till en större del av populationen (Berndtsson m.fl., 2008). Åtgärder söks och jämförs sedan med andra möjliga åtgärder som löser samma problem på olika sätt. Innovativa tekniker som "dumma klienter", "client-side prediction", "server reconciliation", "interpolation", historik och tidsstämplar i datapaketen analyseras och värderas om de passar till pvp-spel. På grund av att vissa av lösningarna även skapar nya problem är några av de andra lösningarna enbart åtgärder på de problemen som ytterligare har skapats.

2 Bakgrund

Onlinespel är väldigt populärt idag och den totala mängden pengar spenderat på onlinespel 2020 var 36,9 miljarder dollar (Kahraman & Kazançoğlu, 2022). Därför är det förståeligt om många spelutvecklare är intresserade av att skapa onlinespel. Första steget till att skapa ett bra onlinespel är att förstå hur uppbyggnaden av nätverket ser ut.

2.1.1 Nätverksprotokoll

Det finns två olika vanliga protokoll som används för att skicka datapaket över internet (Ali, Nasir & Qazi, 2013). När TCP, eller ”transmission control protocol”, används för att skicka ett datapaket och paketet av någon anledning inte kommer fram försöker den igen flera gånger (Fiedler, 2008). Detta fungerar med hjälp av en “handshake” mellan klienten och servern, där den ena först skickar en förfrågan till den andra för att etablera en anslutning innan data skickas över (Nasir & Qazi, 2013). Detta leder till ökad fördröjning, vilket kan ökas ytterligare om datapaketet förloras flera gånger i rad. Paket skickade med TCP kommer också fram i rätt ordning, vilket innebär att om första paketet förloras och blir försenat, kommer nästa paket också bli det. På grund av åtgärderna som TCP tar för att garantera att datapaket kommer fram används även relativt mycket data (Fiedler, 2008).

UDP, eller ”user datagram protocol”, skickar istället datapaketen utan att bry sig ifall de kommer fram (Ali, Nasir & Qazi, 2013). Om ett meddelande förloras någonstans på vägen är det helt enkelt borta och varken sändaren eller mottagaren får någon information om att det är borttappat. Ordningen som datapaket skickade med UDP anländer är slumpat, alltså att paket som skickas senare kan komma fram snabbare än de som skickats tidigare (Nasir & Qazi, 2013). I det bästa fallet för båda protokollen skickas UDP snabbare och kräver betydligt mindre resurser än TCP, och i värsta fall skickas TCP för sent, medans UDP inte skickas alls (Ali, Nasir & Qazi, 2013).

2.1.2 Nätverksuppsättningar

Det finns två vanliga sätt att ansluta klienter till varandra: “peer-to-peer” och “klient/server” (Chernyak, 2014). Klient (eng. “client”) hänvisar i denna text till spelapplikationen som körs på spelarens hårdvara. I en *peer-to-peer*-uppsättning ansluter klienterna och skickar datapaket direkt till varandra. I en *klient/server*-uppsättning ansluter klienterna och skickar datapaket till en server som sedan skickar datapaket tillbaka till klienterna. Nedan antecknas några av för- och nackdelarna med respektive uppsättning enligt Chernyak (2014).

Peer-to-peer:

- **Fördelar**:
  - Spelföretag behöver inte betala för servrar.
  - Mindre fördröjning ifall klienterna är geografiskt nära varandra.
  - Spelare kan fortfarande spela online med varandra även om spelutvecklaren överger spelet.
  - Så länge klienterna är geografiskt nära varandra kan en klient ansluta sig från var som helst.
- **Nackdelar:**
  - Svårt att hantera större antal spelare
  - Alla data hanteras lokalt hos klienterna, vilket ökar möjligheterna till fusk.

Klient/server:

- **Fördelar:**
  - En klient med opålitlig internetåtkomst påverkar andra klienter mindre.

    - En server som kontrolleras av ett spelföretag kan upptäcka ifall en klient försöker fuska eller störa nätverket genom att skicka otillåtna datapaket och därmed stoppa den.
    - Kan hantera en större mängd klienter.
- **Nackdelar:**
    - Servern måste vara uppsatt på en geografisk plats, vilket gör att klienter som befinner sig långt bort får sämre uppkoppling.
    - Om servern slutar fungera kommer även stora delar av spelet göra det.
    - Företaget måste betala för att hålla uppe servern.

Efter att ha satt upp nätverket med hjälp av peer-to-peer eller klient/server och använt tidigare nämnda nätverksprotokoll har en spelutvecklare tekniskt sätt grunderna för att ett onlinespel ska fungera över ett nätverk. Att enbart programmera så att datan skickas och tas emot är dock inte tillräckligt för ett färdigt spel. Det finns flera problem som en spelutvecklare som är van vid att skapa singleplayerspel kan stöta på, som sedan kan påverka andra viktiga aspekter av spelet, inte minst *game feel*. För att förstå dessa problem bör en spelutvecklare först få en förståelse om vad game feel innebär.

2.1.3 Game feel

Game feel har ingen standardiserad definition, men Swink (2009) skriver att det kan förklaras genom tre byggnadsblock; realtidskontroll, simulerad rymd och polish, och beskriver dem på följande vis: Realtidskontroll är en form av interaktion, där spelaren skickar sina inmatningar till datorn, som då räknar ut och visar resultatet på skärmen för att låta spelaren tänka ut och skicka nya inmatningar. Blir fördröjningen mellan att spelaren utför inmatningarna och att resultaten visas på skärmen över 150 millisekunder kan spelet upplevas som trögt. Simulerad rymd hänvisar till de fysiska interaktionerna som uppstår i den virtuella rymden, genom kollision och respons mellan spelarkaraktären och andra objekt i spelvärlden. Sist men inte minst är polish de effekter som läggs till för att stärka dessa interaktionerna utan att ändra på den underliggande simulationen. Ett exempel på en genre av onlinespel vars nätverkshantering har stor påverkan på dessa byggnadsblock är pvp-spel.

2.1.4 Pvp-spel med realtidskontroll

I denna text hänvisar “pvp-spel” till onlinespel där spelaren har direkt kontroll över en karaktär i realtid (alltså inte turordningsbaserat) och tävlar mot andra spelare om att vinna. Inom pvp-spel kan snabba beslut vara kritiska för spelarens framgång och det är upp till nätverkshanteringen att förmedla dessa snabba beslut till de andra klienterna (Levkoff, 2014). Det är även viktigt att pvp-spel är så rättvisa som möjligt, spelare bör inte förlora på grund av att deras hårdvara eller internetåtkomst är sämre än andra spelare. Fusksäkerhet är också viktigt då en enda fuskare kan förstöra spelupplevelsen för alla andra spelare i matchen (Levkoff, 2014). Nätverkshanteringen kan ha stor påverkan på flera av de saker som tagits upp och det är därmed viktigt att implementera den med allt detta i åtanke. Denna text kommer fokusera på just skjutarspel även om flera av lösningarna kan vara relevanta för andra typer av pvp-spel.

2.2 Vanliga problem

För att kunna skapa ett onlinespel utan att nätverket påverkar spelarens upplevelse är det viktigt att förstå de problem som kan uppstå. Nedan förklaras några vanliga problem som kan uppstå vid just skapandet av onlinespel.

