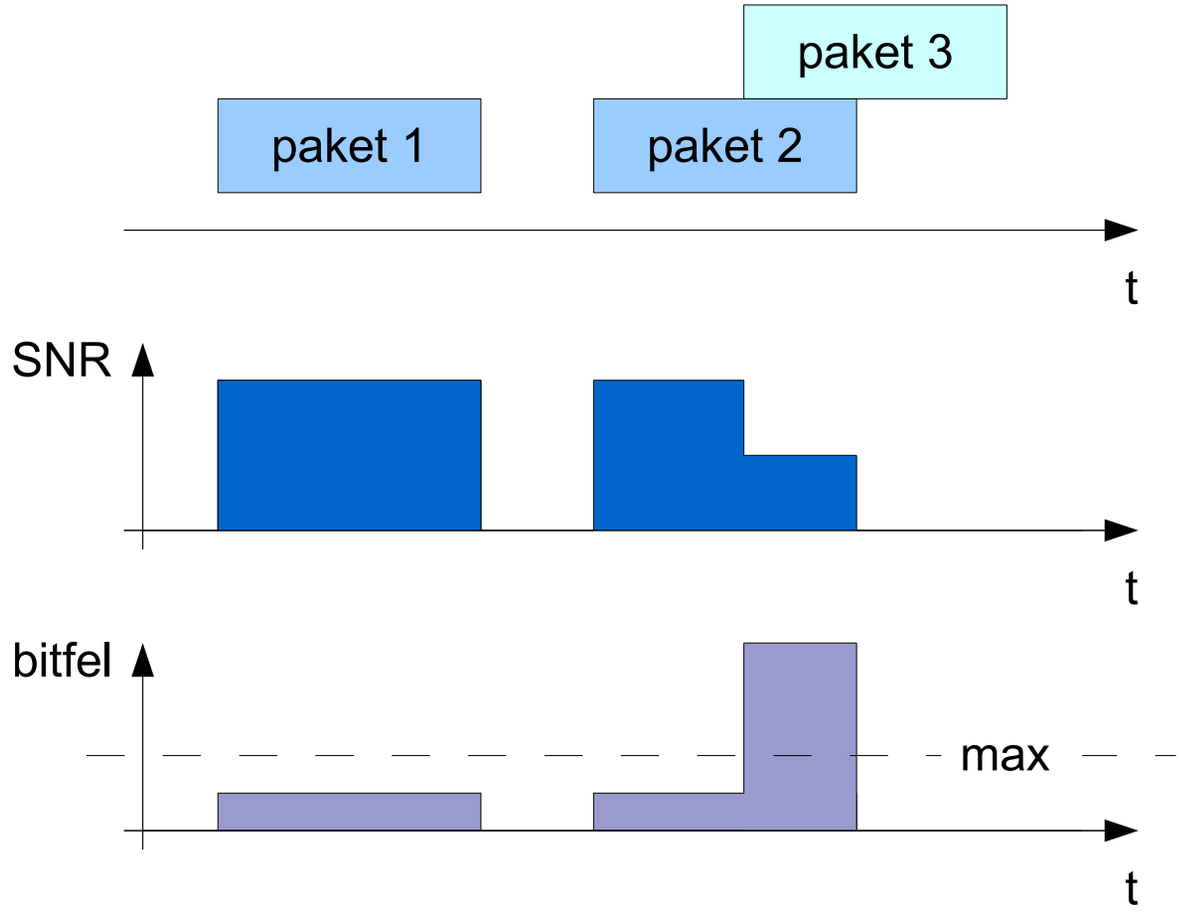
# Teknik för radionätsimuleringar med karaktäristisk kanalinformation

ULF STERNER




FOI är en huvudsakligen uppdragsfinansierad myndighet under Försvarsdepartementet. Kärnverksamheten är forskning, metod- och teknikutveckling till nytta för försvar och säkerhet. Organisationen har cirka 1000 anställda varav ungefär 800 är forskare. Detta gör organisationen till Sveriges största forskningsinstitut. FOI ger kunderna tillgång till ledande expertis inom ett stort antal tillämpningsområden såsom säkerhetspolitiska studier och analyser inom försvar och säkerhet, bedömning av olika typer av hot, system för ledning och hantering av kriser, skydd mot och hantering av farliga ämnen, IT-säkerhet och nya sensorers möjligheter.

