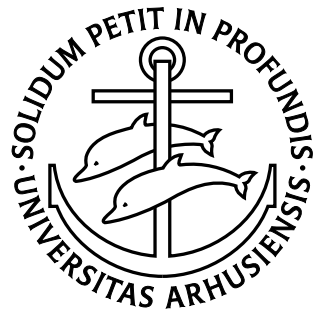
CPN-modellering og analyse af routing i mobile ad hoc-netværk

Niels O. Jensen

Speciale, maj 2006

Årskortnr. 19920533
Email: noj@noj.dk
Vejleder: Jens Bæk Jørgensen



Datalogisk Institut
Aarhus Universitet
Danmark

Resumé

Dette speciale undersøger mobile ad hoc-netværk betegnet MANET'er. Jeg beskriver MANET-protokoller generelt og fokuserer herefter på MANET-protokollen Dynamic Source Routing. Jeg beskriver og modellerer denne og forskellige, valgfrie optimeringer hertil – f.eks. knuders brug af deres caching af netværkstopologidata – i Coloured Petri Nets. Jeg beskriver problemområder i den officielle specifikation af protokollen, og jeg kommer med forslag til, eller giver retningslinier for, løsninger for disse. Desuden laver jeg et forslag til en ekstra optimering af protokollen. Herefter undersøger jeg to af de modellerede optimeringers indvirkning på protokollens effektivitet bl.a. målt i pakkefremkomst-procentdel og antal transmissioner i netværket under varierende forhold.

Abstract

This thesis examines mobile ad hoc networks known as MANETs. I describe MANET protocols in general and subsequently focus on the MANET protocol Dynamic Source Routing. I describe and model this and various optional optimisations to it – e.g. nodes' use of cached network topology data – in Coloured Petri Nets. I describe problem areas in the official specification of the protocol and I make proposals or give guidelines to solutions for these. Furthermore, I suggest an extra optimisation of the protocol. After this I examine how two of the modelled optimisations influence the efficiency of the protocol among others measured in packet delivery ratio and the number of transmissions in the network under varying circumstances.

Indhold

Vigtigste figurer og tabeller

Vigtigste forkortelser

AODV: MANET-protokollen "Ad Hoc On-Demand Distance Vector Routing", se afsnit 1.3.3
Bidir-only: Netværk, hvor alle links er bidirektionelle (d.v.s. symmetriske), se afsnit 2.2.5
CPN: Modelleringssproget "Coloured Petri Nets", se afsnit 3.1
CRR: DSR-udvidelsen "Cached Route Reply", se afsnit 2.3.6
CTS: "Clear To Send"-pakke, se afsnit 1.5.3
DHCP: IP-adresse-tildelingsprotokollen "Dynamic Host Configuration Protocol", se afsnit 1.6.3
DSDV: MANET-protokollen "Destination-Sequenced Distance Vector", se afsnit 1.3.3
DSR: MANET-protokollen "Dynamic Source Routing", se afsnit 1.3.3 og kapitel 2
Fast: Model-setup, hvor knuder bevæger sig hurtigt, se afsnit 5.1.3
FIFO: "First In First Out" – lister, der behandles som køer
Frequently-unidir: Netværk, hvor de fleste links er unidirektionelle (d.v.s. ikke-symmetriske), se afsnit 2.2.5
IEEE: "Institute of Electrical & Electronic Engineers"
IETF: "Internet Engineering Task Force"
IP: "Internet Protocol", se afsnit 1.5.1
LIFO: "Last In First Out" – lister, der behandles som stakke
MAC: "Media Access Control"-protokol, se afsnit 1.5.3
MANET: "Mobile ad hoc network", se afsnit 1.2
MB: DSR-datastrukturen "Maintenance Buffer", se afsnit 2.4.5
Medium: Model-setup, hvor knuder bevæger sig med medium hastighed, se afsnit 5.1.3
Mini-SO: Den foreslåede DSR-udvidelse "Mini-Salvage Operations", se afsnit 3.5.5
ML: Det funktionelle programmeringssprog "Metalanguage", se afsnit 3.1
Mostly-bidir: Netværk, hvor de fleste links er bidirektionelle (d.v.s. symmetriske), se afsnit 2.2.5
NIQ: DSR-datastrukturen "Network Interface Queue", se afsnit 2.4.5
ns-2: Modelleringsværktøjet "The Network Simulator", se afsnit 3.1
OLSR: MANET-protokollen "Optimized Link State Routing Protocol", se afsnit 1.3.3
OSI: "Open Systems Interconnect", se afsnit 1.5.1
RC: DSR-datastrukturen "Route Cache", se afsnit 2.2.4
RD: "Route Discovery", se afsnit 2.3
RecvSide: Modtagersiden af DSR-modellen, se kapitel 3
RM: "Route Maintenance", se afsnit 2.4
RRTE og RRTI: DSR-datastrukturerne "Route Request Table" for henholdsvist eksternt og internt skabte pakker, se afsnit 2.3.5
RTS: "Request To Send"-pakke, se afsnit 1.5.3
SB: DSR-datastrukturen "Send Buffer", se afsnit 2.3.5
SendSide: Afsendersiden af DSR-protokollen, se kapitel 3
Slow: Model-setup, hvor knuder bevæger sig langsomt, se afsnit 5.1.3
SO: DSR-udvidelsen "Salvage Operations", se afsnit 2.4.6
TCP: "Transmission Control Protocol", se afsnit 1.5.1
TTL: "Time To Live"
WPA: "Wi-Fi Protected Access", se afsnit 1.6.4

Indledning

Dette speciale beskæftiger sig med såkaldte MANET-protokoller. MANET står for "mobile ad hoc-netværk". En MANET-protokol er en netværksprotokol, der er i stand til at oprette et netværk mellem mobile netværksdeltagere, hvor og når der er brug for det ("ad hoc") – uden brug af allerede eksisterende infrastruktur, men alene ved hjælp af en trådløs teknologi indbygget i de enkelte netværksdeltagere selv.

En MANET-protokol forudsætter ikke, at alle deltagere har en så kraftig trådløs teknologi, at alle deltagere altid vil være inden for rækkevidde af hinanden. Dette betyder, at hvis en netværksdeltager vil sende netværkspakker til en anden i et MANET, og de to deltagere er uden for hinandens rækkevidde, skal en eller flere netværksdeltagere imellem de to videresende disse pakker fra afsenderen til modtageren.

En MANET-protokol skal være selv-organiserende. Dette betyder bl.a., at den automatisk selv skal kunne finde disse ruter mellem netværksdeltagere. Den skal også kunne opdage, hvis en rute ikke længere fungerer – f.eks. fordi en af deltagerne i ruten har bevæget sig væk fra de øvrige – og kunne finde en ny rute mellem de to deltagere. Kommer der nye netværksdeltagere til i MANET-området, skal disse automatisk kunne inkluderes i netværket.

Mål med specialet

I de senere år er adskillige forslag til MANET-protokoller blevet præsenteret, men ingen af disse er endnu hverken blevet valgt som en standard inden for området eller blevet brugt i en produktionsklar implementation (jvf. oversigterne [IETa, IETb, IEE, PRG], se afsnit 1.4). Det er derfor interessant at undersøge, i hvor høj grad de præsenterede MANET-protokoller er klar til at kunne blive benyttet i disse to sammenhænge. Specielt kan man undersøge protokollerne på to områder:

Specifikationskvalitet: Mange protokoller har som grundlag en specifikation, der beskriver protokollen i klartekst. En sådan specifikation kan derfor indeholde flertydigheder, og der kan være specialtilfælde, der ikke bliver dækket af den.

En måde, hvorved man kan undersøge, hvor høj kvaliteten af en protokolbeskrivelse er, er at bygge en model over den og herigennem gennemgå protokollen minutiøst. Undervejs i dette arbejde kan man støde på problemområder i protokolspecifikationen – man kan f.eks. finde specialtilfælde, hvor der ikke er specificeret, hvad protokollen skal gøre, eller tilfælde, hvor den opførsel, protokollen dikterer, at netværksdeltagerne skal følge, gør, at noget uhensigtsmæssigt i netværket forekommer – f.eks. at visse netværkspakker vil blive sendt rundt i en ring i et netværk uden nogensinde at nå frem.

Derudover kan problemområder også identificeres ved at foretage en såkaldt state space-analyse på modellen. Dette er desværre kun en teoretisk mulighed, hvilket jeg vender tilbage til i det nedenstående.

Valgfrie optimeringer: I specifikationen af MANET-protokoller angives ofte en række områder, hvor en implementation eller model af protokollen kan udformes på flere forskellige måder. Dette kan f.eks. dreje sig om hvilken strategi, der skal benyttes, når en

rute til en netværksdeltager skal udvælges blandt adskillige, eller det kan være valgfrie optimeringer af protokollen, der kan inkluderes eller ej.

Under dette punkt kan man både undersøge, om visse strategier er klart bedre end andre, og om de valgfrie optimeringer bør tages med i en endelig implementation eller ej – altså om man kan se, at medtagelsen af en optimering entydigt vil forbedre eller forværre protokollens effektivitet i forhold til en eller flere parametre. En sådan parameter kan f.eks. være hvor mange pakker, der når frem af dem, der bliver forsøgt routet i netværket.

Målet med dette speciale er at undersøge MANET-protokoller nærmere m.h.t. disse to områder.

Metode og opnåede resultater

For at kunne gå mere i dybden med de to punkter har jeg valgt at fokusere på en enkelt MANET-protokol.