2.2.1 Fördröjning

Ett av de största problemen som förekommer i alla onlinespel är fördröjning (eng. “latency”). Med detta menas tiden det tar för data att överföras från en punkt till en annan över ett nätverk (Zhao, 2021, Brun, Safaei & Boustead, 2006). Just inom onlinespel används ofta termen “ping" som betyder tiden det tar för ett datapaket att åka från klienten till servern och sedan tillbaka till klienten. I ett onlinespel kan exempelvis en klient skicka att spelaren vill röra sig men om den har lång fördröjning till servern kan det ta flera hundra millisekunder tills den får göra det. Detta är ett problem som ger en negativ påverkan på realtidskontroll, som tidigare nämnts är en av de tre byggnadsblock för game feel (Swink, 2009, Zhao, 2021). Det finns ett antal faktorer som kan påverka fördröjningen inom onlinespel, några exempel beskrivs nedan.

Distans:

Om klienterna (och servern om klient/server-uppsättningen används) som måste skicka datapaket mellan varandra inte befinner sig på exakt samma plats kan datapaketen behöva färdas en bit vilket oundvikligt tar tid (Brun, Safaei & Boustead, 2006, Spacey, 2018).

Överbelastning hos klienten:

Det finns en gräns för hur mycket data som kan hanteras åt gången genom till exempel spelarens router och om denna gräns överskrids kan det bidra till ökad fördröjning (Liu m.fl., 2022).

Överbelastning hos servern:

Även hos servern finns det en gräns för hur mycket data som kan hanteras och bidrar också till ökad fördröjning om den överskrids (Liu m.fl., 2022).

2.2.2 Paketförlust

Paketförlust (eng. “packet loss”) inom onlinespel hänvisar till en situation där vissa av de datapaket som skickas över nätverket inte når sin destination (Spacey, 2018). Detta kan förekomma på grund av en rad olika anledningar som exempelvis glappande anslutningar, överbelastad server, överbelastad klient eller överbelastad internetanslutning (Pandora FMS team, 2022). Paketförlust kan ha en betydlig påverkan på en spelares upplevelse, det påverkar dock på olika sätt beroende på om ett datapaket förlorades på väg till servern eller klienten.

Med nätverksmodellen klient/server skickar servern datapaket som innehåller information om hur spelvärlden ser ut för servern (Brun, Safaei & Boustead, 2006, Liu m.fl., 2022). Ifall ett datapaket som är på väg till klienten från servern förloras är det inte ett speciellt stort problem eftersom en nyare version av hur spelvärlden ser ut för servern skickas bara några millisekunder senare. För en spelare kan detta exempelvis upplevas som att världen “hackar till lite”.

2.2.3 Fusk

Med fusk inom onlinespel menas användningen av metoder eller verktyg för att få ett orättvist övertag över andra spelare (Parizi, 2019). Fusk kan ta flera olika former, till exempel kan spelare utnyttja buggar som redan finns i spelet, automatisera sina inmatningar genom tredjepartsprogram, ändra i spelets kod, redigera spelets variabler eller till och med manipulera sin egen nätverksåtkomst och skicka modifierade datapaket. Inom onlinespel går det sällan att lita på att spelarna kommer att bete sig så som spelutvecklaren hade önskat, speciellt inte inom pvp-spel där det finns större motivation att vinna över andra spelare (Gambetta, 2020).

En av de vanligaste formerna av att fuska i spel är att använda sig av “hacks” eller “mods” som tillåter spelaren att få kontroll över funktioner eller attacker som inte är tillgängliga för andra spelare. Nedan är några exempel på vanliga fusk som använder mjukvara för att manipulera spelet på olika sätt i onlinespel.

- "Aimbots" kan få spelarkaraktären att automatiskt sikta mot fiender inom skjutarspel (Blaukovitsch, 2022).
- "Wallhacks" ritar någon form av markering där spelarens motståndare befinner sig över spelets bildruta, även när motståndarna exempelvis befinner sig bakom en vägg (Blaukovitsch, 2022).
- "Triggerbots" skjuter automatiskt när spelaren siktar på en motståndare (Blaukovitsch, 2022).

En annan form av fusk är "scripting", där spelaren använder programmerad kod för att automatisera funktioner som att sikta, skjuta eller röra sig (Parizi, 2019). Vissa använder sig även av "bots", som är automatiserade program som tar över hela spelet och spelar istället för spelaren.

Som tidigare nämnts skickas datapaket inom peer-to-peer från klienterna direkt till varandra och leder därmed till lättare användning av fusk. Ett exempel på detta i ett onlinespel är om spelaren försöker skicka falsk data (exempelvis spelarkaraktärens position).

Användningen av fusk kan inte enbart förstöra de andra spelarnas kortsiktiga spelupplevelse utan även deras långsiktiga syn på spelet om de möter fuskare för ofta. Detta kan i sin tur minska antalet personer som är villiga att spela spelet vilket kan leda till minskning av inkomst för spelutvecklarna (Blaukovitsch, 2022).

2.2.4 Skalbarhet

Förmågan hos onlinespelets nätverksinfrastruktur att hantera ett ökande antal spelare eller handlingar utan en märkbar påverkan på prestanda benämns skalbarhet (Ferretti & D'Angelo, 2018). Servern måste ge tillräckligt med kapacitet för att hantera all data och även skala med antalet spelare och samtidigt inte slösa kraft vid tillfällen då få spelare spelar (ibid.).

När antalet klienter anslutna till en server ökar, kommer mängden data som måste överföras också öka, vilket kan belasta både servern och spelarnas nätverk. Detta kan leda till bland annat tidigare nämnda problem som fördröjningar och paketförlust eller i värsta fall att servern kraschar.

2.2.5 Synkronisering

Med synkronisering menas att ett onlinespel försöker se till att alla klienter och servern ser samma *speltillstånd* (eng. "game state") vilket är viktigt för att spelet ska hållas rättvist och för att undvika frustration hos spelare (Liu m.fl., 2022). Det finns ofta många delar av ett speltillstånd som behöver synkroniseras, till exempel spelarkaraktärernas positioner, animationer och resurser såväl som alla interagerbara objekts egenskaper.

Att synkronisera många spelarkaraktärer och interagerbara objekt kräver att stora mängder data skickas fram och tillbaka mellan server och klient vilket kan leda till tidigare nämnda problem som överbelastning och därmed paketförlust och fördröjning.

2.2.6 Uppdateringsfrekvens

I klient/server-upplägget har servern oftast en bestämd uppdateringsfrekvens (eng. "tickrate") (Gallagher, 2022). I detta sammanhanget innebär uppdateringsfrekvensen den frekvens som servern skickar datapaket på. I pvp-spel ligger uppdateringsfrekvensen ofta på 64 eller 128 hz och är oftast inte densamma som klientens vilket kan skapa problem. Om denna frekvens på servern exempelvis är 64 hz och spelarens skärm är 120 hz kan det se ut som att de andra spelarnas karaktärer teleporterar fram istället för att kontinuerligt förflytta sig (Gambetta, 2022).

2.2.7 Säkerhet

Med säkerhet menas åtgärderna som vidtas för att skydda spelet och spelarna från olika typer av cyberattacker och hot (Parizi, 2019).

Något som är viktigt för säkerheten inom onlinespel är skyddet av spelets servrar och klienter från hackning och DDoS-attacker (Distributed Denial of Service). En DDoS-attack innebär att överbelasta nätverket och därmed störa det genom att skicka en stor mängd förfrågningar per sekund till servern (Wang & Li, 2022, Parizi, 2019). Med peer-to-peer-uppsättningen där klienterna skickar datapaket direkt till varandra behöver varje klient en direkt anslutning till de andra klienterna. Detta innebär att de måste dela med sig av sin publika ip-adress som de andra spelarna sedan kan använda för att överbelasta nätverket med en DDoS-attack. Detta påverkar även andra användare som är anslutna till samma router. Om en DDoS-attack används i klient/server-uppsättningen kan den få servern att bli överbelastad vilket kan påverka alla ansluta spelare genom att göra spelet otillgängligt tills spelföretaget fixar servrarna (Parizi, 2019). Eftersom hackning ligger utanför omfattningen av denna text så kommer det inte nämnas i lösningarna. Det tas upp för att det kan vara bra att veta om, även om en lösning inte presenteras.

