Multiplayer-spel i realtid på smarttelefoner

JESPER ANNEBÄCK
och CARL HENRIK KLÅVUS

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2012

Multiplayer-spel i realtid på smarttelefoner

JESPER ANNEBÄCK
och CARL HENRIK KLÅVUS

DD143X, Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik 300 högskolepoäng
Kungliga Tekniska Högskolan år 2012
Handledare på CSC var Mårten Björkman
Examinator var Mårten Björkman

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2012/
anneback_jesper_OCH_klavus_carl_henrik_K12007.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Referat

Följande rapport är skriven av Jesper Annebäck och Carl Henrik Klåvus som studerar Datateknikprogrammet på Kungliga Tekniska högskolan. Denna uppsats är skriven för att erhålla kandidatexamen inom datateknik.

Dagens fem populäraste mobilspel på marknaden är antingen turbaserade eller singelplayer-spel, vilket inte alls är fallet då man ser till PC-spel - där de mest föredragna spelen är realtidsbaserade multiplayer-spel. Av denna anledning så torde det finnas en stor marknad för realtidsmultiplayer-spel på mobiltelefonsmarknaden om kraven på dessa spel uppfylls.

Rapporten berör frågan huruvida realtidsmultiplayer-spel för smarttelfoner står sig idag, med avseende på smarttelefoners hårdvaru-prestanda och 3G-latenstider. Denna rapport kommer gå genom de grundstenar och standarder av 3G i Europa samt vad som kan göras åt latenstider på en applikationsnivå för att förminska känslan av fördröjning i multiplayerspel. Mätningarna av latenstider över olika nätverkstekniker utfördes 10:e April 2012. Resultaten visade att laggkompensationsteknik är betydelsefullt när man ska utveckla spel som är ämnade för 3G-näten och att hårdvaran i dagens smarttelefoner kan rendera enkel 3D-grafik.

Abstract

The following report is made by Jesper Annebäck and Carl Henrik Klåvus who attend the Computer Science program at Kungliga Tekniska högskolan. This essay is written in order to achieve a Bachelor's degree in the field of computer science.

Today's top five games for modern mobile phones, smartphones, on the market are either turned-based multiplayer or singleplayer games, unlike the case you see at the market for PC games where the real-time multiplayer genre is the most favored choice. Therefor there is a large possible market for real-time multiplayer games on mobile platforms if the requirements for these kind of games are met.

This report addresses the issue of real-time multiplayer gaming on smartphones in aspect of hardware performance and 3G latencies. This paper will describe the fundamentals of 3G networks in Europe and what can be done in application level to reduce the feeling of latency in multiplayer games. The measurements made in the study of latency over different network mediums were conducted 10th of April 2012. The results from the study shows that lag compensation is important when developing games that will be used by smartphones using 3G and that the hardware on today's smartphones can render simple 3D graphics.

Innehåll

Kapitel 1

Introduktion

1.1 Bakgrund

Online-spelandet och E-sport är en starkt växande industri, idag finns det exempelvis 11 miljoner unika spelare i Massive-multiplayer-spelet *World of Warcraft*.[1] First-person-shooter-spelet (härmed benämnt FPS i stora bokstäver) *Modern Warfare 3* sålde 6,5 miljoner kopior på enbart första dagen efter lansering.[2] Det ökade datorspelsintresset är förstås tack vare bättre ljud och fysikaliska effekter i spelmotorerna men även på grund av åtkomsten till bättre internettjänster och nätverksstöd.

Beroende på vad det är för datorspel blir behovet av en bra uppkoppling och mindre latens mer betydelsefull. Latens är den fördröjningstid som uppstår vid kommunikation över ett nätverk, denna fördröjningstid försöks hållas på en så pass låg nivå som möjligt.[3] Latens är alltså den tid det tar att skicka ett paket från en destination till en annan.

Turbaserade datorspel är mindre känsliga för latenstider i och med att spellogiken inte tillåter spelarna att spela under samma tidsperiod. Latens har mycket större influens i FPS-spel då höga latenstider i spelen påverkar interaktionen i uppstående situationer som kräver korta reaktionstider, exempelvis att en spelare ska skjuta en annan spelare innan den hinner ta skydd runt ett hörn. Detta ger upphov till konflikter då båda spelarna kan ha en egen uppfattning av vad som är realtid och det kan sluta med att de jagar varandras skuggbilder där de stod för exempelvis

[1] A Ziebart, "World of Warcraft dips to a mere 11.4 million", för AOL Inc, uppdaterad 2011-05-09, läst 2012-03-06, `http://wow.joystiq.com/2011/05/09/world-of-warcraft-dips-to-a-mere-11-4-million-subscribers/`[1]

[2] B Crecente,"Call of Duty: Modern Warfare 3 Shatters All Sales Records, Nears Half a Billion Dollars in Day One Sales", för Kotaku, uppdaterad 2011-11-11, läst 2012-03-06, `http://kotaku.com/5857400/call-of-duty-modern-warfare-3-shatters-all-sales-records-surprises-no-one`[2]

[3] G Howlan, "What is Lag?", för GameDev.Net LLC, uppdaterad 1999-09-14, läst 2012-03-19 `http://www.gamedev.net/page/resources/_/technical/multiplayer-and-network-programming/what-is-lag-r712`[3]

400 millisekunder(ms) sedan.

Dagens topp fem populäraste mobilspel på marknaden (App Store och Google Play) är antingen turbaserade eller singelplayer-spel vilket inte alls är fallet då man ser till PC-spel - där de regerande spelen är realtidsbaserade multiplayer-spel.[4,5,6] Av denna anledning torde det finnas en stor marknad för realtidsmultiplayer-spel på mobiltelefonsmarknaden om bara de krav som ställs på dessa uppfylls. Dessa krav anpassas utefter de begränsningarna som finns hos bland annat 3G och den hårdvara som dagens smarttelefoner besitter.

1.2 Syfte

Detta arbete har som syfte att undersöka om dagens smarttelefoner har en prestanda, i ett äldre realtidsmultiplayer-spel, som kan ge en önskad spelupplevelse med avseende på latenstider och uppdateringfrekvens.

1.3 Problembeskrivning

Detta arbete kommer undersöka huruvida smarttelefoner, med Android som operativsystem, står sig som ett alternativ till PC när det kommer till multiplayerspelande. Huvudfrågeställningen kommer vara:

Vilka latenstider kan fås på olika nätverk som smarttelefoner använder? Exempelvis 3G, trådlöst nätverk (WLAN).

Två andra mindre frågor som kommer behandlas är:

Är hårdvaran idag i smarttelefoner tillräcklig för att nå acceptabla Frames per second-nivåer (härmed benämt som fps i små bokstävergenment)?
Är dessa latenstider tillräckligt stabila och korta för att det ska vara möjligt spela realtidsmultiplayer-spel?

Några monetära och grafiska aspekter av skillnaden mellan PC och smarttelefon kommer inte jämföras, utan endast ovanstående hårdvarurelaterade frågor.

[4]Apple Inc., "Popular Apps", läst 2012-04-03 , `http://itunes.apple.com/us/genre/ios-games/id6014?mt=8`[4]

[5]Google Inc., "Bästsäljande i Appar", läst 2012-04-03, `https://play.google.com/store/apps/collection/topselling_paid?feature=top-paid`[5]

[6]CBS Interactive Inc., "Most Popular PC Games", läst 2012-04-03, `http://www.gamespot.com/games.html?platform=5&mode=all&sort=views&dlx_type=all&sortdir=asc&official=all`[6]

1.4 Samarbetsredogörelse

Under detta projekt skötte Carl Henrik främst implementationsdelen, det vill säga skapandet av Android-applikationen och testning, men undersökte även de bakomliggande teknikerna som används i multiplayer-spel för att förbättra den upplevda latenstiden.