For at få en baggrund for at kunne udvælge en sådan starter jeg specialet med i **kapitel 1** at give en nærmere beskrivelse af, hvordan en MANET-protokol mere præcist defineres, hvilke specielle forhold, der gælder for mobile ad hoc-netværk, og hvilke forskellige hovedtyper af MANET-protokoller, der findes.

Forslag til mange forskellige MANET-protokoller er lavet i løbet af de sidste år. I afsnit 1.3.4 argumenterer jeg for, at ud af disse vil det være fornuftigt at undersøge specielt protokollen DSR ("Dynamic Source Routing", beskrevet i bl.a. [JM96, JMB01, JMH05]) nærmere. Dette gør jeg hovedsageligt ud fra det faktum, at den har klaret sig godt i en række hidtidige eksperimenter med modellerede versioner af MANET-protokoller ([BMJ+98, CK04]), og derfor kan have en reel chance for at blive valgt som en standard inden for området.

Specifikationskvalitet

DSR-specifikationen [JMH05] nævner selv to mangler ved protokollen: Der tages ikke højde for hverken sikkerhed m.h.t. ondsindede netværksdeltagere eller for, hvordan IP-adresser skal tildeles i DSR-netværk. Begge disse punkter er generelle problemer for MANET-protokoller, og behandles nærmere i afsnit 1.6 i **kapitel 1**. Her beskriver jeg, at sikkerhedsproblematikken på et tidspunkt højst sandsynligt vil kunne løses med en generel udvidelse til en vilkårlig MANET-protokol (omend en sådan ikke er færdigudviklet og testet endnu), men at det stadig er et åbent spørgsmål, om IP-adresser kan uddeles dynamisk i et MANET.

I **kapitel 2** gennemgås DSR-protokollen selv. Gennemgangen er delt op i de tre logiske dele, der findes i protokollen: Hvordan pakker sendes fra en netværksdeltager til en anden via en rute i protokollen (afsnit 2.2), hvordan disse ruter findes (kaldet "Route Discovery" eller RD, afsnit 2.3), og endelig hvordan protokollen opdager, at fundne ruter ikke længere fungerer (kaldet "Route Maintenance" eller RM, afsnit 2.4).

På flere områder argumenterer hverken specifikationen [JMH05] eller den tilhørende litteratur [JM96, JMB01] for, hvorfor protokollen skal opføre sig, som referencerne siger, den skal i visse specialtilfælde. Her sørger jeg for at komme med bud på argumenter for disse valg, og nævner også ulemper ved valgene, hvis der er sådanne.

For at kunne undersøge DSR i større detaljegrad, benyttes gennemgangen i kapitel 2 til at til at lave en model af DSR i **kapitel 3**. Modellen bliver lavet i CPN ("Coloured Petri Nets", [Jen92, Jen94]), et grafisk modelleringssprog beskrevet i afsnit 3.1, og følger den samme opdeling som i kapitel 2 m.h.t. pakkeforsendelse (afsnit 3.3), RD (afsnit 3.4) og RM (afsnit 3.5).

Undervejs i gennemgangen af opbygningen af modellen identificerer jeg en række punkter, hvor det har været nødvendigt at bygge modellen anderledes, end specifikationen [JMH05] dikterer. Jeg identificerer tre problemområder i RD og to i RM. De fundne problemområder svinger i alvorsgrad:

- I afsnit 3.4.6 vises et specialtilfælde, der er udeladt i specifikationen, og hvor den fremgangsmåde, specifikationen generelt dikterer inden for området, kan lede til en deadlock i fremsendelsen af pakker. Dette kan kun afhjælpes, hvis man enten bryder med de generelle retningslinier beskrevet i specifikationen eller udvider datastrukturerne i forhold til den opbygning, specifikationen beskriver.
- I afsnit 3.5.4 vises et punkt, hvor den dikterede fremgangsmåde ikke kan lade sig gøre med den opbygning af datastrukturer, specifikationen beskriver.
- I afsnit 3.5.5 vises et specialtilfælde, specifikationen ikke behandler, og hvor udeladelse af en speciel fremgangsmåde i dette specialtilfælde vil få protokollen til at virke suboptimalt. Fremgangsmåden, der er et forslag til en optimering af protokollen, vises i detaljer.
- I afsnit 3.4.5 vises et specialtilfælde, hvor ruter, der ikke helt fungerer, kan blive fundet i en RD, selvom protokollen bevidst forsøger at undgå disse. Jeg viser samtidigt, hvordan problemet kan løses.
- I afsnit 3.4.4 vises en detalje, der ikke er specificeret ordentligt, og som folk, der laver en implementation af protokollen, derfor kan risikere at få besvær med at få til at fungere, hvis de ikke selv udleder den rette fremgangsmåde. Jeg viser samtidigt den rette fremgangsmåde.

Det falder uden for rammerne af dette speciale at finde færdige løsninger på alle de her identificerede punkter, men der gives som minimum retningslinier til hvordan, en løsning kan udformes. I visse af tilfældene er et decideret forslag til en løsning alligevel inkluderet i modellen – opdelingen i problemområder med og uden løsning er til dels sket automatisk, idet modellen ikke ville kunne bringes til at fungere ordentligt, hvis der ikke blev modelleret en eller anden form for løsning i de specialtilfælde, der er beskrevet – enten med de løsninger, der er foreslået, eller med en alternativ løsning.

En stor force ved brugen af CPN-modeller er, at der findes værktøjer, der er i stand til at beregne en state space-graf for dem. En sådan graf kan f.eks. benyttes til at finde deadlocks i modellen, hvilket automatisk vil kunne identificere yderligere problemområder i protokollen. Imidlertid fungerer dette ikke for modeller, der ikke er trivielt små. I **kapitel 4** redegør jeg for, at CPN-modellen, der bliver præsenteret i kapitel 3, er for stor til at kunne blive brugt i en state space-beregning.

Valgfrie optimeringer

Til opfyldelse af punktet "Valgfrie optimeringer" beskriver jeg i gennemgangen af DSR i **kapitel 2** en række mulige optimeringer/udvidelser af DSR. I DSR-modellen i **kapitel 3** modellerer jeg to af disse udvidelser: En RD-udvidelse kaldet "Cached Route Reply" (CRR, som tillader netværksdeltagere at svare på ruteforspørgsler på vegne af andre netværksdeltagere) og en RM-udvidelse kaldet "Salvage Operations" (SO, som tillader netværksdeltagere at forsøge at videresende netværkspakker via en ny rute i stedet for blot at smide pakken ud, hvis den rute, der skal benyttes, viser sig at være blevet invalideret).

CPN-modellen af DSR benyttes i **kapitel 5** til at gennemføre en række eksperimenter. Jeg undersøger bl.a., hvordan DSR-modellens effektivitet påvirkes, når man slår de to forskellige valgfrie udvidelser til eller fra. "Effektivitet" bliver her målt på flere forskellige måder, f.eks. som antal pakker, der når frem, og som antal brugte pakketransmissioner pr. afsendt pakke.

Modellen kan konfigureres på en række forskellige måder. En fuldstændig analyse vil inkludere eksperimenter under alle kombinationer af konfigurationsparameterværdierne, men dette ville tage længere tid at udføre, end der var til rådighed til eksperimenterne. Jeg udvælger derfor en enkelt parameter, der kan varieres på. I afsnit 5.1 finder jeg frem til, at det vil have størst effekt at variere på netværksdeltagernes hastighed i modellen.

En række eksperimenter bliver kørt under hver konfigurering, og statistiske analyser bliver brugt på måledata herfra til at uddrage konklusioner om effektiviteten af optimeringerne. En oversigt over de benyttede statistiske analyser kan ses i appendix G

Herudfra finder jeg, at hvis man benytter RD-udvidelsen CRR, forbedres effektiviteten af DSR-modellen – men kun, når de modellerede netværksdeltagerne ikke bevæger sig *meget* hurtigt rundt mellem hinanden.

Det lykkes til gengæld ikke at påvise en effekt af RM-udvidelsen SO uanset netværksdeltagernes modellerede hastighed. Dette kan enten skyldes, at mine benyttede eksperimenter er for korte eller for få til at situationerne, hvor SO kommer i brug, optræder ofte nok til, at det skinner igennem i statistikken, eller det kan skyldes, at SO slet ikke har en effekt. Endelig kan det også skyldes, at den optimering, der beskrives i afsnit 3.5.5, påvirker resultatet af modellen af SO negativt. Om SO skal benyttes i en implementation, vil derfor kræve yderligere analyser.

Forudsætninger, downloads og tak

Læseren af dette speciale forventes at have et basalt kendskab til distribuerede systemer. Et basalt kendskab til modellering i Coloured Petri Net vil også være anbefalelsesværdigt, men er ikke strengt nødvendigt. Derimod kræves intet kendskab til trådløs kommunikation, til mobile ad hoc-netværk og problemstillinger heri eller til Dynamic Source Routing-protokollen på forhånd.

CPN-modellen, der gennemgås i kapitel 3, kan downloades fra `http://www.noj.dk/thesis/`. Minimum version 2.0.0 af CPN Tools kræves for at kunne benytte modellen.

Jeg vil gerne sende en stor tak til Kari Brandt, Peter Riishøj Brinkler, Trine Kornum Christiansen, Sebastian Adorján Dyhr, Hanne Gottliebsen, Søren Louring, Sophie Mikkelsen, Thea Køllgaard Olsen, Marie Louise Skram Pedersen, Jesper Thomsen og Thomas Widmann for udlån af litteratur, udlån af processorkraft til eksperimentkørslerne, LaTeXnisk hjælp og korrekturlæsning.