FOI
Totalförsvarets forskningsinstitut
Ledningssystem
Box 1165
581 11 Linköping

Tel: 013-37 80 00
Fax: 013-37 81 00

www.foi.se

FOI-R–2757–SE
ISSN 1650-1942

Teknisk rapport
April 2009

**Ledningssystem**

Ulf Sterner

# Teknik för radionätsimule-ringar med karaktäristisk kanalinformation

| | |
|---|---|
| Titel | Teknik för radionätsimuleringar med karaktäristisk kanalinformation |
| Title | Radio network simulations utilizing characteristic channel information |
| Rapportnr / Report No. | FOI-R–2757–SE |
| Rapporttyp | Teknisk rapport |
| Report Type | Technical Report |
| Månad / Month | April / April |
| Utgivningsår / Year | 2009 |
| Antal sidor / Pages | 25 |
| ISSN | 1650-1942 |
| Kund / Customer | FMV |
| Kompetenskloss | 22 Samband och Telekom |
| Projektnr / Project No. | E7478 |
| Godkänd av / Approved by | Åsa Waern |



FOI, Totalförsvarets Forskningsinstitut
Avdelningen för Ledningssystem
Box 1165
581 11 LINKÖPING

FOI, Swedish Defence Research Agency
Command and Control Systems
P.O. Box 1165
SE-581 11 LINKÖPING

## Sammanfattning

Inom projektet ”Vågutbredning för bredbandiga vågformer” har möjligheten att förse OPNET med en mer realistisk länkmodell som på ett bättre sätt kan modellera ett modernt radiosystem undersökts. I denna rapport beskriver vi de OPNET-relaterade frågorna som har behandlats i projektet. Två olika länkmodeller har implementerats inom projektet. Den ena modellen bygger på att mer realistiska länkdämpningar används medan den andra bygger på att kanalkapaciteten estimeras. De båda framtagna länkmodellerna baseras på den befintliga kommunikationslänksstrukturen i OPNET så att de enkelt ska kunna användas av existerande OPNET-modeller.

Nyckelord: OPNET, kanalmodell, länkmodell

## Abstract

In the project “Wave propagation for wide band waveform” the possibility to provide OPNET with a more realistic link model is investigated. In this report we describe all OPNET related questions that are considered in the project. Two different link models have been implemented in the project; one based on using more realistic link attenuation values and one based on estimating the channel capacity. Both models uses OPNET:s existing link model structure so that it should be simple to use them in existing models.

Keywords: OPNET, channel modell, link model

# Innehållsförteckning

# 1 Inledning

Inom projektet ”Vågutbredning för bredbandiga vågformer” har möjligheten att förse OPNET [1] med en mera realistisk länkmodell som på ett bättre sätt kan modellera radiokanalen undersökts. I denna rapport beskrivs de OPNET-relaterade frågorna som har behandlats i projektet. För en beskrivning av teorin bakom de använda länkmodellerna se [2].

Ett delmål för projektet har varit att implementera en eller flera lämpliga länkmodeller i OPNET. Målsättningen har här varit att basera en ny länkmodell på OPNET:s nuvarande länkstruktur. Detta val innebär ett antal begränsningar för de tänkta modellerna då det övergripande programflödet är givet av den nuvarande strukturen. Samtidigt ökar dock kompabiliteten med befintliga modeller avsevärt genom detta val, vilket minskar kostnaden för att använda länkmodellen i andra projekt.

Förutom en bibehållen grundstruktur har vi även ansett att beräkningskostnaden för en ny kanalmodell är viktig. Vid simuleringar av radionät i OPNET ligger en mycket stor del av simuleringstiden på själva radiodelen. En viss ökning av simuleringstiden är givetvis rimlig om noggrannheten i resultaten blir bättre då tack vare en bättre länkmodell. Ökningen i simuleringstid får dock inte vara hur stor som helst. En simulering som normalt tar några minuter får inte ta flera dagar att köra om en ny länkmodell används. Målsättningen har här varit att beräkningstiden inte får öka med mer än en faktor tre.

Den nuvarande implementationen innehåller två länkmodeller, dels en enkel modell som enbart sätter länkdämpningen men som för övrigt använder OPNET:s standardmodeller och dels en som bygger på kanalkapacitetsberäkningar.

Rapporten är uppdelad i fem kapitel. I det andra kapitlet ges en översiktlig beskrivning av OPNET:s nuvarande modell för radiosändare och radiomottagare. I det tredje behandlas vilka förändringar som är möjligt att göra inom det nuvarande ramverket. Vi ger sedan en övergripande beskrivning av hur de nya programkomponenterna är designade. Avslutningsvis finns en användarguide för programkomponenterna.

# 2 OPNET:s nuvarande länkmodell

I OPNET modelleras radiosändare och radiomottagare som ett antal pipeline-steg [1]. För att skapa en mera realistisk länkmodell i OPNET behöver pipeline-steg på både sändarsidan och mottagarsidan förändras. För att skapa en bättre förståelse för hur dagens modell fungerar ger vi här en kort beskrivning av OPNET:s radiomottagare och radiosändare. Vi avslutar sedan med ett exempel på hur beräkningarna i mottagaren sker vid mottagning av flera parallella paket.