Jespers del under detta projekt var att forska omkring 3G:s grunder som innefattar det allt från radiokommunikation i luft till utvecklade egenskaper. Utöver det gjorde han mycket dokumentation bland annat lade upp strukturen för detta dokument.

Tabell 1.1: Lista över arbetsuppdelningen.

Jesper Annebäck	Carl Henrik Klåvus
Skrev 2.1, 3.1 och 3.2.2	Skrev 2.2, 3.2.1, 3.3
Satte upp fungerande Kwakk3-körning	Skrev samt utförde tester på Android-applikation
Skrev LaTeX-dokumentet	Analys av testdata

Alla andra punkter i detta dokument gjordes tillsammans.

1.5 Tillvägagångssätt

Nästa kapitel kommer gå igenom mer ingående om hur 3G och relaterade nätverksprotokoll fungerar, samt olika tekniker som spelutvecklare har utvecklat för att förbättra latensuppfattning i datorspel. Därefter, i kapitel 3, förklaras och utförs våra egna tester för latenstider i både en eget skriven applikation och Quake3-motorn Kwakk3 för Android. Följande kapitel består av tolkning av resultat och mätvärden från de utförda testerna och en mer överskådlig blick över vad som utförts och vilka slutsatser som gjorts.

Kapitel 2

Litteraturstudie

2.1 Smartphones olika nätverksprotokoll

2.1.1 Översikt av 3G som används i Europa

Om man räknar mobilradiokommunikationen som den första (0G) är 3G den tredje generationen av än så länge sex stycken standarder för mobil- och telekommunikation. Generation 4 till 6 (4-6G) är ännu under utveckling och har inte satts som standard.

3G utvecklades fram som standard framförallt på grund av den snabba ökningen av efterfrågan på IP-tjänster.[1] 3G:s genomslagskraft kommer främst från den höga datatransportshastigheten som denna standard kunde erbjuda mobilmarknaden. 3G är ett mobilradio- och nätverkåtkomstschema som ger trådlösa mobila nätverk, tack vare den höga överföringshastigheten, tillgänglighet att överföra mer än bara text- och rösttjänster, såsom att skicka grafiska gränssnitt och föra videosamtal.

IMT-2000, *International Mobile Telecommunications-2000*, är den specifkation, som är en sammanslagning av de olika metoder och plattformar för snabba datatransportstjänster, som använder sig av en eller ett fåtal nätverksplattformar.[2] Enligt IMT-2000 tillhandahåller 3G-enheter en hastighet av 2 Mbps(Megabits per sekund) för stillastående apparater inomhus, 384 kbps(kilobits per sekund) för apparater utomhus och även i låg rörelse (gånghastighet) samt 144 kbps för apparater i fordonshastigheter.

När 3G och specifikationen IMT-2000 skulle implementeras världen över föreslogs olika tekniska lösningar, som skulle uppfylla IMT-2000:s krav, från olika standardiseringsorganisationer. I Europa och för Sverige beslutade ETSI, *European Telecommunications Standards Institute*, att använda lösningen *Wideband Code Division Multiple Access* (WCDMA) med *Frequency Division Duplex* (FDD).[3]

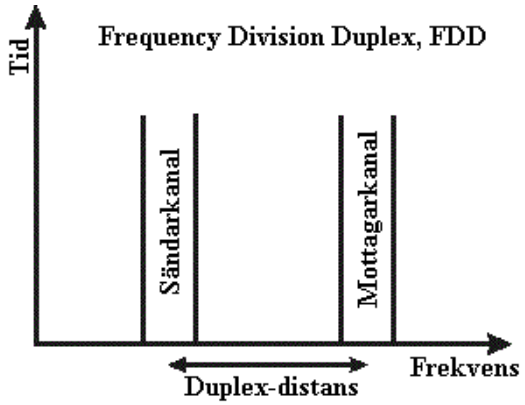
[1] C Smith, D Collins, *3G Wireless Networks*, The McGraw-Hill Companies, USA, 2002, sid. 136[7]

[2] C Smith, D Collins, sid. 136[7]

[3] C Smith, D Collins, sid. 141[7]

2.1.1.1 Frequency Division Duplex, FDD

FDD är en teknik som används vid radiokommunikation och bygger på att sända och mottagna operationer skickas på olika frekvenser, vilket kan skådas i figur 2.1.



Figur 2.1: Ett diagram över FDD-tekniken med avseende på tid och frekvens. Duplex-distans är det mellanrum som krävs för att det inte ska uppstå konflikter mellan de olika frekvenserna på sändar- och mottagarkanalerna.

För Europas valda kommunikationsgränssnitt i luft (WCDMA som tas upp i 2.1.1.2) är storleken på kanalerna, det vill säga både mottagar- och sändarkanaler, är 5 MHz(mega Hertz) och har en duplex-distans på 190 MHz. Sändarkanalerna har en frekvens mellan 1920 MHz och 1980 MHz medan mottagarkanalerna ligger mellan 2110 MHz och 2170 MHz. [4] När en signal skickas allokeras en sändar- och mottagarkanal, ett frekvenspar, och för att FDD ska fungera korrekt måste avståndet mellan kanalerna uppfylla kravet på 190 MHz.[5] I och med en fixt avstånd blir det lättare att byta mellan att skicka och ta emot då det endast krävs att lägga till, eller ta bort, 190 MHz.

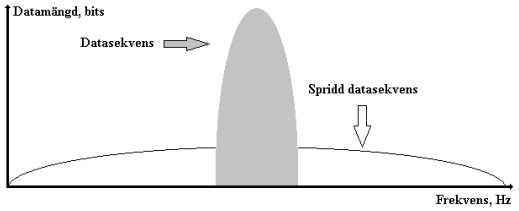
2.1.1.2 Wideband Code Division Multiple Access, WCDMA

WCDMA är som tidigare angivet ett gränssnitt för kommunikation i luft och bygger på tekniken *Directed-Sequence Code Division Multiple Access*, DS-CDMA. En signal, datasekvens, som skickas med metoden DS-CDMA sprids över ett större frekvensintervall med multiplikation av en *spridningskod*, vilket kan skådas i figur 2.2.[6]

[4] C Smith, D Collins, sid. 149[7]

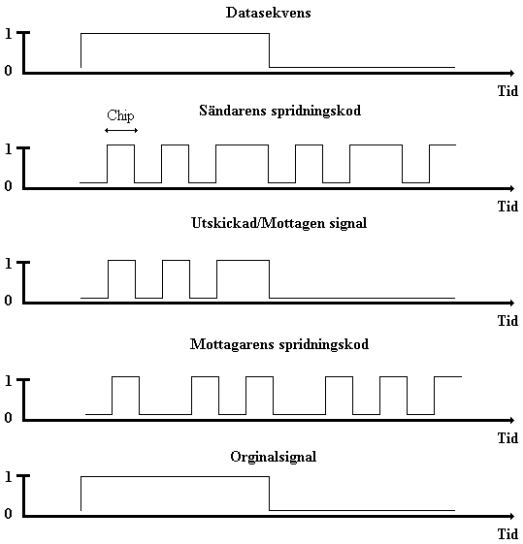
[5] Radio-Electronics.com, "TDD FDD Duplex Schemes", för Adrio Communications Ltd, läst 2012-04-05, `http://www.radio-electronics.com/info/cellulartelecomms/cellular_concepts/tdd-fdd-time-frequency-division-duplex.php`[8]

[6] Moshavi S, "Multi-user detection for DS-CDMA Communications", uppdaterad 2012-01-25, läst 2012-04-10, `http://www.stanford.edu/class/ee360/archived_2010/suppRead/read1/mud_moshavi.pdf`[9]



Figur 2.2: Visar skillnaden mellan en datasekvens som har multiplicerats med en spridningskod (Spridd datasekvens) och orginalfrekvensen.