Desuden er jeg Peter Rickers, Kim Oechsle Hansen, Kåre Fiedler Christiansen og Jens Bæk Jørgensen en meget stor tak skyldig for mange gode råd, idéer og diskussioner under specialeskrivningen.

Specialet består af i alt 131 sider ekskl. indledning, konklusion og appendix – heraf ca. 31 sider figurer og tabeller.

1

Introduktion til mobile ad hoc-netværk

I dette kapitel vil jeg give en introduktion til en ny type netværksprotokoller, det er blevet populært at forske i inden for de seneste år – de såkaldte mobile ad hoc-netværks-protokoller – ofte forkortet som "MANET"-protokoller.

Jeg vil starte med at se på, hvor de nuværende netværksprotokoller kommer til kort, og i hvilke situationer en ny protokoltype vil kunne løse specifikke problemer. Dette foregår i afsnit 1.1. I afsnit 1.2 gives en mere præcis gennemgang af hvilke egenskaber, et netværk, der er lavet af en sådan protokol, skal have for at kunne bruges i situationerne præsenteret i afsnit 1.1.

I afsnit 1.3 gennemgår jeg de hovedtyper af MANET-protokoller, der findes i dag. Jeg finder samtidigt en MANET-protokol, der i en række eksperimenter i den øvrige litteratur har vist sig at fungere godt ud fra en række kriterier i forhold til andre eksempler på protokoller. I afsnit 1.4 vil jeg give et overblik over hvor langt, man generelt er kommet med MANET-protokol-specifikationsarbejdet og implementationen af disse.

I afsnit 1.5 vil jeg se på, hvordan en MANET-protokol kan indpasses i OSI-modellen (en standard, der sorterer protokoller i abstrakte lag), og hvilke udfordringer og muligheder, det umiddelbart underliggende lag i OSI-modellen giver en MANET-protokol. Og endelig vil jeg i afsnit 1.6 se nærmere på hvilke problemområder, der er specifikke for MANET'er, og som MANET-protokoller derfor måske bør behandle.

1.1 Baggrund for dannelse af en ny protokoltype

Den traditionelle måde at koble en computer til et netværk er at trække et kabel hen til computeren fra en form for netdelingsudstyr (f.eks. en switch). Netdelingsudstyret kan herefter være tilkoblet andet netdelingsudstyr, der tilsammen sammenkobler alle computerne i netværket samt eventuelle centrale servere og gateways til andre netværk (f.eks. internettet).

At trække dette kabel til computeren kan involvere, at man skal bore gennem vægge for fysisk at få kablet hen til computeren. Hvis computeren derefter skal flyttes permanent (f.eks. til et nyt kontor) eller temporært (f.eks. til et møde), skal kablet enten være langt nok til at flytte med, eller det vil være nødvendigt for computerens bruger manuelt at skifte til et andet kabel for at sikre, at computeren konstant er tilkoblet netværket under brugen af den.

I de senere år er det blevet mere og mere udbredt med brugen af trådløs teknologi, når en computer skal kobles til et netværk, hvilket har forbedret denne situation. Et sted, hvor

et antal computere skal indgå i et netværk, opsættes en eller flere basestationer, der er koblet sammen og er forbundet med eventuelle centrale servere og gateways. Når en ny computer skal tilkobles netværket, tilsluttes den blot en form for trådløs teknologi (f.eks. et trådløst netkort), der er i stand til at kommunikere med basestationerne. Flyttes computeren herefter permanent eller temporært, kan den trådløse teknologi i computeren blive ved med at kommunikere med den samme basestation, eller den kan skifte til en anden basestation, der er tættere på den nye placering for computeren, uden at den undervejs mister forbindelsen med netværket, og uden at brugeren af computeren skal foretage sig noget undervejs (udover at foretage den rent fysiske flytning af computeren).

Den fleksibilitet, dette giver, har gjort denne type teknologi mere og mere populær de senere år. Dette har medført, at der også er blevet forsket meget i området, hvilket har resulteret i nyere og mere pålidelige og sikre protokoller mellem basestationerne og den trådløse teknologi i computerne (f.eks. WPA, som jeg vender tilbage til i afsnit 1.6.4).

Der er dog også tænkt på kommunikation direkte fra en enhed til en anden enhed i denne type protokoller – enheder kan kommunikere med hinanden uden at gå gennem en basestation, blot de er tæt nok på hinanden. Denne direkte kommunikation kan ske både med og uden en basestation i nærheden til at styre kommunikationen.

Hvis der ikke findes en præ-eksisterende infrastruktur (d.v.s. et antal basestationer) i et område, og to enheder i dette område er for langt fra hinanden til, at deres trådløse teknologi er i stand til at forbinde til hinanden, vil de med denne "normale" type trådløse netværk dog ikke kunne kommunikere med hinanden.

Dette kan være et reelt problem i flere forskellige situationer – f.eks. hvis et antal udrykningskøretøjer (brandbiler, ambulancer etc.) skal deltage i en stor redningsaktion, hvor en bygning, der vedligeholdt en allerede eksisterende infrastruktur (basestationer, mobiltelefoni etc.) for området, er styrtet sammen, og hvor udrykningskøretøjerne på trods af dette skal kunne kommunikere internt med hinanden for at koordinere redningsaktionen.

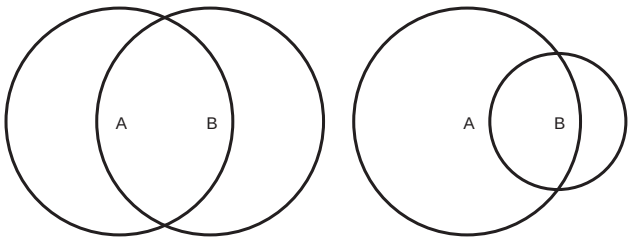
Et andet eksempel kan være en gruppe arkæologer, der skal arbejde på et stort, nyt udgravningssted, og hvor de undervejs skal kunne kommunikere med hinanden om hvad, de har fundet, og hvilke ressourcer, de har brug for, for at funne fortsætte udgravningen de enkelte steder.

Hvis alle de deltagende enheder i disse eksempler altid parvist er tæt nok på hinanden til, at den trådløse teknologi i dem kan kommunikere direkte, kan man bruge en normal trådløs protokol (som beskrevet ovenfor). Hvis dette ikke gælder, er man nødt til at benytte en alternativ strategi. En sådan kan være, at lade de deltagende enheder hjælpe hinanden med kommunikationen ved at danne et netværk, hvor mobile enheder, der (eventuelt temporært) befinder sig mellem to enheder, der vil kommunikere med hinanden, videresender pakker mellem disse to enheder. Med andre ord skal man bruge en protokol, der danner et mobilt ad hoc-netværk – et såkaldt "MANET".

1.2 Præcis definition af et "MANET" og andre begreber

I litteraturen benyttes begrebet "et MANET" normalt som en forkortelse for "et *mobilt ad hoc-netværk*" (i f.eks. [CM99, FL01, JMH05]). Fra tid til anden ses forkortelsen også brugt om "et *multi-hop* ad hoc-netværk" (i f.eks. [MAN]). I anden litteratur, der beskriver samme type netværk, bliver udtrykket "et ad hoc, *wireless* netværk" også ofte brugt (i f.eks. [TV01, Toh02]).

I dette speciale vil jeg benytte begrebet "MANET" om den type netværk, jeg i forrige afsnit fandt frem til, kan løse nogle specifikke problemer. Jeg vil benytte begrebet som en betegnelse, der dækker alle tre ovenstående udtryk – protokoller, der kan skabe et MANET, skal både kunne understøtte, at de deltagende enheder er *mobile*, at de benytter en *wireless* (trådløs) teknologi til at kommunikere med hinanden, og at der kan blive brug for *multi-hop*-ruter i netværket. Det sidste betyder, at selvom enheder i et MANET har trådløs teknologi, er



(a) Symmetrisk link fra A til B

(b) Ikke-symmetrisk link fra A til B

Figur 1.1: To knuder, A og B, med ringe, der angiver deres antennerækkevidder.

det tilladt, at denne ikke altid er kraftig nok til at alle enheder i et MANET kan nå hinanden direkte. Hvis to deltagende enheder vil foretage pakketransmissioner til hinanden, kan det derfor blive nødvendigt at benytte en eller flere enheder, der ligger mellem de to enheder, som *videresendende* enheder.

Bemærk specielt, at når jeg vælger både at forudsætte mobilitet og multi-hop-routing i definitionen af et MANET, betyder det, at hvis man i et MANET kan finde et sæt enheder i netværket, der tilsammen kan videresende en netværkspakke fra en afsenderdeltager til en modtagerdeltager (d.v.s. man har fundet en rute af deltagere imellem de to deltagere), så kan en vilkårlig af deltagerne i denne rute bevæge sig så langt væk i forhold til dens naboer, at den fundne rute holder op med at fungere. Hvis en protokol skal kunne bruges til at danne et MANET, er det altså grundlæggende, at den også er *adaptiv* over for sådanne ændringer i netværkstopologien.