## 2.1 OPNET:s radiosändare

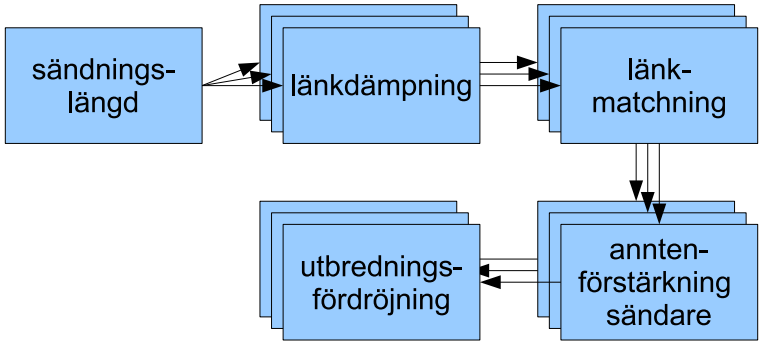
I figur 2.1 visas OPNET:s pipeline-steg i radiosändaren. I det första steget i radiosändaren beräknas hur lång tid sändningen tar. Sändningstiden är definierad som tiden mellan att den första symbolen börjar sändas till att den sista symbolen har sänts i sändaren. Till skillnad från de övriga stegen exekveras detta steg enbart en gång per sändning. I nästa hanteras de storskaliga utbredningsförhållandena. Används exempelvis OPNET:s terrängberoende modeller för att beräkna länkdämpningen sker anropet till dessa här.

I det tredje steget undersöks om sändaren och mottagaren är kompatibla. Normalt testas här egenskaper som sändningsfrekvens och modulationstyp. Länkdämpningssteget och länkmatchningssteget exekveras en gång för varje mottagare. De efterföljande stegen i sändaren exekveras en gång för varje mottagare som på något sätt kommer att påverkas av sändningen. I det fjärde steget beräknas antennförstärkningen i sändarantennen. Slutligen beräknas utbredningsfördröjningen hos kanalen.

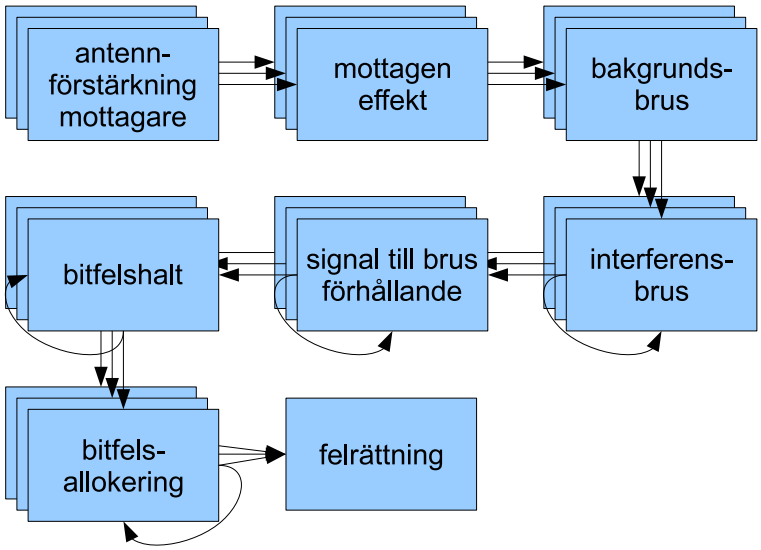
## 2.2 OPNET:s radiomottagare

Radiosändaren i OPNET modelleras med hjälp av 8 stycken pipeline-steg. Dessa steg kan ses i figur 2.2. I det första steget beräknas antennförstärkningen för det mottagna paketet. I nästa steg beräknas den mottagna effekten med beaktande av utbredningsvägen. I det tredje steget beräknas bakgrundsbruset, därefter tas hänsyn till interferenser från andra pågående sändningar. I det femte steget beräknas signal-till-brusförhållandet (SNR) för paketet. I det efterföljande steget beräknas bitfelshalten varefter nästa steg beräknar antalet felaktiga bitar i paketet med hjälp av en slumpfördelning och den tidigare beräknade bitfelshalten.

I det sista steget avgörs om den felrättande koden kan rätta de uppkomna




Figur 2.1: Pipeline-stegen i OPNET:s radiosändare.




Figur 2.2: Pipeline-stegen i OPNET:s radiomottagare.

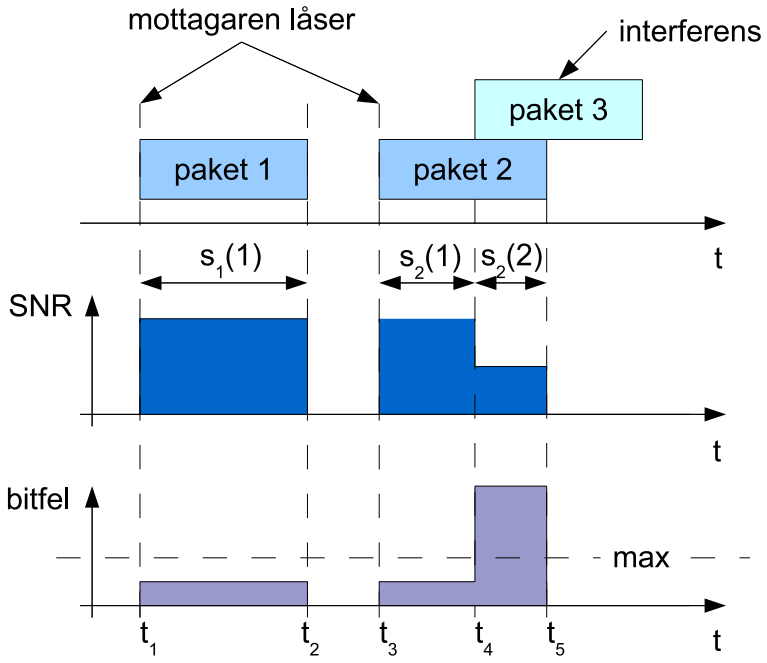
felen eller om paketet ska slängas. Noteras bör här att antennförstärkningen, den mottagna effekten och bakgrundsbruset för ett paket enbart kan ha ett värde. Interferensbruset kan dock variera över ett paket på grund av kollisioner med andra paket. På samma sätt kan SNR och bitfelshalt vara olika för olika delar av paketet. Steg 4-7 i mottagarkedjan kan således komma att exekveras flera gånger för olika paketsegment under mottagandet av ett paket.