Spridningskoden består av en sekvens pseudo-slumpmässigt genererade bitar, även kallade för *chips*, som både sändaren och mottagaren känner till. Frekvensen på denna spridningskod är högre än datasekvensens, det vill säga förekomsten av chip är högre än faktisk data. Genom att utföra denna multiplikation, mellan sändarens datasekvens och spridningskoden, får den utskickade signalen (uppström) en hög frekvens. I och med att spridningskoden är nästintill unik antar uppströmmen ett speciellt utseende. Utan den rätta spridningskoden uppfattas uppströmmen som oförståeligt brus.[7]



Figur 2.3: Spridningsprocessen när ett ensamt sändar-mottagarpar kommunicerar.

I de situationer där datasekvensen har samma frekvens, det vill säga fler sändare kommunicerar samtidigt med samma frekvens och destination, behöver spridningskoderna vara olika för respektive sändare. Om mottagaren vill få ut en specifik datasekvens *avsprids* signalen med den rätta spridningskoden och resultatet kommer bli den sökta datasekvensen plus ett litet brus. Detta lilla brus är förekomsten av de

[7]C Smith, D Collins, sid. 146[7]

andra skickade signalerna, eftersom den mottagna signalen (nedström) innehåller alla sända signaler.

För att mottagaren ska kunna tolka en avspridd datasekvens rätt måste sambandet mellan antalet *avspridda bitar* och *brusfaktorn*, det vill säga bruset som uppstår av andra skickade signaler, vara hög. En annan relevant faktor för att kunna återfå uppströmsdata vid mottagning är den så kallade *spridningsfaktorn* (SF):

$$SF = \frac{chipförekomst}{dataförekomst} \tag{2.1}$$

WCDMA har en bestämd chipförekomst på $3,84 \cdot 10^6$ chips/s och en dataförekomst beroende på vilken tjänst man utnyttjar.[8] Eftersom WCDMA även har stöd för felkorrigering av hälften data som skickas, vilket innebär extra kod, tillåts endast värdena i tabellerna nedan för upp- respektive nedströmning av data.

Spridningsfaktor	(Brutto) Överförings-hastighet i kbps	Faktisk dataförekomst i kbps med antagande att hälften är felkorrige-ringskod
256	15	7.5
128	30	15
64	60	30
32	120	60
16	240	120
8	480	240
4	960	480

Tabell 2.1: Överföringshastigheter och spridningsfaktorer för dataöverföring uppströms.

[8] UMTSWorld.com, "CDMA Overview", uppdaterad 2012-05-09, läst 2012-04-10, `http://www.umtsworld.com/technology/cdmabasics.htm`[10]

Spridningsfaktor	(Brutto) Överföringshastighet i kbps	Teoretiskt värde för dataförekomst i kbps med mindre felkorrigeringskod	Faktisk dataförekomst i kbps med antagande att hälften är felkorrigeringskod
512	15	3-6	1-3
256	30	12-24	6-12
128	60	42-51.2	21-25
64	120	90	45
32	240	210	105
16	480	432	216
8	960	912	456
4	1920	1872	936

Tabell 2.2: Överföringshastigheter och spridningsfaktorer för dataöverföring nedströms. Märkvärt är skillnaden i antal spidningsfaktorer och dubbel dataförekomst.

Utifrån tabell 2.2 kan skådas några skillnader mellan upp- och nedströmning av data. Nedströmning har stöd för ytterligare en spridningsfaktor och att dataförekomsten är den dubbla vid mottagande, som är bestämt utifrån UMTS. Felkorrigeringenskoden mellan ett flertal sändare samt hur deras respektive uppströmar är sammansatta till en och samma datasekvens, ger upphov till att den faktiska dataförekomsten för nedströmning inte är exakt dubbla uppstömmens.[9]

WCDMA erbjuder simultan upp- och nedströmning på multipla kanaler, för att uppfylla kravet på 2Mbps för en ensam användare. Exempelvis görs detta genom att öppna fem kanaler med en spridningsfaktor av 4 vilket resulterar i en hastighet av 2,4 Mbps. En annan viktig egenskap som WCDMA har är att strömningen av data inte behöver vara statisk utan kan ändras tack vare *10 ms-strukturen* som sändarkanalerna använder sig av. Detta är en teknik som utvecklats vidare och resulterat i form av *High Speed Packet Access*, som har en *2ms-struktur*, som förklaras i avsnitt 2.1.1.3.

2.1.1.3 High Speed Packet Access en utveckling av 3G

High Speed Packet Access eller HSPA är en annan benämning av *High Speed Uplink Packet Access* (HSUPA) och *High Speed Downlink Packet Access* (HSDPA) som är protokoll utvecklande för att förbättra WCDMA. Denna service är energikrävande och aktiveras temporärt vid höga belastningar på bandbredden.[10] HSDPA kan

[9] C Smith, D Collins, sid. 148[7]

[10] Seidel E, "Technology of High Speed Packet Access (HSPA)", uppdaterad Oktober 2006, läst 2012-04-11, `http://www.nomor.de/uploads/b0/2m/b02mwrVvIa5ZUtVrFeSP1w/Technology_of_HSPA.pdf` sid. 1[11]

komma upp i en temporär hastighet av 14,4 Mbps och HSUPA 5,76 Mbps, men användare av denna service är garanterade av en genomströmmande hastighet av minst 1 Mbps.[11] Tekniken som HSPA använder sig av baserar sig på att allokeringen för strömingen av data delas upp dynamiskt mellan olika kanaler, istället för att vara statisk. Tack vare 2ms-strukturen, det vill säga datasekvensens längd, går det att modulera den skickade datasekven som en ihopsättning av flera olika kanalers dataströmmar. Detta betyder i sin tur att det går att få mer gjort under en kortare tid eftersom fler kanaler skickar sin data samtidigt istället för att vänta på sin tur, vilket medför mindre latenser.[12]

2.1.2 Transportprotokollet UDP

UDP eller *User Datagram Protocol* är ett protokoll för att hantera transport av datatrafik över LAN och WLAN. För att ska skapa en UDP-koppling mellan datorer krävs inget genmensamt utbyte av data, utan UDP kan sända data mot en IP-adress vare sig den tar mot eller inte.[13] Det är alltså upp till mjukvaruutvecklaren som använder sig av UDP att implementera hur den ska hantera paket eller paketförluster.

UDP används främst av applikationer som inte behöver en absolut pålitlighet för den data som skickas.[14] De applikationer som påverkas negativt av latensökningar, som är konsekvensen av andra protokolls pakethantering är också ett bra användningsområde för UDP. Exempelvis I FPS-spel där de senaste uppdateringarna är det viktigaste medan vad som hänt inte är lika viktigt.

