Avdelning för datavetenskap

Fredrik Nilsson och Jonas Wånggren

Utveckling av en radiolänkssimulator

Development of a radio link simulator

Examensarbete (20p)
Civilingenjör Informationsteknologi

Datum: 2006-02-01
Handledare: Thijs Holleboom
Examinator: Donald Ross
Löpnummer: D2007:09

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Utveckling av en radiolänkssimulator

Fredrik Nilsson och Jonas Wånggren

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en civilingenjörsexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Fredrik Nilsson

Jonas Wånggren

Godkänd, Date of defense 2006-02-01

Handledare: Thijs Holleboom

Examinator: Donald Ross

Sammanfattning

Den här uppsatsen handlar om utvecklingen av en radiolänkssimulator åt företaget Icomera. Icomera inriktar sig på att sälja och distribuera internetåtkomst på tåg. Företagets affärsidé bygger på att sälja system som använder sig av olika trådlösa länkar som simultant skickar och tar emot data med hjälp av ett egenutvecklat protokoll. Icomera har sålt sitt system till bl.a. Statens järnvägar (SJ), för att möjliggöra internetåtkomst från tågen. Radiolänkssimulatorn används idag i Icomeras laboratorium vid testning och verifiering av tågsystemet, i och med att den ger en mer kontrollerad miljö än ett riktigt tåg.

Rapporten redogör för hur trådlösa länkar beter sig i verkligheten och hur deras beteende kan simuleras. Rapporten berör främst de problematiska delarna av implementationen, men också teori i form av bl.a. statistiska modeller.

Radiolänkssimulatorn har till största del skrivits i programspråket C# .NET, vilket har lett till flera olägenheter som behandlas i rapporten. De flesta olägenheter har främst med prestandaförluster att göra.

Abstract

This master thesis deals with the creation of a link simulator to be used by the company Icomera. Today the link simulator is used in Icomera's laboratory for simulating wireless links. Icomera's business concept is to sell and distribute internet access on trains and other vehicles. They use different wireless connections for simultaneously transfer data wrapped in their own protocol. Icomera has sold the system to Statens järnvägar (SJ), to enable internet access from trains. The link simulator is used for testing and verification of Icomera's train system. Hence it can be done in a more controlled environment than onboard a real train.

The report describes how wireless links acts in reality and how this behavior can be simulated. The report handles mainly the problematic parts of the implementation but also theory, e.g. statistic models.

The link simulator has primarily been written in the programming language C# .NET which has lead to several inconveniences which is discussed in the report. Most of the inconveniences have to do with performance loss.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Introduktion

1.1 Projektmål

Syftet med projektet var att skriva en programvara som simulerar trådlösa radiolänkar åt företaget Icomera. Programvaran ska användas i företagets laboratorium vid test och verifiering.

När projektet började hade Icomera redan en simulator, som led av bristande funktionalitet. Målet med vårt arbete var att utveckla en ny simulator som åtgärdare de brister som fanns i den gamla. Den nya simulatorn skulle vid projektets slut ersätta den tidigare versionen av simulatorn. Icomeras krav var att den nya simulatorn skulle ha ökad funktionalitet och ett bättre grafiskt gränssnitt, för att kunna ge en bättre verklighetsanknytning och få fler användningsområden. Simulatorn skulle även ge Icomera nya möjligheter vid test och verifiering av sitt system.

1.2 Bakgrund

Icomera startade sin verksamhet 1999 i Göteborg. Affärsidén bestod från början av att sälja mjukvara som möjliggjorde för kunder att utnyttja multipla länkar vid anslutning till Internet (exempel på två länkar är mobiltelefon och trådlöst nätverkskort). Syftet med att använda multipla länkar är två. För det första ger fler länkar ökad bandbredd, i och med att alla länkarnas kapacitet kan utnyttjas samtidigt. För det andra ger multipla länkar en stabilare uppkoppling, i och med att en länk kan koppla ned utan att anslutningen bryts. Produkten var tänkt att rikta sig till personer som använde en bärbar dator och flera mobiltelefoner för att ansluta sig till Internet.

Idén vidareutvecklades sedan till en produkt som möjliggjorde trådlös internetåtkomst på tåg, vilket också är vad Icomera ägnar sig åt idag. I mars 2003 fick Icomera sitt första stora kontrakt med tågföretaget Linx AB, som var en tågoperatör i Skandinavien. Efter Linx-kontraktet har även brittiska GNER och svenska SJ skrivit kontrakt med Icomera. Dessa kontrakt har

gjort att Icomera idag är världsledande inom området trådlöst Internet på tåg och deras produkter tillåter personal samt passagerare att t.ex. surfa på Internet, läsa mejl och komma åt sitt företagsnätverk via virtuellt privat nätverk (VPN) under tågresans gång.

1.3 Miljö

All utveckling har skett i Icomeras kontor i Göteborg. För att testa simulatorn har vi dels använt oss av Icomeras laboratorium, men även byggt en temporär testmiljö vid vår arbetsstation. Detta för att undvika att störa andra tester i laboratoriet, se avsnitt 2.6 för information om laboratoriet.

1.4 Problemställning

Icomera hade en tidigare version av simulator, men den uppfyllde inte simuleringsbehoven. För att Icomera skulle kunna utföra mer avancerad testning och verifiering krävdes en simulator med mer funktionalitet. Den tidigare versionen av simulatorimplementationen var alltför begränsad för att vidareutvecklas. Icomera valde istället att starta detta projekt för att designa och implementera en ny radiolänkssimulator från grunden.

1.5 Krav

Kraven från Icomeras sida var att vi skulle bygga en simulator som skulle kunna simulera en eller flera radiolänkar. Länkarna skulle kunna anta olika karakteristik med avseende på bandbredd, latens och paketförluster. För att efterlikna en verklig länk skulle olika statistiska fördelningar användas för bandbreddsfördröjningar, latensfördröjningar och paketförluster, till exempel binomial- och poissonfördelningar. Det skulle även gå att fördefiniera länktyper med en viss karakteristik, t.ex. 3G- och GPRS-länkar. För att få fram så verkliga värden som möjligt på de fördefinierade länktyperna skulle vi sammanställa RTT-data (round-trip-time) från tågsystem i drift. Denna data skulle sedan ligga till grund för kalibreringen av de fördefinierade länktypera. Det skulle även finnas möjlighet att skapa

enkelriktade länkar som endera tar emot eller skickar data, likt en satellitlänk.

Utvecklingen skulle ske i utvecklingsmiljön Microsoft Visual Studio och programmet skulle sedan köras på en Windows XP-plattform. Verktyget skulle ha ett enkelt och funktionellt grafiskt användargränssnitt, där länkkarakteristik ska kunna ändras.

Simulatorn skulle även kunna avkoda data som skickas med Icomeras egenutvecklade protokoll. Icomeras egna protokoll tillhandahåller information om varje länk, något som ska analyseras och tillämpas i simulatorn. Resultatet av en simulering skulle sparas på lämpligt sätt för analyseras i efterhand. För en detaljerad kravlista samt beskrivning av kraven se avsnitt 4.2.

Slutligen hade Icomera som krav att vi skulle utelämna all källkod och beskrivning av Icomeras egna protokoll i rapporten.

1.6 Verifiering

Verifiering som skulle göras på simulatorn innebar att ett antal testfall skulle specificeras och köras (testfall finns i Bilaga A – Verifieringsplan). När simulatorn var implementerad och ansågs klar skulle testerna utföras och om simulatorn inte klarade testerna skulle den kompletteras till dess att testfallen var uppfyllda. Verifieringen användes för att bekräfta att Icomeras krav var uppfyllda. Verifieringen skulle ske i Icomeras laboratorium.

1.7 Metod

Metoden som skulle användes för att utveckla simulatorn byggde både på teoretisk och på praktisk grund. Med det menas att det fanns en teoretisk grund för hur implementationen utfördes med avseende på analys av länkkarakteristik som t.ex. bandbredd, latens och paketförluster. Den praktiska grunden utgörs av implementationstester. All utveckling har skett i Icomeras kontorsmiljö för att få bättre insikt i systemet och kunna interagera med de verktyg som redan fanns. En expert på Icomeras system fanns hela tiden tillgänglig för att ge riktlinjer och hjälp. Designen av simulatorn

skedde i rådfrågning och konsultation med utvecklarna på Icomeras utvecklingsavdelning

Utvecklingsmetoden vi använde kan liknas vid vattenfallsmodellen (se referens [1] för information om vattenfallsmodellen) och bestod av dessa följande steg:

Steg 1 (Utredning): Sätta sig in i Icomeras verksamhet och system.
Steg 2 (Utredning): Ta fram en kravspecifikation och verifierings-dokument i samarbete med Icomera.
Steg 3 (Design): Göra en design av simulatorn och alla dess komponenter.
Steg 4 (Prototyp): Identifiera och göra prototyper av de kritiska systemdelarna (de delarna som vi hade lite kunskap om eller som de anställda på Icomera ansåg kunna orsaka eventuella problem)
Steg 5 (Design): Justera designen utefter prototyperna i steg 4.
Steg 6-9 (Implementation): Göra fyra implementeringsiterationer där vi implementerar designen från steg 5.
Steg 10 (Verifiering): Verifiera simulatorn genom att gå igenom alla testfall i verifieringsplanen.
Steg 11-N (Implementation & Verifiering): Göra justeringar på simulatorn så att den klarar alla testfall. Upprepa detta steg till dess att simulatorn klara alla testfall.

Den gamla simulatorn byttes ut mot den nya under projekttiden för testning och kalibrering.

1.8 Resultat

Vid projektets slut fanns ett färdigt verktyg för simulering av radiolänkar. Mjukvaran har sin tyngdpunkt på stabilitet och tillförlitlighet. Simulatorn klarar av Icomeras behov i deras laboratorium samt är designad för att även klara av framtida behov och vidareutveckling. Simulatorn är lätt att använda och tillhandahåller den funktionalitet som ställdes som krav vid projektets början.

1.9 Disposition

I kapitel 2 redogörs för hur Icomeras system är strukturerat. Det görs för att ge en bild över systemet och förståelse för simulatorns roll i systemet. Det beskrivs också vilka för- och nackdelar det finns med att simulera.

I kapitel 3 beskrivs vilka olika typer av länkar som Icomera i nuläget använder i sitt system för trådlös internetanslutning på tåg. Det redogörs också för vilka parametrar som påverkar en länk, samt ges ett histogram över latensbeteende för de olika länktyperna.

I kapitel 4 beskrivs kraven och de beslut som tagits under projektets gång. Detta följs av en redogörelse för hur simulatorn är uppdelad i olika modulerna.

I kapitel 5 och 6 beskrivs simuleringsmotorns och användargränssnittets implementationen. Det redogörs för flödet genom simulatorn samt interaktionen med användargränssnittet.

I kapitel 7 beskrivs två användarfal. Ett användarfall där interaktionen mot simulatorn sker manuellt och ett användarfall där det sker automatiserat.

I kapitel 8 redogörs för de problem som har uppkommit under projektet. Förslag på eventuella lösningar ges.

I kapitel 9 redovisas en sammanfattning av projektet samt slutsatser. Vi reflekterar över kraven och verifierar att de är uppfyllda. Egna tankar om projektet redogörs, t.ex. vad vi skulle ha gjort annorlunda.

2 Bakgrund och Icomeras system

2.1 Introduktion

I detta kapitel förklaras fördelarna med simulering vid utveckling av mjukvara och hur Icomera använder sig av simulering. Kapitlet ger även en genomgång av Icomeras system, det vill säga trådlös internetanslutning på tåg. Systemet beskrivs först övergripande och sedan beskrivs de enskilda komponenterna.

Slutligen ges även en beskrivning av hur Icomeras simuleringssystem fungerar och hur dess olika komponenter är sammankopplade och konfigurerade.

2.2 Varför simulera?

Ett system kan vara allt från ett distributionsnätverk till ett kommunikationssystem. Simulering kan bland annat användas till att studera och jämföra olika typer av design eller för att felsöka i dessa system. Med hjälp av simuleringsmodeller ges möjlighet att testa existerande systems beteende efter ändringar, eller göra tester på ett nytt system redan innan en prototyp har hunnit bli klar. Möjligheten att enkelt konstruera och exekvera modeller för att få fram statistik, har alltid varit den stora fördelen med mjukvarusimulering.[2]

Det pratas mycket om vikten av att testa mjukvara vid utveckling, men för att kunna testa system måste det till att börja med finnas en slutmiljö tillgänglig. Problemen uppstår när utvecklingen riktar sig mot en slutmiljö som inte finns tillgänglig på ett enkelt sätt på utvecklingsplatsen. Ett exempel på detta är t.ex. företag som utvecklar mjukvara för nätverk, såsom Internet. Dessa företag skapar knappast en kopia av Internet med tusentals noder i sitt laboratorium för att kunna testa mjukvaran, trots att användare kan komma att köra mjukvara på just ett sådant nätverk. Icomera är ytterligare ett exempel på ett företag där slutmiljön inte finns tillgänglig. Icomera kan inte fysiskt återskapa alla de olika delarna av

kommunikationskedjan mellan tågen och nätslussen (eng: gateway) (se avsnitt 2.4 för beskrivning av hela kommunikationskedjan) som är nödvändigt för att testa systemet. För dessa företag är det nödvändigt att simulera delar av systemet för att kunna utföra tester. Hade inte Icomera haft möjlighet att simulera sin slutmiljö hade alternativet varit att testa alla ändringar av tågsystemet på ett tåg. Att göra alla dessa tester på ett tåg skulle vara ohållbart, i och med att det blir dyrt och tar lång tid. Möjligheten att testa med en simulator i ett laboratorium är således något som sparar både tid och resurser.[3][4][5]

Slutsatsen av detta är att simulering i sig själv inte har något egenvärde för utvecklingen. Simulering utgör grunden för andra viktiga steg i utvecklingsprocessen, så som testning. Ett konkret exempel på detta fick vi i ett möte med Johan Berts på Icomera, där det bl.a. nämns att den tidigare versionen av simulatorns bristande funktionalitet bidrog till lägre grad av testing än önskat. Detta visar på behovet av att simulera delar av systemet och hur det ligger som grund för andra steg i utvecklingsprocessen.

2.3 För- och nackdelar med simulering.

Att simulera har både för- och nackdelar. Dessa är dock svåra att identifiera ur ett generellt perspektiv och varierar mellan olika företag. För att kunna identifiera för- och nackdelar med simulering har vi valt att inrikta oss på Icomeras testbehov. Här följer en lista med de punkter, som vi i samtal med Icomera anser vara de största fördelarna samt nackdelarna med att simulera.

Fördelar:

- *Kunna testa nya system utan att påverka driften av befintliga system.*

  Det ses som ytters olämplig att tvinga slutkunden att stänga ned något av sina system för att låta Icomera testa nya system. Det skulle även vara olämpligt att testa på ett tåg i drift. Eventuella

störningar i systemet skulle ge användarna ett dåligt intryck av systemet och därmed riskera att tågbolagen förlorar kunder.

- *Kunna testa uppdateringar i systemet utan att påverka driften av befintliga system.*

  Precis som förgående punkt skulle det vara olämpligt att testa uppdateringar på tåg i drift. Det kan resultera i störningar i systemet. Att först kunna testa uppdateringen i en simulator minskar risken för störningar i systemet, när väl uppdateringen genomförs på ett tåg.

- *Kunna utföra tester i kontorsmiljö, behöver inte vara på ett tåg för att kunna testa.*

  Detta är en stor fördel då det sparar mycket tid. Att testa på ett tåg kräver planering. Helst måste utvecklarna hitta ett tåg som har ett system installerat, men som inte är i drift. Om ett sådant tåg inte finns tillgängligt måste utvecklarna få tillåtelse av tågbolagen att köra tester på ett tåg med betalande kunder. Slutligen måste utvecklarna ta sig till och från tåget för att utföra tester. Ett enkelt test kan på detta sätt ta hela dagar istället för minuter.

- *Det krävs mindre resurser när väl simulatorn är på plats och fungerar.*

  När väl simulatorn är utvecklad och installerad krävs det bara små enkla konfigureringar för att använda den. Det gör att en utvecklare kan utföra tester på egen hand. Något som inte är fallet på ett tåg, där det oftast krävs minst två personer för att utföra tester.

- *Testerna kan utföras i en kontrollerad miljö.*

  Genom att alla tester körs i en simulator där alla parametrar sätts explicit, så har utvecklarna fullständig kontroll på alla variabler vid testtillfället. Kör utvecklarna tester på ett tåg är det svårt att kontrollera alla variabler. Många variabler påverkas av system utanför Icomeras egna. Ett exempel på sådana variable är t.ex. det nätverksjitter som finns i mobilnätet.

- *Det kostar mindre att testa med hjälp av en simulator än på ett tåg.*

  Som det nämndes i tidigare punkter så sparar simulatorn både tid och resurser vid testning. Detta är något som indirekt innebär billigare tester.

Nackdelar:

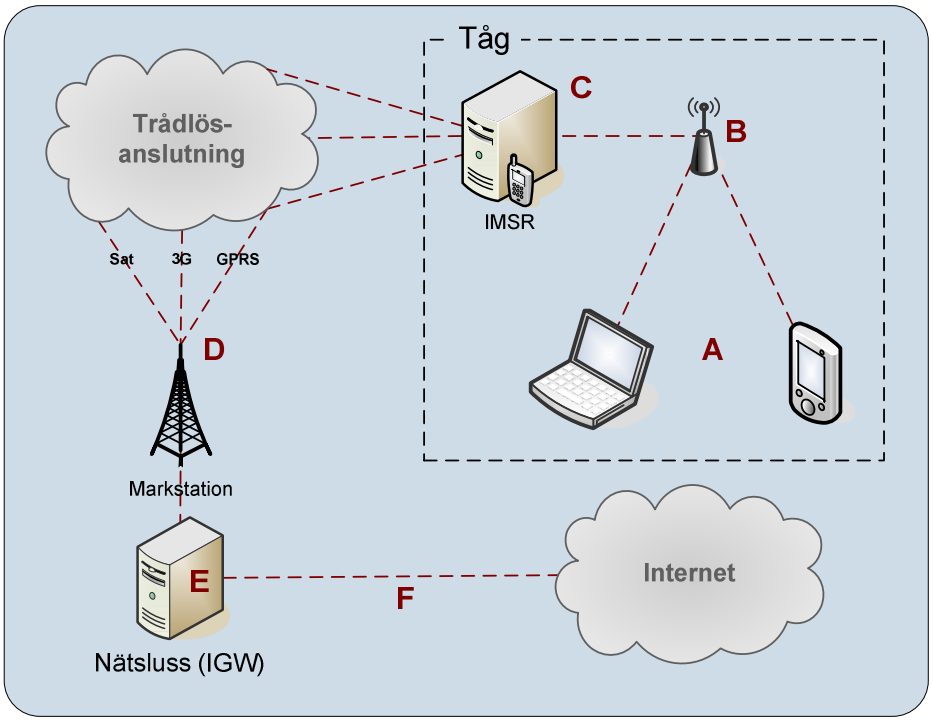
- *Simulering kan aldrig vara så exakt som verkligheten.*

  En simulator kan aldrig vara lika exakt som verkligheten eftersom den måste följa lagarna för ett datorprogram. Med det menas att datorn är begränsad av processorhastigheter och minnesstorleker etc. Det gör att simulatorn inte kan ta hänsyn till alla de variabler som finns i den riktiga världen. En simulator försöker bara efterlikna verkligheten utan att för den skull behöva ta hänsyn till alla dess variabler.

- *Det kräver tid och resurser att ta fram en simulator.*

  För att skapa en simulator krävs en analys av det rikitga systemet för att b.la. identifiera påverkande variabler. Det kräver både tid och resurser.

2.4 Systembeskrivning



Figur 2-1: De olika delarna i Icomeras system

Att förstå konfiguration av Icomeras tågsystemets och dess sammansättning är en väsentlig del för att förstå simulatorns roll och varför den behövs för att Icomera ska kunna testa sina system. Därför ges här en övergripande beskrivning av hur systemet fungerar med hjälp av Figur 2-1.

När passagerarna sätter sig på tåget kan de ansluta sig till Internet via en bärbar dator, handdator eller någon annan enhet som har ett har ett trådlöst gränssnitt (se A, Figur 2-1). Passagerarna börjar med att ansluta till ett trådlöst nätverk som tillhandahålls av tågoperatören via accesspunkter, som strategisk placerats ut i tågvagnarna (se B, Figur 2-1). Accesspunkter, som utgör en brygga mellan ett trådlöst gränssnitt och kabelburet gränssnitt, finns bara i de vagnar som tillhandahåller Internetåtkomst. På så sätt kan tågoperatören begränsa åtkomstmöjligheterna till exempelvis förstaklassvagnarna. Nätverket som passagerarna ansluter till är helt öppet och ingen autentisering krävs för att ansluta. Alla accesspunkterna i tågen är sammankopplade och anslutna till en gemensam server på tåget som kallas Icomera Mobile System Rack (IMSR) (se C, Figur 2-1). När passagerarnas

datatrafik når IMSR:en verifierar IMSR:en att den specifika passageraren är inloggad innan den skicka trafiken vidare ut på Internet. Om IMSR:en upptäcker att avsändaren inte är inloggad omdirigeras han/hon till en webbportal för inloggning. För att logga in krävs ett lösenord. Lösenordet går att få tag i på flera sätt bl.a. genom att betala en avgift via sitt kreditkort direkt i webbportalen. Lösenorden finns även i flera olika typer såsom tidsbegränsade, hela resan eller datamängdsbiljetter. När passageraren har loggat in med ett korrekt lösenord kan han/hon surfa tills dess att lösenordets giltighetstid har gått. När detta sker omdirigeras passageraren åter tillbaka till webbportalen, för att logga in igen med ett nytt lösenord.

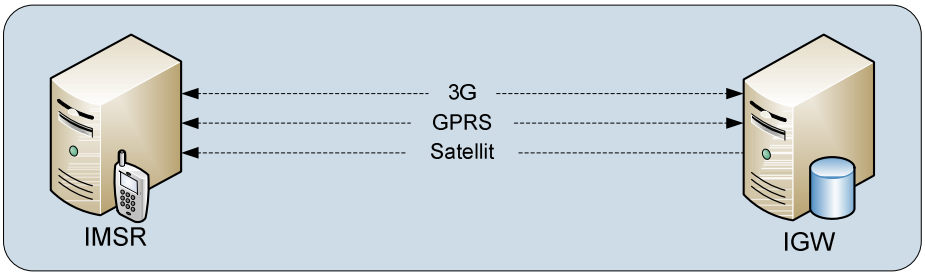
För att en IMSR ska kunna skicka vidare datatrafik till Internet har den ett antal trådlösa länkar som förbinder tåget med olika markstationer (länkarna är av flera olika typer bl.a. 3G och GPRS, se D i Figur 2-1). När datatrafiken har nått en markstation fortsätter den till en nätsluss, som kallas Icomera Gateway (IGW) (se E, Figur 2-1), och därifrån vidare ut på Internet igen (se F, Figur 2-1). När datatrafiken ska skickas tillbaka, från Internet till passagerare på tåget, sker det på samma sätt fast i omvänd riktning.

Att låta IMSR:en använda flera länkar ger både ökad bandbredd samt ökad tillgänglighet eftersom det minskar risken att förlora uppkopplingen, i och med att datatrafiken är fördelad på flera länkar. Genom att använda sig av olika typer av länkar, så blir Icomeras system mindre beroende av täckningsgraden på de enskilda nätverken. Det finns dock tillfällen när även detta system blir sårbart t.ex. vid färd genom tunnlar där det ofta saknas täckning från alla typer av nät. En lösning på problemet med tunnlar är att placera antenner i början och slutet av tåget. När första vagnen åker in i tunneln kommer sista vagnen fortfarande ha täckning, när den sista vagnen åker in i tunneln har förhoppningsvis den första vagnen kommit ut på andra sidan tunneln och återfått täckning. En annan lösning är att montera repeatrar i tunnlarna, som innebär att den trådlösa signalen förstärks inuti tunneln.

2.5 Detaljerad beskrivning

I de följande avsnitten kommer IMSR:en och IGW:n att redovisas i detalj. Det ges även en förklaring till vad en, ur Icomeras perspektiv, länk är. De här tre delar är grundstenarna i Icomeras system och en förståelse för hur de fungerar är nödvändig för att förstå vad som krävs för att konstruera en simulator.

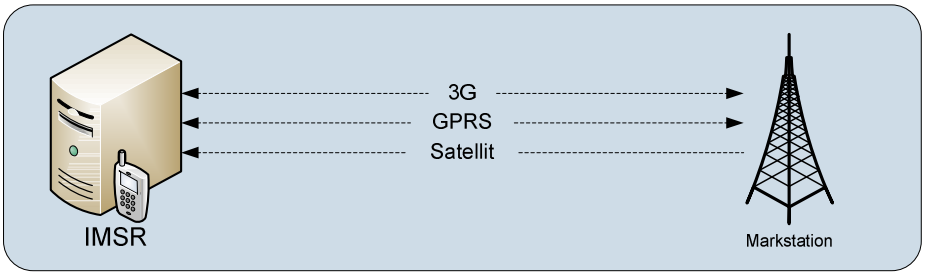
2.5.1 Länkar



Figur 2-2: Länkar mellan IMSR:en och IGW:en.

En länk är en koppling mellan IMSR och IGW. Kopplingen är trådlös och sker med olika tekniker t.ex. 3G, GPRS och satellit. Varje länktyp har olika karakteristik som t.ex. bandbredd, latens och paketförluster. Dessa kan variera med tid och omgivning. En länks paketförluster ökar när mottagaren befinner sig långt från sändaren och minskar när den befinner sig nära. Det finns flera trådlösa länktekniker, men Icomera använder sig bara av 3G, GPRS och satellit (Rapporten kommer i fortsättningen bara inrikta sig på dessa tre länktyper). För att en exakt specifikation på dessa länktypers karakteristik, se avsnitt 3.3.

2.5.2 Icomera Mobile System Rack (IMSR)



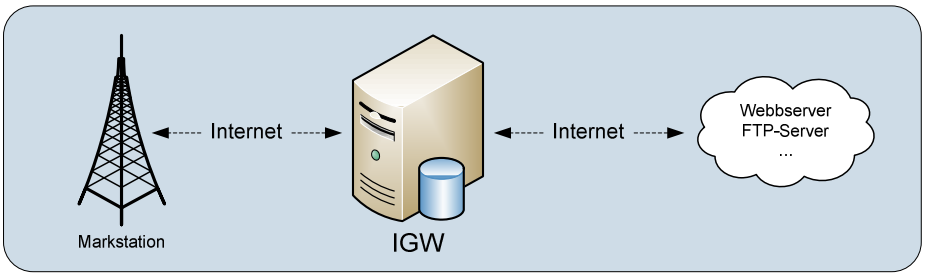
Figur 2-3: IMSR:ens koppling till markstationen.