## 2.3 SNR-beräkning vid parallella paket

I figur 2.3 visas ett exempel på hur mottagarkedjan fungerar. Vid tidpunkten $t_1$ låser mottagaren på paket 1. Vid denna tidpunkt exekveras steg 1-5 i mottagarkedjan. Vid slutet av mottagningen av paket 1, vid tidpunkten $t_2$, exekveras slutligen steg 6-8. Då inga andra paket interfererar med paketet kommer SNR och bitfelshalten att vara konstant under hela paketet.

När paket 2 kommer fram vid $t_3$ låser mottagaren återigen och steg 1-5 exekveras. Vid tidpunkten $t_4$ kommer paket 3 fram. Det normala är att mottagaren bara kan låsa på det första paketet medan andra överlappande paket behandlas som brus. Något behov av att beräkna mer än den mottagna effekten för paket som behandlas som brus finns normalt inte vilket gör att enbart steg 1-2 behöver exekveras för paket 3. Därefter exekveras steg 6-7 för segment $s_2(1)$ hos paket 2 innan steg 4-5 exekveras för segment $s_2(2)$. Vid tidpunkten $t_5$ körs sedan steg 6-7 för segment $s_2(2)$ innan steg 8 körs för hela paket 2.

Det är vidare viktigt att observera att alla bitfelsberäkningar i OPNET:s nuvarande radiomottagarmodell sker på informationsbitsnivå. Detta gäller även i fallet då större symboler som innehåller många informationsbitar används.




Figur 2.3: Beräkning av SNR och antal bitfel i OPNET.

# 3 Möjliga förändringar

En avgörande fråga vid förändringar av OPNET:s radiomodell är om den nuvarande strukturen ska behållas eller om en mera generell struktur ska införas. Eftersom införande av en ny struktur innebär att befintliga modeller som använder dagens struktur måste modifieras så innebär en ny struktur ett avsevärt merarbete. Inom detta projekt är målsättningen därför att behålla den befintliga strukturen och enbart ändra de ingående pipeline-stegen.

Vad som är möjligt att göra givet dagens struktur begränsas av flera faktorer. I detta kapitel analyserar vi de, ur vårt perspektiv, allvarligaste begränsningarna. Vi börjar med att analysera hur beräkningstung en ny länkmodell får vara varefter vi fortsätter med frågan hur data kan aggregeras för att avgöra om en sändning har lyckats.

## 3.1 Simuleringstid

En begränsande faktor för en ny länkmodell är simuleringstiden. För att modellen ska vara användbar får inte simuleringstiden för ett scenario i OPNET öka allt för mycket. Att sätta ett absolut tak är dock svårt. Simuleringstiden för ett scenario är ungefär proportionell mot antalet paket vi sänder samt antalet noder, som blir involverade i en sändning. Om en ny länkmodell ökar beräkningstiden för en paketsändning med en faktor 2 så kommer således den totala simuleringstiden också öka med ungefär en faktor 2.

Tester med dagens länkmodell tyder på att 1000 additioner ökar simuleringstiden med drygt $25\%$. Skulle en ny länkmodell kräva 10000 additioner ökar simuleringstiden med över $260\%$ vilket kan vara lite mycket. Ett rimligt designmål torde vara ca 7000 additioner vilket gör att simuleringstiden ökar med mindre än $200\%$, ser tabell 3.1. I praktiken krävs givetvis även andra ope-

| #+ | $t[s]$ | $\Delta\%$ |
|---:|---:|---:|
| 0 | 50 | 0 |
| 10 | 50 | 0 |
| 100 | 51 | 2 |
| 1000 | 63 | 26 |
| 10000 | 183 | 266 |

Tabell 3.1: Kostnaden för olika antal additioner i ett 10 nods nät. Testerna är gjorda med icke optimerad kod.

| funk. | rel. tid |
|---|---|
| $+$ | 1.0 |
| $-$ | 1.0 |
| $*$ | 1.1 |
| $/$ | 2.1 |
| $\sin$ | 5.9 |
| $\arcsin$ | 8.8 |
| $\exp$ | 3.5 |
| $\ln$ | 4.7 |
| rand[1] | 9.9 |
| rand[2] | 22.1 |

Tabell 3.2: Kostnaden för olika matematiska operationer relativt $+$ operationen. Testerna är gjorda med icke optimerad kod där rand[1] är OPNET:s egna slumptalsgenerator [1] och rand[2] Boost:s implementation av Mersenne Twister [3].

rationer än +. För att få en uppfattning av kostnaden för dessa kan tabell 3.2 användas.