2.2 Om spelmotorer och latenskompensationstekniker

De allra flesta multiplayer-spel idag har en enkel server-klientside-arkitektur.[15] Anledningen till varför det fungerar bäst är helt enkelt att den server som hanterar spellogiken kan förhindra klienter (spelare) från att fuska. Om spelscenarion som inträffar lokalt på en klient skulle gälla för alla klienter kan detta utnyttjas för att skriva fusk. Därför väljer spelutvecklare att ha en auktoritär server som beslutar vilka kommandon som går igenom.[16,17]

[11] GSMA, "HSPA", uppdaterad 11-04-2012, läst 11-04-2012, `http://www.gsma.com/hspa`[12]

[12] Seidel E, "Technology of High Speed Packet Access (HSPA)", uppdaterad Oktober 2006, läst 2012-04-11, `http://www.nomor.de/uploads/b0/2m/b02mwrVvIa5ZUtVrFeSP1w/Technology_of_HSPA.pdf` sid. 1-2 [11]

[13] R Prasad, L Muños, WLANs and WPANs towards 4G Wireless, Artech House, USA, 2003 sid. 87[13]

[14] R Prasad, L Muños, sid. 87[13]

[15] Y W Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization", uppdaterad 2003-10-28, läst 2012-04-12, `http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf`, sid. 2[14]

[16] Y W Bernier, sid. 7[14]

[17] Fiedler G, "What every programmer needs to know about game networking", uppdaterad 2010-01-24, läst 2012-04-12, `http://gafferongames.com/networking-for-game-programmers/`

Server-klient-arkitekturen fungerar på så vis att klienten skickar de kommando eller rörelser som användaren utför till en server. Serven, som är kopplad till en eller flera andra klienter, tar mot klientskapad data och uppdaterar alla spelobjekt i den virtuella världen. Den uppdaterade versionen skickas sedan till alla klienter så att de kan rendera samma spelvärld.

De problem som kan uppstå i spelupplevelsen, beskrivet i enkla drag, är den upplevda tidsdifferens(latens) som uppstår när spelaren utför kommandon lokalt som ska skickas till servern för att sedan vänta på svaret. Svaret från servern krävs för att klienten ska kunna rendera den nya skärmbild som kommandot gett upphov till.[18]

En teknik man utvecklat för att få spelarens handlingar att kännas mer direkta, är att man simulerar samma uppdatering som sker på servern, lokalt på klienten.

2.2.1 Klientsidoprediktion (Client Side Prediction)

Klientsidoprediktion bygger alltså på att servern delar med sig av sitt sätt att uppdatera spelvärlden till alla andra klienter. Med hjälp av den informationen kan klienterna simulera spelvärldsuppdateringen parallellt med servern, vilket leder till att spelaren upplever att deras kommandon utförs direkt. I och med att klienten renderar spelarens kommandon parallellt som den skickar de till servern så behöver klienten inte vänta på serverns respons för att visuellt visa resultatet som spelarens handling har i spelvärlden.[19]

Om klientsimuleringen stämmer överens med serverns version kommer alltså spelaren inte märka av när klienten byter från den lokalt simulerade spelvärlden till den slutgiltiga från servern. Däremot om den lokalt simulerade spelvärlden inte är synkroniserad från serverns kommer leder det till att spelklienten får rätta sig efter serverns upplaga av spelvärlden. Detta är ofta något som inte märks om skillnaderna är små, eftersom spelklienterna har algoritmer som som gör jämna övergångar vid små korrektionsrörelser. Om skillnaderna är för stora kommer spelklienten att teleportera spelaren till det stadium och position som servern skrev att den befann sig i.[20]

Klientsidoprediktion har vissa brister som främst träder fram när klienten uppdaterar position och dess rörelser i realtid eftersom det tar tid innan klientdatan når fram till servern vilket leder till att fienden hinner flytta sig under den tiden.[21] Detta leder till att spelaren måste gissa vart motspelaren kommer vara en liten bit in i framtiden och måste, för exempelvis FPS-spel, sikta på den troligaste platsen där fienden kommer stå "framåt i tiden" för att träffa.

`what-every-programmer-needs-to-know-about-game-networking/`[15]

[18] Y W Bernier, sid. 4[14]

[19] Y W Bernier, sid. 5[14]

[20] Y W Bernier, sid. 5[14]

[21] Y W Bernier, sid. 8[14]

Idag finns det flertalet tekniker för att minimera den upplevda latensen som uppstår vid liknande situationer. Men för att gå in på det så bör det förklaras hur rendering av motståndare på klientsidan fungerar vilket görs härnäst, i följande delkapitel.

2.2.2 Frame-extrapolation och frame-interpolation

För att skapa en realistisk och mjuk rörelse för ett objekt i ett spel, istället för en hackig varje gång servern skickar en uppdatering av spelvärlden, använder sig spelskapare främst utav två tekniker, den ena är frame-extrapolation och den andra frame-interpolation.[22]

Frame-extrapolation är när man på klientsidan använder motståndarens senaste position och tar dess hastighet och riktning och renderar den framåt i realtid utifrån det, till exempel om fienden rör sig i 400 enheter/s. När klienten interpolerar över 100 ms flyttas motståndaren 40 enheter under den tiden som följd av detta blir fiendens rörelser mer konstant istället för hackiga, nästan teleportliknande.

Svårigheten för denna teknik ligger i hur spel hanterar acceleration och deacceleration då spelare tillåts byta riktning fort i flertalet fps-spel, exempelvis Quake III Arena.[23] Den beräknade motståndarpositionen på klientsidan kan således vara långt ifrån korrekt om spelaren byter riktning precis innan servern skickade ut nästa uppdatering av världen. Detta kommer bli uppenbart i nästa uppdatering eftersom användaren kan observerat att spelare plötsligt teleporteras åt ett annat håll. I FPS-spel generellt är detta inget som uppskattas.[24]

Det andra sättet är att hantera renderingen av objektrörelser på klientsidan är interpolation. Frame-interpolation bygger på att det klienten visar är händelse förloppen mellan de senaste två uppdateringarna från servern. Detta försäkrar att rörelsen som objekt gör, i dåtid, är korrekt klientens perspektiv i och med man flyttar de mellan kända punkter och inte längre punkter som klienten antog att de skulle ta. Nackdelen med användning av frame-interpolation är att tekniken som krävs skickar spelarens händelsehorisont bakåt i tiden med en hel serveruppdatering, ty 2 punkter krävs för att den ska bli korrekt. Resultatet blir att latensen på klientsidan ökar med ett helt speluppdateringsintervall.

Om en server nu skulle skicka ut 66 (som är standardvärde för Counter-Strike 1.6 [25]) speluppdateringar i sekunden kommer klienten interpolera spelobjekt mellan det som skedde 15 ms innan till den senaste mottagna uppdateringen, för att därefter

[22] Y W Bernier, sid. 8[14]

[23] Y W Bernier, sid. 8[14]

[24] M Claypool, K Claypool, Damaa Feissal, uppdaterad 2006-01-19, läst 2012-04-12, `http://web.cs.wpi.edu/~claypool/papers/fr-rez/paper.pdf`[16], sid. 11[16]

[25] GameData Inc., "Tickrate / Net_Graph Overview", läst 2012-04-12, `http://www.counter-strike.com/faqs/tickrate/`[17]

få en ny uppdatering och interpolera vidare från den nu äldre versionen. Detta kommer dock bli proportionellt till spelarens egna latenstid, ty den senaste upptagna uppdateringen är beroende av hur snabbt data kommer från servern. Exempelvis om spelaren har 100 ms i latens kommer då det som visas visuellt hur fienden rörde sig för 100-115 ms sedan.

Med fördel används interpolation tillsammans med extrapolation vid händelser av paketförlust, i och med att det saknas punkter att interpolera mellan kan klienten övergå till extrapolation tillfälligt för att sedan gå tillbaka.[26] Trots att klienten har en väldigt korrekt visuell representation av fiendens position lider fortfarande renderingstekniken av det faktum att det visas en skuggbild, eller en dåtidens spelvärld. Detta på grund av att händelsehorisonten flyttas bakåt i tiden för att få en jämnare och mer korrekt visuell representation.

I vissa spel, till exempel Quake III Arena har spelutvecklarna valt att spela upp ett ljud varje gång spelaren träffar, detta gör att spelaren själv kan korrektera sitt sikte för att öka träffsäkerheten.