3 Problemformulering

När en spelutvecklare ska lägga grunderna för nätverkshanteringen i ett onlinespel kan den stöta på flera olika problem som kräver specifika åtgärder. Vanliga problem innefattar bland annat fördröjning, paketförlust, fusk, skalbarhet, synkronisering, uppdateringsfrekvens och säkerhet. De olika åtgärderna kan komma att påverka spelupplevelsen på olika sätt och vissa åtgärder passar några typer av onlinespel bättre än andra. Därför lades fokus på enbart på *en* genre av onlinespel. Syftet med denna texten är alltså att först jämföra och förklara åtgärderna för att sedan komma fram till vilken som passar bäst för en specifik genre av onlinespel.

Valet av spelgenren som studeras baseras på nätverkshanteringens påverkan på spelarens upplevelse. Co-op-spel uteslöts eftersom samtliga spelare i denna spelgenre arbetar mot ett *gemensamt* mål, vilket innebär att användningen av fusk från en av spelarna i vissa fall kan *gynna* de andra spelarna. Detta kan jämföras med pvp-spel där motståndarna kan uppleva det som orättvist. Detta innebär att fusksäkerheten egentligen inte *behöver* vara lika strikt som i pvp-spel där spelarna istället tävlar *mot* varandra. Inom pvp-spel uteslöts sedan turordningsbaserade spel eftersom problemet med fördröjning i dessa spel inte har lika stor påverkan på spelarens upplevelse då spelaren ändå får en förbestämd tid att tänka ut sin respons på. Därmed lades fokuset på pvp-spel med realtidskontroll eftersom denna spelgenre kräver en striktare nätverkshantering än de som nämndes tidigare. Nätverkshanteringen inom denna spelgenre har alltså som störst påverkan på spelarens upplevelse. Det är också viktigt att det är så rättvist som möjligt för alla spelare då dessa spel baseras på att vinna över varandra (Levkoff, 2014). Spelutvecklaren måste alltså se till att spelupplevelsen är så lik som möjligt för alla spelare, oavsett stabiliteten av internetåtkomsten och mängden fördröjning hos klienterna. På grund av att dessa kan skiljas mycket från klient till klient har därmed även *pålitligheten* av nätverkshanteringen stor påverkan på spelarnas upplevelse.

Många av de problem som förekommer i pvp-spel kommer i grunden från spelarnas användning av fusk (Gambetta, 2020). I singleplayerspel spelar det oftast ingen roll om en spelare fuskar eftersom det endast påverkar den spelarens upplevelse. Detta är dock inte fallet i pvp-spel, där användningen av fusk kan påverka de andra spelarnas upplevelse på ett *negativt* sätt. Det är därmed viktigt för en spelutvecklare att veta hur nätverket bör sättas upp för att öka fusksäkerheten. Målet med nätverkshanteringen i denna spelgenre är alltså att skapa en så rättvis miljö för alla spelare med minimal negativ påverkan på game feel.

Frågeställningen som valts är följande:

Vilka åtgärder för de grundläggande nätverksproblem vid skapandet av onlinespel passar bäst till pvp-spel med realtidskontroll?

3.1 Metodbeskrivning

För att få reda på vilka åtgärder som är bäst anpassade för några av de grundläggande problem som kan uppstå vid implementationen av nätverkshantering i pvp-spel genomförs en litteraturanalys. En litteraturanalys innebär en systematisk examination av ett problem där källor analyseras med ett specifikt syfte i åtanke (Berndtsson m.fl., 2008). Frågan i detta sammanhang är vilka åtgärder som passar bäst vid skapandet av pvp-spel. Litteraturanalyser har fördelarna att förbättra förståelsen och sammanfatta faktan som funnits sedan tidigare samt att generalisera resultaten till en större del av populationen (i detta tillfälle spelutvecklare). Främsta prioritering ligger på akademiska tidskrifter som söks genom högskolans databaser. Vid tillfällen där inga sådana texten hittas letades det istället efter bloggar (från exempelvis gamedeveloper.com och GDC). Vissa av de nyare källorna som hittats antar att läsaren redan har de förkunskaper som krävs inom området och förklarar därmed inte vissa begrepp på en tillräckligt djup nivå. Några av källorna från tidigt 2000-tal gör däremot det vilket gör att de fortfarande är aktuella. Därmed används de äldre källorna för att förklara begrepp på en djupare nivå, samtidigt som de nyare validerar att de fortfarande är relevanta och används idag. Ett känt problem inom litteraturanalyser som Van Wely (2014) tar

upp är “GIGA-principen” (eng. “garbage in - garbage out”). Denna princip innebär att om data är dålig, kommer även resultatet av analysen vara det. Detta är ett problem vid tillfällen då lösningarna inte beskrivs utförligt nog i artiklar och bloggar skrivna av kända spelföretag som annars hade varit trovärdiga. Vid sådana tillfällen söks övriga artiklar eller bloggar för att få en utförlig förklaring till lösningen.

Om det visar sig finnas fler än en lösning till ett problem kommer samtliga lösningar att jämföras för att komma fram till vilken som passar bäst för just pvp-spel. Vilken lösning som passar bäst baseras på hur väl den hanterar de problem som förklarats i bakgrunden. Det minsta kravet för en lösning är att den inte påverkar spelupplevelsen på ett *negativt* sätt.

Som sökverktyg för akademiska källor kommer *LibSearch, SpringerLink, Web of Science* och *Scopus* att användas. Sökord kommer till grunden att innefatta ord som “networking”, “video game”, “problems”, “solutions”, “latency”, “packet loss” och “cheating”. Ytterligare sökord som lagts till under arbetets gång är “peer-to-peer”, “client/server”, “dumb client”, “client-side prediction”, “server reconciliation” och “interpolation”.

4 Analys

Med hjälp av att analysera artiklar och bloggar har denna litteraturanalys kommit fram till de lösningar som kan användas när problem framkommer vid skapandet av online pvp-spel. Specifika lösningar måste användas för att minska de grundläggande problem som nämns i bakgrunden. Innan några *nya* åtgärder tas upp måste först grunderna till nätverkshanteringen för spelgenren i fokus läggas. Vissa spel som fortfarande har problemen som beskrivs har även tagits upp för att ge exempel där spelutvecklarna har misslyckats.

4.1 Val av nätverksprotokoll

Vilket nätverksprotokoll som ska användas är inte någonting som väljs *en* gång utan behöver väljas varje gång ett nytt sorts datapaket ska skickas. Vilket som bör väljas beror på olika faktorer:

En faktor som är viktig inom just pvp-spel är fördröjning. På grund av alla åtgärder som TCP använder sig av för att garantera att datapaketen kommer fram och i rätt ordning kräver de mer resurser från både sändaren och mottagaren jämfört med UDP. Detta leder till ökad fördröjning så om låg fördröjning är viktigt bör istället UDP användas. Inom pvp-spel önskas så låg fördröjning som möjligt när det kommer till spelarens interaktioner med sin karaktär och spelvärlden. Därmed används UDP för nästan allting i onlinespel som styrs med realtidskontroll (Pavee, 2022).

En annan faktor är ifall det är viktigt att datapaketet kommer fram till slut även om det kommer fram för sent. Det kan finnas vissa typer av datapaket som skickas inom pvp-spel där det är viktigt att datapaket kommer fram, där fördröjning inte är kritiskt. Exempel på detta är anslutning mellan klienterna och servern, item-transaktioner, chattmeddelanden och spelregler. I dessa fall används oftast TCP (Pavee, 2022).