Icomera Mobile System Rack (IMSR) är den server som är installerad på tågen. IMSR:en har ett antal trådlösa länkar som är uppkopplade till markstationer, se Figur 2-3. Via dessa länkar kan IMSR:en skicka och ta emot data från Internet. Antalet länkar som används varierar mellan olika konfigurationer, men i den senast versionen av IMSR:en kan det vara upp till sexton olika länkar uppkopplade samtidigt.

IMSR:en fungerar som en lokal nätsluss på tåget, via vilken all datatrafik passerar för att nå ut på Internet. För att få så bra genomströmning som möjligt försöker IMSR:en optimera trafikflödet genom de olika länkarna. Detta görs med hjälp av olika algoritmer som väljer ut den mest lämpliga länken att skicka datatrafik på. Algoritmerna använder sig av olika realtidskarakteristik, hos de anslutna länkarna, för att hitta den mest optimala länken.

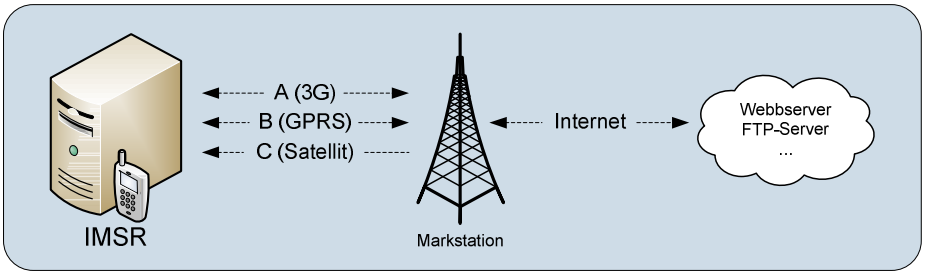
Innan IMSR:en skickar ut trafiken på Internet omsluter den datatrafiken med Icomeras egna protokoll samt omdirigerar trafiken till IGW:en istället för sin tidigare slutdestination (den ursprungliga destinationen finns kvar under det överliggande icomeraprotokollet). Detta görs för att ytterligare optimera och förbättra överförningen mellan tågen och Internet. En beskrivning hur det sker och varför Icomeras protokoll behövs samt varför trafiken omdirigeras till IGW:n ges i avsnitt 2.5.3.

2.5.3 Icomera Gateway (IGW)



Figur 2-4: IGW:ens koppling mellan markstation och ändservrar.

Icomera Gateway (IGW) är en nätsluss där all datatrafik från de olika IMSR:erna passerar innan de skickas vidare ut på Internet. I avsnitt 2.5.2 beskrevs det hur IMSR:en har flera länkar anslutna till markstationer via vilka de kan skicka data direkt ut på Internet. Det kan då ses som onödigt att skicka all datatrafik via IGW:n och sedan ut på Internet igen. Detta är dock nödvändigt för att maximalt kunna utnyttja flera länkar samtidigt. Det är även här som styrkan med ett eget protokoll blir synligt. För att lättare förklara varför det finns en IGW så antar vi istället motsatsen, att varje IMSR skickar datatrafiken direkt till sin slutdestination utan att passera IGW:n. Kommunikationskedjan ser då ut som i Figur 2-5.

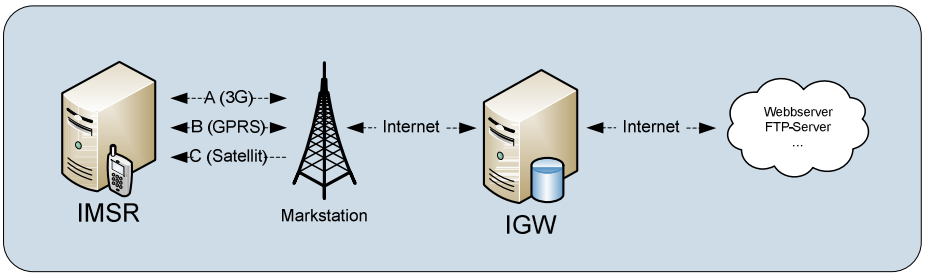


Figur 2-5: Kommunikationskedjan utan IGW.

Om passageraren begär hämtning av en hemsida så kommer IMSR:en välja en av länkarna A, B och C för att skicka begäran på (Låt oss anta begäran skickas på länk A). När begäran når webbservern, skickas svaret tillbaka till länk A:s IP-adress. När svaret når passageraren visas hemsidan i webbläsaren. Problemet med detta är om länken (A), som begäran skickades på, bryts innan svaret når passageraren så kommer passageraren få en

timeout på sin begäran och webbläsaren kommer visa att adressen inte kunde hittas. Detta trots att det fanns två till länkar som svaret kunde ha skickats på. Webbservern känner dock inte till de andra länkar och kan därför inte prova att skicka svaret på dem. Detta är ett stort problem eftersom det ofta händer att länkar går upp och ned, under korta stunder på tågen. Det är dock sällan att alla länkar går ned samtidigt. Med de här förutsättningarna är det därför önskvärt att kunna skicka data på en länk och sedan kunna ta emot dem på en annan.

För att göra detta behövs det en punkt i systemet som hela tiden är uppkopplad och som kan hålla reda på vilka länkar som finns tillgängliga för varje IMSR. Denna punkt utgörs av IGW:n. Låt oss ta samma exempel med skillnaden att all datatrafik från IMSR:erna passerar genom IGW:n, se Figur 2-6.



Figur 2-6: Kommunikationskedjan med IGW.

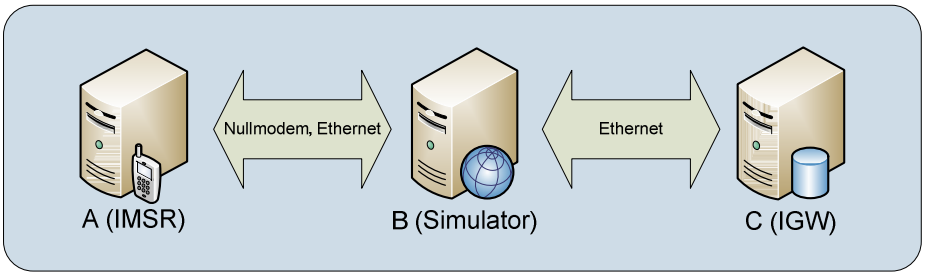
I denna situation när passageraren begär hämtning av en hemsida, kommer IMSR:en välja en av länkarna A, B och C för att skicka begäran på (Låt oss anta att begäran skickas på länk A). När begäran når markstationen skickas den vidare till IGW:n och därifrån vidare till webbservern. Från webbservern skickas svaret tillbaka precis som förut. Skillnaden är dock att svaret skickas till IGW:n och inte till direkt tillbaka länk A:s IP-adress. När paketet når IGW:n väljer den vilken av de tillgängliga länkarna som svaret ska skickas på. På detta sätt blir systemet stabilare eftersom om länk A, som paketet skickades på, kopplar ned så kan IGW:n notera det och istället skicka svaret på länk B eller C. Med andra ord gör IGW:n det möjligt för IMSR:en att skicka begäran på en länk och ta emot svaret på en annan. Detta är en fördel i och med att det möjliggör att användandet av enkelriktade satelliter (en enkelriktad satellit fungerar genom att en begäran

av data skickas via någon form av upplänk t.ex. en 3G-länk och svaret kommer tillbaka via satellitlänken). Detta är praktiskt då satellitlänken för tillfället är den snabbaste av de länktyper som stöds av Icomeras system.

Det finns dock fortfarande ett problem. Hur ska IGW:n kunna veta vilka paket som kommer från vilken IMSR? Det är här Icomeras egna protokoll kommer in i bilden. När paketet skickas från IMSR:en till IGW:en paketeras paketet om och information, som t.ex. vilken länk paketet skickats på och vilken IMSR den skickats ifrån, läggs till. Detta gör det möjlig för IGW:en att skilja på trafik som kommer från olika IMSR:er och länkar. IGW:en tar sedan bort den extra informationen innan paketet skickas vidare ut på Internet. För att IGW:n ska kunna vidarebefordra svaren på det utskickade paketen, binds varje IMSR mot en port eller IP-adress. Det innebär att varje IMSR får en unik IP-adress eller port. När IGW:n tar emot ett svarspaket slår den upp vilken IMSR som är bunden till den IP-adressen eller porten och vidarebefordrar paketet på en av IMSR:ens anslutna länkar.

2.6 Icomeras simuleringsmiljö

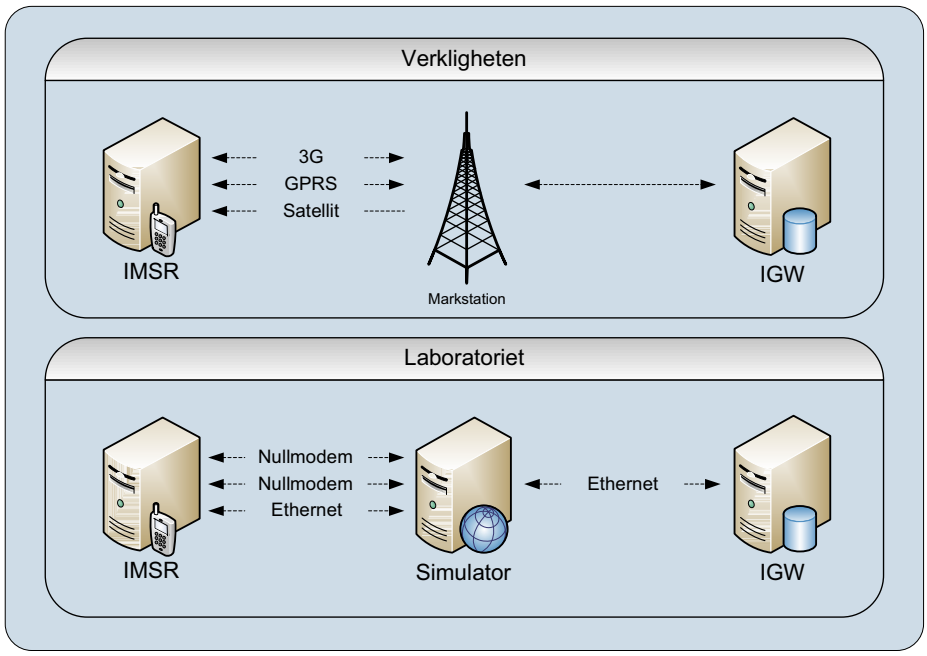
Icomera har ett laboratorium med utrustning för att kunna testa mjuk- och hårdvara. Laboratoriet består bl.a. av tre sammankopplade datorer, se Figur 2-7, som utgör simluleringsmiljön.



Figur 2-7: Kopplingen mellan de tre olika delarna i simuleringskedjan.

På dator A körs en IMSR och på dator C körs en IGW. Dator A och C är sedan sammankopplade via dator B, på vilken simuleringsmjukvaran körs. Simulatorn kan ses som en markstation, men till skillnad från en vanlig markstation, där trafiken från en länk direkt slussas vidare mot IGW:n, så gör simulatorn det möjligt att påverka och begränsa trafikflödet för varje

enskild länk. I Figur 2-8 vissas hur simuleringsmiljön förhåller sig mot verkligheten. I figuren ser vi hur markstationen ersätts av simulatorn och hur 3G-, GPRS- och satellitlänkarna ersätts av nullmodem och ethernet.

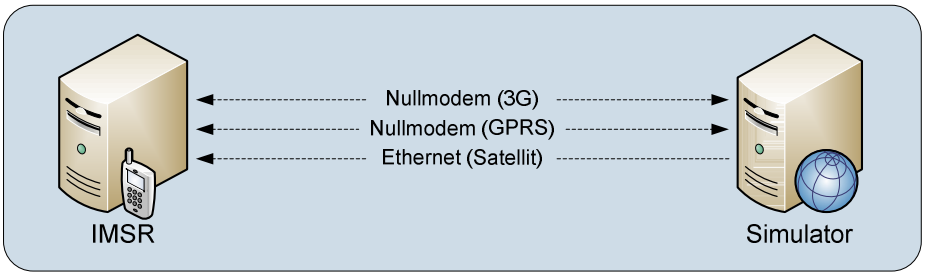


Figur 2-8: Jämförelse av labboratoriemiljön och verkligheten.

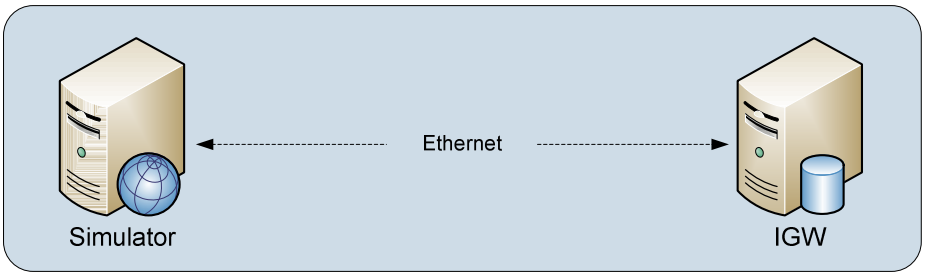
För att kunna skapa länkar mellan IMSR:en och simulatorn använder sig Icomera av åtta seriellkablar och två nätverkskablar. De åtta seriellkablarna används för att simulera 3G- och GPRS-länkar och de två nätverkskablarna för att simulera satellitlänkar, se Figur 2-9. Icomera använder nullmodem istället för ethernet för 3G- och GPRS-länkarna eftersom de vill få med det uppringningsförfarande via remote access service (RAS) som sker mellan två nullmodem. Anledningen är att samma uppringningsförfarande sker med 3G- och GPRS-modemen på tågen.

Simulatorn är kopplad till IGW:n via en nätverkskabel (100 Mbit/s), se Figur 2-10. Detta innebär indirekt att den maximala hastigheten genom simulatorn begränsas till 100 Mbit/s. Detta beror på att oavsett hur mycket den totala bandbredden är på de länkar som finns mellan IMSR:en och simulatorn, så kommer de dela på samma 100 Mbit/s-länk mellan simulatorn och IGW:n.

När testar ska göras i simulatorn börjar utvecklaren med att installera den versionen av IMSR:en och IGW:n som ska testas. Konfigurerar IMSR:en så att den använder rätt antal och typer av länkar. Utvecklaren konfigurerar även simulatormjukvaran (dator B i Figur 2-7) med vilken karakteristik de olika länkar ska ha. Slutligen utför de testerna på det konfigurerade systemet (i det gamla systemet gjordes alla tester för hand. Det gick inte att konfigurera simulatorn så att den automatiskt körde olika testfall, något som är möjligt i den nya simulatorn). För en genomgång av hur utvecklaren använder simuleringsmjukvaran, se kapitel 7 och kapitel 8, där ett användarfall beskrivs.



Figur 2-9: Kommunikationen mellan IMSR:en och simulatorn.



Figur 2-10: Kommunikationen mellan simulatorn och IGW:n.

2.7 Sammanfattning

I det här kapitlet har det givits en övergripande samt detaljerad beskrivning av hur Icomeras tågsystem fungerar. Det har även givits en beskrivning hur Icomeras simuleringssystem är uppbyggd i laboratoriet. Innan vi fortsätter är det två saker som bör förtydligas. Den första är att när rapporten i fortsättningen refererar till simulatorn (och inget annat nämns) är det INTE i form av hela simuleringsmiljön, utan bara den mjukvara som körs på dator B i Figur 2-7. Det andra som bör förtydligas är att det inte är hela

simuleringssystemet som ligger till grund för ex-jobbet, utan bara en omarbetning av simulatormjukvaran som körs på dator B i Figur 2-7. Med andra ord kommer vi inte att designa om eller undersöka eventuella förbättringar på Icomeras simuleringsmiljö.

3 Simulering av Icomeras trådlösa länkar

3.1 Introduktion

Karakteristiken för trådlösa länkar består av ett antal parametrar. Vid simulering, är det dessa parametrar som justeras för att efterlikna en riktig länk. I avsnitt 3.2 redogörs de parametrar som påverkar karakteristiken på en länk.

För att kunna anpassa simulatorn för Icomeras behov studeras även de länktyper som de använder sig av, i avsnitt 3.3.1(GPRS), 3.3.2 (3G) och 3.3.3 (satellit) .

3.2 En trådlös länks parametrar

Det talas mycket om hastighet på en uppkoppling i dagligt tal. Det kan t.ex. vara att ladda hem filer snabbt eller att kunna prata i telefon över Internet. Hur man upplever de två scenarierna beror på en länks karakteristik. De parametrar som påverkar karakteristiken på trådlös länk är bandbredd, latens och paketförluster. För att få en bättre förståelse för hur dessa påverkar datatrafiken och vilket förhållande som finns mellan dem, ges en ingående förklaring i avsnitt 3.2.1-3.2.3.

3.2.1 Bandbredd

Med bandbredd menas i vilken hastighet data kan skickas ut på länken, vanligtvis används enheten bitar/sekund. Mer tekniskt betyder bandbredd hur stort frekvensutrymme en signal tar upp. Det är alltså signalspektrumets bredd. Bandbredden sätter en gräns för hur fort en länk kan skicka data, inte hur fort den kommer fram till mottagaren. [6][7]

3.2.2 Latens

Latens på en länk motsvarar den tid det tar från det att ett paket skickas, tills att det tas emot. På Internet används ofta ordet tur-och-retur-tiden (RTT)(eng: round-trip-time), vilket är tiden det tar för ett paket att skickas till destinationen och sedan tillbaka. Latensen är halva RTT:n med

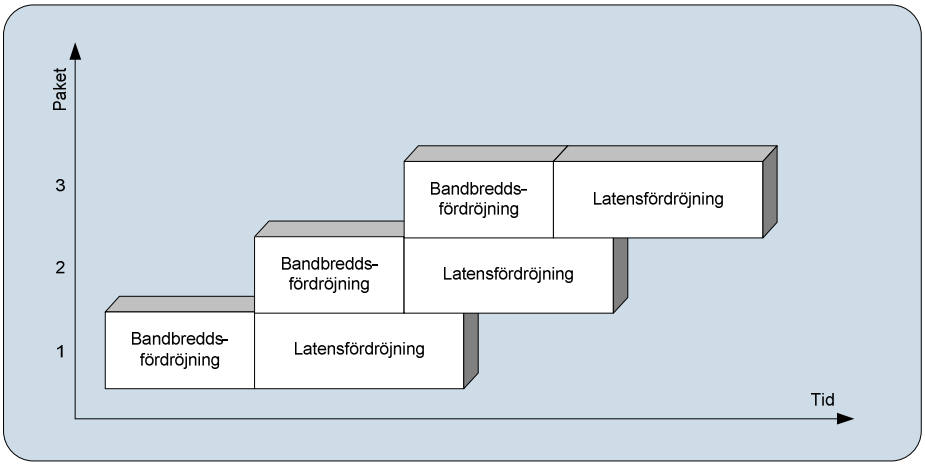
förutsättningen att det tar lika lång tid för paketet att komma fram och tillbaka.

3.2.3 Paketförluster

Paketförlust är när ett paket försvinner innan det når destinationen. Anledningen till att en paketförlust kan bero på flera olika orsaker. Det kan till exempel vara att signalstyrkan är för svag i trådlösa system eller att en buffert i en router blir full och inte kan ta emot fler paket.

3.2.4 Förhållande mellan bandbredd och latens

Det kan vara lätt att blanda ihop storheterna bandbredd (se 3.2.1) och latens (se 3.2.2). Figur 3-1 visar hur bandbredden förhåller sig till latensen. [6]



Figur 3-1: Förhållande mellan bandbredd och latens.

Som Figur 3-1 visar ligger Paket 2, Paket 3 och väntar på att Paket 1 ska bli utskickat på länken. Länken kan endast skicka ut ett paket åt gången och tiden det tar beror på hur stort paket är samt hur många bitar per sekund länken skickar ut. När länken väl har skickat ut paketet på länken är det latensfördröjningen som bestämmer när paketet kommer fram. Det är alltså fördröjningar i routrar, t.ex. på grund av paketstockning. [6]

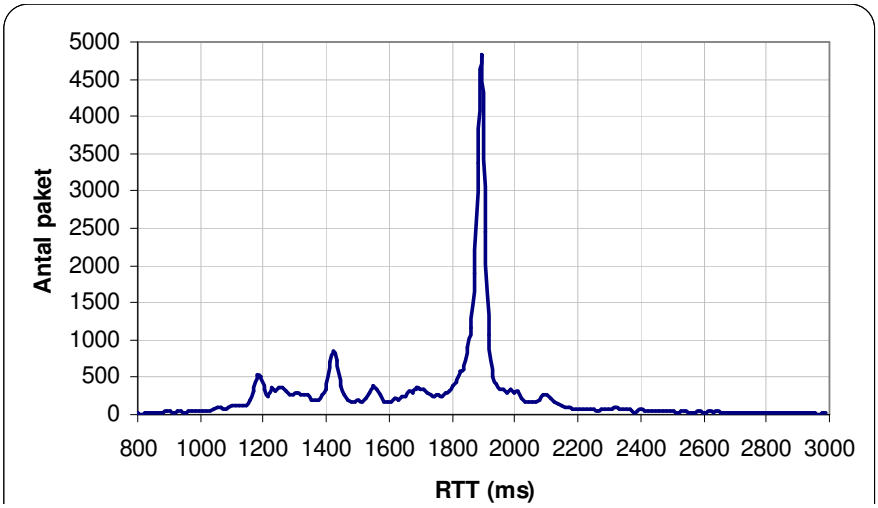
3.3 Icomeras länktyper

Icomera använder idag tre olika typer av trådlösa länkar, som beskrivs i följande underavsnitt. För varje länktyp visas ett RTT-histogram med mätvärden från Icomeras databas, som innehåller data från de tåg som är i drift, för att visa vilka RTT-områden länkarna ligger inom. Ett histogram över hastigheten kan tyvärr inte visas i och med att hastigheten på en länk beror på belastningen på länken, vilket Icomera inte lagrar i databasen. Samma gäller för paketförluster, som också är beroende av hur mycket data som skickas över länken.

3.3.1 GPRS.

General Packet Radio Service (GPRS), är ett paketdatatillägg till Global System for Mobile communications-systemet (GSM). GPRS kallas ibland för 2.5G (2.5 Generationen) för att det är tekniken mellan 2G- (GSM) och 3G- (UMTS) systemen.

GPRS använder sig av tekniken Time Division Multiple Access (TDMA) för att överföra data. Det innebär att dataöverföringen sker via tidsluckor, vilken i sin tur gör att GPRS-hastigheten varierar kraftigt beroende på belastningen på systemet. Den maximala hastigheten i teorin ligger runt 144 kbit/s, vilket är mycket svårt att uppnå i praktiken. När Icomera gjorde mätningar på sina Nokia12-modem, uppmättes hastigheter på runt 40 kbit/s för data som går till modemet och runt 20 kbit från modemet. Latens som uppmäts på GPRS-länkar från tågen ligger på ca 925 ms, vilket jämfört med 3G- och satellitlänkar är mycket högt. I Figur 3-2 visas ett histogram över RTT:n på Icomeras GPRS-länkar. Där syns en tydlig topp med RTT på 1850 ms vilket ger en latens på 925 ms. Mätvärdena är tagna från 1 december, 2005 till 7 december, 2005. [8]



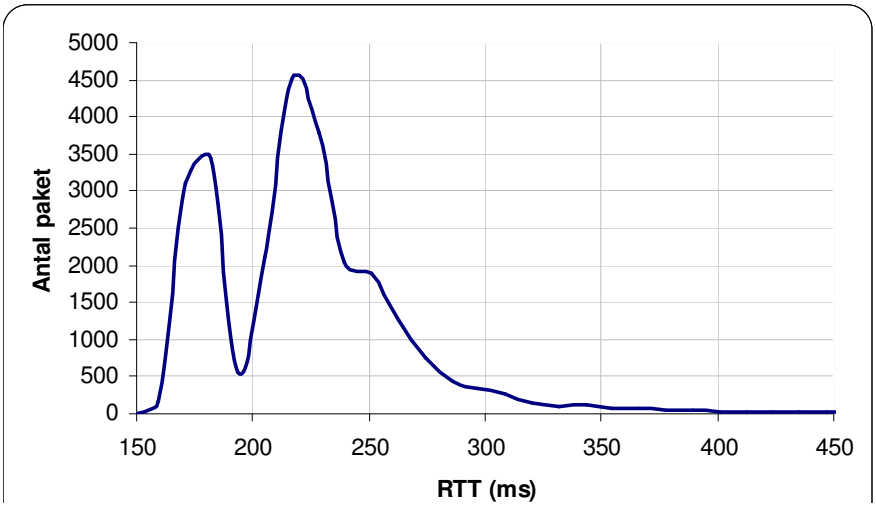
Figur 3-2: Histogram över RTT för GPRS

3.3.2 3G

3G står för tredje generationens mobiltelefonsystem och är uppföljaren till de tidigare systemen Nordic Mobile Telephony (NMT) och GSM. 3G är ett samlingsnamn för flera olika tekniker bl.a. Universal Mobile Telecommunication System (UMTS) som används i Sverige. Den största fördelen med 3G är att överföringshastigheten har ökat markant i jämförelse med GPRS. 3G stödjer bland annat musiköverföring och bildöverföring i form av videosamtal med mobiltelefon. [9]

UMTS använder sig av tekniken Wideband Code Division Multiple Access (W-CDMA). Tekniken bygger på att alla mobiltelefoner sänder över samma frekvensband och särskiljs med kodsignaler. Hastigheten på en 3G länk kan komma upp i max 384 kbit/s. [8][10]

I Figur 3-3 syns två stycken toppar där RTT:en varierar kring, 170 ms och 220 ms. En teori till att RTT:n kretsar just kring dessa två värden är att det sker en växling mellan stadsnät och landsbyggdsnät. Teorin har inte bekräftats. Mätvärdena är tagna från Icomeras databas mellan 1 december, 2005 till 7 december, 2005.