I praktiken används en blandning av interpolation och extrapolation för att rendera spelobjekt. Detta är för att extrapolation är ett bättre alternativ om spelpaket skulle förloras, då servern fortfarande kan generera en rörelse baserad på den senaste uppdateringen. Rörelser med latenstider som vanligtvis brukar extrapoleras och interpoleras beror på hur många speluppdateringar servern skickar ut. Exempelvis om servern skickar ut 20 uppdateringar per sekund, interpoleras rörelsen för en optimistisk prediktion med 50ms och extrapoleras med 50ms om man missar en uppdatering. För en mer pessimistisk prediktion interpoleras rörelsen med 100ms men istället undviker extrapolering. Nackdelen med interpolering framför extrapolering är att latens byts ut mot en bättre visuell representation av rörelsen, vilket gör att den upplevda latensen blir högre. Enligt Professor Claypools undersökning[27] så visas det en förlust på ungefär 35% träffsäkerhet med högprecisionsvapen i FPS-spel vid en latensnivå på 100ms, alltså bör en spelutvecklare inte använda sig av mer än 100ms interpolation om den inte tänkt använda sig av lagkompenserande tekniker.

2.2.3 Laggkompensation

Laggkompensation är när servern för går genom och beräknar varje spelare i spelet deras nuvarande latenstid, tar dess kommandon och sen “spolar tillbaka tiden”med hänsyn till latenstiden plus interpolationstiden (alltså hur lång tid det var mellan de två interpolerade punkterna när spelar kommandot utfördes). Först nu låter servern spelaren utföra sitt kommando, till exempel om spelaren sköt kollar servern om siktet var korrekt och om det i sin tur kan leda till en annan spelares död.[28]

[26] Y W Bernier, sid. 9[14]

[27] M Claypool, K Claypool, uppdaterad November 2006, läst 2012-05-22, `http://web.cs.wpi.edu/~claypool/papers/precision-deadline/final.pdf`[18]

[28] Y W Bernier, sid. 11[14]

Därefter när alla spelarkommandon har utförts spolas tiden, även alla objekt, fram till serverns realtid för att sedan skickas med nästa spelvärldsuppdatering. Detta kan leda till situationer i latenskänsliga spel där små tidsintervall har stor betydelse. Exempelvis om en spelare har som mål att ta skydd bakom ett runt hörn medan motståndaren med en förhållandevis hög latens siktar och skjuter spelaren som springer runt hörnet kommer känsligheten visa sig märkbar. Spelaren, som sprang runt hörnet, kommer uppleva att sig själv skjuten runt hörnet. Det är för att laggkompensationentekniken tar den laggade (hög latens) spelarens latens i beräkning och för den spelaren var skottet välskjutet medan den andra spelaren upplever att han kom undan.

Ett sätt att undvika de anomalier som uppkommer av användadet av laggkompensationstekniker är att sätta ett tak för hur hög latenstiden får vara, för en klient, för att undvika liknande situationer som den nämnd ovan.

Kapitel 3

Metod

3.1 Material

3.1.1 Hårdvara

Bärbar dator, Hewlett Packard Pavillion dv2, kopplad till Kungliga Tekniska högskolans trådlösa nätverk eduroam.

Smarttelefon, Android HTC Desire, med Tele2 SWIPnet upp till 6Mbps nedströms.

Smarttelefon, Android Samsung Google Galaxy Nexus, med Tele2 SWIPnet upp till 3 Mbps nedströms.

USB-kabel, USB-hane till microUSB-hane.

3.1.2 Mjukvara

Bredbandskollen, applikation för smarttelefon, mäter hastigheter över nätverk.

Kwakk3, ett mobilspel i Android som bygger på PC-spelet Quake III Arena.

UDP Droid, egenutvecklad applikation för latenstesting över UDP.

3.2 Utförande

3.2.1 Skapande av serverapplikation i Android

Testapplikationen som utvärderade latensen för 3G skrevs i för UDP. Anledning till detta var att UDP som transportprotokoll används i majoriteten av FPS-spel för att skicka data mellan server och klient. För att kunna efterlikna ett spel-protokoll så mycket som möjligt skrevs därför applikationen även den för UDP.

För testprogrammet utvecklades UDP-server och UDP-klient i Java från ett exempelskal och portade sedan dessa till ett Android-projekt därefter lades det till

funktionalitet som behövs för att mäta latenstider och och utvärdera paketförluster. Server-programmet utrustades även så det kunde skicka olika stora mängder data vilket gav en realistisk skildring av ett autentiskt spel.

För att mäta latenstider hämtades datorns klocka (*System.nanoTime*), därefter skickades ett datapaket. Efter att servern tog mot svaret på paketet som skickades skrev programmet ut tidsdifferansen i millisekunder. Server-programmet för att ta ut latensen är ungefär skrivet som nedan, se även bilaga A.

```
long start=System.nanoTime();
        sender.sendMessage(start);
        sender.receiveMessage();
        System.out.println((System.nanoTime()-start)/1000000);
```

För att kolla paketförluster skickade serverprogrammet 100 paket till Android-applikationen och kollade hur många klienten tog upp och vise versa.

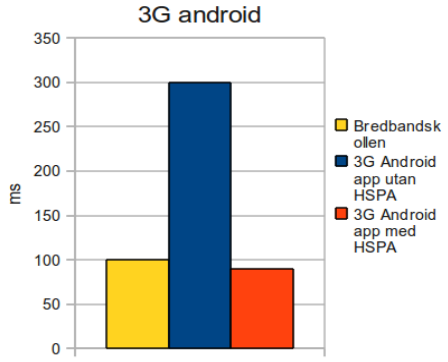
3.2.2 Setup av Kwakk

Kwakk3 är en mobilanpassad version av PC-spelet Quake III Arena(Quake3), som var ett mycket populärt spel under början av 2000, och har inbyggt stöd för prestandaövervakning - fps och latens.

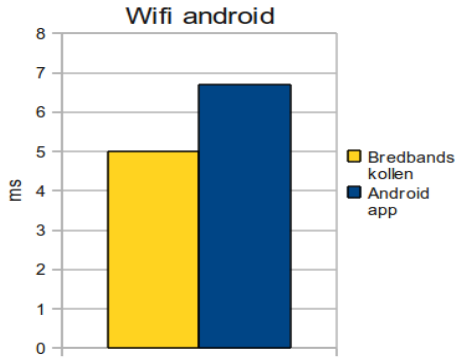
Först laddades applikationen kwaak3.apk ned och installerades på smarttelefoner. Kwaak3 kräver själva spelmotorn, som finns i demoversionen av Quake3, och även senaste versionen(1.32) av datorspelet. Spelmotorn kunde extraheras från filerna pak0.pk3 från demot, och pak1-pak8.pk3 från versionspaketet. Dessa filer lades sedan in på smarttelefonerna i mappen quake3/baseq3.

För att få multiplayerspel att fungerar mellan smarttelefonerna skrevs IP-adressen för servern (någon av smarttelefonerna) in i filen autoexec.cfg i quake/base3. Latensmätningsverktyget i Kwaak3 aktiverades genom att skriva till “seta cg_lagometer ‘1´” i q3config.cfg.

3.3 Empiriskt resultat

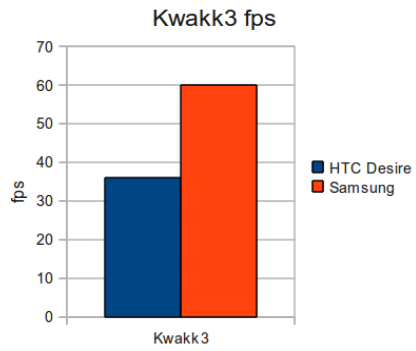


(a) Test över 3G med UDP Droid, mot egen server (bärbardator) och Bredbandskollen.