## 3.2 Dataaggregering

De paket som OPNET sänder mellan olika noder kan innehåller flera olika informationstyper. Dels innehåller alla paket information för loggning och debuggning som tex. tidsstämplar, dels innehåller pakten olika typer av användardata som tex. adresser och nyttoinformation. Det finns två olika sätt att lagra/hämta användardata i ett paket. Dels kan användaren definiera förformaterade paket som innehåller namngivna fält för olika datatyper. Dessa fält kan sedan användas för att lagra/hämta data genom att fältets namn anges. Dels innehåller alla paket, både förformaterade och oformaterade, ett indexbaserat system som kan lagra heltal, flyttal och pekare. På länknivå används normalt det senare systemet då paket av godtyckligt format därigenom kan hanteras.

Det högsta index som OPNET:s radiomodell använder för att lagra data i det indexbaserade systemet definieras via makrot `OPC_TDA_RA_MAX_INDEX`. Vill vi lagra ett värde i indexdelen av ett paket definierar vi lämpligen först ett nytt makro enligt

```
#define NEW_INDEX (OPC_TDA_RA_MAX_INDEX + 1)
```

För att läsa, skriva samt undersöka existensen av data finns ett antal funktioner. I samtliga fall antas `pkptr` vara en pekare till paketet, `index` datans index och

`value` värdet hos datan vid skrivning. För att avgöra om ett index är definierat används

```
bool    op_td_is_set(Packet* pkptr,
                     int index)
```

För att läsa ut data av typerna `int`, `double` och `class T*` ur ett paket används

```
int     op_td_get_int(Packet* pkptr,
                      int index)

double  op_td_get_dbl(Packet* pkptr,
                      int index)

void*   op_td_get_ptr(Packet* pkptr,
                      int index)
```

där hantering av `class T*` kräver en `dynamic_cast<T*>` med efterföljde test för att vara säker. För att skriva data av typerna `int`, `double` och `class T*` till ett paket används

```
void op_td_set_int(Packet* pkptr,
                   int index,
                   int value)

void op_td_set_dbl(Packet* pkptr,
                   int index,
                   double value)

void op_td_set_ptr(Packet* pkptr,
                   int index,
                   T* value)
```

OPNET:s minneshantering saknar idag automatisk destruktion av minne som inte används längre. De pekare som sparas i ett paket måste därför destrueras av den som sparat dem. Vidare kopieras bara pekarens adress och inte innehållet i objektet den pekar på om OPNET skapar en kopia av paketet. Det är således förknippat med vissa risker och problem att spara mera komplexa objekt än objekt av typerna `int`, `double`. I OPNET:s nuvarande mottagarmodell kopieras dock inte paketen utan det är samma paket alla beräkningar sker på rent minnesmässigt även om paketet beräkningsmässigt delas upp i flera segment. Vill vi använda möjligheten att spara objekt i paketen är det således främst destrueringen av dessa som kan vara ett problem.

I den nuvarande mottagarmodellen används enbart möjligheten att spara data av typerna `int`, `double`. Detta fungerar bra eftersom en ny SNR-beräkning sker vid starten av varje nytt segment varefter antalet bitfel beräknas och adderas till de tidigare i slutet av varje segment. Förutom de minnesfält som aggregerar bitfelen kan minnesfälten som används i mottagarmodellen för att beräkna kvaliteten hos paketet därigenom återutnyttjas för konsekutiva segment.

Inom projektet ”Vågutbredning för bredbandiga vågformer” har flera olika länkmodellkoncept diskuterats. I flera av dessa finns behov av att kunna beräkna SNR för andra segmentstorlekar än de som uppkommer på grund av interferens från andra paket. Diverse testimplementationer har dock visat att det inte är något principiellt problem att beräkna SNR över hela paket eller en godtycklig symbolstorlek. Införandet av nya symbolstorlekar på länknivå kan dock resultera i att befintliga protokoll som 802.11 får svårt att fungera över dessa länkar utan modifieringar eftersom ett befintligt protokoll har förutsatt en viss symbolstorlek när paketformaten definierats.

# 4 Programdesign

För att testa de i detta projekt designade länkmodellerna i OPNET har vi gjort en implementation av två modeller. Dels en ren länkdämpningsbaserad modell och dels en mer avancerad modell som baseras på kanalkapacitetsberäkningar. Vid framtagandet av modellerna har vi använt samma övergripande programdesign.

I detta kapitel ger vi en övergripande beskrivning av programmet med fokus på de designval vi gjort. Avslutningsvis ger vi också en kort beskrivning av hur programfilerna är organiserade. Vi börjar dock med en principiell beskrivning av vilka pipeline-steg vi har ändrat samt vad dessa nya steg gör.