4.2 Val av nätverksuppsättningar

Det finns flera olika faktorer och perspektiv som kan påverka en spelutvecklares beslut när det kommer till att välja mellan peer-to-peer och klient/server:

En faktor är mängden spelare som ska kunna interagera med varandra. Ett problem med peer-to-peer är att antalet spelare i en match kan behöva begränsas mer än för ett spel med klient/server-uppsättning (Liu m.fl., 2022). Detta beror på att klienterna inte kan förväntas ha tillräckligt med bandbredd för att skicka och ta emot datapaket från alla andra klienter i spelet om mängden spelare blir för stor. Servrar har däremot oftast mycket mer bandbredd vilket leder till att de kan hantera större mängder spelare utan problem (Fouquet, 2014). På grund av detta hjälper klient/server-uppsättningen med skalbarhet.

En annan faktor är hur synkronisationen av speltillståndet ska hanteras. När peer-to-peer används kan det bli svårt att synkronisera speltillstånd, vilket betyder hur spelvärlden ser ut vid ett visst tillfälle (exempelvis var spelarkaraktärer och föremål befinner sig) (Moraal, 2006). I en situation där ett spel redan har påbörjats och ytterligare en klient ansluter sig kan andra klienter behöva skicka datapaket om det nuvarande speltillståndet till den nya klienten (Fiedler, 2010). Detta kan leda till överbelastning hos de andra klienterna. Med en klient/server-uppsättning kan istället *servern* skicka speltillståndet till den nya spelaren utan att de andra spelarna påverkas. Ifall en peer-to-peer-uppsättning används och en konflikt uppstår där två klienter inte är överens om speltillståndet kan det även bli svårt att ta reda på vilket som ska användas.

Ytterligare en faktor är till vilken nivå spelare ska kunna fuska. Som tidigare nämnts är fusk ett stort problem inom online pvp-spel. Ta ett exempel där en spelare har modifierat sin klient

så att den kan ändra i speltillståndet för att kunna förflytta sin karaktär dit den vill *direkt*. Om peer-to-peer används kan det som tidigare nämnts vara svårt att avgöra vilken utav klienternas speltillstånd som är korrekt (Moraal, 2006). Om klient/server används i denna situation kan servern se att den modifierade klientens speltillstånd inte stämmer överens med sitt egna och välja att inte skicka vidare den manipulerade klientens speltillstånd till de andra klienterna. Med en klient/server-uppsättning kan alltså möjligheten till fusk begränsas. För att bevisa detta ytterligare jämförs två spel som båda innehåller pvp men har olika nätverksuppsättningar. *Grand Theft Auto V* är ett spel som använder sig utav peer-to-peer för sitt onlineläge. I detta spelet kan en fuskare exempelvis ta fram ett flygplan från ingenstans i en korsning mitt i staden, eller klona en annan karaktärs utseende och namn (Marshall, 2021). Alltså kan modifierade klienter direkt förändra speltillståndet hos andra klienter. Jämför detta med *Counter Strike: Global Offensive* som använder sig av klient/server (Bryant & Saiedian, 2021). Här kan spelare som fuskar exempelvis använda wall-hacks, triggerbots och aimbots (Bhatti, 2021). Dessa fusk kan inte modifiera andra klienters speltillstånd direkt utan enbart samla information och automatisera inmatningar. Med en klient/server-uppsättning kan servern även analysera dessa inmatningar för att hitta och banna fuskare (McDonald, 2018). Om ett online pvp-spel utvecklas där det går att lita på att ingen klient kommer försöka fuska (i exempelvis vissa militärsimulationer, som nämns senare i texten) kan peer-to-peer-uppsättningen fungera men annars bör klient/server-uppsättningen användas då möjligheterna till fusk blir betydligt mer begränsade.

I fallet av denna text som fokuserar på pvp-spel bör alltså klient/server-uppsättningen användas för att öka skalbarheten, samtidigt som den minskar möjligheten till fusk. Därmed kommer resten av analysen gå in på djupet av flera vanliga åtgärder till problem som kan uppstå främst vid användningen av nätverksuppsättningen klient/server. Även om problemet med säkerhet återstår oavsett val av nätverksuppsättning skiftas konsekvenserna av DDoS-attackerna till företagets servrar istället för att riskera hot mot privatpersonerna. Det krävs också mer ansträngning att skicka DDoS-attacker till servern på grund av dess större bandbredd, vilket minskar möjligheterna till överbelastning (Reza, 2019).

4.3 Begränsning av möjligheter till fusk

Stycket 4.2 har etablerat att användningen av nätverksuppsättningen klient/server kan ha fördelen att en klient inte direkt kan påverka en annan klients speltillstånd. Detta är dock inte en direkt konsekvens av att använda klient/server utan beror på system som kan implementeras *tack vare* nätverksuppsättningen klient/server (Liu m.fl., 2022). Det främsta systemet som bidrar till detta är användningen av en så kallad *"dum klient"* (eng "dumb client").

Användningen av en dum klient innebär att klienten aldrig skickar sitt speltillstånd utan skickar enbart sina inmatningar till servern (Gambetta, 2020). I ett exempel där spelaren vill att sin karaktär ska gå framåt skickar klienten att spelaren har klickat på respektive knapp istället för att skicka sin nya position till servern. Servern tar sedan emot inmatningen, flyttar spelarens karaktär och skickar det nya speltillståndet till alla anslutna klienter.

Klienter går inte att lita på, de kan exempelvis göra missberäkningar på grund av dålig hårdvara och manipuleras eller tas över av tredjepartsprogram i syfte att fuska (Gambetta, 2020). Genom att använda sig av dumma klienter flyttas alla beslut om hur spelarna interagerar med varandra till servern. Detta bidrar till att minska möjligheten till fusk och kan öka jämställdheten mellan spelare som har olika bra hårdvara och internetåtkomst (Liu m.fl., 2022). Servern bör även endast tillåta *ett* datapaket med inmatning per uppdatering av speltillståndet. Detta är för att förhindra klienter från att exempelvis skicka datapaket som säger att de utförde en viss inmatning mer frekvent än vad som är tänkt vara möjligt. Inom pvp-spel är rättvishet och fusksäkerhet två väldigt viktiga aspekter vilket leder till att dumma klienter passar väldigt bra.

4.4 Åtgärder mot fördröjning inom förflyttning

En dum klient fungerar bra så länge som alla klienter och servern befinner sig geografiskt på samma plats med stabila internetuppkopplingar. Detta är dock nästan aldrig är fallet vilket gör att oundvikliga fördröjningar uppstår mellan att klienten skickar inmatningar och får tillbaka ett uppdaterat speltillstånd som baserats på dessa inmatningar. Detta kan alltså orsaka betydlig fördröjning mellan inmatning och händelse på skärmen vilket kan förvärra en av de tre byggstenarna till game feel betydligt, nämligen realtidskontroll.

Detta problem kan åtgärdas genom en teknik som kallas "client-side prediction" (Bernier, 2003). Säg att spelaren trycker på höger piltangent för att flytta sig ett steg åt höger från position (0, 0) till (1, 0) och har 100 millisekunder fördröjning till servern och tillbaka. Med lösningen som beskrivits hittills skulle spelaren behöva vänta minst 100 millisekunder mellan att trycka på piltangenten och att se sin karaktär flytta sig till den nya positionen på skärmen. Så länge som spelvärlden är deterministisk (det vill säga att given en viss situation och en viss inmatning räknar den ut vad som ska hända på samma sätt varje gång) kan klienten i detta exemplet förflytta sig till (1, 0) direkt utan att vänta på svar från servern. Klienten förutspår alltså vad som kommer hända för att undvika fördröjning. Säg att en spelare manipulerar sin klient så att den förflyttar sig till (2, 0) med ett enda tryck på höger piltangent. Med denna lösning kommer bara spelaren själv se sin karaktär på (2, 0) medan servern och alla andra spelare ser den på (1, 0). Alltså kan en spelare manipulera sin klient hur den vill utan att påverka vad de andra på servern ser. Genom att implementera denna lösning återfår spelet sin realtidskontroll, oavsett fördröjningen mellan klient och server.