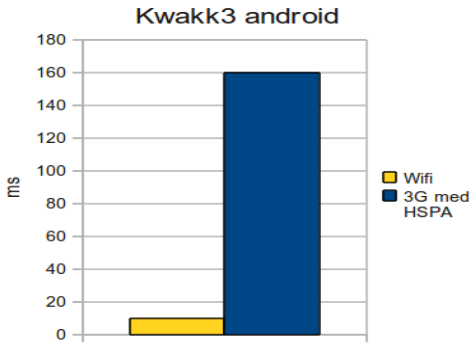


(b) Test över WLAN med UDP Droid, mot egen server (bärbardator) och Bredbandskollen.

noindent



(c) Medelvärde av fps-nivåer mätt med Kwakk3s egna latensmätningsverktyg.



(d) Latenstider i Kwakk3 mätt med HTC Desire.

Figur 3.1: Mätdata från tester.

Samtliga mätdata är tagna klockan 11:00-12:00 samma dag, den 10:e april, i Kungliga Tekniska högskolans datorsal på våning fyra, i E-huset. Det relativa avståndet mellan smarttelefon och dator var kort för samtliga tester. Testning med en smarttelefon, för WLAN och 3G (figur 3.1 a) och b)), utfördes i tre mätningar för varje deltest och värdet mellan varje test hämtades ut. Samma sak gjordes för mätningarna mot Bredbandskollen. Värderna som visas i graferna är ett aritmetiskt medelvärde taget från den egna applikationen, där vi helt enkelt tagit alla latenstider för alla paketen och dividerat dem på antalet mottagna paket.

Kwaak3-testet kördes mellan de två smarttelefonerna som låg precis intill varand-

ra. Mätvärdena var tagna genom att låta en av smarttelefonerna köras som server, som den andra kopplade upp sig mot, sedan användes spelets egetberäknade latensfunktion för att få fram mätvärdena. Vi kunde även mäta, med hjälp av Kwakk3s inbyggda funktion “cg_drawfps ‘1”’ fås-nivåerna vid spel.

Resultaten visade inga större paketförluster, 1/100 paket i ett par fall i användandet av 3G och inga för WLAN. Däremot hade 3G-testen stora latensvariationer där de varierade mellan 250-450 ms utan HSPA och 60-120 ms med HSPA aktiverat.

Genom att aktivera spelapplikationen Wordfeuds meny, automatiskt hämtar uppdateringar till spelet, aktiverades HSPA i smarttelefonen. Således reducerades latenstiderna i applikationen med ungefär 210 ms, vilket gjorde medellatensen gick från 300 ms till 90 ms.

Kwakk3s fps-nivåer på den hårdvara som testade gav önskevärda resultat där fps-nivåer kring 30-40 anses godtagbart för att kunna spela utan avsevärda prestanda förluster.

Kapitel 4

Analys

I början av testen med den egetproducerade applikation uppnåddes latenser kring 300 ms, ty applikationen automatiskt använde "vanlig 3G" och inte HSPA. Det var möjligt att kringå detta genom att köra applikationen i bakgrunden för att sedan öppna en annan applikation som använder sig utav HSPA.

Det finns ett stort behov av lagg-kompensationstekniker framförallt vid mycket ojämna latenstider, vid FPS-spel, eftersom det blir mer påfrestande för spelaren då man överlåter denne själv att kompensera för fördröjningen genom att sikta framför fienden. Eftersom latensen helatiden pendlar kraftigt blir det ännu svårare för spelaren att förutse sin motståndares rörelser, då den reella positionen varierar proportionellt med latensen. Lagg-kompensationen tillåter en spelare att sikta rakt på en fiende och träffa. I och med detta bidrar lagg-kompensationen till ett jämnare spel även om latensena mellan dem är märkbart olika.

Latenstiderna i den egna applikationen blev likgiltiga Bredbandskollens applikation. Eftersom testerna via 3G-nätet inte körs parallellt bör skillnaderna tolkas som normala variationer i 3G-nätet, ty mätningarna skedde vid olika tidpunkter.

Det var dock svårare att tolka utifall HSPA användes i Kwakk3 eller inte, då spelet använder ett eget sätt att mäta latenstider. Indikationsikonen för vilken nätverksuppkoppling som Android-operativsystemet använder sig av syns inte inne i Kwakk3-applikationen. De mätvärden vi fick med applikationens inbyggda latensmätare låg omkring 160-200 ms. Vilket torde tyda på att 3G-nätet inte prioriterar trafiken korrekt, eller att mätningarna gav ett inkorrekt resultat.

Mer positivt och kanske mer väntat erhölls väldigt låga latenstider med vanligt WLAN där mätningen visade ett medelvärde på 10 ms. Detta påvisar att alternativet, med WLAN som vald nätverksuppkoppling, högst lämpligt för gaming i FPS-spel. Däremot utifrån hårdvarusynpunkt erhölls goda resultat då smarttelefonerna visade en fps-nivå på minst 40, vilket är enligt mätresultaten acceptabla nivåer för gaming.

Kapitel 5

Slutsats

Förutsättningarna för spelutveckling och taktilutveckling för realtidsmultiplayer-spel med smarttelefoner som plattform är att det finns en grund som gör de möjliga att utvärdera. Grunden i det här fallet är att det finns en hårdvara som presterar tillräckligt bra och ger då bra latenstider för kommunikation mellan mobila enheter.

Våra resultat visar att hårdvaran i dagens smarttelefoner kan hantera 3D-spel av enklare sort och med WLAN som vald uppkoppling få bra latenstider. Däremot framgår av våra tester att latenstiderna på 3G-nätet är väldigt varierande och detta medför att det skulle bli näst intill omöjligt för en spelare att kompensera för sin egen latens då den svänger kraftigt. Av den anledningen rekommenderar vi att spelutvecklare använder sig av laggkompensation när de utvecklar framtida multiplayer-spel för smarttelefoner.

Kapitel 6

Framtid

Att utvärdera huruvida latensen påverkade spelbarheten i Kwakk3 visade sig väldigt svårt då det finns så pass många andra faktorer när man spelar på en mobil enhet. Ett exempel är att fingerrörelserna inte är lika precisa som spel med datormus och anomalier där skärmbilden väldigt snabbt ändras när man drar på skärmen med två fingrar. Detta låg utanför vår projektspecifikation men känns ändå som ett värdefullt område att forska vidare i. Detta hade exempelvis kunnat göras med att koppla in smarttelefonen till en bildskärm samt ett tangentbord för att styra och förbättra upplevelsen. Alltså att utvärdera smarttelefonen som en spelplatform i helhet. Detta var en konceptuell tanke som tilltalade oss i projektets start, men visade sig svår att genomföra då vi saknade hårdvara för att genomföra det.