Figur 3-3: Histogram över RTT för 3G

3.3.3 Satellit

Satelliter används idag bland annat för olika typer av kommunikation såsom TV, radio och telefoni. Icomera använder i nuläget endast en enkelriktad satellitlänk för mottagande av data. Så för att skicka data krävs att någon typ av upplänk finns tillgänglig. På Icomera studeras även dubbelriktade satellitantenner för att vidga sin målgrupp och kunna erbjuda Internet på de ställen där det inte finns tillgång till vare sig 3G eller GPRS. Fördelen med att använda satellit är att överföringshastigheterna är betydligt högre än både 3G och GPRS, samt att täckningsgraden är större. Histogrammet i Figur 3-4 visar hur RTT är fördelad. Det syns en klar dominans på RTT mellan 410-440 ms där kurvan visar en markant topp. Mätvärdena är tagna från Icomeras databas mellan datumen 1 december, 2005 till 7 december, 2005. Latensen för en satellit går även enkelt att räkna ut matematiskt. Avståndet från Sirius2-satelliten som Icomera använder sig av, till Sverige är ungefär 42000 km. Tiden det tar för ett paket att sändas räknas ut med formeln:

$$t = \frac{s}{v}$$

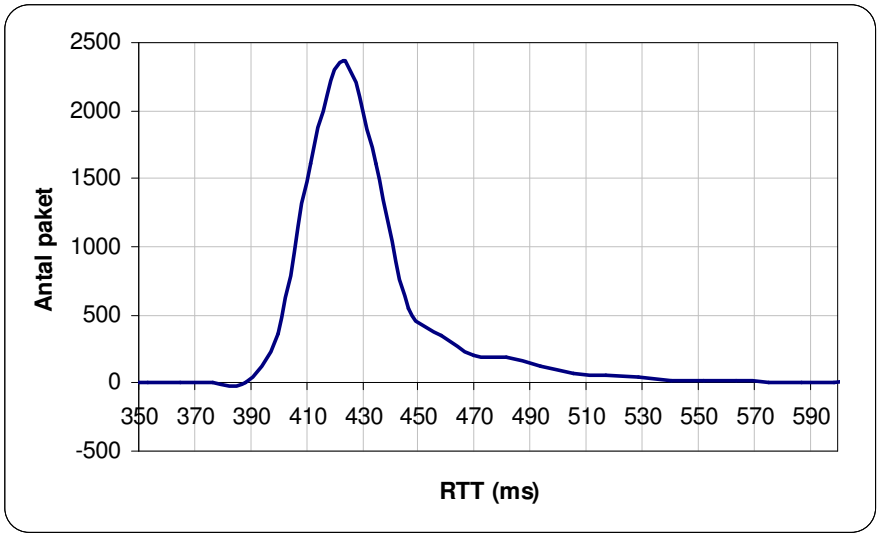
där

s = 42 000 000 m

v = ljusets hastighet = $3*10^8$ m/s

t blir efter uträkning 0.14 s.

Det betyder att endast paketets färd genom luften tar 140 millisekunder. Eftersom Icomera använder sig av en enkelriktad satellit kommer mätvärdena i histogrammet i Figur 3-4 även har mätvärden från GPRS och 3G. [11]



Figur 3-4: Histogram över RTT för satellit

4 Krav & Design

4.1 Introduktion

I detta kapitel listas alla krav med en kort förklaring, se avsnitt 4.2. Det redogörs även för val av programspråk och utvecklingsmetod i avsnitt 4.3.1 och 4.3.2. Avslutningsvis beskrivs simulatorns moduler, samt dess komponenter, i avsnitt 4.3.3.

4.2 Krav

Nedan följer Icomeras detaljerade kravlista på simulatorn. Kraven är prioriterade mellan noll och tre, där noll är högst och tre lägst prioritet. Listan på krav under varje prioritet är inte inbördes prioriterade utan numrerade endast för identifiering.

Prioritet 0:

På en länk ges möjlighet att:

A. Få in länkkarakteristik i simulatorn, utläst ur Icomeras egna protokolls trafikdata.

   När en anslutning av en länk upprättas skickas karakteristik om länken med Icomeras egna protokol. Denna information ska läsas in i simulatorn.

B. Specificera bandbredd.

   Att kunna ange bandbredd på en länk för att t.ex. simulera en 3G-länk med en hastighet på 384 kbit/s. Bandbredden ska variera efter en konfigurerbar normalfördelning.

C. Specificera latens.

   Att kunna ange latens på en länk för att t.ex. efterlikna en GPRS-länk med en latens på 950 ms. Latensen ska variera enligt en konfigurerbar poissonfördelning.

D. Specificera bitfel.

   Ange en sannolikhet att en bit blir fel, vilket ska resultera i en paketförlust. Bitfelen ska variera enligt en konfigurerbar binomialfördelning.

E. Stänga av upp- samt nedtrafik.

Att kunna strypa trafik på en länk åt båda hållen för att efterlikna t.ex. en enkelriktad satellitlänk, där data endast tas emot.

Det ska finnas förinställda värden av ovanstående egenskapar för följande länkar:

GPRS – Dataöverföring i GSM-nätet, se avsnitt 3.3.1.

UMTS 128 – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

UMTS 384 – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

UMTS HSDPA – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

Satellit – Dataöverföring via satellit, se avsnitt 3.3.3.

Prioritet 1:

A. XML-standard på konfigurationsfiler.

Där textfiler förekommer för konfiguration eller datalagring ska XML användas.

B. Ett användarvänligt grafiskt användargränssnitt (GUI).

GUI:t ska vara lätt att använda och det ska gå att övervaka datatrafiken genom simulatorn. GUI:t ska exekveras som en applikation.

C. Simulatormotorn ska exekveras som en Microsoft windowstjänst (eng: Microsoft Windows Service).

Simulatormotorn ska köras i bakgrunden och exekveras som en windowstjänst.

Prioritet 2:

A. Beräkna kostnad på en länk med avseende på megabyte och tid.

Det ska vara möjligt att ange en kostnad i kronor på en länk, antingen med avseende på tid eller på mängd överförd data

B. Använda skript för att automatisera händelseförlopp.

Det ska vara möjligt att via en skriptfil ändra data i simulatorn vid angiven tidpunkt.

C. Ha stöd för flera tågklienter samtidigt.

Det ska ges möjlighet att koppla upp flera tågklienter (IMSR:er) till simulatorn.

Prioritet 3:

A. Spara ned Icomeras egna protokolls trafikdata.

Det ska vara möjligt att spara ned Icomeras egna protokolls trafikdata i ett lämpligt filformat. Det ska i efterhand vara möjligt att öppna filen och studera trafikdatat.

4.3 Design

I följande avsnitt presenteras designen av simulatorn. De beslut som tagits om design har i stor del gjorts på egen hand men i vissa fall har konsultation med utvecklarna på Icomera skett. Vi har försökt att följa de designmönster som används i Icomeras övriga system för att främja att Icomera utan svårighet ska kunna arbete vidare på simulatorn efter projektets slut. Det finns även dokument skrivna av Icomera om bland annat kodstandard, vilka vi följt.

4.3.1 Utvecklingsmetod

När vi började diskutera projektet med Icomera diskuterades hur utvecklingen skulle ske. Det pratades om att en speciell utvecklingsprocess skulle användas. Det fanns inte någon bestämd process i företaget vid den tidpunkten och den metod vi använde blev en variant på vattenfallsmodellen.

Det som först gjordes var en förstudie där Icomeras system studeras. Efter förstudien gjordes en genomgående design. När designen godkänts av vår handledare på Icomera kunde implementationen börja och även den stegvisa utbytningen av den gamla simulatorn. Efter att arbetet med simulatorn var klart verifierades den, vilket visade att kraven var uppfyllda. Varje steg i processen förklaras mer utförligt i följande punktlista:

Steg 1 (Utredning – Tid: 1 vecka): Sätta sig in i Icomeras verksamhet och system. Denna fas bestod av att förstå Icomeras system, genom att läsa dokumentation.

Steg 2 (Utredning – Tid: 1 vecka): Ta fram en kravspecifikation och verifieringsdokument i samarbete med Icomera. Denna fas bestod av att tillsammans med Icomera gå igenom hur de vill att projektet skulle läggas upp, vilka funktionella krav de hade, i vilket programspråk utvecklingen skulle ske och om vi skulle återanvända kod eller skriva om allt från början.

Steg 3 (Design – Tid: 2 veckor): Göra en övergripande design av systemet. Denna fas innebar att vi tog kraven från steg 2 och den kunskap vi hade om systemet från steg 1 och försökte göra en övergripande design av systemet. Den övergripande designen innebar att hitta naturliga avgränsningar och indelningar av moduler. När dessa var identifierade gjordes en design för varje enskild modul.

Steg 4 (Prototyp – Tid: 1 vecka): Identifiera och göra prototyper av de kritiska systemdelarna. Denna fas innebar att vi identifierade, tillsammans med utvecklarna på Icomera, de delar av designen som kunde orsaka eventuella problem. Vi tog sedan de problematiska delarna och gjorde prototyper.

Steg 5 (Design – Tid: 1 vecka): Justera designen utefter prototyperna i steg 4. Denna fas innebar att vi justerade designen efter de problem och möjliga förbättringar som vi hittade i steg 4.

Steg 6-9 (Implementation – Tid: 4 veckor): Göra fyra implementerings-iterationer där vi implementerade designen från steg 5. I dessa iterationer implementerades kraven. De högst prioriterade kraven implementerades först. Vid slutet av varje iteration hade vi ett möte med vår handledare på Icomera och gick igenom hur vi låg till tidsmässigt och gjorde justeringar i

kravlistan ut efter detta (lågt prioriterade krav togs bort om de inte ansågs hinnas med inom tidsramen för projektet).

Steg 10 (Verifiering – Tid: 1 vecka): Verifiera simulatorn mot alla testfall i verifieringsdokumentet. Denna fas innebar att vi satte upp simuleringsmiljön i Icomeras laboratorium och kontrollerade att simulatorn uppfyllde alla testfall i verifieringsdokumentet.

Steg 11-N (Implementation & Verifiering): Justera simulatorn tilldess att den uppfyllde alla testfall. Dessa faser innebar att vi fick göra konfigurerings- och implementationsändringar tills dess att simulatorn uppfyllde alla testfall.

4.3.2 Programspråk

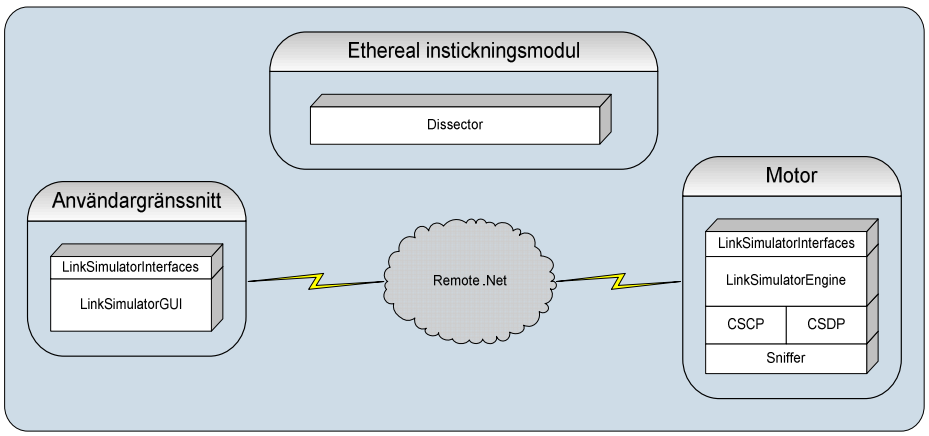
När vi från början diskuterade valet av programspråk handlade det om C++ eller .NET C#. Det fanns två huvudsakliga anledningar att använda C++. Den första var att den tidigare versionen av simulatorn var skriven i C++ och vi kunde återanvända kod från den. Den andra anledningen var att utvecklarna på Icomera till största delen bara kunde C++ och därför kunde få svårt att ge guidning om vi skulle få problem relaterade till programspråket.

Det fanns även två huvudsakliga anledningar att använda .NET C#. Den första var att programmet, enligt våra egna erfarenheter, skulle gå snabbare att utveckla eftersom ramverket .NET tillhandhåller färdig funktionalitet. Den andra anledningen var att .NET tillhandahåller en skräpsamlare samt en mer utbyggd undantagshantering vilket underlättar både i prototyp- och implementationsfasen.[12]

I slutändan valde vi att använda .NET C#, i och med att Icomera ville använda simulatorprojektet som ett test för hur .NET C# fungerar vid prestandakritiska operationer. Studier av den gamla simulatorkoden visade också att den inte gick att återanvända och vi beslutade att börja om från början. Vi ansåg att .NET C# skulle göra både design och prototypfasen enklare än om vi använde C++.

Den mesta av koden i rapporten är skriven i .NET C# och en förståelse för dess syntax och ramverk är nödvändig för att förstå vissa delar av koden. Att ge denna förståelse ligger utanför rapporten och istället hänvisas läsaren till referens [13].

4.3.3 Moduler och Komponenter



Figur 4-1: Simulatormodulernas sammankoppling.

För att göra designen lätt att underhålla och uppdatera i framtiden har vi delat in simulatorn i tre stycken moduler, användargränssnitt, motor och Ethereal instickningsmodul. Modulerna är i möjligaste mån fristående från varandra. För att få en bild över hur de olika modulerna sitter samman och vilka komponenter de innehåller se

Figur 4-1.

4.3.3.1 Motor

Motorn är den modul där simulerandet av länkar sker. Motormodulen körs som en windowstjäns på simulatordatorn i labboratoriet (dator B Figur 2-7). Motormodulen består av fem stycken komponenter, LinkSimulatorEngine (LSE), LinkSimulatorInterfaces, CSCP- och CSDP-avkodare och Sniffern. För en detaljerad beskrivning av motorn och dess implementation, se avsnitt 5.

LinkSimulatorEngine (LSE)

LinkSimulatorEngine-modulen (LSE) är hjärtat i systemet och är den komponent där simuleringen av länkarna sker. LSE:n knyter samman andra komponenters funktionalitet samt tillhandahåller egen för att kunna simulera trådlösa länkar. För att undvika att LSE:n blir för stor har vi flyttat ut användargränssnittet till en egen modul (se 4.3.3.2) och all interaktion mot LSE:n sker via de gränssnitt som finns i LinkSimulatorInterfaces.

LSE:n fungerar enligt följande; trafik från och till IMSR:en passerar genom LSE:n. LSE:n ser vilken länk paketet skickades på och fördröjer paketet enligt den inställda länkkarakteristiken. För att LSE:n ska kunna identifiera från vilken länk de inkommande datapaketen kom, används CSCP- och CSDP-avkodarkomponenten. Icomera hade även som krav att all datatrafik skulle sparas så att användaren skulle kunna studera den i efterhand. För att åstadkomma det använder LSE:n snifferkomponenten.

LinkSimulatorInterfaces

LinkSimulatorInterfaces-modulen innehåller alla de gränssnitt som används för kommunikation mellan LinkSimulatorEngine och olika användargränssnitt (i den här versionen av simulatorn finns det bara ett användargränssnitt, nämligen LinkSimulatorGUI). Gränssnitten ligger i en egen modul för att skapa en tydlig separation mellan motorn och användargränssnitt. Eftersom all kommunikation sker via denna komponents gränssnitt så behöver inte LSE:n känna till något om användargränssnitten. Detta gör det möjligt att lägga till nya användargränssnitt samt ändra i de gamla utan att ändra i LSE:n.

CSCP- och CSDP-avkodare

Client Server Control Protocol (CSCP) och Client Server Data Protocol (CSDP) är Icomeras egenutvecklade protokoll för kommunikation mellan IMSR och IGW. CSCP- och CSDP-avkodarkomponenten konverterar de inkomna paketen från byte-arrayer till protokollstrukturer. Protokollstrukturerna innehåller bl.a. information om vilken IMSR och länk paketet skickas till/från.

Sniffern

Snifferkomponenten avlyssnar en specifik IP-adress och port. All datatrafik som passerar genom sniffern sparas till en fil med filformatet libpcap, se avsnitt *5.6.1* för information om libpcap. Filen kan sedan öppnas i t.ex. Ethereal för analys.

4.3.3.2 Användargränssnitt

Användargränssnittsmodulen består bara av en komponent, nämligen LinkSimulatorGUI-komponenten (LSGUI). LSGUI:t komponent sköter all interaktion från användaren och förmedlar den vidare till LSE:n via de gränssnitt som exponeras via LinkSimulatorInterfaces. Anropen till LSE:n sker via .NET Remoting, se avsnitt 6.3. En beskrivning av användargränssnittet med dess uppbyggnad och funktionalitet ges i avsnitt 0.

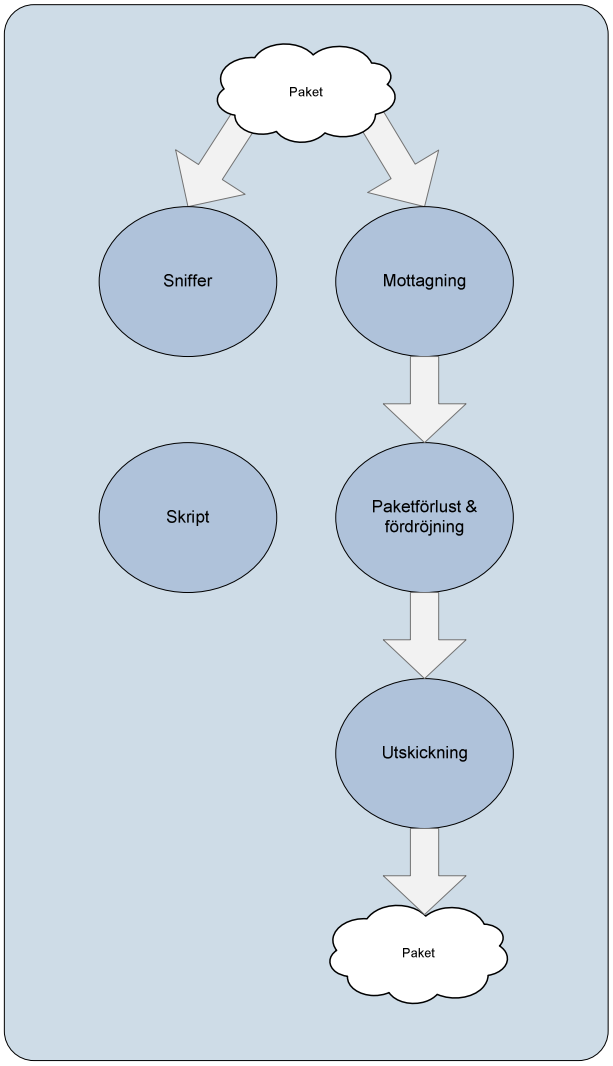
4.3.3.3 Etherealdissektor

Ethereal är en paketsniffer med funktioner för att analysera datatrafik. Ethereal är ett projekt med öppen källkod och används av Icomera då det inte kräver licensavgifter. Ethereal har i grundinstallationen bara stöd för standardprotokoll så som HTTP, FTP, m.m. men det är möjligt att lägga till stöd för nya protokoll via instickningsmoduler. Etheraldissektorn är en sådan instickningsmodul som ger stöd för CSCP-protokollet. Detta gör det möjligt att studera Icomeras protokoll på samma sätt som standardprotokollen. För ytterligare information om Ethereal hänvisas till deras officiella hemsida [14].

5 Implementation - Motorn

5.1 Introduktion

När simulatordesignen gjordes var tanken att få ett flöde som efterliknar verkligheten. Med det menas att simulatorn ska efterlikna de markstationer som visas i Figur 2-8, med skillnaden att det i simulatorn ska vara möjligt att justera länkarnas karakteristik.



Figur 5-1: Paketflödet genom motorn.

Vi ville också skapa en simulator som var oberoende från IMSR:n och IGW:n. Anledningen till det var att vi inte ville att IMSR:n eller IGW:n skulle behöva specialkonfigureras när de användes i simuleringsmiljön.

I Figur 5-1 visas det övergripande paketflödet genom motorn. Figuren visar hur ett paket först skickas både till snifferlagret och till mottagningslagret. Snifferlagret ansvarar för att spara ned alla inkommande paket, för att användaren ska kunna analysera datatrafiken i efterhand. Snifferlagret beskrivs i avsnitt 5.6. I mottagningslagret avkodas paketet och information om t.ex. vilken länk och IMSR paketet skickats från läses ut. Mottagningslagret beskrivs i avsnitt 5.2. Efter att paketet har passerat mottagningslagret åker det vidare till paketförlust- och fördröjningslagret. I paketförlust- och fördröjningslagret tas eventuellt paketet bort, vilket resulterar i en paketförlust, eller så räknas en paketfördröjningen ut. Paketförlust- och paketfördröjningsuträkningarna baseras på karakteristiken för den länk som paketet skickades på. Paketförlust och fördröjningslagret beskrivs i avsnitt 5.3. Från det lagret åker paketet vidare till utskickningslagret. I utskickningslagret appliceras den uträknade paketfördröjningen och paketet fördröjs enligt den uträknade tiden. Efter att paketet har fördröjts, skickas det mot sin slutdestination. Utskickningslagret beskrivs i avsnitt 5.4. Det finns även ett skriptlager som hanterar all skriptfunktionalitet. Skriptlagret interagerar med motorn via samma gränssnitt (LinkSimulatorInterfaces, se avsnitt 4.3.3.1) som användargränssnittet och gör det möjligt att skapa automatiserade testflöden. Skriptlagret beskrivs i avsnitt 5.5.

5.2 Mottagning

I det första steget i motorn fångas paketen upp med hjälp av sockets som lyssnar efter alla paket som skickas på port 1 (CSDP) och 3 (CSCP) (socketimplementationen har begränsningar som förklaras i avsnitt 8.4). När ett paket anländer används CSDP- och CSCP-avkodarmodulen för att läsa ut paketinformation. CSDP- och CSCP-avkodarmodulen konverterar bitströmmen till ett objekt, där t.ex. information om vilken länk paketet skickades på, går att avläsa (att konvertera från en bitström till ett objekt

innebär problem, vilka är förklarade i avsnitt 8.5). Om avkodarmodulen inte kan avkoda ett paket kastas paketet bort. Paket som innehåller bitfel kommer alltså inte att kunna skickas genom simulatorn, i och med att avkodarmodulen inte kan avkoda dem. Simulatorn kan därför inte användas för att testa huruvida IMSR:en och IGW:n klarar att hantera paket innehållande bitfel.

För att kunna ta emot data via sockets asynkront används trådar. Det används en tråd för varje socketanslutning, för att asynkront kunna ta emot data. Dessa trådar ställer till precisionsproblem, vilket är förklarat i avsnitt 8.3.

Slutligen, om paketet lyckats avkodats av CSDP- och CSCP-avkodarmodulen, skickas paketet till ett länkobjekt. Länkobjektet representerar en enskild 3G-, GPRS eller satellitlänk mellan IMSR:n och IGWN:n, med inställbara egenskaper för bandbredd, latens och paketförluster. När paketet anländer till länkobjektet beräknas paketförlust och fördröjning, vilket är förklarat i avsnitt 5.3.

5.3 Paketförluster & Fördröjning

När ett paket anländer till ett länkobjekt räknas paketförlust och fördröjning ut med hjälp av länkobjektets karakteristik. Att räkna ut paketförlusten för ett paket innebär att länkobjektet räknar ut om ett paket ska kasta eller inte. Hur paketförlusten räknas ut är förklarat i avsnitt 5.3.1. Om länkobjektet konstaterar att paketet inte ska kastas räknas paketfördröjningen ut för paketet. Paketfördröjningen räknas ut genom att addera ihop latensfördröjningen och bandbreddsfördröjningen för att få den totala paketfördröjningen, se Figur 3-1 för bild av förhållandet mellan latens och bandbredd. Hur bandbreddsfördröjning och latensfördröjning räknas ut förklaras i avsnitt 5.3.2, respektive avsnitt 5.3.3.

Slutligen när paketfördröjningen är uträknad läggs paketet i utskickningskön, vilket beskrivs i avsnitt 5.4.

5.3.1 Paketförluster

Varför paketförluster uppstår diskuteras i avsnitt 3.2.3. För att implementera paketförluster studerade vi två olika metoder. Det första var att kasta paket efter en viss fördelning och sannolikhet. Problemet med denna metod är den inte tar hänsyn till storleken på paketen. Ett paket på 1500 bytes har lika stor sannolikhet att kastas bort som ett som är 40 bytes.