Med lösningen som presenterats hittills kommer alltså en spelare som försöker manipulera sin klient hamna i osynk med servern och de andra spelarna. För att förhindra detta sätter klienten spelarens karaktär på den plats som servern bestämde när meddelandet väl kommit fram till klienten. I exemplet skulle alltså spelaren som modifierat sin klient först förflyttas från (0, 0) till (2, 0) och 100 millisekunder senare tillbaka till (1, 0).

I en situation där en spelare trycker på höger piltangent två gånger inom 100 millisekunder skulle spelarens karaktär med nuvarande lösning förflyttas från (0, 0) till (1, 0) och (2, 0), sedan tillbaka till (1, 0) när serverns meddelande kommer fram och till sist (2, 0). För att förhindra detta används en teknik som kallas "server reconciliation" (Gambetta, 2020). Detta innebär att klienten inte bara skickar sin inmatning till servern utan även vilken exakt tidpunkt som den tryckte på knappen (Bernier, 2003). Servern skickar sedan inte enbart tillbaka vilken plats spelarens karaktär hamnade på utan även vilken tidpunkt som spelaren tryckte på knappen som ledde till att den hamnade där. Klienten sparar även en historik av vilka inmatningar som utförts och vilka tidpunkter de utfördes. Istället för att förflytta sig dit servern bestämt varje gång den får information av servern kan klienten med hjälp av dessa tidsstämplar och historik jämföra hur den själv räknade ut var den skulle hamna baserat på en viss inmatning med hur servern räknade ut det. Ifall den räknade fel på grund av exempelvis bugg eller försök till fusk förflyttar sig spelarens karaktär dit servern bestämde och räknar ut var den hamnar baserat på alla inmatningar sedan den tidpunkten igen.

Client-side prediction kan vara jobbigt att implementera men med hjälp av den kan ett onlinespel åstadkomma realtidskontroll som är lika bra som i ett singleplayerspel *samtidigt* som det kan ha rättvisa och fusksäkerhet som är lika bra som om enbart dumma klienter användes. Dessa faktorer gör client-side prediction till en teknik som passar väldigt bra för pvp-spel.

4.5 Åtgärder mot skillnader i uppdateringsfrekvens

Uppdateringsfrekvensen hos servern bör väljas baserat på hur snabbt servern kan räkna ut allt som kan behöva räknas ut per tidssteg och hur många matcher som servern ska kunna hantera samtidigt (Gallagher, 2022).

I bakgrunden ges ett exempel där serverns uppdateringsfrekvens är långsammare än klientens vilket leder till att det ser ut som att de andra spelarna teleporterar fram istället för att

kontinuerligt förflytta sig. För att åtgärda detta kan en teknik som kallas “interpolation” användas (Gambetta, 2020). Detta innebär att klienten inte omedelbart ändrar andra spelarkaraktärers positioner så fort den tar emot dem utan uppdaterar positionerna i flera mindre steg beroende på klientens bildfrekvens. Detta ökar polish då spelkaraktärerna nu ser ut att röra sig kontinuerligt istället för att teleportera fram. Polish är ett av byggnadsblocken till game feel och att öka polish leder till en positiv påverkan på spelupplevelsen (Swink, 2009). Nackdelen med denna teknik är dock att fördröjningen på andra spelarkaraktärers positioner ökas.

En åtgärd mot att fördröjningen ökas är att använda en teknik som kallas “dead reckoning” (Gambetta, 2020). Med denna teknik behöver klienten information från datapaketen om hur snabbt andra spelarkaraktärer rör sig, vilken riktning de rör sig i och hur gammal informationen är. Detta för att försöka förutspå var de befinner sig i nutid, givet att de inte bytt riktning eller ändrat hastighet. Ifall de har ändrat riktning eller hastighet kan deras position behöva rättas när senare datapaket kommit fram. Denna teknik är nödvändig inom exempelvis racingspel där det är viktigt att spelarens motståndares positioner inte visas med fördröjning för att kunna bedöma vem som leder (Gambetta, 2020). Anledningen till att det är passande att användas i racingspel är för att fordon oftast inte ändrar riktning eller hastighet så drastiskt, vilket gör det enklare att förutspå var den kommer befinna sig vid senare tillfällen. Den passar dock inte lika väl inom pvp-spel där spelaren styr en karaktär som kan ändra sin riktning och hastighet väldigt snabbt. I ett exempel där en klient med hög fördröjnings karaktär springer mot en öppning men stannar precis innan kan en annan klient visa detta som att den springer fram till öppningen och sedan backar lite. Detta skulle kunna avslöja spelarkaraktären med hög fördröjnings position enbart för att den hade högre fördröjning vilket har en negativ påverkan på spelets rättvisa. Denna teknik kan alltså vara användbar för vissa typer av pvp-spel där snabba förändringar inom riktning och hastighet inte är vanliga men den passar inte för skjutarspel som är vad denna text fokuserar på.

4.6 Åtgärder mot fördröjning inom skjutning

En enkel implementation av att låta spelare skjuta varandra i ett onlinespel är ett system där klienten får räkna ut hur sitt eget skott åkte och vad det träffade. Sedan skickar klienten ett datapaket som innehåller denna informationen till servern. Servern litar på informationen den fick av klienten och skickar vidare informationen om vad skottet träffade till alla andra klienter. Med denna implementation blir klienten auktoritativ vilket leder till ett stort problem (Bernier, 2003). En manipulerad klient kan skicka ett datapaket till servern där det står att den sköt en annan spelarkaraktär även utan att spelaren utförde inmatningarna som ledde till det. Denna typen av implementation är vanlig inom militärsimuleringar där alla klienter går att lita på eller onlinespel som använder sig av peer-to-peer (fast datapaketet med träff-informationen åker då direkt till andra klienter istället för att skickas till servern först) (Bernier, 2003). Den passar dock inte till ett pvp-spel där främlingar kan möta varandra genom internet och har möjligheten att manipulera sina klienter.

En annan implementation som passar pvp-spel bättre är att endast låta klienter skicka datapaket som bara säger att spelaren *vill* skjuta. När datapaketet kommer fram till servern räknar den ut hur skottet åkte, vad/vem som blev träffad och skickar vidare informationen till de andra klienterna. Med denna implementation blir servern auktoritativ och problemet med att manipulerade klienter kan skicka att de sköt vem som helst när som helst undviks (Bernier, 2003). Däremot uppstår ett nytt problem som syntes i onlinespelet Quake3 som använde sig av denna implementation. När en spelare använde sig av ett *instant-hit*-vapen (vars skott träffar exakt samma tidpunkt de avfyrades) behövde den ändå skjuta där den trodde att spelarkaraktären den sköt på skulle befinna sig när datapaketet om att den vill skjuta kommer fram till servern (Bernier, 2003). Alltså behövde spelarna leda sina skott beroende på hur lång fördröjning till servern de hade. Detta leder dels till försämrad simulerad rymd eftersom ett skott kan åka rakt igenom en spelarkaraktär utan att träffa vilket bryter immersion. Det leder även till ökad orättvisa då enbart spelare med hög fördröjning behöver leda sina skott.