## 4.1 Nya pipeline-steg

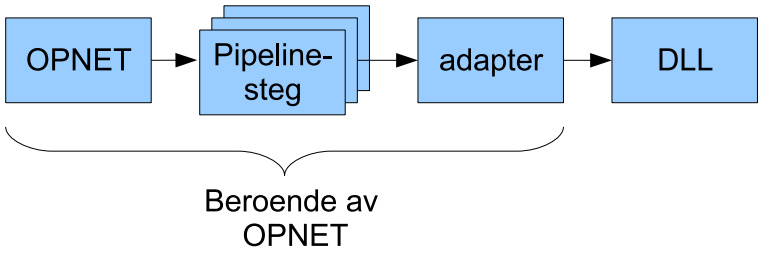
För den länkdämpningsbaserade modellen har vi bara ersatt det gamla länkdämpnings-pipeline-steget, `closure model`, med ett nytt som beräknar länkdämpningen. För övrigt bygger modellen på att OPNET:s radiomodell fungerar som vanligt.

För den mer avancerade modellen har vi, förutom länkdämpningssteget, även ersatt stegen som hanterar mottagen effekt, signal till brusförhållande, bitfelshalt, bitfelsallokering och felrättning. I denna modellen har vi flyttat beräkningen av länkdämpningen till pipeline-steget som beräknar den mottagna effekten. I pipeline-steget som tidigare beräknade signal till brusförhållande sätts här enbart en tidsstämpel så att vi vet när ett visst segment börjar. I pipelinesteget som tidigare beräknade bitfelshalten sker en aggregering av SNR hos paketsegmentet så att vi får tillgång till ett medel SNR för hela paketet. Det pipeline-steg som tidigare hanterade bitfelsallokering är helt tomt och finns bara för att OPNET:s länkmodell har en viss given struktur. I steget som tidigare hanterade felrättning sker en kanalkapacitetsberäkning med medel SNR och aktuella kanalegenskaper som inparametrar. Denna kapacitetsberäkning används sedan för att avgöra om ett paket kan tas emot.

## 4.2 Struktur

Styrande för programdesignen har varit att implementationen, så långt som möjligt, ska vara testbar med enhetstester[1] så att kvaliteten hos koden kan hanteras på ett systematiskt sätt. Vidare har vi velat att programmet ska ha en

[1] Tester där varje i programmet ingående klass och funktion testas var för sig.



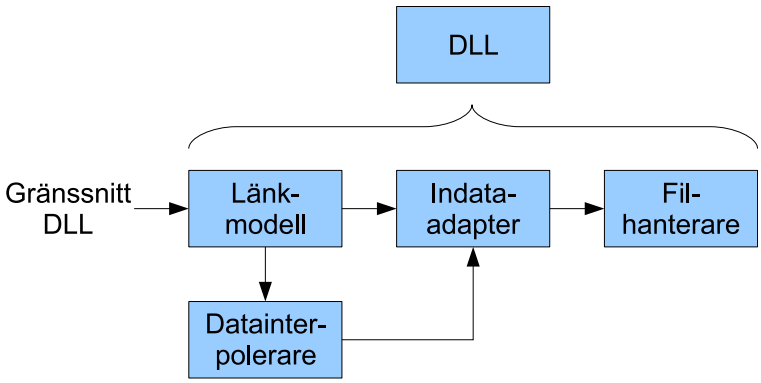

Figur 4.1: Övergripande struktur hos de programdelar som är beroende av OPNET. De av OPNET oberoende programdelarna är förpackade i en DLL-fil.

struktur som underlättar framtida återanvändning av koden. Den övergripande strukturen är därför samma för de båda implementerade modellerna.

Eftersom OPNET inte erbjuder något stöd för enhetstester har vi här valt att använda det ramverk för enhetstester som tagits fram för simuleringsmiljön Aquarius. De implementerade länkmodellerna bygger alla på att förbearbetad data läses upp och bearbetas ytterligare i pipeline-stegen. Viktigt att notera här är att ett pipeline-steg i OPNET är en funktion. För att kunna lagra uppläst data mellan olika anrop av pipeline-stegen krävs därför att extra datastrukturer tillförs. Huvuddelen av koden ligger därför inte i själva pipeline-stegen utan i diverse objekt som anropas från pipeline-stegen. Detta har i sin tur lett till att implementationen har gjorts i C++ med stöd av biblioteket Boost till skillnad mot de av OPNET implementerade pipeline-stegen som är skrivna i ren C-kod.

Implementationen består i princip av tre delar, en del som kapslar in alla kopplingar till OPNET, en del som hanterar inläsning och mellanlagring av länkdata samt en del som hanterar själva beräkningarna. De två senare delarna, vilka innehåller huvuddelen av koden, är implementerade i form av ett extern projekt som länkas in till OPNET med hjälp en DLL-fil. Då vissa deklarationer i OPNET inte fungerar med programbiblioteket Boost [3] som används av både vårt ramverk för enhetstester och den nyutvecklade koden har de OPNET-beroende delarna av koden inte testats med enhetstester.