I metod två räknar vi istället på att en bit blir fel. Det gör att vi tar hänsyn till paketens storlek. För att slippa räkna på varje bit använder vi oss en binomialfördeling. Biomialfördelningen tar två inparametrar, antal bitar och antal bitar med fel. Svaret blir sannolikheten att paketet innehåller fel. Vi tar svaret från biomialfördelningen och om det är noll låter vi paketet vara, annars slänger vi det eftersom det innehåller ett eller flera bitfel (ett eller x fel spelar ingen roll, paketet kastas i vilket fall). [15]

En stokastisk variabel X säges vara binomialfördelad med parametrarna n och p om:

$$\Omega = \{0,1,2,...,n\}$$

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k} \qquad k \in \Omega$$

där i vårat fall

n = antal bitar i paketet

p = sannolikheten att en bit går fel

k = antal bitar som går fel

I vårat fall vill vi bara veta om ett paket ska kastas eller inte. Det betyder att antal bitar som går fel måste vara noll (k=0). Det spelar alltså ingen roll om det är x eller y fel, paketet kommer att slängas i båda fallen. I och med att k=0 kan vi förenkla formeln:

$$P(X = 0) = \binom{n}{0} p^0 (1-p)^{n-0} = (1-p)^n$$

Programkoden för implementationen av binomialfördelningen ser ut enligt följande:

```
public static double BinomialProbability(int n, double p)
{
      return Math.Pow((1-p), n);
}
```

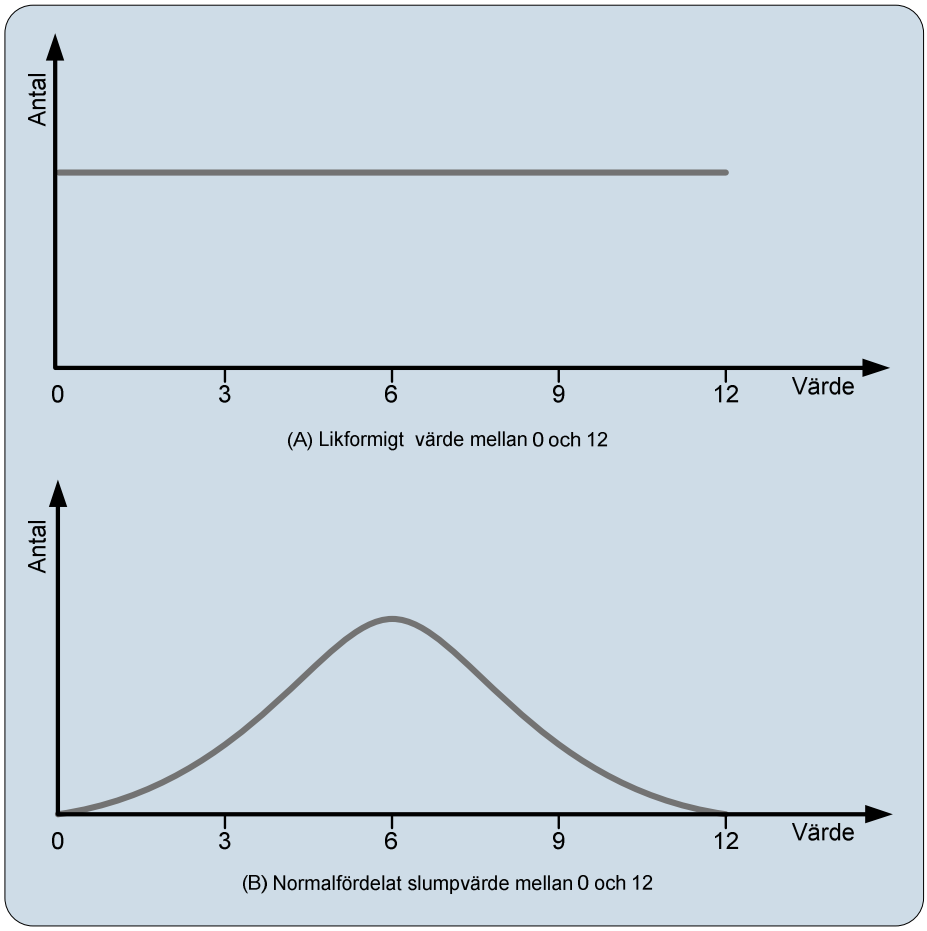
där *n* och *p* är definierade som i formeln $P(X=0)$. Returvärdet från den här funktionen jämförs sedan med ett slumptal mellan 0 och 1. Om slumptalet är större är returvärdet, kastas paketet bort. Den funktionen ser ut enligt följande:

```
private bool PacketLoss(Entity entity)
{
      bool drop = false;
      double bitErrorRateProbability = (entity.FromClient ?
            characteristics.OutBitErrorRateProbability :
            characteristics.InBitErrorRateProbability);

      if(random.NextDouble() > Distribution.BinomialProbability(
            entity.Packet.Size*BITS_IN_BYTE,
            bitErrorRateProbability))
            drop = true;

      return drop;
}
```

Funktionen börjar med att räkna ut vad sannolikheten är att ett paket av en viss längd blir fel. Värdet kommer att ligga mellan 0 och 1, där 1 innebär att det inte blir något fel och 0 att det garanterat blir fel. Vi slumpar sedan ett tal mellan 0 och 1 med hjälp av funktionen NextDouble. De två talen jämförs med varandra och om slumptalet är större än sannolikheten att paketet blir fel tas paketet bort.

5.3.2 Bandbredd

Bandbredd talar om hur fort data kan skickas på en länk (se definition i avsnitt 3.2.1). Vi har använt oss av två parametrar för att specificera bandbredd, förväntad bandbredd och avvikelse. Vi har valt att använda en normalfördelad slumptalsgenerator för att räkna ut bandbredden. Det gör vi för att få variation på bandbredden som kretsar runt det förväntade värdet. Skillnaden att använda en normalfördelad variabel jämfört med en likformig, visas i Figur 5-2.



Figur 5-2: Jämförelse av en likformig och normalfördelad variabel.

Figuren visar hur X antal slumpade variabelvärden mellan 0 och 12 fördelar sig enligt de två olika fördelningarna. Eftersom fördelningsfunktionen för normalfördelning är given av en integral går det inte att finna en enkel invers utan vi förlitar oss istället på centrala gränsvärdessatsen. Centrala gränsvärdessatsen säger att en summa av oberoende likfördelade stokastiska variabler är normalfördelade. Vi genererar 12 stycken likformigt fördelade tal U_i från alla reella tal mellan 0 och 1, d.v.s. $R(0,1)$. Summan av dessa 12 tal kommer ligga mellan 0 och 12, men oftast runt 6 enligt centrala gränsvärdessatsen. Genom att sedan subtrahera 6 från summan, så kommer summan ligga mellan -6 och 6, men kretsa mest kring 0. Den matematiska formeln ser ut enligt:

$$\sum_{i=1}^{12} U_i - 6 \approx N(0,1)$$

Översättningen av den här formeln till programkod ser ut enligt följande:

```
public static double GenerateStandardNormal()
{
        double x = -6;
        for(int i=1; i<=12; i++)
                x += random.NextDouble();
}
```

Funktionen är statisk och returnerar ett värde mellan -6 och 6. Funktionen *NextDouble* returnerar ett värde mellan noll och ett. Funktionen kommer returnera 0,5 som medeltal, vilket gör att -6 + 12*0,5 är lika med 0. Därmed kommer medeltalet från funktionen att vara noll och de flesta värden kommer att kretsa runt noll. [15]

5.3.3 Latens

Tiden det tar för ett paket att ta sig från A till B kallas för latens (se definitionen i avsnitt 3.2.2). Efter att ha studerat graferna av latens från de trådlösa länkarna som Icomera använder i avsnitt 3.3, kom vi fram till att graferna liknar i stor del en poisson 4 fördelning. Alltså en poissonfördelning med det förväntade värdet 4.

En stokastisk variabel X sägs vara poissonfördelad med parametern λ om:

$$\Omega = \{0,1,2,3,...\}$$

$$P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda} \qquad k \in \Omega$$

För att kunna räkna ut en poisson 4-fördelad variabel används vi sambandet som fås från referens [15]:

$$P(X = k+1) = \frac{\lambda}{k+1} P(X = k)$$

Med hjälp av dessa två formler kan vi skapa en algoritm för att få fram en poissonfördelad variabel:

```
public static int RandomPoisson4()
{
        int a = 4;
        int x = 0;
        double p = System.Math.Exp(-a);
        double F = p;
        do
        {
              double u = random.NextDouble();
              if(u<F)
                    return x;
              p = (a * p)/(++x);
              F += p;
        }while(true);
  }
```

I och med att F ökar i varje cykel blir det större och större sannolikhet att u blir mindre än F. Sannolikhet för att u är större än F i de första cyklerna beror på vilket λ (a) som används. I vårt fall använder vi oss av $\lambda = 4$ vilket ger det initiala värdet på $F = e^{-4} \approx 0{,}0183$. Det är inte så stor sannolikhet att ett slumptal i domänen $R(0,1)$ ligger under det F den första cykeln. Efter varje cykel ökar F vilket ger en poissonfördelning av variabeln x. [15]

5.4 Utskickning

Efter att paketfördröjningen är uträknad anländer paketet till utskickningslagret. I utskickningslagret får paketet en tidstämpel när det ska skickas ut och placeras sedan in i en sorterad lista. Listan är sorterad efter utskickningstid, där paketen som ska skickas ut först, ligger överst i listan. Funktionen för insättningen i den sorterade listan ser ut enligt följande:

```
public void Add(Entity entity)
{
	System.Threading.Monitor.Enter(sendList);
	int i = 0;
	while(sendList.Count > i && entity.OutTime > ((Entity)sendList[i]).OutTime)
		i++;
	sendList.Insert(i, entity);
	System.Threading.Monitor.Exit(sendList);
}
```

Först låser vi listan så att ingen annan tråd kan manipulera den samtidigt. Sedan itererar vi igenom listan för att hitta indexet där paketet ska ligga. Den första entiteten (paketet) i listan är det som ska skickas ut härnäst. Sedan ligger alla paket sorterade efter den tid som de ska skickas ut.

Vi har nu en sorterad lista som fylls på med paket. Paketen har en tidstämpel när de ska skickas ut. För att skicka paketen används följande funktion:

```
public void Send()
{
	Entity entity = null;
	while(true)
	{
		if(shutdown == true)
			break;
		try
		{

			System.Threading.Monitor.Enter(sendList);
			while(sendList.Count > 0 &&
				((Entity)(sendList[0])).OutTime <= Time.Instance.Current)
			{
				entity = (Entity)sendList[0];
				sendList.RemoveAt(0);
				if(entity != null && entity.OutSocket != null)
				{
					entity.OutSocket.SendTo(
						entity.Packet.Data,
						entity.Packet.Size,
						System.Net.Sockets.SocketFlags.None,
						entity.Packet.ToEndPoint);
					entity.InSocket.ReleaseEntity(entity);
				}
				else
					entity.OutSocket = null;
			}
			System.Threading.Monitor.Exit(sendList);
			System.Threading.Thread.Sleep(1);
		}
		catch(Exception e)
		{
			Log.Instance.AddException(e);
		}
	}
}
```

Varje millisekund tittar vi på första paketet i listan. Om paketets uttid är mindre är den nuvarande tiden plockas paketet bort från listan och skickas iväg. Detta upprepas tills dess att alla paket har skickas som har en uttid som

är mindre än den nuvarande tiden. När det är gjort väntar tråden i en millisekund innan den upprepar processen igen.

Pressionen i utskickningsmomentet är något problematisk. Anledningen är att utskickning intervallerna inte kan vara mindre än 1 millisekund (Vi skulle kunna göra så att tråden hela tiden försöker skicka, med det skulle resultera i att andra trådar skulle bli påverkade negativt) och därav kan vi bara uppnå en maxhastighet på 12 Mbit/s innan vi får problem med precisionen. I avsnitt 8.2 beskrivs detta problem mer i detalj och det ges även förslag på eventuella lösningar på problemet.

5.5 Skript

För att automatisera händelser i simulatorn används skript. Händelser som kan ske i skript kan vara att t.ex. en länks bitfelssannolikhet ändras eller att bandbredden sätts till noll för att strypa länken. Anledningen till att ett skript är användbart för Icomera är att länkarnas karakteristik varierar kraftigt under en tågresa. Skripten kan då användas för att efterlikna en tågresa, utan att användaren manuellt måste justera länkkarakteristiken. Skript gör det även möjligt att upprepa samma beteende flera gånger, t.ex. vid verifiering.

Skriptmotorn kan ses som ett användargränssnitt som tar emot händelser från en fil istället för användaren. Skriptmotorn använder samma gränssnitt (LinkSimulatorInterface) som det grafiska användargränssnittet för att utföra de olika skripthändelserna.

Skriptet är uppbyggt med Extensible Markup Language (XML). I Figur 5-3 visas hur ett skript kan byggas upp i XML. I Tabell 5-1 visas alla kommandon som användaren kan tillämpa i ett skript.

Kommando	Beskrivning
UpExpBandwidth	Talar om vad den förväntande upphastigheten ska vara på länken. Värdet måste vara numeriskt och enheten är bits/s
DownExpBandwidth	Talar om vad den förväntande nedhastigheten ska vara på länken. Värdet måste vara numeriskt och enheten är bits/s
UpExpLatency	Talar om den förväntade upplatensen som ska vara på länken. Värdet måste vara numeriskt och enheten är nanosekunder.
DownExpLatency	Talar om den förväntade nedlatensen som ska vara på länken. Värdet måste vara numeriskt och enheten är nanosekunder.
UpBitRateError	Talar om det uppbitfel som ska vara på länken. Värdet är ett flyttal och anger hur stor sannolikhet det ska vara att ett bitfel inträffar i ett paket.
DownBitRateError	Talar om det nedbitfel som ska vara på länken. Värdet är ett flyttal och anger hur stor sannolikhet det ska vara att ett bitfel inträffar i ett paket.
UpAllowTraffic	Sätter om upptrafik ska tillåtas. Värdet är ett booleanskt värde.
DownAllowTraffic	Sätter om nedtrafik ska tillåtas. Värdet är ett booleanskt värde.
CaptureConfig	Anger om paketsniffning ska startas eller stoppas. Värdet måste vara ett numeriskt värde, antigen 0 eller 1.
GeneratePackets	Startar en paketgenerering. Värdet ska vara et sträng med syntaxen ”<<värddator>> <<program>> <<parameters>>

Tabell 5-1: De olika kommandona i ett automatiseringsskript.

Exempelskriptet i Figur 5-3 börjar med deklarationen av en händelselista (*EventList).* Händelselistan innehåller ett godtyckligt antal händelsenoder (*Event*). Händelsenoderna innehåller i sin tur ett *Time*-attribut som talar om efter hur många sekunder händelsen ska utföras. Tid noll är när skriptet startats, så i exempelskriptet i Figur 5-3, kommer första händelsen ske efter 5 sekunder efter att skriptet startats. Händelsenoden innehåller även ett *LinkTag*-attribut som talar om på vilken länk händelsen ska tillämpas.

Vad som kommer att ske i vid varje händelse bestäms av kommandonoden (*Command)* inuti händelsenoden. Det kan finnas ett godtyckligt antal kommandon i en händelse, men det får bara finnas ett kommando av varje typ. Det får t.ex. inte finnas två kommandon med i samma händelse med *CommandType* attributet satt till *UpExpBandwidth.*

```
<EventList StartTime="0">

<Event Time="5" LinkTag="foobar">
<Command CommandType="UpExpBandwidth" Value="1000000"></Command>
<Command CommandType="DownExpBandwidth" Value="1000000"></Command>
</Event>

…

<Event Time="10" LinkTag="foobar">
<Command CommandType="UpExpLatency" Value="10000000"></Command>
<Command CommandType="DownExpLatency" Value="10000000"></Command>
</Event>

</EventList>
```

Figur 5-3: Exempelskript för automatisering

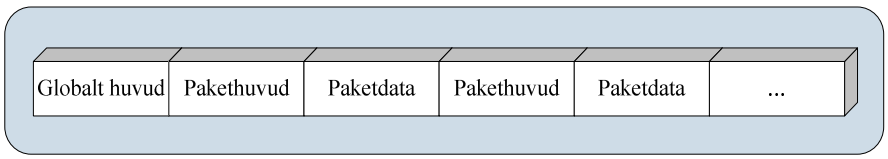
5.6 Sniffer

Sniffern är en komponent som likt motorn lyssnar på datatrafik. Snifferkomponenten sparar den inkommande datatrafiken till en fil med filformatet libpcap. Libpcap-formatet är förklarat i avsnitt 5.6.1. Anledningen till att datatrafiken sparas är för att utvecklarna ska kunna analysera trafikflödet efter olika tester. För att analysera datatrafik använder Icomera en paketsniffer vid namn Ethereal. Ethereal sniffar paket i realtid och visar resultatet i ett grafiskt användargränssnitt. Det är även möjligt att i Ethereal öppna sparade filer, med resultatet från gamla sniffningar. De sparade filerna måste vara i filformatet libpcap, därav anledningen till att vi sparar datatrafiken just i det formatet.

I grundinstallationen kan Ethereal bara avkoda standardprotokoll såsom HTTP, FTP, DHCP m.m. För att Ethereal ska kunna avkoda Icomeras egna CDCP- och CSCP-protokol behövs en instickningsmodul. Hur instickningsmodulen fungerar och är implementerad beskrivs i avsnitt 5.6.2.

5.6.1 Filformat libpcap

Libpcap är ett enkelt sätt att spara datatrafik. Det finns olika versioner av libpcap, men idag är den vanligaste versionen av formatet 2.4. Formatet består av ett globalt huvud för hela filen och därefter ett huvud för varje paket. I Figur 5-4 visas hur filen är uppbyggt.



Figur 5-4: Libpcaps filstuktur.

Det globala huvudet består av ett antal fält enligt:

```
typedef struct pcap_hdr_s {
        guint32 magic_number;   /* magic number */
        guint16 version_major;  /* major version number */
        guint16 version_minor;  /* minor version number */
        gint32  thiszone;       /* GMT to local correction */
        guint32 sigfigs;        /* accuracy of timestamps */
        guint32 snaplen;        /* max length of captured packets */
        guint32 network;        /* data link type */
} pcap_hdr_t;
```

där

guint32 = 32 bitars positivt heltal.
guint12 = 16 bitars positivt heltal
gint32 = 32 bitars heltal

Det första fältet som är ett magiskt nummer talar om i vilken ordning byten kommer. För att få den ursprungliga ordningen anges värdet ”0xa1b2c3d4”. För att byta plats på bytarna används värdet ”0xd4c3b2a1”. Det andra och tredje fältet beskriver vilken version som ska användas. Det är idag 2.4 som används av Ethereal. Det fjärde fältet beskriver hur tiden förhåller sig till ”Greenwich Mean Time” (GMT) för att paketens tidstämpling ska kunna räknas ut rätt. Det femte fältet talar om vilken

upplösning infångandet av paket ska ske. För att få så bra upplösningen som möjlig sätts denna till noll. Det sjätte fältet är den maximala storleken på paketet, i bytes. Det är vanligtvis 65535 bytes, men kan begränsas om så önskas. Det sjunde och sista fältet beskriver vilken typ av nätverk som ska avlyssnas. Det är för oss Ethernet. [16]

Strukturen för pakethuvudet har utformandet:

```
typedef struct pcaprec_hdr_s {
        guint32 ts_sec;         /* timestamp seconds */
        guint32 ts_usec;        /* timestamp microseconds */
        guint32 incl_len;       /* number of octets of packet saved in file */
        guint32 orig_len;       /* actual length of packet */
} pcaprec_hdr_t;
```

Där

guint32 = 32 bitars positivt heltal.

I huvudet kommer alltså information om det enskilda paketet att finnas. Det första värdet talar om vilken sekund sedan första januari 1970 GMT som paketet anlände. Det andra fältet talar om vilken mikrosekund paketet anlände. Värdet kan inte bli mer än 1 000 000, för om så vore fallet ökas sekunderna med 1 istället. Det tredje fältet är storleken på hur mycket data som är sparat i filen, vilket förhåller sig till det fjärde och sista fältet som är storleken på paketet.

Libpcap har vissa nackdelar. Idag när datorerna har blivit snabbare skulle en upplösning på nanosekunder vara önskvärt. En till nackdel är att det inte finns information om vilket nätverksgränssnitt som har använts, t.ex. nätverkskort. Ytterliggare en nackdel är att det inte finns information om t.ex. paketförluster eller annan tänkvärd statistik.

Vår implementation av filformatet och avlyssning av paket går till enligt följande. Först sparas det globala huvudet ned med generell information av paketet. När ett paket sedan anländer tidstämplas det och sparas kontinuerligt ned i filen. En hastighetsaspekt skulle vara att buffra ett visst

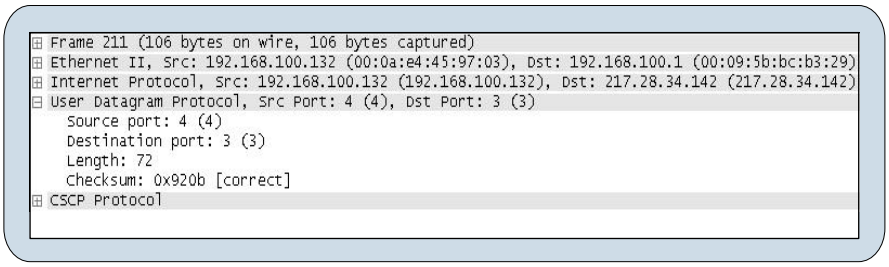
antal paket innan paketet sparas ned för att minska på I/O-anropen. Det har dock inte upplevts någon prestandaförlust.

5.6.2 Etherealdissektor

Ethereal är skrivet i programspråket C och inkluderar flera andra open-source-projekt. För att skriva en dissektor till Ethereal finns det två tillvägagångssätt. Det första är att skriva in det i kärnan i Ethereal. Det andra att skriva en instickningsmodul som sedan läses in vid programstart. Vi valde den andra lösningen, vilket gör att dissektorn blir oberoende av vilken version som används och det gör att även nya versioner av Ethereal kan använda avkodaren. [17]

Det finns en hel del förprogrammerad logik för att implementera en dissektor. Ethereal, som använder sig av WinPcap för att fånga upp paket, tar emot paketen i rådata. Med det menas att all information ända ned från nätverkslagret finns med. Det går att definiera så att alla lager före UDP tas hand om automatiskt. I och med att Icomeras egna protokoll använder UDP passar det bra. Implementationsdetaljerna om hur Icomeras egna protokoll analyseras utelämnas på grund av sekretess.

För att visualisera protokollet i Ethereal används en trädstruktur. Mer detaljerad information hamnar längre ned i trädstrukturen.



Figur 5-5: Skärmdump från Ethereal

Figur 5-5 visar Icomeras egna protokoll (CSCP) längst ned i figuren. Det går även att expandera ”CSCP Protocol” precis som ”User Datagram Protocol” är expanderat i Figur 5-5, för att få mer detaljerad information. [17]

6 Implementation - Användargränssnittet

6.1 Introduktion

Ett av kraven från Icomera var att vi skulle designa om det grafiska användargränssnittet. Det fanns ett flertal anledningar till denna omdesign. Vissa av anledningarna kom tack vare designbeslut, t.ex. att programmet skulle skrivas i C#, till skillnad från det gamla användargränssnittet som var skrivit i C++. Det ansågs då som onödigt att konvertera det gamla utseendet till .NET och sedan försöka lägga till de nya funktionerna som krävs för att uppfylla kraven. En annan anledning till omdesignen var att det gamla användargränssnittet var hårt sammankopplad med motorn. Något som gjorde det omöjligt att lägga motorn på en dator och köra användargränssnittet på en annan. Detta gjorde att användaren var tvungen att sitta i laboratoriet för att använda och övervaka simuleringar. I den nya designen är det grafiska användargränssnittet frikopplat från motorn. För att åstadkomma denna uppdelning användes en design princip som kallas för MVC (Model-View-Controller). Hur denna modelleringsprincip fungerar och vad den ger för fördelar beskrivs i avsnitt 6.2.

Genom att frikoppla användargränssnittet är det möjligt att ändra utseende på applikationen utan att påverka eller ändra motorns beteende. Något som är önskvärt då det ger en möjlighet att skapa individuella vyer av simulatorn. Med det menas att Icomera kan bygga olika utseenden av applikationen beroende på vem som ska använda den och vad den ska användas till. De finns t.ex. önskemål att det skulle gå att köra användargränssnittet i en kommandoprompt, så väl som ett grafiskt användargränssnitt. Detta var önskvärt då Icomera i förlängningen vill kunna styra simulatorn från skript vilket är lättare att göra om simulatorn kan ta in parametrar via textkommandon i stället för knapptryckningar. Samtidigt vill de ha ett grafiskt användargränssnitt där de lätt kan studera grafer och se trender och historik på ett enklare sätt än att bara se siffror utskrivna i det textbaserade gränssnitt.

Ytterligare en anledning till omdesignen var att byta från ett polling-system till ett händelsebaserat system. I den gamla simulatorn låg användargränssnittet och hämtande data från motorn med ett visst tidsintervall, så kallat polling. Det finns två stora problem med denna teknik. Den första är att användargränssnittet hela tiden mellanlagrar data från motorn. Det eftersom motorn hela tiden bara gav en överblicksbild av dess data. Användargränssnittet måste då själv kontroller om den nya datan den får är ändrad sedan föregående uppdatering och då uppdatera de kontroller som är berörda. Mellanlagringen är inte helt nödvändig då det går att designa användargränssnittet så att varje gång information uppdateras så uppdateras alla kontroller, oavsett om dess kontroller är berörda av uppdateringen eller inte. Problemet med det är att uppdateringen mot motorn sker någonstans mellan 10 – 100 gånger i sekunden vilket resulterar i att kontrollerna måste ritas om lika många gånger. Det gör att användargränssnittet får ett flimrande beteende och att processorn belastas i onödan då de flesta av kontrollerna ritas om fast deras data inte har uppdaterats.

Det andra stora problemet med denna design är att det skickas onödigt stora mängder data mellan motor och användargränssnitt. Detta var inget problem i den gamla simulatorn då användargränssnittet och motorn satt ihop i en process. Den nya simulatorn å andra sidan gör det möjligt att köra användargränssnittet och motorn på två olika datorer, vilket gör att data kommer att skickas över någon form av nätverk. För att tydligare se problemet så följer i Tabell 6-1 en sammanställning på data som måste skicka vid varje uppdatering.

IMSR-information	**Antal bytes**
Antal länkar	4
Namn	256
Länkinformation (8 st)	
Id	16
Namn	256
Typ (3G, GPRS, satellit)	4
Upphastighet	4
Nedhastighet	4

Max upphastighet	4
Max nedhastighet	4

Total:	**3164**

Tabell 6-1: Data som skickas om ett inte händelsebaserat system används.

Som Tabell 6-1 visar så måste en IMSR med åtta länkar skicka totalt 3164 bytes vid en uppdatering. Icomera hade som krav att systemet skulle klara av att hantera upp till 200 IMSR:er samtidigt vilket leder till att 632800 bytes måste överföras vid varje uppdatering. Om vi räknar om det till vilka krav det ställer på nätverksuppkopplingen får vi:

6328000 * 8 = 5062400 bits / per överföring

Vi gör 10 – 100 uppdateringar i sekunder för att få en tillräcklig ström av data till användargränssnittet.

5062400 * 10 = 50624000 bits/s ≈ 50 Mbits/s
5062400 * 100 = 506240000 bits/s ≈ 500 Mbit/s

Som synes så är kraven mellan 50 – 500 Mbits/s och det göra att det interna nätverket på Icomera, som har en hastighet på 100 Mbit/s, skulle bli kraftigt belastat varje gång någon använder simulatorn. Det räcker att två användare använder simulatorn samtidigt för att hela nätverket skulle bli överbelastat.

Vid studerande av Tabell 6-1 och vilken data som skickas så är den största delen sådan data som oftast inte uppdateras. Ett bra exempel är namnet på IMSR:en och länkarna, vilket oftast sätts en gång och sedan inte ändras. Namn utgör även den största delen av det data som skickas vid en uppdatering. Att använda ett händelsebaserat system och bara skicka namnet de få gånger det uppdateras skulle reducera kraven på nätverksuppkopplingen rejält. Om vi tar bort all data som inte uppdateras kontinuerlig så får vi Tabell 6-2.

IMSR-information	Antal bytes
Antal länkar	4
Länkinformation (8 st)	
Upphastighet	4
Nedhastighet	4
Max upphastighet	4
Max nedhastighet	4

Total:	**132**

Tabell 6-2: Data som skickas om ett händelsebaserat system används

Genom att räkna som förut för att få fram vad det skulle ställa för krav på nätverksuppkopplingen får vi.