Ett tredje sätt att implementera skjutning inom onlinespel är att klienten inte enbart skickar en förfrågan *om* att skjuta utan även en tidsstämpel på *när* den sköt till servern. Servern håller också en historik av alla spelarnas och flyttbara föremåls positioner en viss tid tillbaka. Hur lång tid tillbaka denna historik går kan påverka spelupplevelsen och diskuteras senare i texten. När servern tar emot ett datapaket om att en spelare vill skjuta återskapar den hur spelvärlden såg ut vid just den tidpunkten då spelaren sköt (Gambetta, 2020). Det är också viktigt att servern vet hur mycket klienterna interpolerar andra spelares förflyttningar i detta steg så att den kan räkna med det när den återskapar var alla stod (Bernier, 2003).

Det finns dock flera problem med denna implementation. Säg att en spelare har en fördröjning på 300 millisekunder och en annan har 10 millisekunder. Första spelaren skjuter den andra men innan informationen om att den sköt har nått servern har den andra spelaren hunnit gömma sig bakom en vägg. Om serverns historik på spelarpositioner går 300 millisekunder tillbaka kan andra spelaren bli frustrerad då den blev träffad av ett skott som den tycker inte möjligtvis kunde ha träffat, medans om den inte går så långt tillbaka kan första spelaren bli frustrerad då den sköt ett skott som åkte rakt igenom den andra spelaren utan att träffa.

Ett annat problem är vad som kallas “peekers advantage” (Stavropoulos, 2021). Säg att en spelare sitter still i ett rum och vaktar en dörröppning medans en annan spelare springer in i rummet. Hade det inte funnits någon fördröjning och spelarna hade lika snabb reaktionstid hade spelaren som vaktade och var beredd skjutit först. Om båda spelarna istället har en fördröjning på 300 millisekunder kan spelaren som springer in upptäcka spelaren som sitter still innan den ens kan se spelaren springa in på sin skärm. I denna situationen får alltså spelaren som springer in en fördel.

En fuskare kan utnyttja detta system genom att manipulera klientens kod så att den kan skicka en felaktig tidsstämpel på sina skjut-meddelanden till servern (Hunink, 2023). Genom att göra detta kan klienten exempelvis skicka ett meddelande som säger att den sköt för 100 millisekunder sedan då den skulle ha träffat en annan spelare med skottet, även om det inte längre är möjligt att träffa spelaren.

Trots alla problem som orsakas är det ändå viktigast att en spelare som utför ett perfekt skott som åker rakt igenom fiendens huvud får en träff inom pvp-spel (Bernier, 2003). Annars bryts både simulerad rymd och rättvisan i spelet.

5 Sammanfattning och diskussion

5.1 Sammanfattning

Det kan framkomma många problem vid skapandet av onlinespel för nya spelutvecklare. Texten beskriver åtgärderna till några av de vanligaste problemen som fördröjning, paketförlust, fusk, skalbarhet, synkronisering, uppdateringsfrekvens och säkerhet. Här fokuseras det på att åtgärda de problem som uppstår inom just pvp-spel, där nätverkshanteringen har som störst påverkan på spelarens upplevelse. De spel och texter som analyserats har bevisat att en klient/server-uppsättning bör användas för pvp-spel för att öka fusksäkerheten. Jämfört med peer-to-peer, där klienterna ansluter sig och skickar sina datapaket direkt till varandra kan servern istället ta hand om alla uträkningar och göra klienten "dum", vilket innebär att den enbart skickar vad spelaren *vill* göra och sen låter servern bestämma om den får det eller inte. Mellan de två vanligaste protokollen för att skicka datapaket inom dataspel måste spelutvecklare välja om det är viktigare *att* paketet kommer fram eller *när* den kommer fram. Inom pvp-spel är det viktigt för spelarna att få information om de andras spelkaraktärerna för att kunna göra snabba beslut, vilket UDP gör på ett effektivt sätt. TCP kan istället användas för att säkerställa en anslutning mellan klienterna och servern eller vid de paket som enbart skickas en gång, exempelvis chattmeddelanden. Detta är med hjälp av en "handshake" mellan klienten och servern, där den ena först skickar en förfrågan till den andra för att etablera en anslutning innan data skickas över. Detta gör att alla paket skickade med TCP garanterat kommer fram, men möjligtvis med en längre fördröjning.

Om servern är fysiskt placerad långt ifrån klienten kan fördröjningen bli märkbart stor även om UDP används. Detta kan leda till att spelarens inmatningar inte uppdateras på spelarens skärm på hundratals millisekunder. För att lösa detta kan en teknik som kallas "client-side prediction" användas (Bernier, 2003). Denna lösning låter klienten förutspå var karaktären kommer hamna även innan den har fått tillbaka paketet från servern och därmed eliminera en del fördröjning. Om spelaren själv då försöker manipulera spelarkaraktärens position kommer det inte påverka vad de andra ser på servern, men fortfarande se ändringarna på sin egna skärm och därmed hamna i osynk med servern och de andra spelarna. För att förhindra detta sätter sen klienten spelarkaraktären på den plats som servern bestämde när meddelandet väl kommit fram. För att stoppa servern från att flytta tillbaka spelkaraktären hela tiden används en teknik som kallas "server reconciliation" (Gambetta, 2020). En historik med spelarens information sparas på hos både klienten och servern för att kunna räkna ut var spelarkaraktären hamnar baserat på vilken tid (som skickas med i datapaketet) klienten faktiskt skickade inmatningarna. Är uträkningarna annorlunda på grund av exempelvis buggar eller försök till fusk flyttas spelarkaraktären tillbaka till platsen som servern bestämde och räknar om var den hamnar baserat på de knappar som tryckts sedan dess. Denna tidsstämpel som skickas med kan även användas för att hantera fördröjningen mellan spelarens inmatningar om att skjuta en annan spelare till att den kommer fram till servern. På grund av att servern som tidigare nämnts sparar informationen om var alla spelare varit till en viss tid tillbaka kan servern återskapa hur spelvärlden såg ut vid den tidpunkten då spelaren sköt.

Det är också sällan servern och klienten uppdateras lika ofta. Om servern skickar de andra spelarkaraktärernas position med en frekvens på 30 hz och spelarens skärm uppdateras med en högre frekvens som exempelvis 120 hz ser det ut som att de andra spelarna teleporterar fram istället för att kontinuerligt förflytta sig. För att lösa detta kan en teknik som kallas "interpolation" användas. Med denna lösning ändras inte spelarkaraktärernas position direkt, utan jämnar den i flera mindre steg beroende på klientens bildfrekvens (Gambetta, 2020).

5.2 Diskussion

De lösningar som beskrivs i texten stoppar inte helt och hållet alla problem som tas upp i bakgrunden. Många fusk är fortfarande tillgängliga för spelarna att använda. Poängen med

vissa av lösningarna i texten är att *minska* dessa möjligheter till fusk. Eftersom fuskare aktivt letar efter nya sätt att fuska på som spelutvecklarna måste skapa nya åtgärder för är det svårt att kunna stoppa det helt. För att stoppa fler sorters fusk som fungerar även med åtgärderna som har presenterats i texten kan "anti-cheat" användas (Björkskog & Gaston, 2019).

Flera av de källor som använts är från tidigt 2000-tal. Detta beror på att de nyare källor som hittades om vissa ämnen och tekniker byggde på de grunder som beskrivs i de äldre källorna. På grund av att ett av målen med denna text var att beskriva grundläggande problem och dess åtgärder var de äldre källorna ofta mer relevanta för att utförligt kunna förklara hur åtgärderna fungerar. Liu m.fl., (2022) skriver exempelvis att Bernier (2003) beskriver vissa tekniker i detalj men att moderna spel använder fler tekniker än de som presenterats.