Herudover forudsætter både jeg og litteraturen generelt, at et MANET også skal være *selv-organiserende* (i f.eks. [Toh02]). Det er ikke nok, at et MANET er adaptivt, opdateringen af information, der er nødvendig, for at deltagende enheder kan tilpasse sig en ny netværkstopologi (f.eks. opdagelse af de førnævnte ruter), skal kunne ske uden bruger-indgreb. Desuden skal et MANET selv være i stand til at opdage, hvis der er andre enheder i nærheden, der kan/vil indgå i det ad hoc-netværk, og skal kunne udføre de nødvendige ting, der skal udføres, for at disse nye enheder kan indgå i netværket (f.eks. udveksling af information om topologi etc.). Præcist hvad der skal ske, og hvilken information der skal spredes/indsamles, når en ny enhed skal tilsluttes, eller en ny rute skal findes, afhænger dog af den enkelte netværksprotokol.

At et MANET er et *ad hoc*-netværk betyder, at netværket skal kunne opstå nærmest "spontant" mellem netværksenheder. Et MANET må gerne være selv-startende (omend det ikke er et fast krav i litteraturen), men det må ikke være nødvendigt med nogen form for forudlavet infrastruktur i området, hvor enhederne, der vil danne et MANET, opholder sig. Dette inkluderer f.eks. afhængighed af basestationer som beskrevet i afsnit 1.1 og øvrige former for centraliserede enheder.

Andre definitioner

I det efterfølgende vil jeg kalde en mobil enhed (som beskrevet i det ovenstående) for en **"knude"**. Dette kan dække over enhver form for MANET-netværksdeltager, f.eks. en trådløs computer, der bliver båret eller kørt rundt. En knude skal blot være en enhed, der kan og vil deltage som en router (altså en videresendende enhed) for netværkspakker i et MANET.

Forbindelser mellem knuder (når de er inden for antennerækkevidde af hinanden) kaldes **"links"**. At der er et link fra en knude A til en knude B betyder ikke nødvendigvis, at der også er et link fra B til A – links er med andre ord ikke nødvendigvis **"symmetriske"**, da der kan være forskel på f.eks. antenneforholdene i de to knuder, kupering mellem knuderne, hvor

meget energi de enkelte knuder har tilbage til at udsende pakker etc. Et eksempel på et symmetrisk link kan ses i figur 1.1(a) på forrige side, mens et eksempel på et ikke-symmetrisk link kan ses i figur 1.1(b) på forrige side. Hvis der er et link (symmetrisk eller ikke-symmetrisk) fra en knude A til en knude B, kaldes B for A's **"naboknude"**.

I det ovenstående er det allerede antydet, at der kan forekomme netværkstrafik, der bliver genereret af MANET-protokollen i en knude og bliver sendt til MANET-protokollen i en anden knude – f.eks. når to knuder udveksler information om ruter i et MANET. Sådan trafik vil jeg fremover kalde **"kontroltrafik"**. Den anden type trafik, altså netværkstrafik, der indeholder rent faktiske datapakker fra en knude til en anden, vil jeg kalde **"datatrafik"**.

Fremover benytter jeg betegnelsen et **"normalt"** trådet eller trådløst netværk om netværk, der ikke er MANET'er – f.eks. et trådløst netværk, der er opbygget med basestationer som beskrevet i afsnit 1.1.

1.3 Hovedklassifikationer af MANET-protokoller

MANET-protokoller kan være vidt forskellige. Dette afsnit vil se lidt nærmere på hvilke grundlæggende hovedtræk, der er inden for MANET-protokol-området. De to vigtigste hovedtræk, man kan gruppere en MANET-protokol efter, er hvilken topologikendskabsstrategi og hvilket informationsindhentningstidspunkt, knuderne i MANET-protokollen skal benytte sig af.

1.3.1 Topologikendskabsstrategi ("link state" eller "distance vector")

I protokollerne i dag findes der en række forskellige tilgangsvinkler til hvilken topologikendskabsstrategi, de enkelte knuder kan have. De to mest brugte strategier er "Link state" og "Distance vector":

"Link state"-routing går ud på, at hver knude holder styr på den komplette topologi af hele netværket. Dette gøres typisk ved, at hver knude overvåger sine naboer og distribuerer information om knudens nabo-link-status ("link state") til resten af netværket med jævne mellemrum – f.eks. ved at broadcaste den ud i netværket. Når en knude skal sende en pakke til en anden knude, er det ud fra knudens interne netværkstopologiinformation en simpel sag at beregne den korteste rute til en given destination i netværket.

"Distance vector"-routing sænker informationsniveauet i den enkelte knude i forhold til "Link state"-routing. I "Distance vector"-routing kender hver enkelt knude ikke til hele netværkstopologien, men vedligeholder kun information om, hvilken knude den skal sende eller videresende en pakke til, for at denne vil nå til en bestemt destination. Informationen kan f.eks. for hver mulig destination i netværket bestå af en række naboknuder associeret med en "cost" ved at videresende pakken til denne naboknude – denne "cost" kan f.eks. være, hvor mange videresendelser det vil kræve, før pakken er fremme hos den endelige destinationsknude, hvis det er den associerede naboknude, der benyttes som "next hop"-knude. Filosofien bag denne type routing er altså, at i stedet for, at en knude kender den præcise rute, den skal sende en pakke via, sender den pakken i den retning, der vil gøre, at pakken benytter den korteste rute – heraf navnet "Distance vector".

De to ovenstående strategier kan give det indtryk, at knudernes topologikendskabsstrategi generelt er et spørgsmål om "niveau" – altså blot hvor *meget*, de enkelte knuder ved. Dette er dog ikke helt sandt. Andre strategier findes også, hvor den information, knuderne skal huske på, ikke blot kan defineres som et fravalg i forhold til de øvrige strategier. I den efterfølgende

	"Link state"	**"Distance vector"**
I netværket udveksler knuder information om...	Alle links	Retninger til destinationer
Løkker undgås...	Automatisk	Kun hvis MANET-protokollen specifikt tager sig af det
Størrelse af interne tabeller afhænger af...	Antal links i netværket	Antal naboknuder gange antal destinationsknuder i netværket

Tabel 1.2: Forskel på "Link state"- og "Distance vector"-protokoller

gennemgang vil jeg dog begrænse mig til at fokusere på de to introducerede typer af topologikendskabsstrategier: "Link state" og "Distance vector", da disse er de mest benyttede i de MANET-protokoller, der indtil videre er lavet.

Men hvilke konsekvenser har det så, hvis man vælger den ene strategi frem for den anden? Det afhænger helt af den MANET-protokol, der bruges. Blandt andet kan man i "Distance vector"-protokoller risikere, at der opstår "løkker" – en knude A videresender en pakke til en knude B, som videresender den til A igen, idet de begge har hinanden som korteste rute til pakkens destinationsknude. Dette vil naturligt ikke kunne forekomme i "Link state"-protokoller, men her skal der til gengæld udveksles mere information. I tabel 1.2 kan en oversigt over forskellene på "Link state"- og "Distance vector"-protokoller ses.

I afsnit 1.3.3 giver jeg eksempler på protokoller inden for hver type, og i afsnit 1.3.4 viser jeg resultater fra litteraturen, der viser, hvilken af disse der klarer sig bedst ud fra visse resultatkriterier. Det viser sig, at det er en "Link State"-protokol, der klarer sig bedst af de præsenterede eksempler.

1.3.2 Informationsindhentningstidspunkt ("proaktivt" eller "on demand")

I den ovenstående inddeling af MANET-protokoller efter deres topologikendskabsstrategi har jeg beskrevet protokollerne som om, at når en knude opdager, at der er sket ændringer i netværkstopologien, vil den del af det øvrige netværk, dette påvirker, i løbet af kort tid få opdateret deres interne tabeller, så disse afspejler ændringen. Dette behøver dog ikke være tilfældet. Visse protokoller distribuerer først opdateret information om netværkstopologien ud, når den skal benyttes af de enkelte knuder. Denne forskel i, hvornår knuderne får information om topologien, giver en anden inddeling af MANET-protokollerne. Igen kan man dele protokollerne op i to grupperinger: "Proaktiv" og "On demand":

"Proaktive" protokoller (også kaldet "Table driven" protokoller) er de protokoller, der straks (eller med jævne mellemrum) distribuerer information ud om netværkstopologien og ændringer i disse til de knuder, det berører. Dette betyder, at der kan forekomme kontroltrafik i netværket, selvom der ikke forekommer datatrafik, specielt hvis der forekommer bevægelse blandt de deltagende knuder.

"On demand"-protokoller (også kaldet "Reaktive" protokoller) er protokoller, hvor der ikke vil forekomme sådan kontroltrafik, når der ikke forekommer datatrafik. Knudernes interne tabeller opdateres med andre ord ikke automatisk, efterhånden som der sker ændringer i netværkstopologien. Denne information indhentes først, når der er brug for den, og knudens nuværende information enten viser sig at være mangelfuld eller forældet. Knuden holder med andre ord kun styr på de ruter (ved "link state"-protokoller) eller "next hop"-knuder (ved "distance vector"-protokoller), den umiddelbart har brug for, i stedet for *alle* ruter eller "next hop"-knuder i netværket.