Litteraturförteckning

[1] Ziebart A, “World of Warcraft dips to a mere 11.4 million”, för AOL Inc, uppdaterad 2011-05-09, läst 2012-03-06, http://wow.joystiq.com/2011/05/09/world-of-warcraft-dips-to-a-mere-11-4-million-subscribers/,

[2] Crecente B,“Call of Duty: Modern Warfare 3 Shatters All Sales Records, Nears Half a Billion Dollars in Day One Sales”, för Kotaktu, uppdaterad 2011-11-11, läst 2012-03-06, http://kotaku.com/5857400/call-of-duty-modern-warfare-3-shatters-all-sales-records-surprises-no-one

[3] Howlan G, “What is Lag?”, för GameDev.Net LLC, uppdaterad 1999-09-14, läst 2012-03-19 http://www.gamedev.net/page/resources/_/technical/multiplayer-and-network-programming/what-is-lag-r712

[4] Apple Inc., “Popular Apps”, läst 2012-04-03 , http://itunes.apple.com/us/genre/ios-games/id6014?mt=8

[5] Google Inc., “Bästsäljande i Appar”, läst 2012-04-03, https://play.google.com/store/apps/collection/topselling_paid?feature=top-paid

[6] CBS Interactive Inc., “Most Popular PC Games”, läst 2012-04-03, http://www.gamespot.com/games.html?platform=5&mode=all&sort=views&dlx_type=all&sortdir=asc&official=all

[7] Smith C, Collins D, *3G Wireless Networks*, The McGraw-Hill Companies, USA, 2002

[8] Radio-Electronics.com, “TDD FDD Duplex Schemes”, för Adrio Communications Ltd, läst 2012-04-05, http://www.radio-electronics.com/info/cellulartelecomms/cellular_concepts/tdd-fdd-time-frequency-division-duplex.php

[9] Moshavi S, “Multi-user detection for DS-CDMA Communications”, uppdaterad 2012-01-25, läst 2012-04-10, http://www.stanford.edu/class/ee360/archived_2010/suppRead/read1/mud_moshavi.pdf

[10] UMTSWorld.com, “CDMA Overview”, uppdaterad 2012-05-09, läst 2012-04-10, http://www.umtsworld.com/technology/cdmabasics.htm

[11] Seidel E, "Technology of High Speed Packet Access (HSPA)", uppdaterad Oktober 2006, läst 2012-04-11, http://www.nomor.de/uploads/b0/2m/b02mwrVvIa5ZUtVrFeSP1w/Technology_of_HSPA.pdf

[12] [gsma] GSMA, "HSPA", uppdaterad 2012-04-11, läst 2012-04-11, http://www.gsma.com/hspa

[13] Prasad R, Muños L, WLANs and WPANs towards 4G Wireless, Artech House, USA, 2003

[14] Bernier W Y, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization", uppdaterad 2003-10-28, läst 2012-04-12, http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf

[15] Fiedler G, "What every programmer needs to know about game networking", uppdaterad 2010-01-24, läst 2012-04-12, http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/

[16] Claypool M, Claypool K, Damaa Feissal, uppdaterad 2006-01-19, läst 2012-04-12, http://web.cs.wpi.edu/~claypool/papers/fr-rez/paper.pdf

[17] GameData Inc., "Tickrate / Net_Graph Overview", läst 2012-04-12, http://www.counter-strike.com/faqs/tickrate/

[18] Claypool M, Claypool K, uppdaterad November 2006, läst 2012-05-22, http://web.cs.wpi.edu/~claypool/papers/precision-deadline/final.pdf

Bilaga A

Kod till Android-applikation

A.1 KexudpandroidActivity.java

```
package my.first.udpandroid;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;

public class KexudpandroidActivity extends Activity {

TextView t;
//A button used to create the listener on udp ports on the android device to activate t
Button button2;
// list with containing messages
private ListView msgList;
// Not super usefull but provides the neccessary functionality required to use the msgl
private ArrayAdapter<String> received;
private EditText input;


    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.main);
        input = (EditText) findViewById(R.id.txt_input);
        received = new ArrayAdapter<String>(this, R.layout.msgitem);
        msgList = (ListView) findViewById(R.id.txt_output);
        msgList.setAdapter(received);
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(btnSendListener);
    }
    //Help method to write to the list of messages, not really used but gives some sort
        private void sendMessage2(String message) {
     String newStr = input.getText().toString();
     if(newStr!=null&&newStr!="") {
received.add("me:"+newStr);
input.setText("");
     }
    }
    // listener for the Onclick event that is triggered by pressing the button,
    // starts a new thread which listens on the android ports and sends back messages w
    // to simulate a ICMP protokoll
    private OnClickListener btnSendListener = new OnClickListener() {
        public void onClick(View v){
         sendMessage2("initializing udp ports:");
         new Thread(new UDPReceiver()).run();
        }
    };

    }
```

A.2 UDPReceiver.java

```
package my.first.udpandroid;
/**
 * This Java J2SE 5.0 source code is in the public domain. You are free to use
 * and modify it as you wish, but it comes with absolutely no warrantee.
 */

import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;


/**
 * Sample class illustrating UDP networking by receiving simple text messages
 * via UDP; use UDPSender to send the messages.
 *
 * @author Jennifer Hodgdon, Poplar ProductivityWare, www.poplarware.com in
 *         conjunction with an article at:
 *         http://www.poplarware.com/udpjava.html
 *
 * @see UDPSender
 */
public class UDPReceiver extends Thread {

    /** Port to use for UDP receiving. */
    public static final int Recieving_PORT = 7777;
    public static final int Sending_PORT = 9000;

    /** Size of receive buffer, in bytes. */
    public static final int BUFFER_SIZE = 200;

    private String TargetAddress="130.229.136.132";
    /** UDP Socket object to use for networking. */
    private DatagramSocket sock;

    /** UDP Packet object to use for receiving message. */
    private DatagramPacket pack;

    /** Buffer to store received text in. */
    private byte[] receiveBuffer;

    /** Default constructor: sets up networking. */
    public UDPReceiver(){
```

```
        receiveBuffer = new byte[BUFFER_SIZE];

        try {
            sock = new DatagramSocket( Recieving_PORT );


            pack = new DatagramPacket( receiveBuffer, receiveBuffer.length );
        } catch( Exception e ) {
            System.out.println( "Exception in creating DatagramSocket or Packet: "
                    + e.getMessage() );
        }
    }

    /**
     * Receives one UDP message, sends the message back, and resets buffer so it's
     * ready to receive again.
     */
    private void receiveSendMessage() {
        try {

            sock.receive( pack );
            String msg = new String( receiveBuffer, 0, pack.getLength() );
            sendMessage(msg);
            System.out.println( "Message received: " + msg );
            pack.setLength( receiveBuffer.length );
        } catch( Exception e ) {
            System.out.println( "Exception in receiving packet: "
                    + e.getMessage() );
        }
    }
/**
 * skickar medelandet tillbaka till den förvalda IP-adressen
 * @param msg message to be sent
 */
    private void sendMessage(String msg) {
        try {
         DatagramPacket spack=null;
         DatagramSocket ssock = new DatagramSocket();
         byte[] buf = msg.getBytes();
         spack = new DatagramPacket( buf, buf.length,
                    InetAddress.getByName(TargetAddress), Sending_PORT );
            ssock.send( spack );
            ssock.close();
            System.out.println( "Message sent." );
```

```
        } catch( Exception e ) {
            System.out.println( "Exception in sending packet: "
                    + e.getMessage() )
                    ;
        }
    }

    /**
     * Repeatedly receives simple UDP messages from the specified port.
     *
     * @param args Arguments are ignored.
     */
    @Override
    public void run(){
     for(int i=0; i<200; i++) {
     receiveSendMessage();
     }
     sock.close();
    }



}
```