132 * 200 * 8 = 211200 bits / per överföring

Vi gör 10 – 100 uppdateringar i sekunder för att få en tillräcklig ström av data till användargränssnittet.

211200 * 10 = 2112000 bits/s ≈ 2 Mbits/s
211200 * 100 = 21120000 bits/s ≈ 20 Mbit/s

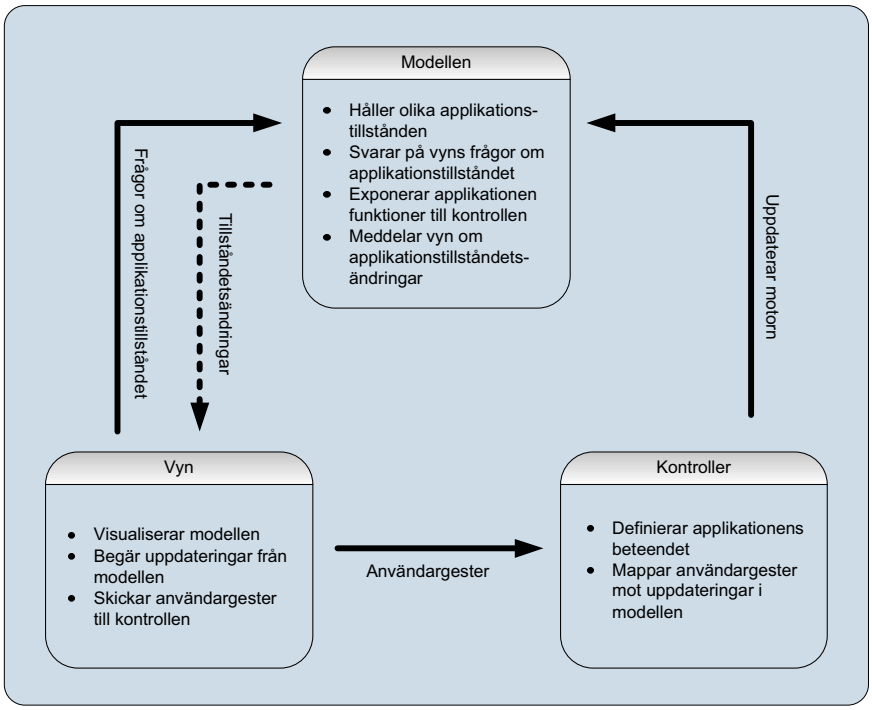
Detta ger en förbättring med faktor 25 jämfört med det gamla polling-systemet. Detta tillsammans med de andra faktorerna, som nämnts tidigare i avsnittet, gjorde att vi bestämma oss för att helt designa och skriva om gränssnittet. Det nya gränssnittet skall byggas efter MVC-principen för att separera gränssnittet och motor och göra det möjligt att köra dem på olika datorer, samt att kommunikationen mellan gränssnittet och motorn skall vara händelsebaserad för att minska kraven på uppkopplingen mellan de två.

6.2 MVC (Model – View - Controller)

MVC-mönstret är ett sätt på vilket applikationen bryts ned i tre olika komponenter; modellen, vyn och kontrollen. MVC var utvecklat för att efterlikna beteendet i användargränssnitt i form av inmatning, bearbetning och utmatning. [18]

Inmatning	>	Bearbetning	>	Utmatning
Kontroller	>	Modellen	>	Vyn

De olika objekten beskrivs i den efterföljande punktlistan samt i Figur 6-1.



Figur 6-1: Övergripande beskrivning av MVC.

- Kontrollen behandlar mus- och tangentbordstryckningar från användaren och översätter dessa kommandon mot operationer i motorn.

- Modellen hanterar all applikationsdata och exponerar gränssnitt för att manipulera den. Den ser även till att notifiera eventuella lyssnare om applikationstillståndet ändras.

- Vyn visualiserar modellen genom olika grafikobjekt och text. Vyn förser även kontroller med användargester, såsom musklickningar m.m.

MVC är inget fristående designmöster utan bygger på flera andra mönster. Till exempel kan flera vyer lyssna på samma händelse och det ger möjligt att lägga till nya vyer dynamiskt utan ändringar i motorn. Förfarandet kan även beskrivas mer generellt genom att en förändring i ett objekt kan påverka flera andra objekt, utan att de förändrade objektet känner till detaljer om de lyssnande objekten (detta är även känt som designmönstret observer, se referens [19] för mer information). Ett annat exempel är vyerna som alla har en instans av ett kontrollobjekt som hanterar inmatningen från användaren. Vyn skickar användarkommandon till kontrollen som då översätta användarkommandon mot anrop till motorn och uppdateringar i vyn. Detta gör det möjligt att byta vykontroller för att få ett nytt beteende på vyn (detta är även känt som design mönstret strategy, se referens [20] för mer information). [18][19][20]

Nu när vi vet vad MFC är så utgår vi från de tre huvudsakliga krav som Icomera ställde på användargränssnittet och ser om MFC är lämpligt att använda.

1. Det ska gå att ha flera olika vyer av användargränssnittet.
   Detta är möjligt då vyer är helt fristående i MVC. Det kan finnas flera olika vyer som använder sig av samma kontroller och modell. Lika så är det möjligt att bara behålla modell och låta varje enskilt användargränssnitt både ha sina egen kontroller och vyer.

2. Det ska vara lätt att lägga till funktionalitet i simuleringsmotorn utan att påverka användargränssnittet.
   Detta är möjligt då modellen, likt vyn och kontrollen, är fristående i MVC. Skulle någon lägga till något i modellen så behöver bara de vyer där den nya informationen ska var synlig uppdateras (så länge de gamla funktionerna och gränssnitten inte påverkas).

3. Det ska gå att kommunicera händelsebaserat mellan motorn och användargränssnittet för att minska belastningen på nätverket.

MVC bygger på att vyn och modellen kommunicerar via händelse och det sker inga onödiga anrop i MVC som belastar nätverket i onödan.

Som vi ser så har MVC en lösning på alla de tre krav Icomera ställde på användargränssnittet och det är därför vi valde MVC som grund för användargränssnittsdesignen.

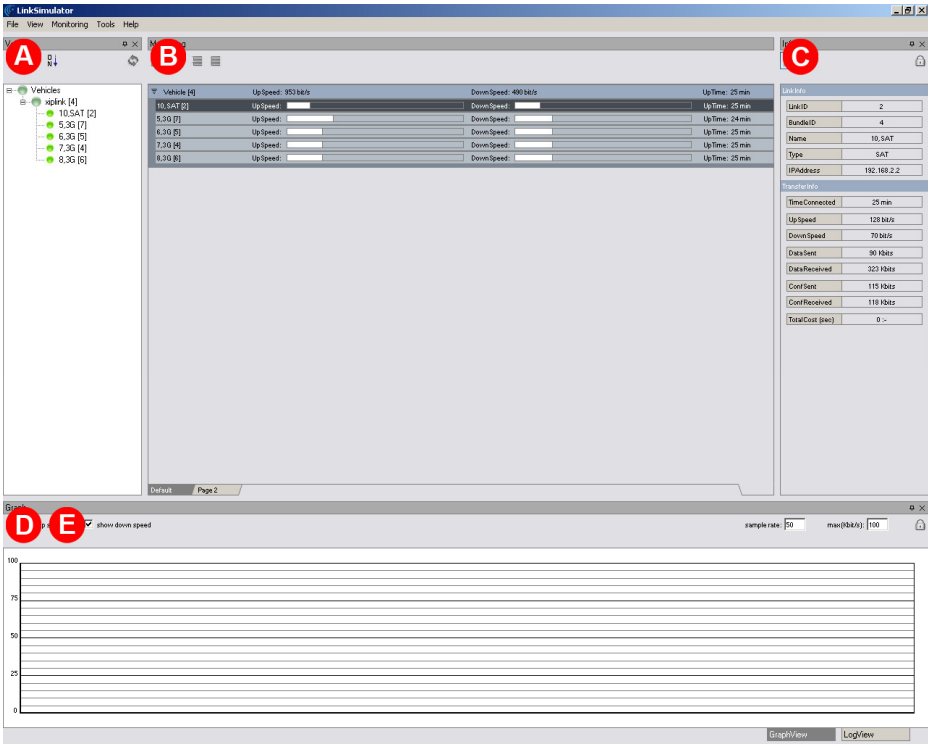
6.3 .NET Remoting

Kraven från Icomera var att utveckla ett användarvänligt grafiskt gränssnitt. Vi skulle även designa så att simulatorn skulle vara enkel att underhålla och uppdatera i framtiden. Med de kraven som grund valde vi att separera användargränssnittet från motorn. För att koppla samman dessa två moduler användes .NET-remoting som är ett sätt att koppla samman applikationer i ett distribuerat system.

.NET Remoting gör det möjligt att göra Remote Process Call (RPC) mellan två .NET-program. RPC är ett sätt att anropa funktioner på andra datorer så länge de är sammankopplade via ett nätverk. RPC är inte unikt för .NET och .NET Remoting är bara en av flera implementationer av RPC. [21]

Anledningen till att vi använde oss av .NET Remoting istället för att skicka egendefinierade paket är att .NET Remoting gör det möjligt att abstrahera objekt så att utvecklarna inte behöver ta hänsyn till om objektet är lokala eller ligger i de distribuerade delarna av systemet. .NET remoting fungerar genom att alla objekt som är distribuerade skapar ett proxy-objekt Detta objekt hämtas sedan av den lokala applikationen som använder proxy-objektet för att kommunicera med det distribuerade objektet via nätverk. Proxy-objekten gör att utvecklaren kan kalla på funktioner som på vilket lokalt objekt som helt och kommunikation som sker över nätverket abstraheras bort. [22][23]

6.4 Beskrivning av användargränssnittet



Figur 6-2: Skärmdump av användargränssnittet.

Det grafiska gränssnittet är uppdelat i fem delar A, B, C, D och E, som kan ses i Figur 6-2. För att enklare kunna förklara hur gränssnittet fungerar ges först en övergripande förklaring av de fem delarna och hur de är sammankopplade. Efter den övergripliga beskrivningen av delarna, ges en mer detaljerad beskrivning av varje del i avsnitten 6.5-6.10.

- **VehicleView (A)** – Denna vy är utgångspunken i användargränssnittet. Den innehåller en hierarkis trädstruktur över anslutna IMSR:er och dess länkar. Från denna vy kan användaren ta tag i en IMSR eller en länk och dra den till MonitoringView.

- **MonitoringView (B)** – I denna vy kan användaren övervaka länkar och IMSR:er. Användaren ser länkarnas överföringshastighet relativt deras maxöverföringshastighet samt den totala IMSR-överföringshastigheten. Om utvecklaren markerar en länk kommer detaljerad information om just denna länk synas i InfoView:n

- **InfoView (C)** – I denna vy syns detaljerade information om den markerade länken/IMSR:en i MonitoringView. Det är även möjligt att konfigurera länkens karakteristik.

- **GraphView (D)** – I denna vy visas en graf över upp- och nedhastighet för de länkarna och IMSR:erna som är markerade i MonitoringView. Grafen kan användas till att se hur hastigheten varierar över ett tidsintervall.

- **LogView (E)** – I LogView syn alla loggutskrifter som simulatormotorn genererar. Detta är till för att kunna ge användaren en snabb feedback om något skulle gå fel. Den skriver även ut information som skapas av skript, något som ger utvecklarna möjlighet att följa och se vart någonstans i skriptet simulatorn befinner sig.

Varje panel förutom MonitoringView har följande två knappar upp i sitt högra hörn:

 Denna knapp dockar panelen mot ena sidan av applikationen. Vill användaren få tillbaka panelen kan han/hon antingen trycka på tabben som läggs till vid sidan av applikationen då panelen dockas mot den eller så kan användaren använda *View* i huvudmenyn, se 6.5.2.

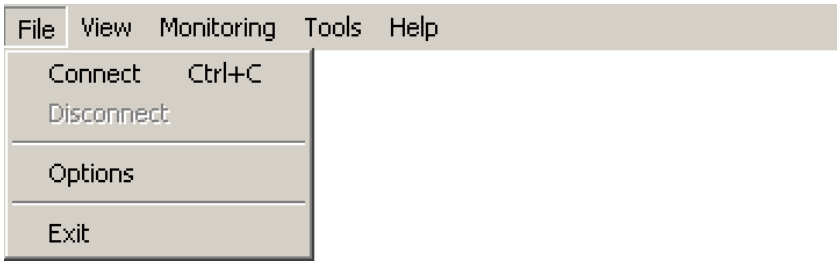
 Denna knapp gömmer panelen helt och vill användaren få tillbaka panelen kan han/hon gå till *View* i huvudmenyn, se 6.5.2.

6.5 Huvudmeny

Huvudmenyn består av fem stycken underkategorier. Dessa underkategorier beskrivs i de följande underavsnitten. Innehållet i parenteser

som står efter vissa av kommandona är snabbkommandon. Ett snabbkommando är en tangentbordskombination som gör det möjligt att komma åt kommandot utan att gå in i huvudmenyn.

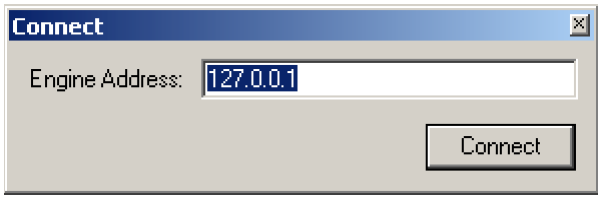
6.5.1 File



Connect (Ctrl+C)

Connect används far att ansluta användargränssnittet mot en specifik motor. När användaren väljer *connect* vissas en dialog (se Figur 6-3) där han/hon ombeds att specificera IP-adressen till motorn. I Icomeras fall ligger motorn på simulatordatorn i laboratoriet (Dator B Figur 2-7). Gränssnittet kan bara vara anslutet mot en motor åt gången. Vill användaren övervaka en annan simulering i en annan motor måste denna göra en *disconnect* (se 6.5.1.Disconnect) och sedan välja *connect* igen.

Motorn kan bara ha ett gränssitt anslutet åt gången om användaren ansluter till en motor där det redan är ett gränssitt anslutet visas en dialog där den nya användaren ombeds välja om han/hon vill slänga ut den redan aktiva användaren eller avbryta sitt anslutningsförsök.



Figur 6-3: Connect-dialogen.

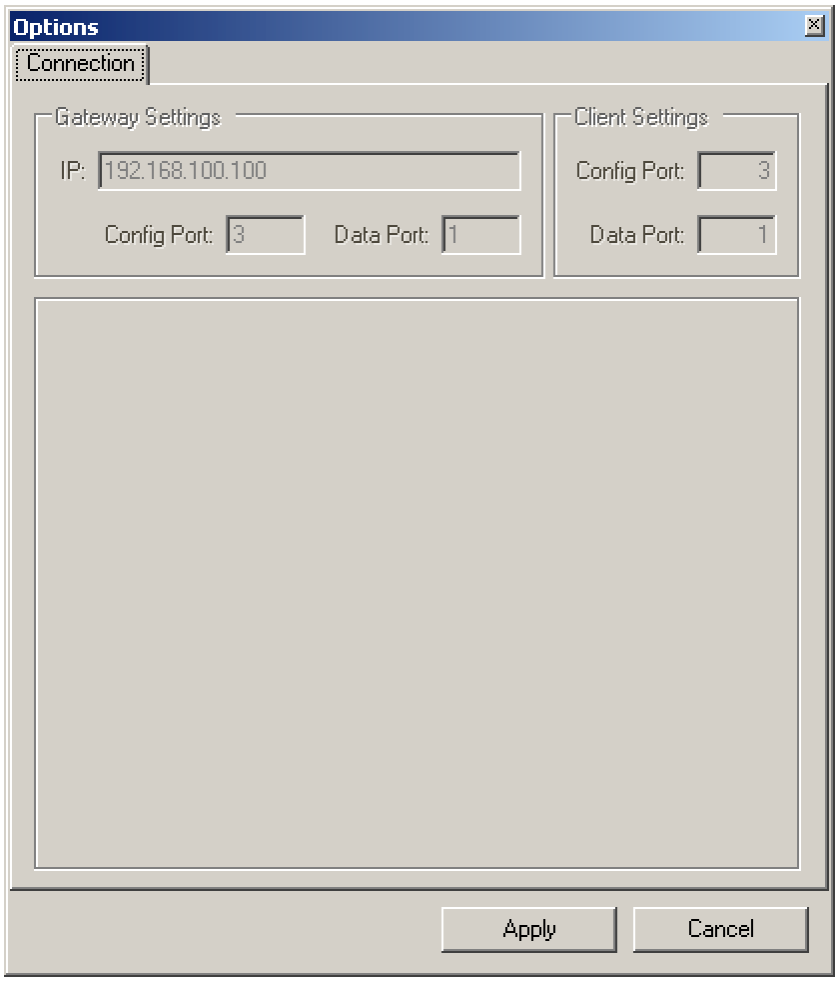
Disconnect

Disconnect användas för att stänga ned förbindelsen mellan gränssnittet och motor. En *disconnect* avbryter INTE den aktiva simuleringen i motorn,

utan kopplar bara ifrån det övervakande användargränssnittet. Det är praktiskt i och med att användaren slipper ha användargränssnittet uppe under hela simuleringen samt att han/hon kan överlåta övervakningen till en annan användare.

Options

Options tar fram en dialog (se Figur 6-4) där användaren kan göra inställningar för både användargränssnittet och motorn.



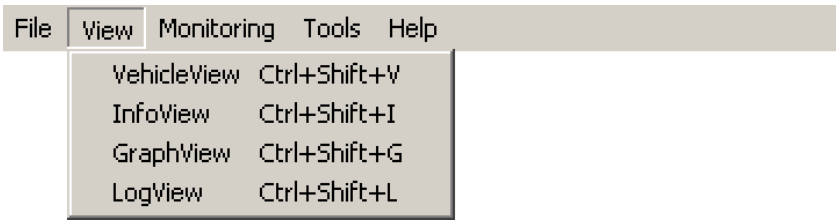
Figur 6-4: Options-dialogen.

Exit

Exit stänger ned användargränssnittet. Exit gör en indirekt *disconnect* för att stänga förbindelsen med motorn.

6.5.2 View

I View-menyn kan användaren säga vilka av de olika panelerna i gränssnittet (se Figur 6-2) som ska synas respektive gömmas. Genom att välja ett menyalternativ kan användaren gömma en panel som är synlig eller ta fram en panel som är gömd. Det finns följande menyalternativ:



VehicleView (Ctrl+ShiftV) - *VehicleView* växlar mellan att gömma och visa VehicleView-panelen (panel A i Figur 6-2).

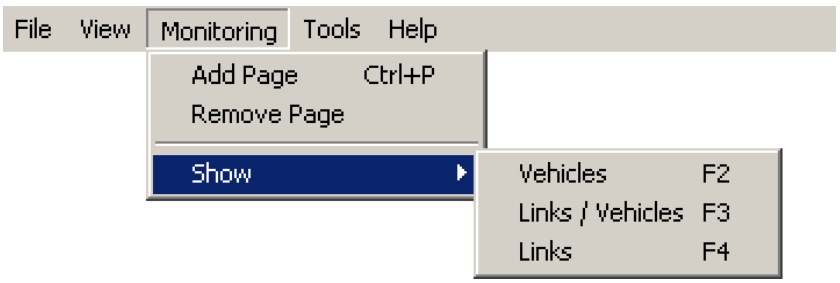
InfoView (Ctrl+Shift+I) - *InfoView* växlar mellan att gömma och visa InfoView-panelen (panel C i Figur 6-2).

GraphView (Ctrl+Shift+G) - *GraphView* växlar mellan att gömma och visa GraphView-panelen (panel D i Figur 6-2).

LogView (Ctrl+Shift+L) - *LogView* växlar mellan att gömma och visa LogView-panelen (panel E i Figur 6-2).

6.5.3 Monitoring

I Monitoring-menyn kan användaren konfigurera och ändra utseendet på MonitoringView (panel B i Figur 6-2). Det finns följande menyalternativ:



Add Page (Ctrl+P)

Add Page lägger till en ny tabb i MonitoringViews tabblista (se 6.7.2). När användaren väjer *Add Page* kommer en dialog fram (se Figur 6-5), där han/hon ombeds att skriva in ett namn på den nya tabben. Den nya tabben lägger sig längs till höger i MonitoringViews tabblista.

Figur 6-5: New Page-dialogen.

Remove Page

Remove Page tar bort den aktiva tabben i MonitoringViews tabblista (se 6.7.2). Om den sista tabben i tabblistan tas bort kommer en ny tom tabb med namnet ”Default” att skapas då det alltid måste finnas minst en tabb.

Show

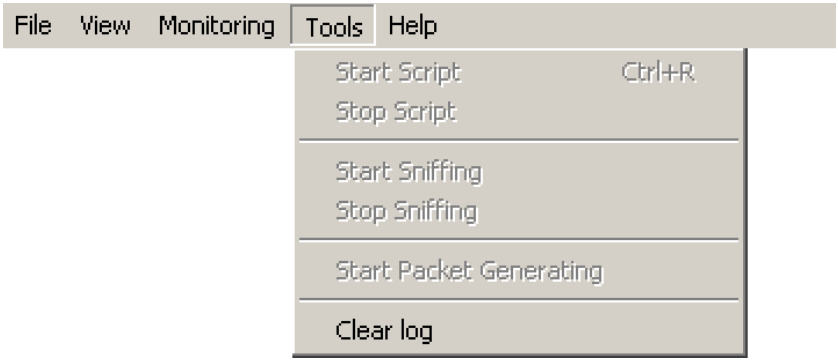
Show låter användaren välja hur IMSR:erna respektive länkarna ska visas i MonitoringView. Användaren kan välja på följande alternativ:

Vehicles (F2) – Visar IMSR-information medan länkinformationen göms undan.

Links / Vehicles (F3) – Visar både länk- och IMSR-information.

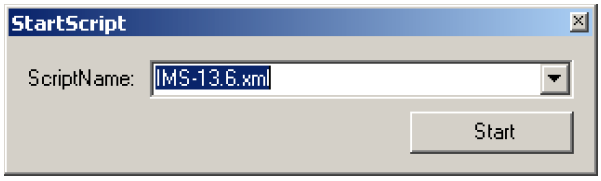
Links (F4) – Visar länkinformation medan IMSR-informationen göms undan.

6.5.4 Tools



Start Script (Ctrl+R)

Start Script startar ett automatiserat simuleringsskript. När användaren väljer *Start Script* visas en dialog där användaren kan välja ett skript ur en drop-down-meny, se Figur 6-6 (användaren kan lägga till nya skript genom att lägga till en ny XML-skriptfil i motorns skriptkatalog). Motorn kan bara ha ett script aktivt. För att byta skript måste användaren först stoppa det aktiva scriptet (se 6.5.4.StopScript) och sedan välja *Start Script* igen.



Figur 6-6: Start Script-dialogen.

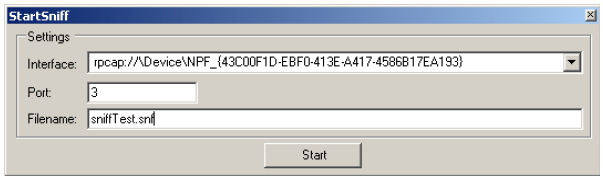
Stop Script

Stop Script stoppar det aktiva simuleringsskriptet.

Start Sniff

Start Sniff startar en inspelning av datatrafikflödet på en specifik IP-adress och port. När användaren väljer *Start Sniff* visas en dialog där han/hon ställer inställningar för sniffern, se Figur 6-7. Användaren börjar med att välja nätverkskort och port som ska användas. Sedan ställer användaren in namnet på filen till vilken trafikflödet sparas. För en förklaring om hur datatrafikflödet sparas, se avsnitt 5.6.

Motorn kan bara ha en aktiv datatrafiksinspelning. För att byta inställningarna för datatrafiksinspelningen måste användaren först stoppa det aktiva datatrafiksinspelningen (se 6.5.4.StopSniff) och sedan välja *Start Sniff* igen.



Figur 6-7: Start Sniffer-dialogen.

Stop Sniff

Stop Sniff stoppar det aktiva datatrafiksinspelningen.

Start Packet Generating

Start Packet Generating startar ett automatiserat trafikflöde genom motorn. När användaren väljer *Start Packet Generating* visas en dialog, se Figur 6-8. I dialogen kan användaren konfigurera paketgeneratorn. Först ställer användaren in adressen till IGW:n och sedan specificerar han/hon argumentlistan till paketgenereringsprogrammet. I den nuvarande versionen av simulatorn kan bara iperf användas som paketgenerator. Beskrivning av iperf ligger utanför rapporten och läsaren hänvisas till referensen [24] för ytterligare information.

Figur 6-8: Start Packe Generator-dialogen.

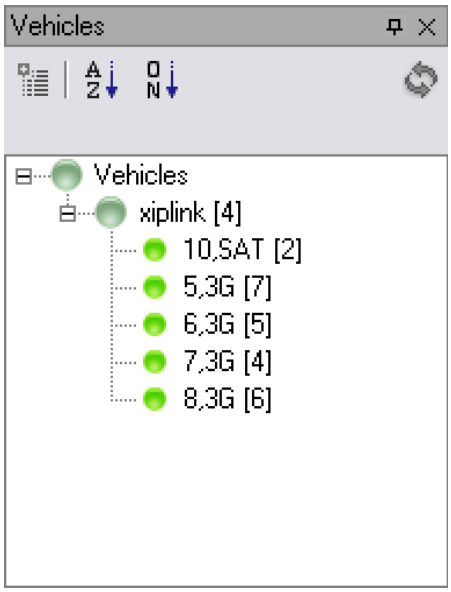
Clear Log

Clear Log tömmer logutskrifterna i LogView (se 6.10 för information om LogView).

6.5.5 Help

Hjälpmenyn består bara av två alternativ; *Help (F1)* och *About*. När användaren väljer *Help* visas hjälpmanualen. När användaren väljer *About* visas en dialog med information om gränssnitts- och motorversion.

6.6 VehicleView



Figur 6-9: VechileView

VehicleView är startpunkten i användargränssnittet. I VehicleView visas alla anslutna IMSR:er i form av noder i en trädvy. En nod representerar en specifik IMSR. Nodens namn består av IMSR:ens namn (konfigurerat av användaren när han/hon konfigurerar IMSR:en) samt två hårda parenteser mellan vilka IMSR:ens id står.