Forskellen på de to fremgangsmåder påvirker både netværkstrafikforbrug, tidsforbrug og størrelsen af de interne tabeller. I "proaktive" protokoller kan knuder spare tid, når de skal sende

	"Proaktiv"	**"On demand"**
Forberedelse af rute kræves?	Aldrig	Ved brug af nye ruter og ruter, der er invaliderede
Kontroltrafik i netværket?	Altid nogen	Kun i forbindelse med forberedelse af rute
Størrelse af interne tabeller afhænger af...	Størrelsen på netværket	Antal ruter, der er i brug

Tabel 1.3: Forskel på "Proaktive" og "On demand"-protokoller

en pakke til en anden knude, fordi de aldrig skal benytte tid til at finde ruten eller "next hop"-knuden – denne information haves allerede. Dette er specielt en fordel, hvis der i netværket ofte laves nye, korte forbindelser mellem knuder. Til gengæld kan de enkelte knuder blive nødt til at holde styr på redundante ruter – hvis knuderne ligger tæt på hinanden, vil der ofte være mere end en rute fra en knude til en anden, og da "proaktive" ruter skal holde styr på enten hele netværkstopologien eller blot alle de destinationer, der kan nås fra hver af naboknuderne, kan dette kræve, at de enkelte knuder har både plads til tabeller med disse informationer og CPU-kraft til kontinuerligt at processere modtagne opdateringer af dem. Desuden er der en baggrunds-kontroltrafik i "proaktive" protokoller, hvilket vil påvirke energiforbruget i de enkelte knuder (se afsnit 1.6.1).

Disse problemer har "on demand"-protokollerne ikke. Her kan der spares på størrelsen af de interne tabeller i de enkelte knuder (fordi der kun skal vedligeholdes information om ruter, der er i brug), og der vil ikke altid være et baggrunds-kontroltrafikforbrug – kontrolbeskedoverheadet vil skalere i takt med en kombination af mobiliteten og det antal pakker, der skal sendes i netværket – helt ned til 0, hvis der enten ikke sendes pakker, og/eller knuderne ikke flytter sig. Til gengæld vil "on demand"-protokollerne specielt spilde tid, hvis en knude deri forsøger at sende pakker via en invalideret rute – først skal den bruge tid på at opdage, at ruten ikke længere fungerer, og derefter bruge tid på at finde en ny rute til destinationen for den afsendte pakke.

Som det kan ses, er det ikke et simpelt valg at vælge mellem at specificere en MANET-protokol, så den bliver "proaktiv" eller "on demand". En oversigt over fordele og ulemper kan ses i tabel 1.3.

1.3.3 Kombination af hovedklassifikationerne

Hvis man kombinerer de ovenfor beskrevne klassifikationer, får man i alt fire underinddelinger af MANET-protokoller:

"Link state", "proaktiv": Alle knuder kender de fulde ruter til alle destinationer i netværket. Et eksempel på en sådan protokol er "Optimized Link State Routing Protocol" (OLSR, [JMC+01, CJ03]). En kort præsentation af denne kan ses i appendix A.1.

"Distance vector", "proaktiv": Alle knuder kender "next hop"-knuden til alle destinationer i netværket. Et eksempel på en sådan protokol er "Destination-Sequenced Distance Vector" (DSDV, [PB94, PB01]). En kort præsentation af denne kan ses i appendix A.2.

"Distance vector", "on demand": Knuder kender "next hop"-knuder til de destinationer, de har brug for – enten fordi de sender pakker til disse destinationer, eller fordi en anden knude sender pakker til destinationen gennem denne knude. Et eksempel på en sådan protokol er "Ad Hoc On-Demand Distance Vector Routing" (AODV, [PR01, PBRD03]). En kort præsentation af denne kan ses i appendix A.3.

"Link state", "on demand": Knuder kender den fulde rute til de destinationer, de sender pakker til, men ikke hvis de kun videresender pakker for andre knuder via denne rute. Sådanne protokoller kaldes også "source-routende protokoller", da det er nødvendigt i pakken at angive den fulde rute til destinationsknuden for pakken. Et eksempel på en sådan protokol er "Dynamic Source Routing" (DSR, [JM96, JMB01, JMH05]). En kort præsentation af denne kan ses i appendix A.4, mens en mere grundig beskrivelse gives i kapitel 2.

1.3.4 Sammenligning af eksempler på MANET-protokoller

Hvilken af de fire underinddelinger, der blev præsenteret i forrige afsnit, er så den bedste at bruge, hvis man skal lave et MANET? Som så meget andet afhænger dette af omgivelserne, MANET-protokollen bliver brugt i. Dette kan være antal knuder, knudernes hastighed, hvor tæt knuderne er på hinanden etc.

Det ligger uden for dette speciale at lave en komplet sammenligning her, men noget tilsvarende er dog allerede gjort i adskillige artikler. Da forskellige protokoller inden for hver underinddeling stadig kan have meget forskellige egenskaber, er det i artiklerne dog ikke underinddelingerne, der er vejet op imod hinanden, men eksempler på protokoller inden for hver underinddeling. Jeg vil præsentere resultatet af to af disse artikler her.

Artiklen [BMJ+98] har modelleret fire forskellige protokoller i en modificeret version af en netværkssimulator kaldet ns-2 [NS2]. Dette inkluderer tre af dem, jeg har refereret til i de foregående afsnit (DSDV, DSR og AODV). Herefter har forfatterne kørt en række simuleringer af hver model under forskellige simulerede forhold, hvor knudernes hastighed og antal knuder, der afsender pakker, varieres, og har sammenlignet tre forskellige resultatkriterier fra hver simulering:

Pakkefremkomstraten – hvor stor en procentdel af de simuleret afsendte pakker, der nåede frem til modtageren: Det bedste resultat her fik "on demand"-protokollerne (DSR og AODV) – disse var i stand til at aflevere 95% af pakkerne uanset hvilke parametre, simuleringen blev kørt under. DSDV's pakkefremkomstraten faldt derimod til under 75%, efterhånden som knudernes hastighed blev sat op.

Antal pakketransmissioner – hvor mange pakketransmissioner, der forekom i alt i hver simulering: I DSDV (den "proaktive" protokol) holdt antallet af pakketransmissioner sig stabilt som forventet (da protokollen dikterer, at information skal broadcastes ud i netværket med jævne mellemrum uanset simuleringens parametre). I "on demand"-protokollerne (DSR og AODV) steg antallet af pakketransmissioner derimod, jo hurtigere knuderne bevægede sig rundt blandt hinanden. Dette giver også fin mening, da ruter oftere bliver invalideret, jo hurtigere knuderne bevæger sig. Der er dog en væsentlig forskel: AODV ("distance vector"-protokollen) brugte konsekvent cirka 4-5 gange så mange pakketransmissioner som DSR ("link state"-protokollen). DSR benyttede generelt et lavt antal transmissioner – selv når knuderne bevægede sig hurtigst, holdt DSR sig under DSDV.

Ruteoptimalitet: Hvor optimal, de benyttede pakkeruter var – målt som forskellen mellem antal brugte pakketransmissioner for routingen af en pakke og antallet af hops i den korteste rute, der fandtes i netværket mellem afsender- og destinationsknuden, da de respektive pakker blev simuleret afsendt. Her fik DSDV og DSR (de to mest forskellige protokoller – den ene en "distance vector", "proaktiv" protokol, den anden en "link state", "on demand" protokol) de bedste resultater – de benyttede ruter var meget tæt på optimale. AODV brugte derimod ruter, der var noget længere fra det optimale – op til fire hop eller mere. Specielt når knudernes hastighed blev sat op, forværredes AODV's ruteoptimalitet.

Alt i alt kan dette tolkes som, at DSR ("on demand", "link state"-protokollen) er den bedste protokol af de tre i denne sammenligning. Det skal naturligvis understreges, at en anden undersøgelse kan give et andet resultat, da der kan være forskelle i modellerne af protokollerne, omstændighederne, der simuleres etc. Desuden skal det naturligvis understreges, at "on demand", "link state"-protokoller ikke ud fra dette kan konkluderes bedre end de øvrige protokoller, idet der kun er modelleret en enkelt protokol i hver af de tre MANET-protokoltyper, der her kigges på, og idet der kun er kigget på enkelte resultatkriterier i hver af dem.

Den fjerde protokol, der blev præsenteret i [BMJ+98], var en "on demand", "link reversal routing"-protokol, altså et eksempel på en protokol, der benytter en anden topologikendskabsstrategi end dem, der blev præsenteret i afsnit 1.3.1. Jeg vil ignorere den her, da den i [BMJ+98] klarede sig dårligere end de øvrige i næsten alle de ovenstående resultatkriterier.

[BMJ+98] sammenlignede dog kun DSDV, DSR og AODV. Der mangler stadig en vurdering af, hvor god OLSR (den "proaktive" "link state"-protokol, jeg refererede til i forrige afsnit) er i forhold til de øvrige. Denne vurdering findes i f.eks. [CK04], der sammenligner bl.a. AODV, DSR og OLSR.

Artiklen viser bl.a. gennem simuleringer af modellerede versioner af protokollerne, at hvor DSR og AODV også her klarer sig fint hvad angår pakkefremkomstraten (her holder de sig begge altid over 70% i de simuleringer, der blev brugt i undersøgelserne i artiklen), så klarer OLSR sig mindre godt – hvis knudernes hastighed sættes op til den maksimale hastighed, der er brugt i simuleringer, falder fremkomstraten til under 45% i OLSR, hvilket er testens dårligste resultat.

Også hvad angår de øvrige resultatkriterier, der kigges på i artiklen, klarer DSR sig bedst, mens OLSR og AODV skiftes til at være den dårligste i simulationsresultaterne.