Den del som kapslar in OPNET-beroendet består av de nya pipeline-stegen samt ett static allokerat adapterobjekt som är gemensamt för alla pipeline-steg. Pipeline-stegen innehåller i sig ytterst lite kod utan anropar en gemensam adapterklass. Adapterklassen har som främsta uppgift att mappa om de identiteter som OPNET använder internt till ett mera användarvänligt system. Själva ommappningen bygger på att de inblandade nodernas `*.trj`-filer är numrerade så att numret i sändarens och mottagarens `*.trj`-fil motsvarar index hos den kanaldata som används. Efter ommappningen anropas den OPNET oberoende koden i DLL-filen som sköter själva beräkningarna, se figur 4.1.




Figur 4.2: Övergripande struktur hos de OPNET-oberoende programdelar. Samtliga OPNET-oberoende programdelar är förpackade i en DLL-fil.

Länkmodellklassen, där anrop från den OPNET-beroende adaptern till DLL-filen hamnar, hanterar alla mer länk specifika beräkningar. I fallet med den länkdämpningsbaserade modellen är klassen relativt enkel, men i fallet med den kanalkapacitetsbaserade modellen utnyttjar länkmodellklassen ett antal hjälpklasser. Dessa hjälpklasser används för att kontrollera tillståndet hos radiokanalerna mellan olika sändar- och mottagarpar.

Den indata som länkmodellen baserar sina beräkningar på antas bestå av tidssampel av olika storheter samplade med ett fixt tidsintervall. Om länkmodellen efterfrågar data mellan samplingspunkterna antas att vi genom interpolation kan få fram efterfrågad data. Funktionaliteten för att interpolera data ligger i en hjälpklass som binds till indataadaptern för sin rådataförsörjning, se figur 4.2.

Inläsning och mellanlagring av indata är uppdelade i två huvudklasser, en filhanterarklass som läser indata från fil och en indata-adapterklass som transformerar inläst data till rätt indataformat. Dessa båda klasser kommunicerar via ett standardiserat gränssnitt. Genom denna konstruktion uppnås en hög grad av frikoppling mellan det använda filformatet och det indataformat som en viss länkmodell efterfrågar, se figur 4.2.

## 4.3 Kodorganisation

Källkoden till de implementerade länkmodellerna är organiserad i ett antal olika kataloger. En lista över dessa samt en kort beskrivning av innehållet i de olika katalogerna ges här.

`dll_test`:
: Innehåller ett testprojekt för att testköra den DLL-fil som innehåller den OPNET-oberoende koden. Solution filen ligger i denna katalog så projektet ska öppnas i från denna katalog.

`foi_opnet_link_model`:
: Innehåller den del av länkmodellerna som är oberoende av OPNET.

  `c3d`: All kod som som är specifik för den mer avancerade länkmodellen förutom den OPNET-beroende koden.

  `channel_gain`:
  : All kod som som är specifik för den mer länkmodellen som enbart sätter länkdämpningen förutom den OPNET-beroende koden..

  `com_def`:
  : Innehåller endel grundläggande klassdefinitioner så som bland annat slumptal, undantagshantering, singleton och fabrik objekt.

  `cpp_test`:
  : Innehåller det använda biblioteket för att köra enhetstester.

  `file_parser`:
  : Innehåller all kod som är gemensam för olika länkmodeller rörande filhantering, indata-adaptering och data-interpolering.

`matlab`:
: Här ligger diverse Matlab-kod som behövs för att konvertera ett scenario till det filformat som länkmodellerna använder.

`op_models`:
: Här ligger all kod som är beroende av OPNET.

# 5 Användarguide

För att underlätta användandet av de två implementerade länkmodellerna har vi i detta kapitel samlat information om hur indata till modellerna skapas, hur koden till projektet kan kompileras samt vilka inställningar som behövs i OPNET.

## 5.1 Skapa indata

Den nuvarande modellen behöver kanaldata för radiokanalen och koordinater till de noder som utnyttjar modellen. Som indataformat för koordinaterna används OPNET:s `trj`-fil format, se [1]. Den nuvarande implementationen kan hantera ett indataformat för kanaldatan. Designen tillåter dock att nya informat enkelt kan läggas till. Det nuvarande filformatet har ändelsen `.lad` och har följande syntax

```
version = |versionsnummer|
size = <|antal rader med data|, ...
        |antal kolumner med data|, ...
        |antal element i en tupel|>
deltaT = |tiden mellan två sampel [s]|
{--} {--}
{--} {--}
{--} {--}
```

där `"..."` har använts för radbrytningar som inte ska vara med i en `.lad` fil utan bara finns med i exemplet. Versionsnumret kan antingen vara 1 eller 2 i den nuvarande implementationen. Version 1 motsvarar data till den länkdämpningsbaserade länkmodellen medan version 2 motsvarar data till den kapacitetsbaserade länkmodellen. Antal rader och antal kolumner ska motsvara yttre strukturen hos de datatupler `{--}` som finns filen. Antalet element i en tupel bestämmer den inre strukturen i datatuplerna. Den tredje raden i indatafilen anger tiden mellan två sampel mätt i sekunder.