Varje IMSR-nod har en eller flera länknoder under sig. En länknod representerar en av IMSR:ens länkar. Länknodens namn består av länkens namn (konfigurerat av användaren när han/hon konfigurerar IMSR:en) samt två hårda parenteser mellan vilka länkens id står. Länkens id skapas slumpmässigt vid uppkopplingen mellan IMSR:en och IGW:n och samma länk kan byta id mellan anslutningstillfällen. Den enda regel som gäller för länk-ID:en är att de är unika inom en IMSR. Med andra ord kan det inte finnas två länkar, från samma IMSR, med samma id anslutna vid ett och samma tillfälle.

För en bild över hur IMSR-noder och länkar är sammankopplade se Figur 6-9. Som bilden visar är både IMSR-noden och länknoderna gröna. Det är för att visa att de för tillfället är anslutna och aktiva. När en IMSR eller en länk kopplar ned kommer deras respektive IMSR- eller länknod att bli blåa

för att indikera för användaren att IMSR:en eller länken har kopplat ned. För att undvika att se IMSR:er som inte är anslutna kan användaren högerklicka på den frånkopplade IMSR- eller länknoden och välja *delete,* för att ta bort den ur gränssnittet.

I VehicleView kan användaren bara studera vilka IMSR:er och länkar som är anslutna. För att kunna studera flödet av datatrafik måste användaren ta tag i de noder som han/hon vill studera och dra dem över till MonitoringView (för en genomgång av MonitoringView fungerar, se avsnitt 6.7). Om användaren tar tag i en IMSR-nod och drar den till MonitoringView kommer alla IMSR:ens länkar att följa med. För att undvika detta kan användaren hålla ned *Ctrl* på tangentbordet samtidigt som han/hon drar IMSR-noden över till MonitoringView.

6.6.1 Knappar

I Figur 6-9 syns det fyra knappar överst i panelen. Dessa knappar underlättar för användaren att navigera i VehicleView och här under följer en kort förklaring av deras funktioner:

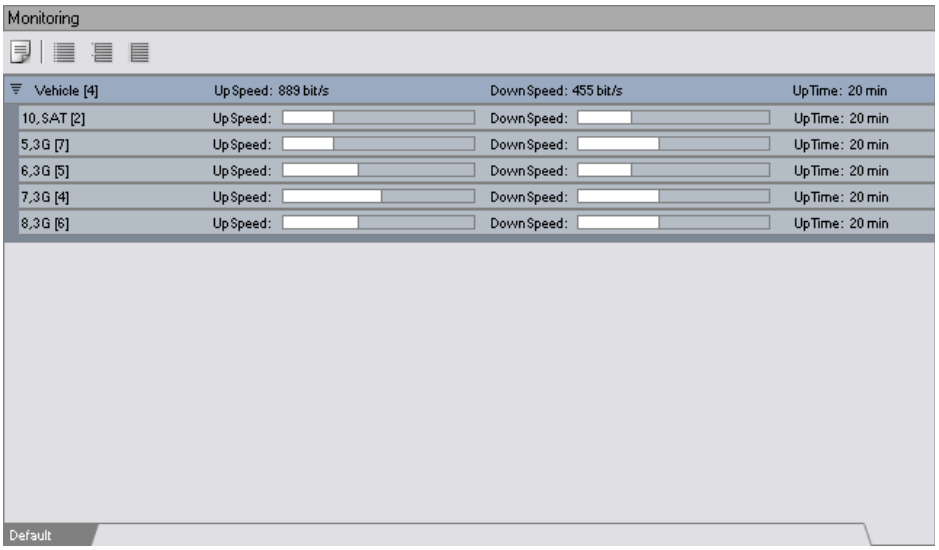
 Den här knappen synkar om alla IMSR:er och dess länkar. Detta användas eftersom gränssnittet i vissa fall kan hamna i osynk med motorn och användaren kan då göra en manuell uppdatering genom att trycka på denna knapp.

 Denna knapp sorterar alla IMSR:er efter hur många länkar som de har anslutna. De IMSR:er som har flest anslutna länkar hamnar överst och sedan följer resten av IMSR:erna i en avtagande ordning.

 Denna knapp sorterar alla IMSR:er i bokstavsordning efter deras namn. De IMSR:erna som börjar på A hamnar övers och sedan följer resten av IMSR:erna i en alfabetisk avtagande ordning.

 Denna knapp expanderar/kollapsar alla noder i VehicleView. Om användaren trycker på knappen när toppnoden är expanderad kommer den och alla dess barnnoder att kollapsa och om toppnoden är kollapsad kommer alla barnnoder att expandera. Detta är för att användaren ska kunna gömma/visa alla IMSR:ernas länkar med en knapptryckning istället för att behöva trycka på varje enskild IMSR-nod.

6.7 MonitoringView



Figur 6-10: MonitoringView

I MonitoringView kan användaren studera datatrafiken genom länkar och IMSR:er. För att lägga till en IMSR eller länk för övervakning tar användaren tag i en IMSR-nod eller länknod i VehicleView och drar och släpper den över MonitoringView. Hur IMSR:er och länkar representeras beskrivs nedan.

Figur 6-11: IMSR-övervakningsinformation.

En IMSR representeras enligt Figur 6-11. Längst ut till vänster i figuren står IMSR:ens namn följt av två hårda parenteser mellan vilka IMSR:ens ID står. Efter det följer två kolumner *UpSpeed* och *DownSpeed* i vilka den sammanlagda upp- samt nedhastigheten skrivs ut. Upp- samt nedhastigheten för en IMSR räknas ut som summan av dess länkars upp- samt nedhastighet. Längst ut till höger skrivs IMSR:ens upptid ut. Upptiden är hur länge en IMSR varit ansluten utan att tappa uppkopplingen till IGW:n. Med andra ord hur länge IMSR:en haft minst en länk ansluten.

Figur 6-12: Länkövervakningsinformation.

En länk representeras enligt Figur 6-12. Längst ut till vänster i figuren står länkens namn följt av två hårda parenteser mellan vilka länkens ID står. Längst ut till höger skrivs länkens upptid ut. Upptiden är hur länge en länk har varit ansluten till IGW:n. Mittemellan länknamnet och upptiden ligger två kolumner *UpSpeed* och *DownSpeed*, men till skillnad mot IMSR-representationen så skrivs inte upp- och nedhastigheten ut i klartext, utan representeras grafiskt. *UpSpeed* och *DownSpeed* representerar likt en progressbar som används för att visar hur mycket som är kvar att göra på en viss operation. Skillnaden på progressbaren som används för *UpSpeed* och *DownSpeed* är att den visar hur mycket av den teoretiska maxhastigheten som faktiskt används. Som exempel tar vi en 3G-länk, som har en teoretisk maxhastighet 384 kbit/s, där vi uppmäter en faktisk hastighet på 190 kbit/s. I det fallet kommer progressbaren vara fylld till hälften för att indikera att vi bara använder 50 % av den tillgängliga hastigheten.

Längst ned i MonitoringView finns en tabblista (tabblistan beskrivs i avsnitt 6.7.2) där användaren kan skapa olika tabbar för att gruppera in länkar och IMSR:er för att lättare kunna övervaka dem.

I MonitoringView kan användaren bara övervaka länkarnas hastighet relativt deras maxhastighet samt att användaren kan se den totala IMSR-hastigheten. För att få fram mer information kan användaren markera IMSR:er och länkar, vilket resulterar i att de markerade IMSR:erna och länkarna visas i GraphView, se 6.9. Användaren kan även välja att markera en enskild länk eller IMSR för att visa detaljerad information om den. Om en länk markeras kan användaren även konfigurera dess karakteristik.

6.7.1 Knappar

 Den här knappen lägger till en ny tabb i MonitoringViews tabblista (se 6.7.2). När användaren trycker på knappen sker samma förfarande som om han/hon väljer *Monitoring/Add Page* i huvudmenyn (se 6.5.3 för mer information om att lägga till tabbar).

Dessa tre efterföljande knappar låter användaren välja hur IMSR- och länkinformationen ska visas i MonitoringView:

 Visar IMSR-information medan länkinformationen göms undan.

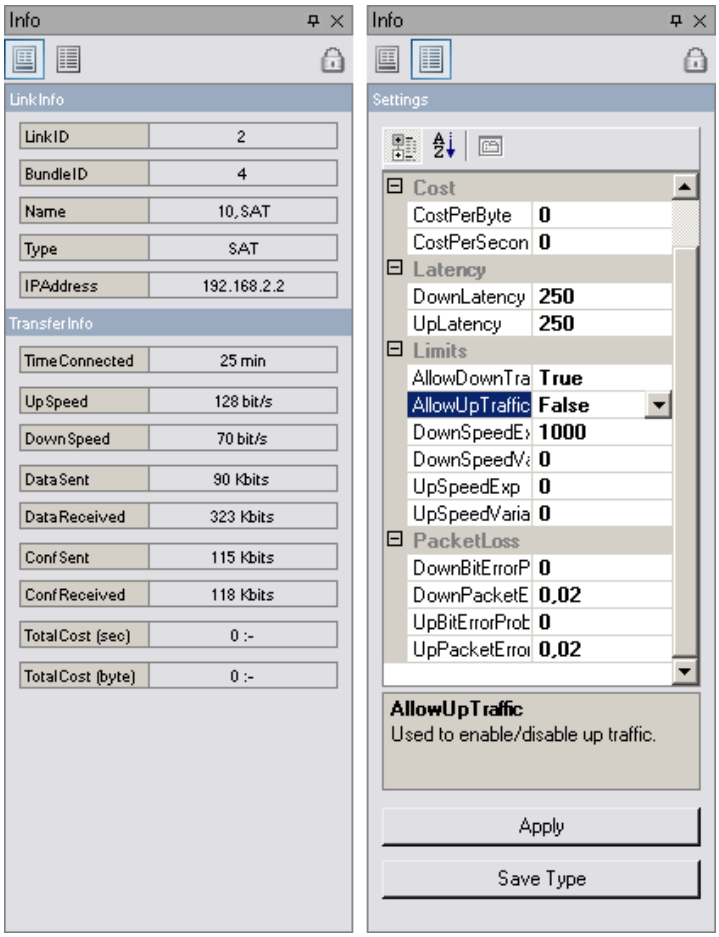
 Visar både länk- och IMSR-information.

 Visar länkinformation medan IMSR-information göms undan.

6.7.2 Tabblista

Tabblistan gör det möjligt för användaren att gruppera in länkar och IMSR:er i olika tabbar. Tabblistan är placerat längst ned på MonitoringView, se Figur 6-10. Antalet tabbar är obegränsat och om antalet tabbar blir fler än vad som får plats på skärmen kommer en rullningslist upp längst ut till höger i tabblistan. Rullningslisten göra det möjligt att skrolla fram de tabbar som inte får plats på skärmen.

6.8 InfoView



Figur 6-13: LinkView

InfoView består av två delar, se Figur 6-13. En del där man kan ser detaljerad information om en IMSR eller länk och en del där man kan konfigurera länkkarakteristik. För att navigera mellan de två delarna används knapparna längst upp till vänster i InfoView.


Informationsdelen visar detaljerad information om IMSR:er och länkar. När användaren markerar en IMSR i MonitoringView visas informationen i Tabell 6-3.

Namn	Beskrivning
IMSR Info	
Boundle Id	ID:et på IMSR:en. Skapas automatisk när en IMSR ansluter till IGW:n.
Name	Namnet på IMSR:en (konfigureras på IMSR:en)
Transfer Info	
Time Connected	Antalet minuter IMSR:en har varit ansluten till IGW:n
Up Speed	Totala hastigheten i bit/s som IMSR:en skickar data till IGW:n
Down Speed	Totala hastigheten i bit/s som IMSR:en tar emot data från IGW:n
Data Sent	Totala antalet bytes som har skickat från IMSR:en till IGW:n
Data Received	Totala antalet bytes som har tagits emot av IMSR:en från IGW:n
Conf Sent	Den, av användaren, konfigurerade maximala sändhasigheten för länken (IMSR till IGW).
Conf Received	Den, av användaren, konfigurerade maximala mottagarhasigheten för länken (IGW till IMSR).
Total Cost (Sec)	Kostnaden i kronor för tiden som IMSR:en har används.
Total Cost (Byte)	Kostnaden i kronor för data som har skickats på IMSR:en.

Tabell 6-3: Informationen som visas när en IMSR är markerat.

När användaren markerar en länk i MonitoringView visas informationen i Tabell 6-4.

Namn	Beskrivning
Link Info	
Link Id	ID:et som användaren har konfigurerat på IMSR:en.
Boundle Id	ID:et för IMSR:en som länken tillhör.
Type	Namnet på länktypen t.ex. 3G, SAT.
Name	Namnet på länken (konfiguerat på IMSR:en)
IP-Address	Länkens IP-adress
Transfer Info	
Time Connected	Antalet minuter länken har varit ansluten till IGW:n
Up Speed	Hastigheten i bit/s som IMSR:n skickar data till IGW:n
Down Speed	Hastigheten i bit/s som IMSR:n tar emot data från IGW:n

Data Sent	Antalet bytes som har skickat från IMSR:n till IGW:n
Data Received	Antalet bytes som har tagits emot av IMSR:n från IGW:n
Conf Sent	Den, av användaren, konfigurerade maxsändhasigheten för länken (IMSR till IGW).
Conf Received	Den, av användaren, konfigurerade maxmottagarhasigheten för länken (IGW till IMSR).
Total Cost (tid)	Kostnaden i kronor för tiden som länken har används.
Total Cost (data)	Kostnaden i kronor för data som har skickats på länken.

Tabell 6-4: Informationen som visas när en länk är markerat.

InfoView tillåter användaren att studera detaljerad information om IMSR:er och länkar och hur de förändras under en simulering. Anledningen till att denna information är placerad i en egen panel, och inte visas tillsammans med upp- och nedhastighet i MonitoringView, är för att det ansågs för rörigt. Icomera ville att MonitoringView bara skulle ha den allra nödvändigaste informationen för att göra den lättöverskådlig.

Konfigurationsdelen av InfoView gör det möjligt att ställa in karakteristik på den markerade länken. Konfigurationsdelen är bara tillgänglig då användaren markerar en länk. I Tabell 6-5 beskrivs de egenskaper som kan ställas in för länkkarakteristiken:

	Namn	**Beskrivning**
Predefined		
	Type	Den fördefinierade länktypen. Om användaren inte vill använda en fördefinierad länk typ lämnas detta fält blankt.
Cost		
	CostPerByte	Kostnaden i kronor för att skicka en byte.
	CostPerSecond	Kostnaden i kronor för att vara ansluten i en sekund.
Latency		
	DownLatency	Mottagningsfördröjningen. Den tid i ms som simulatorn fördröjer paketen innan de skickas vidare till IMSR:n.
	UpLatency	Skickningsfördröjningen. Den tid i ms som simulatorn fördröjer paket innan de skickas vidare till IGW:n.
Limits		

	AllowDownTraffic	Bestämmer om data kan tas emot via länken.
	AllowUpTraffic	Bestämmer om data kan skicka via länken.
	DownSpeedExp	Den förväntade mottagarhastigheten i bytes/s.
	DownSpeedVariance	Variansen i bytes/s på den förväntade mottagarhastigheten.
	UpSpeedExp	Den förväntade skickningshastigheten i bytes/s.
	UpSpeedVariance	Variansen i bytes/s på den förväntade skickningshastigheten.
Packet Loss		
	DownBitErrorProb	Mottagnings-bitfels-sannolikheten. Om användaren specificerar 0.01 innebär det att 1 av 100 bitar kommer att tas bort av simulatorn.
	DownPacketErrorProb	Mottagnings-paketfels-sannolikheten. Om användaren specificerar 0.01 innebär det att 1 av 100 paket kommer att tas bort av simulatorn.
	UpBitErrorProb	Skicknings-bitfels-sannolikheten. Om användaren specificerar 0.01 innebär det att 1 av 100 bitar kommer att tas bort av simulatorn.
	UpPacketErrorProb	Skicknings-paketfels-sannolikheten. Om användaren specificerar 0.01 innebär det att 1 av 100 paket kommer att tas bort av simulatorn.

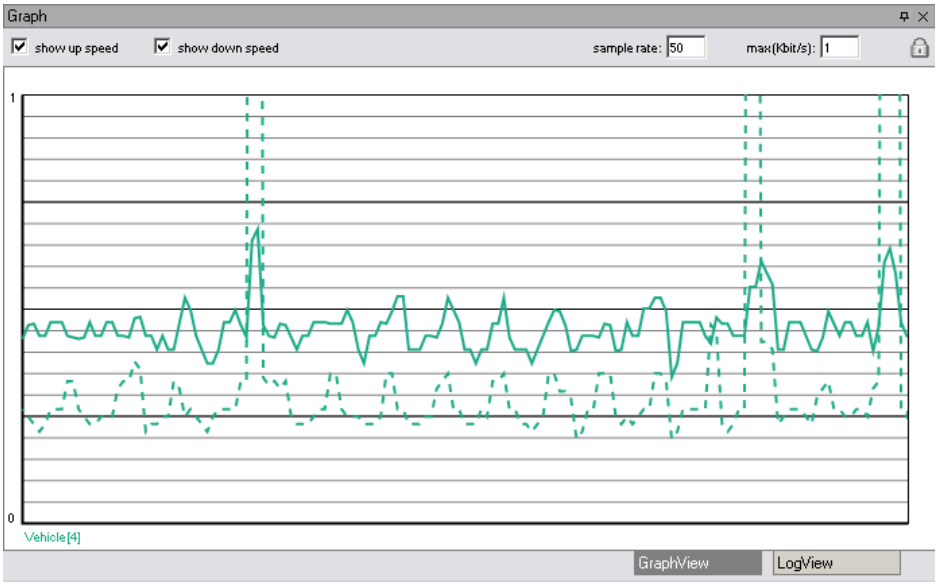
Tabell 6-5: Tillgängliga inställningar när en länk är markerat.

När användaren har ställt in karakteristiken trycker han/hon på *Apply* för att aktivera de nya inställningarna. Om användaren vill återanvända inställningarna kan hon/han trycka på *Save Type* för skapa en ny fördefinierad länktyp. När användaren trycker på *Save Type* visas en dialog där han/hon ombeds att skriva in namnet på den nya länktypen (länktypens namn behöver inte vara unikt, dock är det svårt att skilja på länktyper med samma namn).

6.8.1 Knappar


Denna knapp låser den aktiva länken/IMSR:en. Det innebär att även om användaren markerar en ny IMSR/länk i MonitoringView så kommer den gamla IMSR:en/länken att visas i InfoView. Detta gör det möjligt för användaren att studera en IMSR/länk utan att hela tiden ha den markerat i MonitoringView.

6.9 GraphView



Figur 6-14: GraphView

GraphView (se Figur 6-14) är till för att användaren ska kunna studera IMSR:er och länkars bandbreddsförändringar under längre tidsperiod (inte som i MonitoringView där användaren bara ser realtidsstatistik). För att studera IMSR:er/länkar markerar användaren de önskade IMSR:erna och länkarna i MonitoringView. De markerade IMSR:erna/länkarnas namn skrivs ut med olika färger längst ned i GraphView. Olika färger på grafen gör att det är möjligt för användaren att skilja de olika IMSR:erna/länkarna åt.

För varje IMSR/länk ritas två graflinjer ut, den ena för upphastigheten och den andra för nedhastigheten. Upphastigheten ritas ut som en heldragen linje och nedhastigheten ritas ut som en streckad linje. Graflinjerna ritas även olika tjocka beroende på om de representerar en IMSR (tjock linje) eller länk (tunn linje). Om användaren bara är intresserad av upp- eller nedhastigheten kan han/hon välja bort den andra genom att kryssa ur *show up speed* eller *show down speed*.

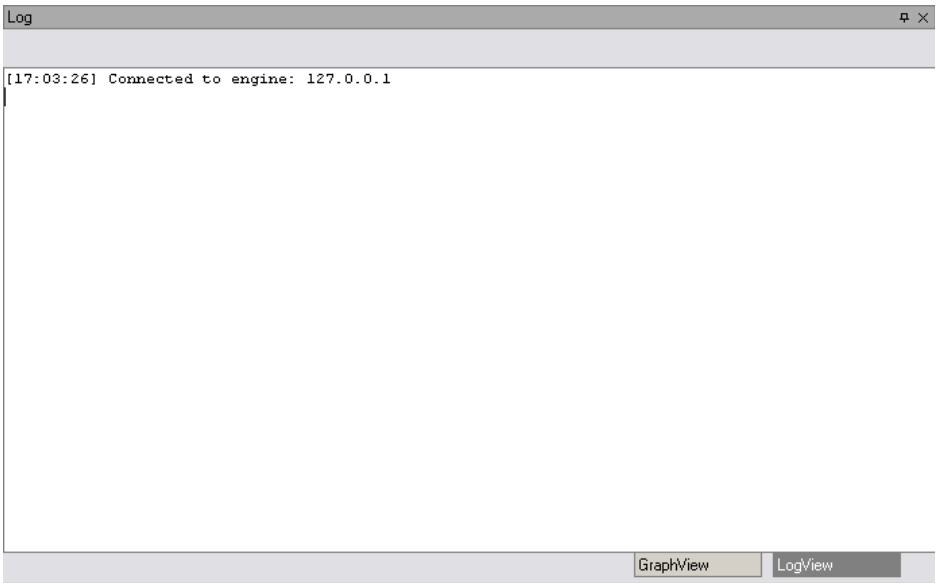
Grafen ritas ut med tiden på x-axeln och hastigheten på y-axeln. Genom att ändra *sampel rate*-värdet kan användaren styra hur stort tidsintervall grafen visar på x-axeln. *Sampel rate* värdet speglar hur många mätvärden som plottas i grafen. Simulatormotorn sparar mätvärden med ett kontinuerlig

intervall på 1 sekund. En *sampel rate* på 50 innebär då att grafen kommer visa ett intervall på 50 sekunder. Användaren kan även styra intervallet på y-axeln. Y-axeln utgår alltid från noll men maxvärdet kan ändras genom att justera *Max(kbit/s)* värdet.

6.9.1 Knappar

 Denna knapp låser de aktiva länkarna/IMSR:erna. Det innebär att även om användaren markerar en ny IMSR/länk i MonitoringView så kommer de gamla IMRS:erna/länkarna fortfarande visas i GraphView. Detta gör det möjligt för användaren att studera en IMSR/länk utan att hela tiden ha det markrat i MonitoringView.

6.10 LogView



Figur 6-15: LogView

LogView (se Figur 6-15) är till för att användaren ska kunna studera och övervaka de händelser som sker i motorn. När något sker i motorn, såsom att en länk byter karakteristik eller att en IMSR eller länk kopplar upp sig, skrivs det ut som ett loggmeddelande i LogView. Loggmeddelande skrivs ut med en tidstämpel följt av ett meddelande som talar om vad som har skett. Nya meddelanden läggs till överst i LogView, så att användaren slipper att skrolla igenom alla meddelanden för att nå de senaste. Om användaren vill ta bort alla gamla meddelanden kan han/hon göra det genom huvudmenyn och *Tools/Clear Logs*, se avsnitt 6.5.4.

7 Användarfall

7.1 Introduktion

I det här kapitlet beskrivs två stycken användarfall. Tanken är att visa hur simulatorn är tänkt att användas. Det första användarfallet beskriver hur användaren manuellt använder simulatorn för att simulera en 3G-länks karakteristik, samt generera ett paketflöde. Det andra användarfallet beskriver hur samma förfarande kan göras med hjälp av ett skript.

Användarfall 1 består av följande steg:

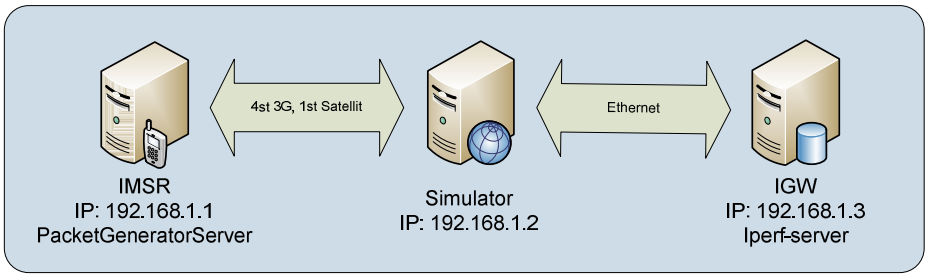
1. Starta paketgenerering utan bandbreddsbegränsning.
2. Begränsa bandbredden.
3. Starta paketgenerering igen.
4. Studera grafen för datatrafiken.

Användarfall 2 består av följande steg:

1. Skriv skript.
2. Starta skript.
3. Studera grafen över datatrafiken.

7.2 Konfigurering

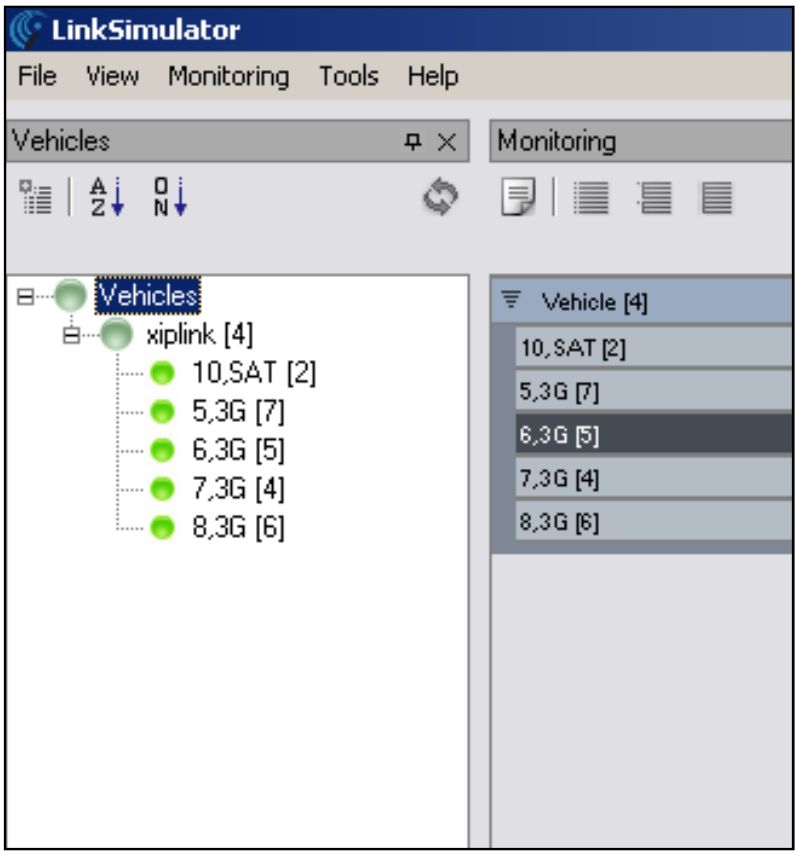
Användargränssnittet är anslutet till simulatorn, som har fem länkar uppkopplade, en satellitlänk och fyra 3G-länkar enligt Figur 7-1.