Det skal bemærkes, at [CK04] benytter et andet simuleringssetup end det, der blev benyttet i [BMJ+98]. Bl.a. benyttes en netværkssimulator kaldet QualNet [SNT] i stedet for ns-2 [NS2], og en række parametre for f.eks. antennerækkevidde i de simulerede omgivelser er defineret anderledes. Dette gør, at man ikke heraf direkte kan slutte, at DSDV også ville have klaret sig dårligere end DSR, hvis den var medtaget i [CK04], eller at OLSR ville have klaret sig dårligere end DSR, hvis den var medtaget i [BMJ+98]. Det kan dog ses som et godt tegn, at der er en vis form for enstemmighed imellem de to undersøgelser, idet de to gengangere (DSR og AODV) klarede sig på samme måde i forhold til hinanden i dem.

Jeg mener derfor, at dette giver en tilstrækkelig baggrund for, at det er værd at undersøge DSR nærmere. Dette gør jeg i kapitel 2 og frem.

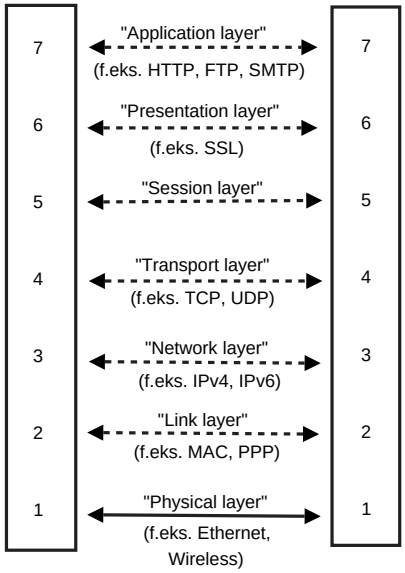
1.4 Status for MANET-protokol-specifikationsarbejdet

Det er først for relativt nyligt, at man er begyndt at udvikle MANET-protokoller for alvor. De fleste bøger, der dækker det generelle netværksområde (f.eks. [Com95, Sta00, For03, CDK05, Koz05]) nævner slet ikke MANET'er eller nævner det kun meget kort som en hypotetisk form for fremtidsnetværk.

Dette betyder dog ikke, at man først er begyndt at forske i emnet for nyligt. Allerede i 1973 begyndte det amerikanske militær at forske inden for området og byggede i denne forbindelse et radionetværk, der kunne videresende netværkspakker mellem specialbyggede, mobile knuder. MANET-protokollen PRNet [JT87] var det første resultat af denne forskning og specificerede et basalt "link state", "proaktivt" MANET, der kunne rumme op til 138 knuder.

Jvf. [MC98] er det dog først i de senere år, at udviklingen for alvor har taget fart, idet det først her er blevet muligt at bygge MANET-deltagende knuder med hardware, som allerede masseproduceres, og som derfor kan anskaffes billigt (f.eks. trådløse netkort).

IETF (Internet Engineering Task Force) har dannet en arbejdsgruppe, der tager sig af koordineringen af udviklingen af MANET-protokoller [IETa, IETb]. I skrivende stund har gruppen stået bag publiceringen af flere MANET-protokoller som såkaldte eksperimentelle RFC'er



Figur 1.4: Konceptuel oversigt over lagene i OSI-modellen baseret på [Sta00, side 53] og [CDK05, side 78].

(bl.a. AODV som [PBRD03] og OLSR som [CJ03]) og er med i udviklingen af andre, der med tiden kan blive til RFC'er, men endnu ikke er helt klare til det (f.eks. DSR [JMH05]). Det nuværende mål med gruppens arbejde er at færdigspecificere protokollerne og vælge to ud (en "proaktiv" og en "on demand") og videregive disse som bud på de MANET-protokoller, der skal benyttes som IETF-standarder inden for området i fremtiden. Planen er, at dette skulle være sket i februar 2006, men jvf. [IETb] var det endnu ikke sket primo maj 2006.

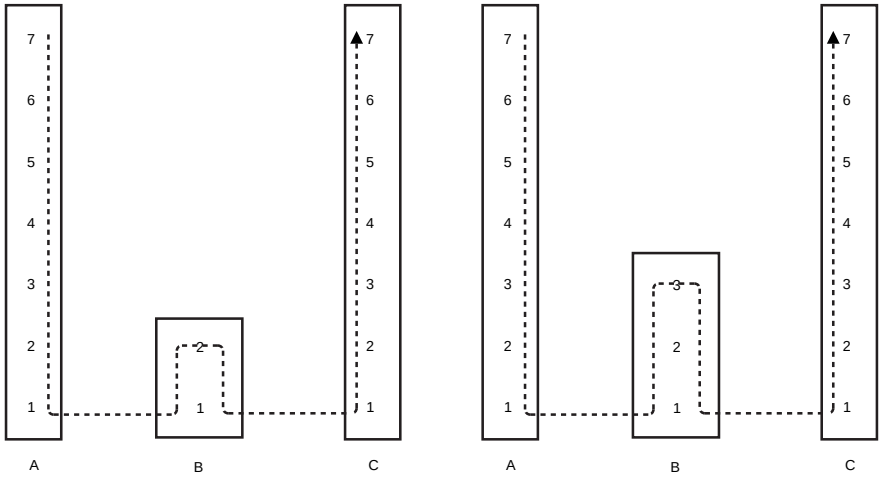
Samtidigt er IEEE i gang med at forberede en ny IEEE-standard for MANET-protokoller (af dem kaldet "Wireless Mesh Networking"). I øjeblikket er man stadig ved at evaluere forskellige protokoller, for at finde ud af hvilken der skal benyttes. Jvf. [IEE] har man siden juni 2005 skåret i alt 15 indsendte protokoller ned til to (pr. november 2005). Det er dog ikke offentliggjort hvilke to, det drejer sig om. Efter planen vil én af disse blive til en ny standard under navnet IEEE 802.11s – i øjeblikket sigter man efter, at dette vil ske senest i juli 2008.

At hverken IETF eller IEEE har lagt sig fast på en standard endnu, betyder dog ikke, at der ikke findes implementationer af protokollerne. Der findes mange "proof of concept"-modeller og -implementationer, hvor de basale dele af en protokol er modelleret/implementeret (f.eks. [RMP00, Dem01, Son01, Nor05] for DSR) og testet (f.eks. i simulationer eller ved at lade et antal bærbare computere i et antal biler i bevægelse foretage pakketransmissioner til hinanden). Men bl.a. fordi man endnu ikke har lagt sig fast på en (eller flere) standarder inden for området, mangler den komplette og produktionsklare implementation af en af disse "nye" MANET-protokoller dog i skrivende stund stadig (jvf. [PRG]).

1.5 Indpasning i OSI-modellen

1.5.1 Om OSI-modellen

Når to knuder skal kommunikere med hinanden, kan man inddele niveauet, de kommunikerer på, i forskellige lag. Et "lag" skal her forstås som en inddeling i forskellige abstraktionsniveauer. Der findes forskellige modeller til at lave denne inddeling. En af de mest brugte modeller



(a) Bridging af en pakke (b) Routing af en pakke

Figur 1.5: Bridging/routing af en pakke fra knude A til knude C gennem knude B i forhold til OSI-modellen.

er OSI-modellen, der i oversigtsform kan ses i figur 1.4 på forrige side.

Et eksempel på et abstraktionsniveau (d.v.s. et lag i OSI-modellen) er "Transport layer". TCP-protokollen, der arbejder på dette lag, sørger bl.a. for, at en række pakker når frem fra en knude A til en knude B i den rækkefølge, de bliver afsendt, og uden pakketab. TCP-protokollen kan nu benytte en underliggende protokol på "Network layer". Dette kan f.eks. være IP-protokollen, der sammen med *sit* underliggende OSI-lag sørger for at foretage de præcise pakketransmissioner, der skal til for at sende en pakke fra A til B, men som f.eks. ikke har en indbygget sikkerhed for, at pakker rent faktisk når frem og i den afsendte rækkefølge.

1.5.2 Bridging og routing

Når en pakke skal videresendes i et "normalt" trådet eller trådløst netværk, er det normalt protokollen, der ligger på enten OSI-lag 2 eller 3, der sørger for dette (jvf. [IBM96, side 37 og 48]). Videresendelse på OSI-lag 2 kaldes "bridging", mens videresendelse på OSI-lag 3 kaldes "routing".

Bridging (figur 1.5(a)) benyttes i "normale" netværk til at sammenkoble lokalnetværk – en bridge sættes op på to netværk, og denne bridge sørger for at se, om der udsendes pakker på det ene netværk, der er addresseret til en enhed på det andet netværk. Disse pakker sørger den derefter for at udsende på det andet netværk – evt. efter at have tilpasset pakkens format, hvis de to netværk benytter forskellige protokoller på OSI-lag 2.

Routing (figur 1.5(b)) benyttes også i "normale" netværk til at sammenkoble lokalnetværk på samme måde som ved bridging, men i modsætning til bridging kan routere gøre dette mere selektivt. F.eks. benytter de kontroltrafik til at kommunikere med hinanden om netværkstopologien, og dermed kan de f.eks. nøjes med at videresende pakker til et netværk, hvis og kun hvis de ved, at en anden router på det netværk er forbundet med et tredie netværk, hvorpå den endelige modtager af pakken opholder sig.

Den ovenstående gennemgang tyder på, at det vil være mest logisk at specificere en MANET-protokol, så den fungerer på OSI-lag 3, da denne lader til at give de mest avancerede muligheder. Der kan dog være årsager til, at man ønsker at lade den fungere på lag 2 i stedet for – f.eks., hvis man ønsker, at forskellige protokoller på lag 3 (IPv4, IPv6 etc.) skal kunne bruge den samme MANET-protokol direkte.