Första metoden som valdes för denna rapport var egentligen att hitta lösningar från andra kända spel som har hanterat dessa problem för att implementera dem i spelmotorn Unity. Tanken här var att försöka replikera de lösningar som använts av spelföretag och därmed inte implementera några nya modifierade versioner. På grund av detta blev dock en artefakt mindre relevant eftersom alla problem och lösningar ändå förklaras i texten, vilket då enbart hade tagit längre tid om en artefakt också hade behövts implementeras. För att validera att lösningarna faktiskt fungerade hade även ett experiment behövts genomföras (Berndtsson m.fl., 2008). Detta hade också lagt till tid på rekrytering och att boka tider för deltagarna. På grund av detta bestämdes det istället att en litteraturanalys skulle genomföras för att enbart beskriva de åtgärder som skapats och komma fram till vilka som passar bäst till pvp-spel.

Dessa lösningar som hittats är definitivt inte de bästa om ens nödvändiga för varje spelgenre och många av dem kräver tid och resurser för att implementera. Implementationen för att kompensera för fördröjning inom skjutning hos spelarna kräver en del minne av servern då den behöver lagra potentiella rörliga objekt och alla spelares information (som position och rotation) en viss tid tillbaka. Denna lösning kan till och med vara *för* krävande för pvp-spel som tillåter ett tresiffrigt eller högre antal spelare.

5.3 Samhälleliga och etiska aspekter

I texten tas det upp att grunden till många av de problem som skapas inom pvp-spel är spelarnas användning av fusk. Inom denna spelgenre är det viktigt att det är så rättvist mellan spelarna för att alla ska kunna få så bra spelupplevelse som möjligt. Anledningarna till att vissa av dessa lösningar väljs är därmed för att skapa en så etiskt jämställdhet som möjligt. En spelutvecklare som skapar online pvp-spel måste *alltid* anta att spelaren kommer bryta mot de etiska aspekterna vilket i sin tur kräver mer arbete. Resultatet blir dock positivt för spelarna eftersom de får en mer rättvis upplevelse i spelet.

Ur ett samhälleligt perspektiv försöker texten ge information inom utvecklingen av onlinespel. Bakgrunden ger en förståelse av problemen som uppstår vid skapandet av onlinespel och analysen redogör för några av de metoder som kan användas för att åtgärda dessa problem. Denna analys kan hjälpa spelutvecklare genom att ge dem en förståelse av vad som kan hända under utvecklingen av spelet samt hur man hanterar det. Poängen med dessa lösningar är att spelaren ska få en så bra upplevelse som möjligt.

Om alla spelare hade spelat rättvist utan försök till fusk eller dataintrång skulle många av dessa åtgärder inte behöva användas. Med en peer-to-peer-uppsättning skulle den enkla möjligheten till användning av fusk som nuvarande är ett stort problem inte längre vara relevant. Därmed hade val av nätverksuppsättning baserats på ett annorlunda sätt. I detta tillfället hade klienterna oavsett nätverksuppsättning kunnat skickat datapaket med spelarkaraktärens information direkt. Med klient/server-uppsättningen hade detta minskat belastningen för servern då den inte längre behöver ansvara för dessa beräkningar. Behovet av client-side prediction hade inte heller varit relevant då inga spelare skulle behöva vänta på svar från servern om sin egna position.

5.4 Framtida arbete

Framtida arbete skulle kunna läggas på att skapa en artefakt för att implementera dessa lösningar som hittats. Eftersom detta enbart hade tagit längre tid denna gången skulle ytterligare ett arbete kunna utföras för att just implementera dem och visa konkreta exempel på hur uträkningarna ser ut i kod. På grund av att vissa av dessa lösningar är avancerade kan det vara svårt att förstå hur en spelutvecklare faktiskt skulle implementera dem i en spelmotor. Texterna som beskriver lösningarna berättar oftast inte exakt hur mycket prestanda de tar att implementera dem vilket skulle kunna mätas om en artefakt hade skapats. Detta i sin tur kan också påverka valet av åtgärd som passar till just pvp-spel. Om beräkningstiden är *för* lång kan den påverka spelupplevelsen på ett negativt och därmed behöva uteslutas från att vara en passande åtgärd.

Ett ytterligare arbete skulle sedan kunna läggas på att optimera de åtgärder som har för långa beräkningstider för att kunna användas inom ett bredare område. Att servern sparar en historik om varje spelare på ett onlinespel med tresiffrigt eller högre antal spelare kan kräva en del minne och därmed inte vara applicerbart.

Den genre av onlinespel som valts till denna text är pvp-spel. De åtgärder som passar till denna spelgenre kan komma att påverka andra spel på ett annorlunda sätt som kanske inte gynnar spelupplevelsen. Inom exempelvis turordningsbaserade pvp-spel har fördröjningen inte lika stor påverkan på spelupplevelsen. Detta innebär att åtgärder som hanterar fördröjning inte har lika stor användning inom denna spelgenre. För att förstå vilka lösningar som passar till andra spelgenrer kan liknande arbeten genomföras med andra spelgenrer i fokus. Alternativt skulle ett av problemen kunna fokuseras på istället för att ta reda på alla möjliga åtgärder som löser det specifika problemet. Ett intressant ämne för detta kan vara att ta reda på alla möjliga metoder en fuskare kan använda sig av och hur man stoppar dem även utanför nätverket.

Referenser

Ali, S.H., Nasir, S.A. & Qazi, S. (2013) 'Impact of router buffer size on TCP/UDP performance', 2013 3rd IEEE International Conference on Computer, Control and Communication (IC4), Computer,Control & Communication (IC4), 2013 3rd International Conference on, s. 1–6. doi:10.1109/IC4.2013.6653751.

Bernier, Y. (2003). Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. *Semantic Scholar* [blogg], u.d. https://www.semanticscholar.org/paper/Latency-Compensating-Methods-in-Client-Server-and-Bernier/330071040ca858ca710a24a03915366fcd46f021 [2023-01-29]

Berndtsson, M., Lundell, B., Hansson, J. & Olsson, B. (2008) A guide for students in computer science and information systems. Springer London. Tillgänglig på Internet: https://search-ebscohost-com.libraryproxy.his.se/login.aspx?direct=true&db=cat09042a&AN=his.oai.his.se.27090&lang=sv&site=eds-live [Hämtad Januari 29, 2023].

Bhatti, F. (2021). Valve admits it was letting cheaters into trusted CSGO games. *Win.gg* [blogg], 30 april. https://win.gg/news/valve-responded-to-spike-in-csgo-cheaters-says-its-fixed/ [2023-03-16]

Blaukovitsch, R. (2022). Cheating in video games: The A to Z. *Irdeto Insights* [blogg], 31 januari. https://blog.irdeto.com/video-gaming/cheating-in-games-everything-you-always-wanted-to-know-about-it/ [2023-01-29]

Brun, J., Safaei, F. & Boustead, P. (2006) 'Managing Latency and Fairness in Networked Games', Communications of the ACM, 49(11), s. 46–51. doi:10.1145/1167838.1167861.

Bryant, B. och Saiedian, H. (2021) An evaluation of videogame network architecture performance and security, *Computer Networks*, 192. doi:10.1016/j.comnet.2021.108128.