Figur 7-1: Användarfall 1 & 2 länkkonfiguration,

För att skapa datatrafik används en paketgenereringsserver och Iperf-server. Paketgenereringsservern ligger på IMSR-datorn och Iperf-servern ligger på IGW-datorn. För att underlätta för utvecklarna kan paketgeneratorn fjärrstyras från simulatorns användargränssnitt. Trafikflödet som skapas mellan IMSR:en och IGW:n är likvärdig med att en användare surfar på tåget.

När det grafiska gränssnittet ansluts kommer användaren se den anslutna IMSR:en, samt dess fem länkar, se Figur 7-2. Detta tillstånd är utgångspunkten för de två efterföljande användarfallen.

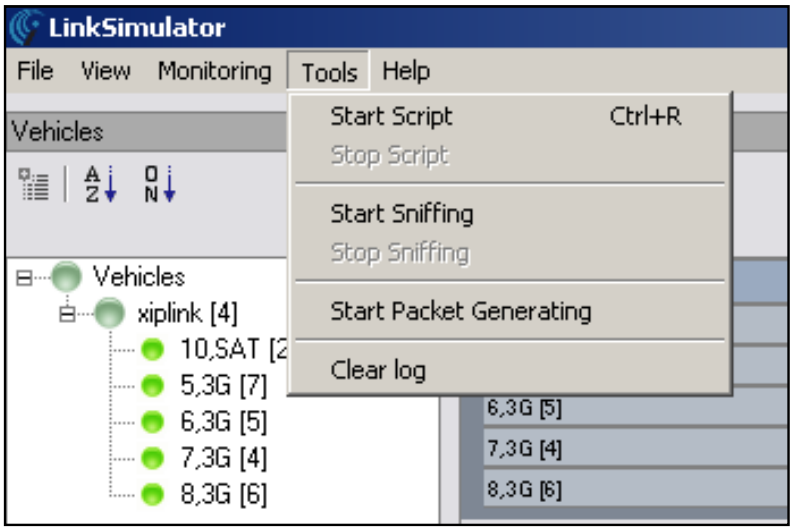


Figur 7-2: Fem länkar ansluta i användargränssnittet.

7.3 Användarfall 1 – manuell manövrering

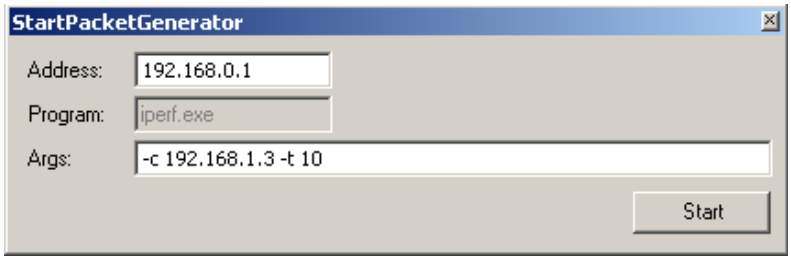
För att mäta hur fort data kan transporteras genom simulatorn startar vi en paketgenerering utan några bandbreddsbegränsningar. Latens och paketförluster är nollställda så de inte ska ha någon inverkan.

Paketgenereringen startas genom att användaren klickar på ”Tools->Start Packet Generating” i huvudmenyn, se Figur 7-3.



Figur 7-3: Starta paketgenerering

Innan paketgenereringen startas kommer en ruta upp där inställningar kan göras, Figur 7-4.



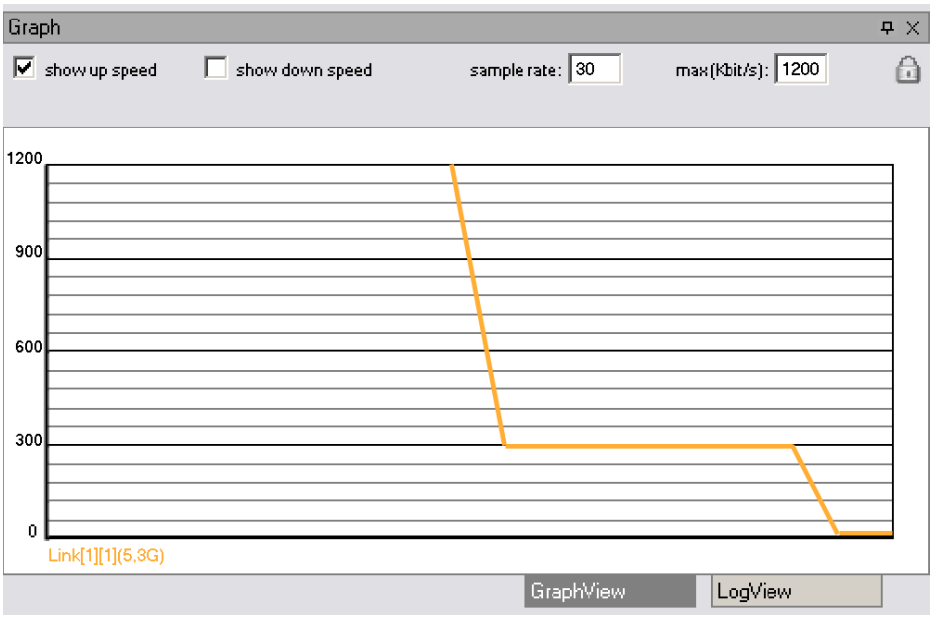
Figur 7-4: Inställningar för paketgenerering.

”Address” är adressen till paketgenereringsservern. Det är en modul som vi själv skrivit. Vad paketgenereringsservern gör är att den startar upp en process med angivna parametrar. Vi har låst den till att alltid starta Iperf som är ett verktyg för att mäta eller generera datatrafik genom TCP och UDP. Under ”Args” ska parametrarna för Iperf anges. ”-c ” betyder att Iperf ska verka som en klient. Sedan kommer IP-adressen till Iperfservern. I vårt fall ligger Iperfservern på samma dator som IGW:n, alltså IP 192.168.1.3. Den sista parametern ”-t 10” betyder att Iperf ska skicka data i 10 sekunder

och sedan avsluta. Genom att klicka på ”Start”-knappen startas genereringen av paket i 10 sekunder. [24]

Nu ska vi prova att begränsa bandbredden och utföra en likadan paketgenerering. Bandbreddsinställningar hittas i InfoView (se 6.8) till höger i användargränssnittet. I det här användarfallet kommer data skickas ifrån IMSR:en till IGW:n. Det betyder att satellitlänken inte kommer att användas i och med att den endast kan ta emot data. Vi ställer in alla 3G-länkar till att ha en bandbredd på 300000 bitar per sekund, 300 kbit/s. Det är en rimlig hastighet på en 3G-länk i goda förhållanden. Vi sätter sedan på paketgenereringen igen.

Nu är två paketgenereringar gjorda. Den första paketgenereringen var utan bandbreddsbegränsningar och den andra paketgenereringen med en bandbreddsbegränsning på 300 kbit/s. De utdata som fås efter ett sådant här test är i form av en graf i användargränssnittet, se Figur 7-5. Det går också att titta på momentana siffror i användargränssnittet under tiden data flödar i länkarna.



Figur 7-5: Resulterande graf efter användarfall 1 och 2.

7.4 Användarfall 2 – skript

I det här användarfallet ska vi titta på hur det går att göra ett liknande test som i avsnitt 7.3, fast med hjälp av ett skript. I avsnitt 5.5 tas det upp hur ett skript byggs upp. Vad vi ska göra är att först ta bort bandbreddsbegränsningar. Det görs genom att en vi sätter en högre bandbredd än nätverket, som har en maxhasighet på 100 Mbit/s. Efter det skapas en paketgenerering i 10 sekunder. Sedan sätts en bandbreddsbegränsning till 300 kbit/s och en till paketgenerering görs. Vi antar i det här testet att latens och paketförluster är nollställda.

```
<?xml version="1.0" encoding="utf-8"?>
<EventList StartTime="0">
<Event Time="0" LinkTag="5,3G">
<Command CommandType="UpExpBandwidth" Value="200000000"></Command>
<Command CommandType="DownExpBandwidth" Value="200000000"></Command>
</Event>
<Event Time="0" LinkTag="6,3G">
<Command CommandType="UpExpBandwidth" Value="200000000"></Command>
<Command CommandType="DownExpBandwidth" Value="200000000"></Command>
</Event>
<Event Time="0" LinkTag="7,3G">
<Command CommandType="UpExpBandwidth" Value="200000000"></Command>
<Command CommandType="DownExpBandwidth" Value="200000000"></Command>
</Event>
<Event Time="0" LinkTag="8,3G">
<Command CommandType="UpExpBandwidth" Value="200000000"></Command>
<Command CommandType="DownExpBandwidth" Value="200000000"></Command>
</Event>

<Event Time="1" LinkTag="anything">
<Command CommandType="GeneratePackets" Value="192.168.1.1 iperf.exe –c
  192.168.1.3 –t 10"></Command>

<Event Time="20" LinkTag="5,3G">
<Command CommandType="UpExpBandwidth" Value="300000"></Command>
<Command CommandType="DownExpBandwidth" Value="300000"></Command>
</Event>
<Event Time="20" LinkTag="6,3G">
<Command CommandType="UpExpBandwidth" Value="300000"></Command>
<Command CommandType="DownExpBandwidth" Value="300000"></Command>
</Event>
```

```
<Event Time="20" LinkTag="7,3G">
<Command CommandType="UpExpBandwidth" Value="300000"></Command>
<Command CommandType="DownExpBandwidth" Value="300000"></Command>
</Event>
<Event Time="20" LinkTag="8,3G">
<Command CommandType="UpExpBandwidth" Value="300000"></Command>
<Command CommandType="DownExpBandwidth" Value="300000"></Command>
</Event>

<Event Time="21" LinkTag="anything">
<Command CommandType="GeneratePackets" Value="192.168.1.1 iperf.exe –c
  192.168.1.3 –t 10"></Command>

</Event>
</EventList>
```

Skriptet börjar med fyra händelser som initierar de 3G-länkarna till en bandbredd på 200 Mbit/s. Som det går att se i Figur 7-2 så har 3G-länkarna namnen ”5,3G”, ”6,3G”, ”7,3G” och ”8,3G”. Dessa namn gör att det går att identifiera länkarna i skripet med attributet ”LinkTag”. ”Time”-attributet är satt till 0 vilket betyder att detta kommer ske direkt skriptet startats. En sekund senare kommer en händelse om att en paketgenerering ska påbörjas. Efter tio sekunders paketgenerering väntar skriptet i ytterligare tio sekunder tills de att 3G-länkarna tilldelas en bandbreddsbegränsning på 300 kbit/s. Efter det kommer en händelse som skapar den sista paketgenereringen. De utdata som fås, är i form av en graf i användargränssnittet, samt momentana siffror i användargränssnittet under tiden data flödar i länkarna.

7.5 Sammanfattning

Två olika metoder har använts för att uppnå samma resultat. Den ena genom att användaren ställer in värden i simulatorn manuellt och den andra genom att använda skript. De har båda olika fördelar när det gäller testning. Men en manuell manövrering kan det vara lättare att se när ett fel uppstår. Används istället skript kan samma test använda om och om igen. Det kan till exempel vara bra vid verifiering.

8 Problem och lösningar

8.1 Introduktion

I detta kapitel redovisas de problem som uppkommit under projektets gång. Den stora anledningen till att dessa problem har uppstått beror på att vi inte tog oss tillräckligt med tid i designfasen för att utveckla en fullständig prototyp. Hade vi gjort det skulle många av dessa problem förmodligen upptäckts och kunna ha åtgärdats i ett tidigt stadium. Istället såg vi inte problemen förrän sent i implementationsfasen och temporära lösningar var tvungna att göras.

8.2 Precision

För att uppnå en bra simulering krävs hög precision. Med hög precision menar vi att simulatorn måste behandla paket inom en viss tid, så att inte paketen buntas ihop innan de skickas ut på länken. Det finns flera faktorer som begränsar precisionen. Det kan bland annat vara datorns snabbhet och operativsystemet, i vårt fall Windows XP. Vi kommer endast att kunna ha en noggrannhet på en millisekund. Det beror på att simulatorn lyssnar efter en tidshändelse. Vid en tidshändelse kontrolleras om något paket är fördröjt med en viss tid och ska skickas vidare. Anledningen till att noggrannheten inte kommer upp i mer än en millisekund i bästa fall, är att operativsystemet Windows XP inte har implementerat en mer högupplöst tidshändelsefunktion. Windows XP garanterar endast en noggrannhet mellan 10-15 millisekunder. Alternativet till att använda tidshändelser vore att kontinuerligt övervaka tiden (eng: busy wait). Möjligtvis kan noggrannheten bli bättre på själva skickningen, men den tråden kommer istället att strypa resurserna för resten av systemet.

Problemet kommer antagligen att lösa sig med tiden i och med att datorerna blir snabbare och nya versioner av .NET släpps. Det är också en av orsakerna till att vi använder .NET-implementationen av tidtagare, istället för en snabbare Win32-implementation.

Med millisekunders noggrannhet kommer simulatorn att hantera hastigheter upp mot 12 Mbit/s, utan att simulatorn påverkar resultatet. Hastigheten 12 Mbit/s är uträknad enligt:

Hastighet = Antal bitar/Tiden

Antal bitar i ett fullt datagram är 1500 bytes, multiplicerat med 8 bitar vilket är 12000 bitar. Data som skickas mellan IMSR:en och IGW:n ligger på 1500 bytes eller strax under för att utnyttja länken maximalt. [6]

Tiden i detta fall är 1 millisekund = 0,001 sekund vilket är den minsta tiden mellan paket som simulatorn kan skicka ut ett paket.

Antal bitar/Tiden = 12000/0,001 = 12 000 000 = 12 Mbit/s

Skulle paketen som skickas vara mindre än 1500 bytes minskas också noggrannheten, så 12 Mbit är den högsta hastighet som simulatorn klarar av om precisionen ska bevaras. Värsta fallet skulle vara om den minimala storleken på ett ethernetpaket skickas som är 46 bytes. Simulatorn klarar med 46 bytes stora paket att simulera en hastighet på max 368 Kbit/s. Simulatorn kommer dock att kunna skicka data snabbare än 12 Mbit/s men paketen kommer att skickas i buntar vilket ger ett oönskat beteende.

Att simulatorn endast har en noggrannhet på 1 millisekund spelar idag mindre roll. Den snabbaste länken som systemet ska behandla är satelliten som i Icomeras fall har en hastighet på ungefär 1 Mbit/s. [6]

8.3 Trådning

Det finns olika modeller för hur program ska designas och implementeras. Ett alternativ är att köra programmet som en tråd, vilket ger ett klart och sekventiellt beteende. Problem uppstår när trådning introduceras. Det handlar om problematiken med att synkronisera trådar vilket är nödvändigt om programmet ska arbeta mot samma data. En annan aspekt är hastighet. Studera Tabell 8-1, där ett antal funktionsanrop har gjorts i .NET. Metoderna som har testats är olika tillvägagångssätt för att synkronisera trådar, förutom den första raden som visar på hur lång tid ett tomt

funktionsanrop tar. Det är intressant att titta på vilken markant ökning det blir när en context switch mellan user mode och kernel mode sker. Att gå igenom hur funktionerna fungerar ligger utanför innehållet i den här rapporten, men slutsatsen är att använda så få trådar som möjligt. Minimering av trådsynkronisering ökar alltså prestandan. [25]

Testtyp	Tid för 200 miljoner operationer (sek)	Övergång till kernel mode
Tomt funktionsanrop	1.0	Nej
Thread.SpinWait(1)	2.4	Nej
Interlocket.Increment(Int32)	2.8	Nej
SwitchToThread()	72.0	Ja
Thread.Sleep()	115,2	Ja
WaitHandle.WaitOne(0,false)	328,2	Ja

Tabell 8-1: Hur hastigheten varierar för olika funktionsanrop

Ett program ska inte använda fler trådar än det finns processorer i datorn. När en dator har fler trådar kommer en schemaläggning ske, vilket resulterar i en prestandaförlust. På en vanlig Microsoft Windows XP-dator körs det runt 400 trådar samtidigt. Det kan bero på att bland annat att Microsoft Windows tillhandahåller API och kodexempel som främjar ineffektiv kod, vilket gör att programmeringstekniken förs vidare till nästa generations programmerare. [25]

I den miljö där simulatorn kommer att användas kommer det att ske ett flertal asynkrona anrop. Det skulle göra systemet mycket komplext om den bara skulle exekveras i en tråd, vilket skulle vara optimalt enligt slutsatsen av Tabell 8-1. Vi valde därför att simulatorn skulle använda sig av en tråd för varje socket, vilket är nödvändigt vid asynkron datamottagning. Det kommer även att vara en tråd som har hand om fördröjning av paket och sedermera skickar iväg paketet. Vi har alltså designat simulatorn med trådningens brister i åtanke och användningen av antalet trådar har försökt att hållas så låg som möjligt. [25]

8.4 WinSock

För att komma åt nätverkstrafik i simulatorn används Windows sockets. Ett tidigt beslut som vi tog var att så mycket som möjligt skulle skrivas i .NET, där det finns inbyggda socketklasser att använda sig av. De funktioner som vi använde oss av i .NET var BeginReceive och

EndReceive. Enligt dokumentationen av .NET på Microsoft Developing Network (MSDN) skapar BeginReceive en tråd för varje paketmottagning, vilket ger möjligheten att asynkront ta emot data. Det gör att implementeringen blir enkel och designen överskådlig.

När programmet skulle testas skarpt mellan IMSR och IGW visade sig brister i .NET:s socketimplementation. Testerna visade på en maxhastighet runt 12 Mbit/s när all ”onödig” kod tagits bort och det enda som simulatorn gjorde var att tunnla trafik (ingen fördröjning av paket). Anledningen till denna låga hastighet beror på att .NET lägger ett lager över WinSock. I det lagret görs flera tester, vilket belastar datorn. Ett naturligt steg var att testa nätverksprestanda för kod som är skriven i native C++ och rena WinSock-metoder. Det går inte att översätta C++ koden rakt av på grund av att .NETs BeginReceive inte finns i native C++. Trådar måste alltså skapas manuellt för att asynkront kunna ta emot data på två olika sockets. När mätning gjordes uppmättes hastigheter på ca 24Mbit/s. Det är en fördubbling mot det som mättes med .NETs inbyggda asynkrona sockets. Ändå var inte resultatet tillräckligt tillfredsställande. IGW:n klarar av hastigheter upp mot 80 Mbit/s, vilken är en hastighet som simulatorn måste närma sig för att kunna användas fullt ut. Vi har ingen förklaring till att WinSock med en native C++-implementation inte klarade högre prestanda på grund av att CPU användningen inte går upp till 100 % utan ligger på runt 70-80 %.

Ett test till gjordes där vi testade att skriva nästan identisk översatt kod i C# .NET där vi hanterade trådhanteringen och den asynkrona mottagningen. Hastigheten uppmättes till 17 Mbit/s. Det är en ökning på 40 % mot den inbyggda funktionen för asynkron mottagningen i .NET. Det är dock bara 70 % av vad native C++ koden klarade av. Tester gjorda på Icomera visar på att hastigheten på C# .NET skulle vara 80 % av native C++, vilket kan ge en verifikation till att endast 70 % uppnås av nätverksprestanda med näst intill identisk översatt kod.

Ett sätt att gå vidare med problemet är att använda sig av en annan implementation istället för sockets. Det finns alternativ som till exempel WinPcap som ofta används av nätverksavlyssnare, bland andra Ethereal. I Tabell 8-2 visas genomsnittlig mätdata gjorda med olika implementeringsmetoder och språk. Testerna är gjorda med programmet

TestTCP och Iperf som använder sig av en TCP-ström för att mäta bandbredd mellan en klient och en server. Mätningen är gjord i ett nätverk med en maxhastighet på 100 Mbit/s. Om IMSR:en kopplar sig direkt till IGW:n uppnås en hastighet på 65 Mbit/s, Det visar på att det är simulatorn som stryper trafiken. När simulatorn används borde alltså maxhastigheten vara 65 Mbit/s. [26]

Metod	Bandbredd
.NET med inbyggd trådhantering	12 Mbit/s
.NET med egen trådhantering	17 Mbit/s
Native C++ med egen trådhantering	24 Mbit/s

Tabell 8-2: Hastigheter med WinSock, implementerat med olika metoder

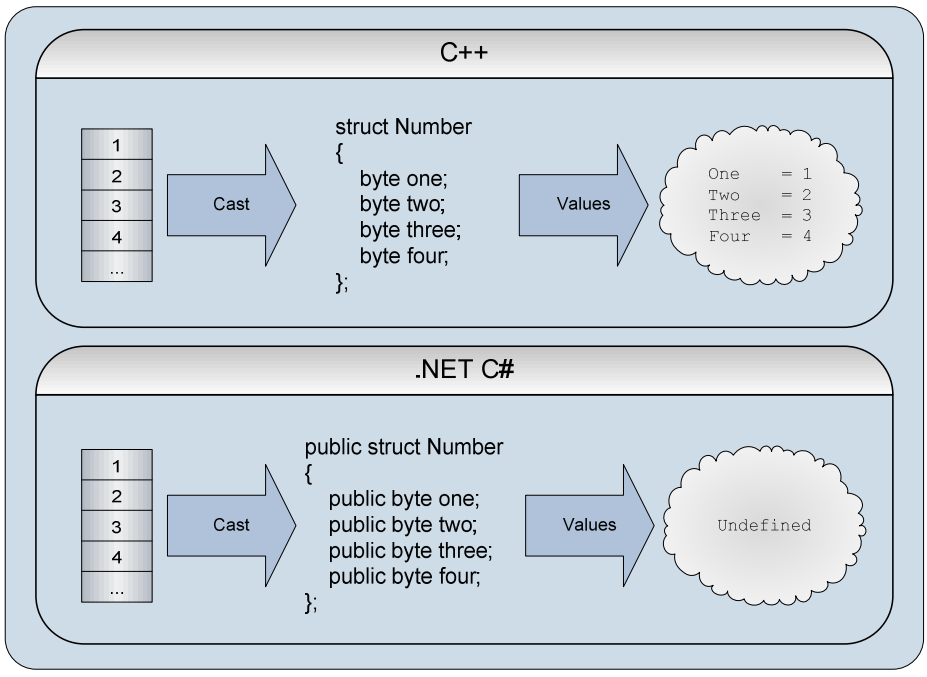
Ett beslut vi tog var att göra en klar separation av kommunikationslagret, så att det i framtiden går att byta ut den mot en snabbare lösning. Idag efterfrågar inte Icomera att komma upp i högra hastigheter än vad som görs på tågen där den snabbaste länken är satelliten med en maxhastighet på 1 Mbit/s. Sammanlagt med alla sju länkarna krävs det inte mer än ett par megabit per sekund för att uppnå de resultat som är önskade.

8.5 C++ till .NET-wrapping

Icomeras system använder sig av ett egendefinierat protokoll för att överföra data mellan IMSR och IGW. För att mjukvaran i IMSR:en och IGW:en ska kunna avkoda protokollet har Icomera skapat ett C++-bibliotek för att representera protokollstrukturerna. Kraven på den nya simulatorn att använda C# .NET som programspråk krävde att biblioteket konverteras till .NET.

Problemet med att konvertera strukturer från C++ till .NET är att dessa två språks använder olika minneshanteringsmodeller. .NET använder sig av en skräpsamlare (eng: garbage collector) och programmeraren behöver inte ta hand om allokering och avallokering av minne, ett ansvar som C++ har lagt på programmeraren. C++ har även stöd för pekararitmetik, något som har lett till att minnesutrymmen alltid allokeras sekventiellt. .NET å andra sidan har inget stöd för pekararitmetik och behöver inte allokera minne

sekventiellt. Problemet med konverteringen mellan dessa två språk illustreras i Figur 8-1, där en bytevektor ska typomvandlas till en struktur via en minneskopiering. De första fyra byten i vektorn har värdena 1 till 4 som efter typomvandlingen ska ha tilldelas variablerna *one*, *two*, *three* och *four* sekventiellt där *one* tilldelas värdet 1 och *two* tilldelas värdet 2 och så vidare. Vid en sådan typomvandling kommer resultatet att bli olika beroende på om det görs i C++ eller .NET. Resultatet i C++ blir att variablerna i strukturen kommer få värdena 1,2,3 och 4 efter ordning de har deklarerats. Vid samma scenario i .NET så blir resultatet att variablernas värden är odefinierade. Det resultatet beror på att strukturens medlemmar inte nödvändigtvis ligger sekventiellt i minnet utan variabel *two* kan ligga för *one* och så vidare. Det kan även vara så att variablerna ligger utsprida i minnet. En kopiering av en bytevektor till ett objekt skulle vid ett sådant tillfälle resultera i att data skrivs till minnesområden som inte är allokerade av objektet. [23][27]

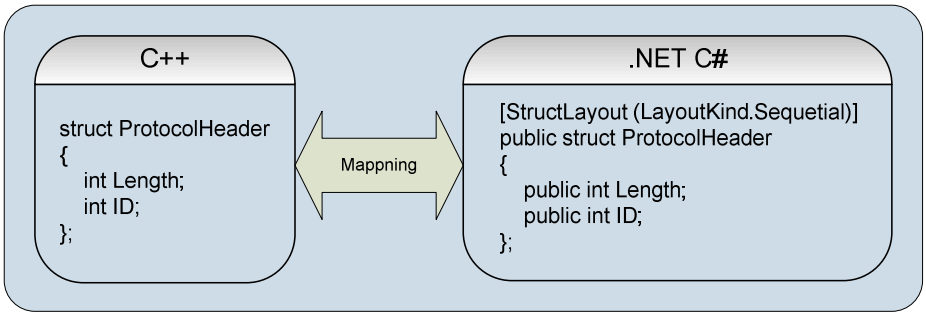


Figur 8-1: Problemen att typomvandla strukturer i .NET C#.

För att lösa problemet har vi angripit det på två olika sätt och testat vilket av sätten som är bäst utifrån de krav som har ställts på simulatorn.