Används version 1 antas varje tupel bestå av ett länkdämpningsvärde. Används version 2 antas varje tupel bestå av fyra värden enligt

```
{ |länkdämpning| ...
  |koherensbandbredd| ...
  |k-faktor| ...
  |korrelationsfaktor| }
```

Antar vi vidare att datan i tuplerna ordnas i matriser med en matris per sampel enligt

$$\begin{pmatrix} v_{11}(1) & v_{12}(1) & v_{13}(1) \\ v_{21}(1) & v_{22}(1) & v_{23}(1) \\ v_{31}(1) & v_{32}(1) & v_{33}(1) \end{pmatrix}, \begin{pmatrix} v_{11}(2) & v_{12}(2) & v_{13}(2) \\ v_{21}(2) & v_{22}(2) & v_{23}(2) \\ v_{31}(2) & v_{32}(2) & v_{33}(2) \end{pmatrix}$$

så består en `lad`-fil av övre triangel i dessa matriser sparade enligt

$$\begin{pmatrix} v_{12}(1) & v_{12}(2) \\ v_{13}(1) & v_{13}(2) \\ v_{23}(1) & v_{23}(2) \end{pmatrix}$$

dvs vi antar att kanalen är reciprok.

## 5.2 Bygg projektet

För att bygga projektet och skapa den DLL-fil som innehåller huvuddelen av koden behövs Microsoft Visual Studio 2008 samt Boost 1.38.0. Vidare måste sökvägen till Boost sättas upp genom att gå till `Tools` -> `Options...` -> `Projects and Solutions` -> `VC++Directories`. Under menyn `Show directories for:` ändrar vi till `Include files` varefter vi kan addera sökvägen till boost. Observeras här bör att filer i Boost inkluderas enligt `#include <boost/*.hpp>`.

Efter att Microsoft Visual Studio 2008 har konfigurerats är det bara att öppna Visual Studio projektet `dll_test.proj` i katalogen `dll_test` och bygga `Release` versionen. Vid byggnationen skapas filerna `foi_opnet_link_model.dll` och `foi_opnet_link_model.lib` vilka innehåller huvuddelen av länkmodellen.

## 5.3 Konfigurera OPNET

För att OPNET ska kunna använda länkmodellerna krävs en del konfigurering i OPNET. Länkmodellerna är bara testade tillsammans med OPNET:s WLAN modell. Om de ska användas i andra sammanhangen kan det krävas ytterligare anpassning av koden. Givet att sceneriet är baserat på Wlan-modeller behöver följande göras:

- Lägg till en extra inparameter "'link data file name"'

  - Gå upp en nivå i hierarkin så att subnätet som innehåller noderna visas (`Ctrl-0`)
  - Öppna attributlistan för subnätet (`Edit Attributes`)
  - Editera alla attribut genom att kryssa för `Advanced`
  - Välj `Extend Attrs`.
  - Välj `Add` och lägg till ett attribut med namnet "'link data file name"' av typen `string`.
  - Stäng Extend Attributes och sätt de nya attributet till den första filen som innehåller aktuell länkdata.

- Sätt `trajectory` filer för alla noder i länkdämpningsmatrisen.
- Byt ut pipeline-stegen enligt:

| Attribut namn | Länkdämpning | Kapacitet |
|---|---|---|
| `closure model` | `channel_gain_closure` | `c3d_closure` |
| `power model` | – | `c3d_power` |
| `snr model` | – | `c3d_snr` |
| `ber model` | – | `c3d_ber` |
| `error model` | – | `c3d_ecc` |
| `ecc model` | – | `c3d_ecc` |

  Modellen sparas sedan lämpligen med ett nytt namn då vi annars kommer att göra en ändring i OPNET:s standardmodellbiblioteket.

- Lägg till katalogen `op_models` till OPNET:s modellbibliotek
- Kryssa för de externa filerna `c3d_radio_system_adapter.ex.cpp` och `channel_gain_adapter.ex.cpp` i `File -> Declared External Files...`
- Lägg till till katalogerna `foi_opnet_link_model` och `boost` till `comp_flags_common` (eller systemvariablen `include`).
- Lägg till `foi_opnet_link_model.lib` till `bind_shobj_libs` i `Preference` (`Ctrl+Alt+P`)
- Lägg slutligen till den katalog som innehåller `Release` versionen av `foi_opnet_link_model.dll` och `foi_opnet_link_model.lib` till systemvariablerna `lib` och `PATH`.

# Referenser

[1] www.opnet.com.

[2] Gunnar Eriksson. Länkmodeller för nätsimuleringar med karaktäristisk kanalinformation. Teknisk Rapport FOI-R–2758–SE, FOI Totalförsvarets forskningsinstitut, Avdelningen för Informatonssystem, Linköping, Sweden, Maj 2009.

[3] www.boost.org.