Chernyak, U. (2014). Peer to Peer Connections for Online Infrastructures. *Game Developer* [blogg], 20 september. https://www.gamedeveloper.com/business/peer-to-peer-connections-for-online-infrastructures [2023-03-16]

Chernyak, U. (2014). Client/Server Functionality for Online Infrastructures. *Game Developer* [blogg], 22 september. https://www.gamedeveloper.com/business/client-server-functionality-for-online-infrastructures [2023-03-16]

Ferretti, S. & D'Angelo, G. (2018). Online Gaming Scalability. Encyclopedia of Computer Graphics and Games. Springer, Cham. https://doi-org.libraryproxy.his.se/10.1007/978-3-319-08234-9_218-1

Fiedler, G. (2008). UDP vs. TCP. *Gaffer On Games* [blogg], 1 oktober. https://www.gafferongames.com/post/udp_vs_tcp/ [2023-03-16]

Fiedler, G. (2010). What Every Programmer Needs To Know About Game Networking. *Gaffer On Games* [blogg], 24 februari. https://www.gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/ [2023-03-16]

Fouquet, P. (2014). Dedicated Servers vs Peer-to-Peer Connections. *Game Skinny* [blogg], 9 december. https://www.gameskinny.com/pqdqx/dedicated-servers-vs-peer-to-peer-connections [2023-03-16]

Gallagher, A. (2022). What is a Game Server? Everything you need to know. *One Qode* [blogg], 18 juli. https://www.oneqode.com/what-is-a-game-server/ [2023-03-16]

Gambetta, G. (2020). Fast-Paced Multiplayer (Part I): Client-Server Game Architecture. *Gabriel Gambetta* [blogg], u.d. https://www.gabrielgambetta.com/client-server-game-architecture.html [2023-03-16]

Gambetta, G. (2020). Client-Side Prediction and Server Reconciliation (Part II): Client-Server Game Architecture. *Gabriel Gambetta* [blogg], u.d. https://www.gabrielgambetta.com/client-side-prediction-server-reconciliation.html [2023-03-16]

Gambetta, G. (2020). Fast-Paced Multiplayer (Part III): Entity Interpolation. *Gabriel Gambetta* [blogg], u.d. https://www.gabrielgambetta.com/entity-interpolation.html [2023-03-16]

Gambetta, G. (2020). Fast-Paced Multiplayer (Part IV): Lag Compensation. *Gabriel Gambetta* [blogg], u.d. https://www.gabrielgambetta.com/lag-compensation.html [2023-03-16]

Hunink, M. (2023). Peeker's advantage in VALORANT explained? *Farming Less* [blogg], 19 februari. https://farmingless.com/what-is-backtracking-csgo/ [2023-03-16]

Kahraman, A. & Kazançoğlu, İ. (2022) 'A Qualitative Research of Young People's Motivation to Start, Continue, Reduce and Quit Playing Online Multiplayer Games on Computer', International Journal of Human-Computer Interaction, s. 1–23. doi:10.1080/10447318.2022.2096041.

Levkoff, S. (2014). On Balance-optimizing the PVP Experience. *Game Developer* [blogg], 2 mars. https://www.gamedeveloper.com/business/on-balance-optimizing-the-pvp-experience [2023-03-16]

Liu, S., Xu, X., & Claypool, M. (2022) 'A Survey and Taxonomy of Latency Compensation Techniques for Network Computer Games', ACM Computing Surveys, 54, pp. 1–34. doi:10.1145/3519023.

Marshall, C. (2021). Rockstar Games fans are sick and tired of all the hackers. *Polygon* [blogg], 2 februari. https://www.polygon.com/2021/2/2/22262564/gta-online-red-dead-online-hackers-rockstar-mod-menus-cheats-community-controversy [2023-03-16]

McDonald, J. (2018). Robocalypse Now: Using Deep Learning to Combat Cheating in 'Counter-Strike: Global Offensive' [video] https://www-gdcvault-com.libraryproxy.his.se/play/1024994/Robocalypse-Now-Using-Deep-Learning [2023-03-16]

Moraal, M. (2006) *Massive Multiplayer Online Game Architectures*. Tillgänglig på internet: https://www.cs.ru.nl/bachelors-theses/2006/Martijn_Moraal___0131903___Massive_Multiplayer_Online_Game_Architectures.pdf [Hämtad Mars 3, 2023]

Pandora FMS team. (2022). Packet loss: problems, causes and solutions. *Pandora FMS Monitoring Blog* [blogg], 25 oktober. https://pandorafms.com/blog/packet-loss/ [2023-01-29]

Pavee. (2022). TCP vs UDP: Why UDP is Preferred Over TCP For Online Multiplayer Games. *Game Dev Planet* [blogg], 13 oktober. https://gamedevplanet.com/tcp-vs-udp-why-udp-is-preferred-for-online-games/ [2023-03-16]

Parizi, RM., Dehghantanha, A., Choo, K-KR., Hammoudeh, M. & Epiphaniou, G. (2019) *Security in Online Games: Current Implementations and Challenges*. Springer International Publishing. doi:10.1007/978-3-030-10543-3_16.

Stavropoulos, A. (2021). What Is Backtracking CSGO? *Dot Esports* [blogg], 25 mars. https://dotesports.com/valorant/news/peekers-advantage-in-valorant-explained [2023-03-16]

Spacey, J. (2018). What is Latency? *Simplicable* [blogg], 2 april. https://simplicable.com/IT/latency [2023-03-16]

Swink, S. (2009) Game feel : a game designer's guide to virtual sensation. Morgan Kaufmann Publishers/Elsevier. Tillgänglig på internet: https://search.ebscohost.com/login.aspx?direct=true&db=cat09042a&AN=his.oai.his.se.9812&lang=sv&site=eds-live [Hämtad Januari 29, 2023]

van Wely, M. (2014) 'The good, the bad and the ugly: meta-analyses', Human Reproduction, 29(8), s. 1622–1626. doi:10.1093/humrep/deu127.

Wang, D. & Li, S. (2022) 'Automated DDoS Attack Mitigation for Software Defined Network', 2022 IEEE 16th International Conference on Anti-counterfeiting, Security, and Identification (ASID), Anti-counterfeiting, Security, and Identification (ASID), 2022 IEEE 16th International Conference on, s. 100–104. doi:10.1109/ASID56930.2022.9996013.

Zhao, M., Zheng, J. and Liu, E.S. (2021) 'Server Allocation for Massively Multiplayer Online Cloud Games Using Evolutionary Optimization', ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 17(2), s. 1–23. doi:10.1145/3433027.

Appendix A – Arbetsfördelning

Arbetet har delats upp mellan studenterna enligt följande beskrivning:

Melvin Olsson har skrivit följande:

- 2.1.2 Nätverksuppsättningar (förutom sista stycket)
- 2.1.4 Pvp-spel med realtidskontroll
- 2.2.1 Fördröjning
- 2.2.2 Paketförlust
- 2.2.3 Fusk
- 3 Problemformulering (enbart frågeställningen)
- 4.1 Val av nätverksprotokoll
- 4.2 Val av nätverks-uppsättningar
- 4.3 Begränsning av möjligheter till fusk
- 4.4 Åtgärder mot fördröjning inom förflyttning
- 4.6 Åtgärder mot fördröjning inom skjutning

Carl-Vilhelm Johansson har skrivit följande:

- Sammanfattning
- 1 Introduktion
- 2.0 Bakgrund
- 2.1.1 Nätverksprotokoll
- 2.1.2 Nätverksuppsättningar (enbart sista stycket)
- 2.1.3 Game feel
- 2.1.4 Pvp-spel med realtidskontroll
- 2.2.0 Vanliga problem
- 2.2.4 Skalbarhet
- 2.2.5 Synkronisering
- 2.2.6 Uppdateringsfrekvens
- 2.2.7 Säkerhet
- 3 Problemformulering (förutom frågeställning)
- 3.1 Metodbeskrivning
- 4.0 Analys
- 4.5 Åtgärder mot skillnader i uppdateringsfrekvens
- 5 Sammanfattning och diskussion

Carl-Vilhelm har varit ansvarig för kontakt med handledare Andreas Jonasson medans Melvin enbart läste Carl-Vilhelms svarsmail och deltog i handledningar.