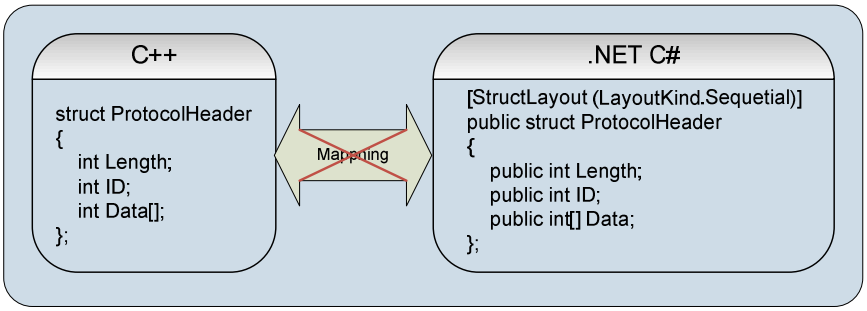
Metod 1

Den första lösningen använder sig av .NETs egna assembly-direktiv. Eftersom strukturer inte nödvändigtvis ligger sekventiellt i minnet så har .NET tillhandahållit ett sätt att tvinga strukturer att vara sekventiella. Ett exempel på mappningen ses i Figur 8-2.



Figur 8-2: Hur strukturer görs sekventiella i .NET C#.

Denna lösning har dock sina brister och kan inte appliceras på alla typer av strukturer. Om t.ex. en strukturs storlek inte kan bestämmas vid kompileringstillfället fungerar inte längre .NETs inbyggda assembly-direktiv för att konvertera strukturer. Ett exempel på en struktur som inte kan bestämmas vi kompileringstillfället illustreras i Figur 8-3. Problemet i figuren är att C++ kommer att tolka *Data* som dynamisk och att försöka matcha den dynamiskt vid en minneskopiering, C# .NET kommer dock att tolka *Data* som en int-pekare med en fast storlek på fyra byte. För att lösa problemet använder vi oss av metod 2.



Figur 8-3: Bristerna med dynamiska arrayer i C# .NET.

Metod 2

Metod 2 använder sig av egendefinierande funktioner för att bygga upp och bryta ned strukturer från och till bytevektorer. För att förtydliga denna process se Figur 8-4 som visar hur en implementation av metod 2 kan appliceras på strukturen i Figur 8-3.

```
public struct ProtocolHeader
{
     public void Serialize( BinWriter binW )
     {
          binW.Write(Length);
          binW.Write(ID);
          binW.Write(Data);
     }

     public void Deserialize( BinReader binR )
     {
          Length  = binR.ReadInt32();
          ID      = binR.ReadInt32();
          Data    = binR.ReadBytes(Length);
     }

     private int Length;
     private int ID;
     private int Data[];
}
```

Figur 8-4: Implementation för att hantera dynamiska strukturer.

När valet mellan dessa metoder skulle göras vägde vi deras för- och nackdelar mot varandra. Fördelarna med metod 1 är att den kan appliceras snabbt och enkelt, dock har den nackdelar i och med den inte kan hantera komplexa strukturer som t.ex. innehåller dynamiska vektorer. Fördelen med metod 2 är att den kan hantera alla former av strukturer hur komplexa de än är. Nackdelarna är dock att den tar lång tid att implementera i och med att alla strukturer måste implementera egna uppbyggnings- och nedbrytningsfunktioner. Vi valde att använda metod 2 på grund av att Icomeras egna protokolls strukturer innehåller flertalet dynamiska vektorer, vilket metod 1 inte kan hantera. [23]

9 Sammanfattning och slutsats

9.1 Introduktion

I detta kapitel utvärderas simulatorn och projektet som helhet. Det redogörs för framgångar och motgångar, samt vad som skulle ha gjorts annorlunda. Frågan ställs också om simulatorn uppfyller de krav som specificerades vid projektets början.

9.2 Simulatorn

Under utvecklingen av simulatorn har vi haft framgångar som beskrivs i avsnitt 9.2.1. Framgångar är bland annat simulatorns stabilitet, användargränssnitt, användarvändlighet, samt dess minimala krav på hårdvara. Vi har även haft några motgångar som tas upp i avsnitt 9.2.2. Generellt för de motgångar vi har haft, är att de är relaterade till simulatorns begränsade prestanda. Det är även kring prestanda som de viktigaste förbättringarna handlar om i avsnitt 9.2.3.

9.2.1 Framgångar

En av de främsta framgångarna med simulatorn är att den är mycket stabil, vilket vi hade stort fokus på. En stor faktor som spelar in är att vi använt oss av .NET, på grund av att den använder sig av skräpsamlare, undantagshantering och ett stabilt ramverk med färdiga datastrukturer och algoritmer.

För att simulatorn ska vara lättanvänd, har vi lagt stor vikt på enkel konfiguration och att det ska gå fort att starta en simulering. Detta visas i användarfallen, i kapitel 7. Där visas hur användaren kan med några få knapptryckningar konfigurera och köra en simulering. Användaren har även möjlighet att göra en mer avancerad konfiguration om så önskas.

Simulatorns grafiska användargränssnitt blev uppskattat av Icomera och kommer även att ligga som grund för omarbetning av IGW:ns användargränssnitt. Grafiska användargrässnitt brukar ofta vara svåra att utveckla på ett bra sätt på grund av att det finns nästintill oändligt antal

designmöjligheter och att Icomera blev nöjda med designen ser vi som en framgång.

När simulatorn körs med maximal belastning går inte processoranvändningen upp markant. Detta är testat på så lågt som en 300 MHz processor. Det borde även gå att lägga till mycket funktionalitet innan datorn överbelastas.

9.2.2 Motgångar

Den största bristen i simulatorn introducerades med .NET sockets. .NET sockets gör att simulatorn inte uppfyller den hastighet som vi hade hoppats på. Detta problem tas upp i avsnitt 8.4, där problemet utreds genom en jämförelse mellan .NET:s och C++ sockethantering. Simulatorn uppfyller ändå Icomeras krav vad det gäller hastighet. Det på grund av att länkarna som Icomera använder sig av inte överstiger simulatorns maxhastighet. En eventuell lösning på problemet beskrivs i avsnitt 9.2.3.

En till brist som påverkar simulatorns maxhastighet är operativsystemets noggrannhet vad gäller tidshändelser. Det handlar om att tidshändelser inte kan skickas med ett tätare intervall än 1 millisekund. Det gör att simulatorn endast klarar av hastigheter upp till 12 Mbit/s innan precisionen blir sämre och paket börjar skickas buntvis. I avsnitt 8.2 ges en fullständig redogörelse för detta problem.

9.2.3 Förbättringar

En av de viktigare förbättringarna vi anser borde göras handlar om att åtgärda bristen i sockethanteringen, för att öka simulatorns prestanda. Genom att byta ut sockets mot en annan lösning vore en stor förbättring som kräver relativt lite resurser. Ett alternativ vore att använda WinPcap som vi använder för att spara ned Icomeras egna protokolls trafikdata. Det är även möjligt att WinPcap skulle kunna användas för att öka prestandan. Det har dock inte gjorts några tester på detta.

En annan förbättring är att utöka funktionaliteten på skriptspråket, som automatiserar händelser. Tanken med ett skriptspråk var att det skulle vara enkelt att använda utan programmeringskunskaper. I Icomeras fall kommer med största sannolikhet alla som använder skriptfunktionaliteten ha

programmeringserfarenhet, vilket gör att mer avancerad funktionalitet skulle kunna läggas till.

9.3 Projektet

För att anse projektet slutfört måste alla kraven vara uppfyllda. I avsnitt 9.3.1 görs därför en reflektion över huruvida kraven är uppfyllda eller inte. Avslutningsvis i avsnitt 9.3.2 redogörs för vad vi skulle ha gjort annorlunda utefter erfarenheter vi fått under projektet.

9.3.1 Kraven uppfyllda

För att verifiera att kraven är uppfyllda återkopplar vi implementationen mot de enskilda kraven. Nedan är en lista med alla krav så som de var specificerade i avsnitt 4.2. Under varje krav ges en kort förklaring varför det är uppfyllt. Varje krav är även kopplat till ett antal testfall som är beskrivna i bilaga A.

Prioritet 0:

På en länk ges möjlighet att:

A. Få in länkkarakteristik i simulatorn, utläst ur Icomeras egna protokolls trafikdata.

För att få ut länkkarakteristiken används CSDP- och CSCP-avkodarmodulen. Hur den är implementerad och hur den används av simulatorn är beskrivet i avsnitt 4.3.3.1 och 5.2. Kravet verifieras av testfall LSE1 i bilaga A.

B. Specificera bandbredd.

Bandbredd är definierad i avsnitt 3.2.1 och ligger till grund för vår implementation. Hur bandbredden är implementerad beskrivs i avsnitt 395.3.2. Bandbredden kan konfigureras manuellt via användargränssnittet, se avsnitt 6.8, eller via skript, se avsnitt 5.5. Bandbredden är fördelad enligt en normalfördelning, vilket är förklarat i avsnitt 5.3.2. Kravet verifieras av testfall LSE2 i bilaga A.

C. Specificera latens.

Latens är definierad i avsnitt 3.2.2 och ligger till grund för vår implementation. Hur latensen är implementerad beskrivs i avsnitt 395.3.3. Latensen kan konfigureras manuellt via användargränssnittet, se avsnitt 6.8, eller via skript, se

avsnitt 5.5. Latensen är fördelad enligt en poissonfördeling, vilket är förklarat i avsnitt 5.3.3. Kravet verifieras av testfall LSE3 i bilaga A.

D. Specificera bitfel.

Begreppet paketförlust är definierad i avsnitt 3.2.3 och ligger till grund för vår implementation. Hur paketförluster är implementerad beskrivs i avsnitt 5.3.1. Paketförluster kan konfigureras manuellt via användargränssnittet, se avsnitt 6.8, eller via skript, se avsnitt 5.5. Kravet verifieras av testfall LSE4 i bilaga A.

E. Stänga av upp- samt nedtrafik.

Implementationsdetaljer om detta krav har utelämnats på grund av dess triviala natur. Kravet verifieras av testfall LSE5 i bilaga A.

Det ska finnas förinställda värden av ovanstående egenskapar för följande länkar:

GPRS – Dataöverföring i GSM-nätet, se avsnitt 3.3.1.

UMTS 128 – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

UMTS 384 – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

UMTS HSDPA – Dataöverföring i 3G-nätet, se avsnitt 3.3.2.

Satellit – Dataöverföring via satellit, se avsnitt 3.3.3.

De förinställda värdena för ovanstående länktyper är tagna från avsnitt 3.3.1, 3.3.2 och 3.3.3. Hur de används beskrivs i avsnitt 6.8.

Prioritet 1:

A. XML-standard på konfigurationsfiler.

Detta krav är uppfyllt och kan bland annat verifieras av att de fördefinierade länktyperna och skriptspråket använder sig av XML.

B. Ett användarvänligt grafiskt användargränssnitt (GUI).

Detta krav är svårt att verifiera. Vi anser det dock som uppfyllt i och med Icomeras positiva reaktioner, vilket kan tolkas att kravet är uppfyllt.

C. Simulatormotorn ska exekveras som en Microsoft windowstjänst (eng: Microsoft Windows Service).

Simulatorn installeras med ett installationsprogram och körs sedan som en windowstjänst, vilket verifierar kravet. Kravet verifieras av testfall LSE0 i bilaga A.

Prioritet 2:

A. Beräkna kostnad på en länk med avseende på megabyte och tid.

Implementationsdetaljer om detta krav har utelämnats på grund av dess triviala natur. Hur kostnaden ställs in är beskrivet i avsnitt 6.8. Kravet verifieras av testfall LSE6 i bilaga A.

B. Använda ett skript för att automatisera händelseförlopp.

Skriptmotorn använder sig av XML-formaterade skripfiler för att automatisera händelser i simulatorn. Detta beskrivs i avsnitt 5.5. Kravet verifieras av testfall LSE7 i bilaga A.

C. Ha stöd för flera tågklienter samtidigt.

Simulatormotorn stödjer flera samtidigt anslutna IMSR:er. Det visas bland annat i användargränssnitt där flera IMSR-noder visas i VehicleView, se 6.6. Kravet verifieras av testfall LSG1 i bilaga A.

Prioritet 3:

B. Spara ned Icomeras egna protokolls trafikdata.

Simulatorn kan spara datatrafik i filformatet libpcap. Dessa filer kan sedan analyseras i bl.a. Ethereal. Implementationen av nedsparningsförfarandet finns beskrivet i avsnitt 5.6. Kravet verifieras av testfall LSE8 och LSE9 i bilaga A.

I Tabell 9-1 visas en matris över hur testfallen i bilaga A, matchas mot kraven. Raderna representerar testfall och kolumnerna representerar krav. Till exempel matchas testfallet LSE1 mot krav 0A. 0A är krav A med prioritet 0. Uppfylls alla tester i verifieringsplanen vet vi att alla krav är uppfyllda och implementationen kan anses färdig.

Testfall	0 A	0 B	0 C	0 D	0 E	1 A	1 B	1 C	2 A	2 B	2 C	3 A
LSE0								x				
LSE1	x											
LSE2		x										
LSE3			x									
LSE4				x								
LSE5					x							
LSE6									x			
LSE7										x		
LSE8												x
LSE9												x
LSG1											x	

Tabell 9-1: Hur kraven är matchade till testfallen

9.3.2 Vad skulle vi ha gjort annorlunda?

När projektet summeras finns det en bra översiktsbild på delar i projektet som vi i ett framtida projekt skulle göra annorlunda. Det handlar framförallt om de administrativa bitarna och inte de implementationsmässiga. Först och främst skulle kravspecifikationen fastställas mycket klarare. Den detaljerade kravspecifikationen har varit oklar på grund av att Icomera inte i visste vad de ville ha. Det uppenbarade sig t.ex. i slutet av projektet att vissa personer på utvecklingsavdelningen missuppfattat vad vi utvecklat och förväntade sig en helt annan typ av simulator. Kravspecifikationen som gjordes i början av projektet förankrades inte av hela utvecklingsavdelningen. Resultatet av det blev att krav lades till och togs bort under hela projektet. Det gjorde att vi spenderade tid på krav som togs bort och därmed fanns det inte tillräckligt med tid för tillkommande krav. Icomeras oklarhet över vad de ville ha resulterade i att många av kraven blev diffusa. Ett exempel är krav 1B, ”Ett användarvänligt grafiskt användargränssnitt”, som är omöjligt att verifiera därmed även omöjligt att tidsestimera.

Att använda sig av någon annan form av utvecklingsmetod än vattenfallsmodellen hade sparat oss mycket tid. Vattenfallsmodellen bygger på att alla krav är fastställda innan projektet början och sedan inte ändras under dess gång. Som vi nämnde ovan så var det inte fallet för oss. Det hade

förmodligen varit bättre att använt sig av en utvecklingsmetod med korta iterationer, t.ex. eXtream Programing (XP), vilket är mer anpassad för kontinuerliga kravändringar och omprioriteringar. Ett exempel är användargränssnittet. Efter att ha arbetat med användargränssnittet i tre veckor skulle vi ha ett möte där en genomgång och synpunkter skulle framföras. Det visade sig att Icomera vill ha något helt annat än vad vi uppfattat från början och det var bara för oss att gå tillbaka till ritbordet. Kortare iterationer i detta läge hade sparat oss mycket tid.

För att sammanfatta hela projektet, så var det alldeles för omfattande för ett examensarbete. Efter att ha studerat andra examensarbeten har vi konstaterat att de är begränsade i sin implementationsdel och därmed får de oftast en mer specifik uppgift. Vår implementations del består av ca 25000 rader kod, varav 5000 i motorn och 20000 i användargränssnittet. Hade vi gjort om projektet skulle vi ha uteslutit användargränssnittet. Vi skulle också bara använt skript för att styra länkkarakteristiken. Interaktionen med simulatorn skulle ske genom ett Command Line Interface (CLI). Dessa tre ändringar skulle resultera i en mer avgränsad och fördjupad rapport.

Förkortningar

3G – Tredje generationens mobiltelefonsystem

CLI – Command Line Interface

CSCP – Client Server Control Protocol

CSDP – Client Server Data Protocol

DHCP – Dynamic Host Configuration Protocol

FTP – File Transfer Protocol

GMT – Greenwich Mean Time

GPRS – General Packet Radio Service

GSM – Global System for Mobile communications

GUI – Graphical User Interface

HSDPA – High Speed Downlink Packet Access

HTTP – Hypertext Transfer Protocol

IGW – Icomera Gateway

IMSR – Icomera Mobil System Rack

I/O – Input / Output

IP – Internet Protocol

LSE – Link Simulator Engine

LSGUI – Link Simulator Graphical User Interface

MSDN – Microsoft Developer Network

MVC – Model View Controller

NMT – Nordic Mobile Telephony

RAS – Remote Access Service

RPC – Remote Procedure Call

RTT – Round Trip Time

SJ – Statens Järnvägar

TCP – Transmission Control Protocol

TDMA – Time Division Multiple Access

UDP – User Datagram Protocol

UML – Unified Modelling Language

UMTS – Universal Mobile Telecommunications System

VPN – Virtuellt Privat Nätverk

W-CDMA – Wideband Code Division Multiple Access

XML – Extensible Markup Language

XP – Extreme Programming

Referenser

[1] Schach, R. (1999) *Software Engineering*, Fourth Edition, McGraw-Hill, Boston, MA.

[2] Whittaker, J.A. (2003, mars) Vad är test av mjukvara? Och varför är det så svårt? *OnTime - ger insikt i tid, 6*-11.

[3] Beck, K. (2004) *Extreme Programming Explained : Embrace Chang.* 2nd Edition, Addison-Wesley Professional.

[4] Crispin, L & House, T (2002) *Testing Extreme Programming*. First Edition, Addison-Wesley.

[5] Beck, K (2003) *Test-Driven Developmen*t, Second Printing, Addison-Wesley.

[6] Kurose, J. F. & Ross, K. W (2003) *Computer Networking.* Pearson Education. Inc

[7] Stallings W. (1997) *Data and computer communications.* Prentice-Hall, Inc.

[8] Fiche, G. & Hébuterne, G (2004) *Communicating Systems & Networks: Traffic & Performance.* Kogan Page.

[9] Post och Telestyrelsen Faktablad (2004) *3G – tredje generationens mobiltelefoni* Tillgänglig på Internet: http://pts.se/Archive/Documents/SE/Faktablad_UMTS%20-%20tredje%20generationens%20mobiltelefoni_PTS-F-2002_4.pdf [Hämtad 05.12.15]

[10] 3GPP (2004) *Overview of 3GPP Release 99 - Summary of all Release 99 Features:* http://www.3gpp.org/Releases/Rel99%20description_v040720.doc [Hämtad 05.12.22]

[11] Sirius officiela hemsida http://www.ses-sirius.com/satellites/sirius-2/ [Hämtad 07.11.25]

[12] Liberty, J. (2001) *Programmning C#.* First Edition, O'Reilly

[13] Marshall, D. (2005) *Programming Microsoft Visual C# 2005: The Language,* 2005 Ed edition, Microsoft Press.

[14] Ethereal officiella hemsida http://www.ethereal.com/ [Hämtad 07.11.21]

[15] Möller, M. (2001) *Statistiska modeller inom datateknik.* Studentlitteratur, Lund

[16] Libpcap officiela hemsida www.tcpdump.org [Hämtad 07.12.03]

[17] Adding a basic dissector http://ethereal.com/docs/edg_html_chunked/ChDissectAdd.html [Hämtad 05.12.15]

[18] Buschmann, F & Meunier, R & Rohnert, H & Sommerlad, P & Stal, M (2001) *Pattern-Oriented Software Architecture : A System of Patterns.* Volume 1, Wiley.

[19] Gamma, E & Helm, R & Johnson, R & Vlissides, J (2005) *Design Patterns : Element of Reusable Object-Oriented Software.* 32nd Printing, Addison-Wesley.

[20] Fowler, M (2005) *Analysis Patterns : Reusable Object Models*. 17th Printing, Addison-Wesley.

[21] Marshall, B & Reeves, R (2001) *Win32 System Services : The Heart of Windows 98 and Windows 2000.* Third Edition, Prentice Hall.

[22] McLean, S & Naftel, J & Williams, K (2002) Microsoft .NET Remoting. First Edition, Microsoft Press.

[23] Nathan, A (2004) *.NET and COM : The Complete Interoperability Guide.* Third Printing, Sams.

[24] IPerf officiella hemsida http://dast.nlanr.net/Projects/Iperf/ [Hämtad 07.09.15]

[25] Richter J. (2005) Concurrent Affairs. *MSDN Magazine Europe, 1*(10), 80-84
[26] WinPcap officiela hemsida http://www.winpcap.org/ [Hämtad 07.12.01]
[27] Prata, S. (2004) *C++ Primer Plus.* 5th edition, Sams.

Bilaga A – Verifieringsplan

Verification Plan

Project:	*Link Simulator*
Author:	*Fredrik Nilsson, Jonas Wånggren*
Checked by:	
Approved by:	*-*
Date:	*2006-01-09*
Version:	*Draft A*

1 TABLE OF CONTENTS

2 INTRODUCTION

2.1 Scope

The scope of this verification plan is to cover all functionality of the Link Simulator Engine and Link Simulator GUI.

2.2 Credentials

The user credentials that are required in the test cases are not available in this document due to security issues.

2.3 Test scripts and files

All required test scripts and test files are available at the Icomera intranet

2.4 Time estimations

All time estimations are made for tests in lab environment.

2.5 Configuration

It should be worth pointing out that all links are set in default configuration mode by using the default values entered in the image. For each particular operation environment, there is a need to fine tune these settings for optimal performance for those specific circumstances.

The different configuration properties are currently under the responsibility of the Operations department.

3 TEST ENVIRONMENTS

3.1 Lab environment (LAB)

Tests are performed in the lab to provide an environment that is as close as possible to an optimal environment, regarding airborne communication such as satellite communication, GPRS and UMTS.

3.1.1 System setup

The tests should be performed on a system in the lab at the Icomera office.

4 TEST CASES

4.1 Installation Procedure

4.1.1 LSE0: Link Simulator Engine Installation

Summary:
Install the Link Simulator Engine as a service

Estimated time:
10 min

Preconditions:
- All hardware components should be available.

Test procedure:
1. Install the Link Simulator Engine

Expected result(s):
- Link Simulator should run as a service

4.2 Link Simulator Engine (LSE) Functionality

4.2.1 LSE1: Protocol characteristic

Summary:
Verify that a correct characteristic was read from the protocol.

Estimated time:
5 min

Preconditions:
- Link Simulator Engine must be up and running and a GUI connected.

Test procedure:
1. Connect a link using Mobility Manager.

Expected result(s):
- The link characteristics should appear in the GUI.

4.2.2 LSE2: Specify bandwidth

Summary:
Verify that the bandwidth varies near the expected value with a normal distribution.

Estimated time:
5 min

Preconditions:
- The Link Simulator Engine is up and running. One link connected with Mobility Manager.

Test procedure:
1. Use iperf to send data between IMSR and gateway

Expected result(s):
- The bandwidth in iperf should agree with the specified value.

4.2.3 LSE3: Specify latency

Summary:
Verify that the latency varies near the expected value with a poisson distribution.

Estimated time:
70 min

Preconditions:
- The Link Simulator Engine is up and running. One link connected with Mobility Manager.

Test procedure:
2. Check time 1.
3. Let Mobility Manager be connected in one hour.
4. Check time 2.

Expected result(s):
- Look in the database that RTT values between time 1 and time 2 are Poisson distributed.

4.2.4 LSE4: Specify packet loss

Summary:
Verify that the packet loss follows the specified value.

Estimated time:
60 min

Preconditions:
- Link Simulator Engine up and running with one link connected with the Mobility Manager.

Test procedure:
1. Specify that 1/3000 possibility of bit error.
2. Ping the gateway with 32 bytes packets 1000 times.
3. Ping the gateway with 128 bytes packets 250 times.

Expected result(s):
- 32*8/3000 = the packet loss from ping.
- 128*8/3000 = the packet loss from ping.

4.2.5 LSE5: Specify up and down traffic

Summary:
Verify that the Link Simulator Engine can disallow at least one way of the traffic on the link.

Estimated time:
10 min

Preconditions:
- Link Simulator Engine up and running.

Test procedure:
1. Establish a link connection with the mobility manager.
2. Disallow the up traffic.
3. Allow up traffic, make sure the link reestablish.
4. Disable the down traffic.

Expected result(s):
- Check if the link connection disappears both in the Mobility Manager and the gateway when the traffic disables.

4.2.6 LSE6: Cost

Summary:
Verify that the cost is correct computed

Estimated time:
10 min

Preconditions:
- Link Simulator up and running with one link connected using Mobility Manager. A GUI connected

Test procedure:
1. Make 100 mega byte of traffic in 1 minute.

Expected result(s):
- The cost should be in the GUI = 100*cost per byte or if 1*cost per minute.

4.2.7 LSE7: Script

Summary:
Verify that the script change values in the Link Simulator Engine

Estimated time:
60 min

Preconditions:
- Link Simulator Engine must be up and running. A GUI connected

Test procedure:
1. Write a simple script.
2. Run the script

Expected result(s):
- The log in the GUI should agree with the script.

4.2.8 LSE8: Save CSCP traffic

Summary:
Verify that the CSCP traffic is saved correct.

Estimated time:
20 min

Preconditions:
- Link Simulator Engine is up and running.

Test procedure:
1. Start Observer packet sniffer.
2. Start the sniffer in Link Simulator Engine.
3. Establish a link.

Expected result(s):
- The traffic saved with the sniffer in Link Simulator Engine agree with the packets shown in Observer.

4.2.9 LSE9: Ethereal disserter

Summary:
Verify that the Ethereal disserter is correct

Estimated time:
5 min

Preconditions:
- Link Simulator Engine up and running, one link connected.

Test procedure:
1. Establish a link with Mobility Manager.

Expected result(s):
- The same result in both Ethereal and Observer.

Link Simulator GUI (LSG) Functionality

4.2.10 LSG1: GUI performance

Summary:
Verify that the GUI can handle at least 100 users

Estimated time:
15 min

Preconditions:
- Link Simulator Engine up and running. User Simulator configured.

Test procedure:
1. Connect 100 users with User Simulator.

Expected result(s):
- The GUI should still be usable.

5 REQUIREMENT VERIFICATION MATRIX

Test Case	0A	0B	0C	0D	0E	1A	1B	1C	2A	2B	2C	3A
LSE0								X				
LSE1	X											
LSE2		X										
LSE3			X									
LSE4				X								
LSE5					X							
LSE6									X			
LSE7						X				X		
LSE8												X
LSE9												X
LSG1							X				X	