Men at man lader en MANET-protokol være en OSI-lag 2-protokol, betyder ikke, at den så ikke må benytte sig af kontroltrafik, blot fordi dette ikke gøres i "traditionelle" netværksprotokoller. OSI-lagene under MANET-protokollen kan ikke se forskel på den trafik, der sendes, og OSI-lagene over MANET-protokollen vil ikke opdage, at der sendes mere trafik end det, den styrer. En protokoldesigner har derfor fuld frihed på dette område.

I praksis findes der både eksempler på MANET-protokoller, der er specificeret til at fungere på OSI-lag 2 (f.eks. DSDV, se afsnit 1.3.3) og OSI-lag 3 (f.eks. DSR, se afsnit 1.3.3). Begge benytter sig af kontroltrafik.

Uanset om en MANET-protokol specificeres til at fungere på OSI-lag 2 eller 3, er det værd at undersøge OSI-lag 2 nærmere, da problemstillinger heri kan smitte af på specifikationen af en MANET-protokol. Dette gøres i næste afsnit.

Bemærk i øvrigt, at i en bridge/router i et "normalt" netværk vil der jvf. ovenstående typisk være to netværksinterfaces, der binder to delnetværk sammen til ét sammenhængende netværk. Bridgen/routeren vil desuden som oftest være et stykke fysisk netværksudstyr, der ikke fungerer som andet end bridge eller router. Begge dele er lavet om i et MANET: Som beskrevet i afsnit 1.2 skal hver knude (d.v.s. hver netværksdeltager) fungere som en bridge/router ud over at være netværksdeltager, men de bridger/router kun inden for samme netværk (d.v.s. inden for samme MANET), så hver knude har typisk kun et netværksinterface.

Bortset fra her i afsnit 1.5.2 vil jeg benytte udtrykket "routing" som en fællesbetegnelse for det at videresende pakker, d.v.s. dækkende både "bridging" og "routing".

1.5.3 Specielt om OSI-lag 2

En "Media Access Control"-protokol (en "MAC-protokol") er en OSI-lag 2-protokol (se figur 1.4 på side 9), der definerer et sæt af regler eller procedurer, der tillader et antal knuder at foretage pakketransmissioner på et delt medie på en effektiv måde. Dette medie kan f.eks. være et trådløst netværk.

En MANET-protokol på OSI-lag 3 kan benytte en underliggende MAC-protokol til at tage sig af kommunikationen fra en knude til dens naboknude.

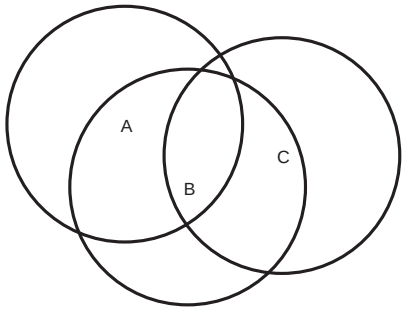
Der findes mange forskellige MAC-protokoller. Generelt kan de deles op i to typer: Synkrone og asynkrone. I en synkron MAC-protokol findes der en centraliseret enhed, der med jævne mellemrum broadcaster et signal, som de øvrige enheder kan synkronisere kommunikationen med. I en asynkron MAC-protokol benyttes en distribueret algoritme til at koordinere pakketransmissionerne i netværket.

Da der i et MANET-netværk ikke findes en centraliseret enhed, der kan nå alle enhederne og styre dem, er det nødvendigt med den asynkrone fremgangsmåde i den underliggende MAC-protokol. Asynkrone protokoller har dog to hovedproblemer, det er værd at se nærmere på her:

"Hidden Terminal Problem"

Hvis to knuder A og C begge er inden for antennerækkevidde af en knude B, men de ikke er inden for antennerækkevidde af hinanden (som illustreret i figur 1.6 på næste side), og de begge foretager en pakketransmission til B, kan pakkerne kollidere i luften, hvorved B ikke vil modtage nogen af dem. Dette vil A og C ikke nødvendigvis opdage, da de ikke kan høre, at der samtidigt med deres transmission er en transmission fra en anden knude uden for deres antennerækkevidde. Dette kaldes for "Hidden Terminal Problem".

En typisk løsning på dette er brugen af bekræftelser – når A sender en pakke til B, sender B en bekræftelse tilbage til A. Hvis A ikke modtager en sådan bekræftelse, efter den har sendt en pakke, kan den vente et lille stykke tid og prøve igen, indtil den får en bekræftelse på, at pakken er nået frem til B. Hvis gentagne forsøg på at sende pakken ikke lykkes, kan A vælge



Figur 1.6: Tre knuder, A, B og C, med ringe, der angiver deres antennerækkevidder. A og B samt B og C kan altså kommunikere med hinanden trådløst, mens A og C er uden for hinandens rækkevidde.

at konkludere, at grunden til, at den ikke modtager en bekræftelse, ikke er pakkekollissioner, men at B f.eks. ikke længere er inden for A's antennerækkevidde.

Tiden, der ventes mellem hver genudsendelse af pakken, kan eventuelt være randomiseret, så hvis A og C samtidigt udsender en pakke til B, og disse kolliderer, vil risikoen for, at det sker igen, når de begge forsøger at gentransmittere pakken, blive formindsket.

Hvis der forekommer en pakkekollission, vil den ovenstående løsningsmodel dog skabe en forsinkelse i afleveringen af begge pakker – både den fra A og den fra C.

Dette kan man afhjælpe med en alternativ løsning: Brugen af "handshaking": Før A sender sin pakketransmission til B, broadcaster den en speciel RTS-pakke ("Request To Send"). B broadcaster en CTS-pakke ("Clear To Send") tilbage som svar på dette, og først herefter foretager A sin egentlige pakketransmission til B. Da C er inden for antennerækkevidde af B, hører den også CTS-pakken og ved derfor, at den skal vente i et stykke tid, før den må forsøge at sende en pakke ud. Tidsintervallet, der skal ventes, bliver ofte valgt tilfældigt for at undgå, at der opstår pakkekollissioner i netværket, hvis andre knuder end C venter på at sende en RTS-pakke ud efter at have hørt B's CTS-pakke.

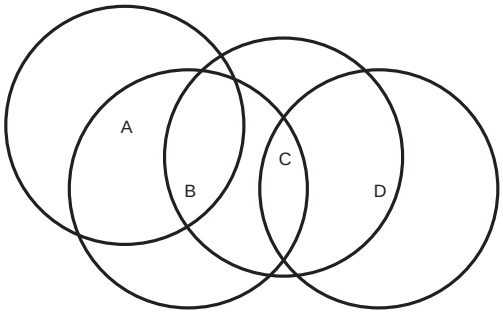
RTS- og CTS-pakker kan dog også kollidere med hinanden – f.eks. hvis B sender en CTS-pakke som svar på A's RTS-pakke samtidigt med, at C sender en RTS-pakke til B. Da RTS- og CTS-pakker er væsentligt mindre end "normale" pakker, vil tiden, hvor de er "i luften" og dermed kan kollidere med hinanden, dog være mindre, men det er klart, at brugen af "handshaking" ikke løser problemet fuldstændigt.

Nogle MAC-protokoller vælger at kombinere brugen af "handshaking" og bekræftelser, så der både foregår en udveksling af RTS- og CTS-pakker, og modtageren sender en bekræftelse tilbage, når den har modtaget den egentlige pakke.

Både for "handshaking"- og bekræftelses-metoderne gælder det, at hvis en knude A vil sende en pakke til en knude B, kræver det, at linket mellem A og B er symmetrisk (se afsnit 1.2). Dette er ikke altid tilfældet i MANET'er, da forskelle i f.eks. antennerækkevidder kan betyde, at B kan høre pakker sendt fra A, men A ikke kan høre pakker sendt fra B. En MANET-protokol (på OSI-lag 3), der benytter en MAC-protokol (OSI-lag 2) med "handshaking" og/eller bekræftelser, kan derfor i disse tilfælde ikke længere foretage direkte pakketransmissioner fra A til B.

"Exposed Node Problem"

Antag, at man har et eksempelnetværk som illustreret i figur 1.7 på næste side. I dette netværk sender C en pakke til D. B overhører dette, og vil egentlig gerne sende en pakke til A, men er nødt til at vente et stykke tid, før den kan gøre det, da dens pakketransmission ellers vil kollidere med C's. Denne situation kaldes for "Exposed Node Problem".



Figur 1.7: Fire knuder, A, B, C og D, med ringe, der angiver deres antennerækkevidder.

Problemet kan f.eks. løses v.h.a. retningsbestemte sendeantenner, så C drejer sin antenne mod D, før den foretager sin pakketransmission, så transmissionen ikke overhøres af B. B kan gøre det samme (i retning mod A), og B og C kan herved foretage deres pakketransmissioner samtidigt. Dette kræver dog to ting: Dels skal knuderne have retningsbestemte antenner, dels skal knuderne kunne finde ud af, i hvilken retning de øvrige knuder rent fysisk er i netværket. Det sidste inkluderer, at C kan finde ud af, at D f.eks. har flyttet sig, siden C og D sidst kommunikerede, og at C kan justere for dette. Hvis dette skal udnyttes, skal altså både hardwaren og MANET-protokollen understøtte det.