A.3 UDPReceiver.java

```
package my.first.udpandroid;
/**
 * This Java J2SE 5.0 source code is in the public domain. You are free to use
 * and modify it as you wish, but it comes with absolutely no warrantee.
 */

import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;

/**
 * Sample class illustrating UDP networking by sending simple text messages via
 * UDP; use UDPReceiver to receive the messages.
 *
 * @author Jennifer Hodgdon, Poplar ProductivityWare, www.poplarware.com in
 *         conjunction with an article at:
 *         http://www.poplarware.com/udpjava.html
 *
 * @see UDPReceiver
 */
public class UDPSender extends Thread{

    /** Port to use for UDP sending. */
    public static final int Sending_PORT = 7777;
    public static final int Receiving_PORT = 9000;

    /** Address to use for UDP sending. */
    public static final String UDP_ADDRESS = "127.0.0.1";


    /** Time to wait between sending messages, in milliseconds. */
    public static final int SLEEP_MILLISECS = 1000;

    /** UDP Socket object to use  for networking. */
    private DatagramSocket sock;
    private DatagramSocket ssock;
    private byte[] receiveBuffer;
    /** UDP Packet object to use for sending message. */
    private DatagramPacket pack;
    private DatagramPacket spack;
    public static final int BUFFER_SIZE = 200;
```

```
    /** Default constructor: sets up networking. */
    public UDPSender(){


    receiveBuffer = new byte[BUFFER_SIZE];

        try {
            sock = new DatagramSocket();
            ssock = new DatagramSocket( Receiving_PORT );
            spack = new DatagramPacket( receiveBuffer, receiveBuffer.length );

        } catch( Exception e ) {
            System.out.println( "Exception in creating socket or packet: "
                    + e.getMessage() );
        }
    }

    /** Sends out one UDP message and notes it was sent on screen. */
    private void sendMessage(long msg) {
        try {
         byte[] buf = new Long(msg).toString().getBytes();
         pack = new DatagramPacket( buf, buf.length,
                    InetAddress.getByName( UDP_ADDRESS ), Sending_PORT );
            sock.send( pack );
            System.out.println( "Message sent." );
        } catch( Exception e ) {
            System.out.println( "Exception in sending packet: "
                    + e.getMessage() )
                    ;
        }
    }


    private long receiveMessage() {
        try {

            ssock.receive( spack );
            String msg = new String( receiveBuffer, 0, spack.getLength() );
            System.out.println( "Message received: " + msg );
            spack.setLength( receiveBuffer.length );
            return new Long(msg);

        } catch( Exception e ) {
            System.out.println( "Exception in receiving packet: "
```

```
                    + e.getMessage() );
        }
return 0;
    }


    /**
     * Repeatedly sends simple UDP messages to the specified port and address.
     *
     * @param args Arguments are ignored.
     */
    @Override
public void run(){
        UDPSender sender = new UDPSender();
        while( true ) {
            sender.sendMessage(System.nanoTime());
            long stop =sender.receiveMessage();
            System.out.println((System.nanoTime()-stop)/1000000);
            try {
                Thread.sleep(SLEEP_MILLISECS);
            } catch( Exception e ) {
//                  do nothing
            }
        }
}
}
```

A.4 UDPSender.java

```
/**
 * This Java J2SE 5.0 source code is in the public domain. You are free to use
 * and modify it as you wish, but it comes with absolutely no warrantee.
 */

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.SocketException;
import java.util.Date;

/**
 * Sample class illustrating UDP networking by sending simple text messages via
 * UDP; use UDPReceiver to receive the messages.
 *
 * @author Jennifer Hodgdon, Poplar ProductivityWare, www.poplarware.com in
 *         conjunction with an article at:
 *         http://www.poplarware.com/udpjava.html
 *
 * @see UDPReceiver
 */
public class UDPSender {

    /** Port to use for UDP sending. */
    public static final int Sending_PORT = 7777;
    public static final int Receiving_PORT = 9000;

    /** Address to use for UDP sending. ANDROID IP*/

    //LOCAL Test ip
    public String UDP_ADDRESS = "127.0.0.1";

    //CH MOBIL IP
    //public static String UDP_ADDRESS = "176.64.115.61";
    // CH MOBIL KTH
//    public static final String UDP_ADDRESS = "130.229.133.94";
//    public static final String UDP_ADDRESS = "130.229.143.246";
```

```
    /** Time to wait between sending messages, in milliseconds. */
    public static final int SLEEP_MILLISECS = 1000;

    /** UDP Socket object to use  for networking. */
    private DatagramSocket sock;
    private DatagramSocket ssock;
    private byte[] receiveBuffer;
    /** UDP Packet object to use for sending message. */
    private DatagramPacket pack;
    private DatagramPacket spack;
    public static final int BUFFER_SIZE = 200;

    /** Default constructor: sets up networking. */
    public UDPSender(String sendtoIP){

    this.UDP_ADDRESS = sendtoIP;
    receiveBuffer = new byte[BUFFER_SIZE];

        try {
            sock = new DatagramSocket();
            ssock = new DatagramSocket( Receiving_PORT );
            spack = new DatagramPacket( receiveBuffer, receiveBuffer.length );

        } catch( Exception e ) {
            System.out.println( "Exception in creating socket or packet: "
                    + e.getMessage() );
        }
    }

    /** Sends out one UDP message and notes it was sent on screen. */
    private void sendMessage(long msg) {
        try {
         byte[] buf = new Long(msg).toString().getBytes();
         pack = new DatagramPacket( buf, buf.length,
                    InetAddress.getByName( UDP_ADDRESS ), Sending_PORT );
            sock.send( pack );
//            System.out.println( "Message sent." );
        } catch( Exception e ) {
            System.out.println( "Exception in sending packet: "
                    + e.getMessage() )
                    ;
        }
    }
```

```
    /*recieves a messages
     *
     */
    private long receiveMessage() {
        try {
            ssock.receive( spack );
            String msg = new String( receiveBuffer, 0, spack.getLength() );
            spack.setLength( receiveBuffer.length );
            return new Long(msg);

        } catch( Exception e ) {
            System.out.println( "Exception in receiving packet: "
                    + e.getMessage() );
        }
return 0;
    }

    /**
     * a very simple package test, that gives each package 2 seconds of time to return
     * then its tossed.
     * @return the ammount of packes that was not dropped out of 100 packages
     * @throws SocketException
     */

    private int packagetest() throws SocketException{

     ssock.setSoTimeout(2000);
     int count=0;
     for(int i=0; i<100;i++){
     sendMessage(1234);
     try {
ssock.receive(spack);
count ++;
} catch (Exception e) {

System.out.println("PACKAGE NR: "+i+" was lost");
}

     }

     return count;
    }
    /**
     * Repeatedly sends simple UDP messages to the specified port and address.
```

```
     *
     * @param args Arguments are ignored.
     * @throws IOException
     */
    public static void main( String[] args ) throws IOException {
     // creates a udp sender with a specific adress to send to
        UDPSender sender = new UDPSender("130.229.179.238");
        // the filestream used to write to our logfile
        FileWriter fstream = new FileWriter("Logfile",true);
        BufferedWriter out = new BufferedWriter(fstream);
        out.append(new Date().toString()+" Starting test"+"\n");

        // PACKAGE TEST FIRST
        int testpackages=0;
        try {
testpackages = sender.packagetest();
} catch (SocketException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}
        // printing package test results to system out and file.
        System.out.println(testpackages+" out of 100 recieved. Succesrate = "+(float)((
        out.append(testpackages+" out of 100 packages recieved. Succesrate = "+(float)(
        int ping=0;
        int totaltime=0;
        long start=0;
        // PING test
        for(int i =0; i<100; i++) {
         start=System.nanoTime();
            sender.sendMessage(start);
            sender.receiveMessage();
            ping=(int)(System.nanoTime()-start)/1000000;
            if(ping>0){
            totaltime=totaltime+ping;
            System.out.println("PINGING "+ping+"ms");
            }
            else{
             System.out.println("package dropped");
            }


        }
        // printing ping result to out and file
        System.out.println("Average ping was "+((float)totaltime/100));
```

```
        out.append("Average ping was "+((float)totaltime/100)+"ms\n\n");
        out.flush();
        out.close();
    }

}
```