Alternativt kan man lade en MANET-protokol udnytte "Exposed Node Problem": Når C i det ovenstående eksempel overhører pakken fra B, kan den eventuelt kigge i den og uddrage informationer om ruter etc. i netværket. Dette gør f.eks. DSR (se afsnit 1.3.3).

1.6 Problemområder, der er specifikke for MANET'er

Når man ser på forskellen mellem et "normalt" trådløst netværk (som beskrevet i afsnit 1.1) og et MANET, er der i MANET'er en række problemområder, der ikke eller kun i mindre grad forekommer i "normale" trådløse netværk. Visse af disse bør tages i betragtning, når man designer en MANET-protokol, mens man for andre ikke nødvendigvis behøver gøre noget specifikt i designet af en ny MANET-protokol for at løse problemområdet – en mere generel løsning kan allerede være lavet eller kan komme senere. I dette afsnit vil en række af disse problemområder blive præsenteret.

1.6.1 Energiforbrug

Et område, man bør kigge på, når man designer en MANET-protokol, er det energiforbrug, protokollen vil forårsage, at de deltagende knuder har.

For at en trådløs knude skal være "rigtig" trådløs, indeholder den ofte en form for genopladeligt batteri, så knuden ikke er konstant afhængig af en ekstern energikilde. Hvis knuden f.eks. er en bærbar computer, og denne er tændt, vil batteriet på et tidspunkt blive tømt, og det vil være nødvendigt (midlertidigt) at sætte den til en stikkontakt. I et "normalt" trådløst netværk er dette normalt ikke et problem; knuden skal alligevel opholde sig inden for en vis radius af en præ-eksisterende infrastruktur (f.eks. en basestation, se afsnit 1.1) for at være forbundet til netværket, og når der er en sådan infrastruktur, vil denne ofte også indeholde en energikilde (f.eks. en stikkontakt).

Det forholder sig anderledes for knuder, der deltager i et MANET. I eksemplerne i afsnit 1.1 kan man se, at der ikke blot mangler en netværksmæssig infrastruktur, der er heller ikke nogen energikildeinfrastruktur. Der kan desuden være langt til den nærmeste energikil-

de – når batteriet i en knude er tømt, kan der gå lang tid, før batteriet kan lades op igen, og knuden igen kan bruges. Det er derfor vigtigt, at MANET-protokoller sørger for at minimere energiforbruget.

Hvis man ser bort fra de faktorer, der også forekommer i "normale" trådløse netværk, d.v.s. f.eks. hvor meget strøm, harddisken kræver, hvor meget strøm, skærmen kræver etc., har man en enkelt faktor tilbage, der er speciel for knuder, der deltager i et MANET: Hvor mange pakketransmissioner knuderne skal foretage.

Når en (trådløs) knude vil deltage i et netværk, vil der i et "normalt", basestationbaseret trådløst netværk kun forekomme kommunikation mellem knuden og en basestation (plus eventuelt overhørt kommunikation jvf. afsnit 1.5.3). Alle pakketransmissioner, der kommer fra basestationen til knuden, er tilsigtet knuden, og alle pakketransmissioner, knuden sender ud, stammer fra et højereliggende OSI-lag i knuden selv.

Deltager knuden i stedet i et MANET, er der som bekendt ikke længere basestationer – alle pakketransmissioner forekommer mellem deltagende knuder. Til gengæld kan MANET-protokollen, knuden skal benytte for at deltage i netværket, kræve, at knuden videresender pakker for andre knuder, hvilket vil sige, at knuden skal foretage pakketransmissioner med pakker, der ikke længere kommer fra et højereliggende OSI-lag i knuden selv. Desuden kan en MANET-protokol specificere, at der skal sendes forskellig kontroltrafik rundt i netværket.

Ifølge [Toh02, kapitel 9] er det specielt *afsendelsen* af pakker gennem et trådløst interface, der bruger strøm – set i forhold til dette kræver modtagelsen af pakker ikke nær så meget strøm. I [Toh02, kapitel 9] foretages der derfor en demonstration af, hvor længe knuder holder, hvis de udsender pakker med forskellige mellemrum. Resultatet af dette blev, at hvor en knudes batteri holdt i omkring 160 minutter, hvis den brugte en MANET-protokol, der dikterede, at knuden skulle foretage en pakketransmission hvert tiende millisekund, kunne det holde i ca. 200 minutter, hvis der kun skulle udsendes en pakke hvert halve sekund. Hvis intervallet blev sat yderligere op til hvert tiende sekund, kunne batteriet holde i 220 minutter. Dette betyder altså, at batteriet holdt over 35% længere, når der blev sendt pakker med det største interval i forhold til, når der blev sendt pakker med det mindste.

Selvom det måske ikke altid i specifikationen af en MANET-protokol kan lade sig gøre at skære antallet af pakketransmissioner helt ned til én hvert tiende sekund, så kan det jvf. dette alligevel betale sig at minimere på antallet af brugte pakketransmissioner i det omfang, det kan lade sig gøre.

1.6.2 Pakkekollissioner

I et "normalt" trådløst netværk er det muligt at have en centraliseret synkronisering af netværkstrafikken. Specifikt kan man på denne måde undgå, at der opstår pakkekollissioner. I andre tilfælde vil det være et underliggende MAC-lag, der sørger for dette – f.eks. v.h.a. handshaking eller bekræftelser som beskrevet i afsnit 1.5.3.

Hvis man designer en MANET-protokol, der skal kunne fungere v.h.a. hardware, der ikke har f.eks. handshaking eller bekræftelser, har designerne af protokollen forskellige valgmuligheder:

Ignorering: Designerne kan vælge at gøre ingenting for aktivt at modvirke pakkekollissioner i netværket. Hvis pakkekollissioner forekommer, og en eller flere pakketransmissioner går tabt, vil det være op til et højereliggende OSI-lag at opdage dette og gensende pakken. Dette kan f.eks. være TCP-laget, hvis denne benyttes som et ovenliggende lag i forhold til MANET-protokollen.

Detektering: Designerne kan vælge ikke at gøre noget aktivt for at modvirke pakkekollissioner i netværket, men kan sørge for, at det bliver opdaget, hvis det sker – f.eks. ved at implementere, at knuder, der modtager en pakketransmission, skal sende en bekræftelse tilbage for pakketransmissionen. Modtager en afsendende knude ikke en bekræftelse,

kan dette f.eks. skyldes en pakkekollission i netværket, og den afsendende knude kan forsøge at foretage pakketransmissionen igen.

Aktiv modvirkning: Hvis en situation opstår, hvor protokollen kan forudsige, at flere knuder vil foretage en pakketransmission samtidigt – f.eks. som svar på den samme pakketransmission – kan designerne af protokollen vælge at gøre noget aktivt for at modvirke dette. Det kan f.eks. være at lave protokollen, så der altid kun er en af knuderne, der foretager en pakketransmission i disse situationer – eller det kan sørge for, at de enkelte knuder venter et randomiseret stykke tid, før de foretager pakketransmissionen (så alle pakkerne stadig bliver udsendt, men ikke længere samtidigt). Designerne af protokollen kan vælge enten at lave forsinkede pakketransmissioner i specifikke situationer eller ved *alle* pakketransmissioner. Dette kræver naturligvis en afvejning, både m.h.t. hvilke pakker der skal forsinkes, og hvor lang tid de skal forsinkes, så risikoen for pakkekollissioner minimeres, men protokollen ikke herved bliver for langsom i brug.

Handshaking: Endelig kan designerne implementere deres egen "Request-To-Send"-"Clear-To-Send"-protokol for at minimere risikoen for pakkekollissionen. Dette kan ske som en kopi af det, visse OSI-lag 2-protokoller implementerer som beskrevet i afsnit 1.5.3. Dette betyder dog også (ligesom i afsnit 1.5.3), at ikke-symmetriske links i netværket ikke længere kan benyttes.

De ovenstående punkter kan naturligvis kombineres – en protokol kan f.eks. vælge, at der både skal være forsinkelser på og bekræftelser for pakketransmissioner.

1.6.3 Tildeling af IP-adresser

IP-protokollen er i høj grad blevet en standard inden for netværk i dag. Når man i et "normalt" trådet eller trådløst netværk vil tildele IP-adresser til de deltagende knuder, kan dette både ske statisk (hver knude får tildelt en IP-adresse, som knuden manuelt skal konfigureres til at benytte) eller dynamisk (v.h.a. DHCP som defineret i [Dro97]).

I DHCP distribueres IP-adresser af en centraliseret enhed (f.eks. en server eller en specielt udvalgt router). Som bekendt findes en sådan centraliseret enhed ikke i et MANET, så her er en anden strategi er nødvendig.

Dette er ikke et problem, der endnu er løst, men det er ikke nødvendigvis et problem, designeren af en MANET-protokol skal løse – hvis en distribueret DHCP-algoritme bliver lavet, vil denne kunne benyttes af knuder, der deltager i et vilkårligt MANET.

DHCP-specifikationen [Dro97] gør dog selv opmærksom på, at da ikke alle knuder nødvendigvis altid kan nå alle knuder i et MANET til enhver tid, giver dette et problem i forbindelse med de forsøg, der har været lavet på området indtil videre:

> "[. . .] distributed address allocation schemes depend on a polling/defense mechanism for discovery of addresses that are already in use. IP hosts may not always be able to defend their network addresses, so that such a distributed address allocation scheme cannot be guaranteed to avoid allocation of duplicate network addresses."

Enten skal der findes en løsning på dette, eller der skal benyttes en anden løsningsstrategi, hvis man ønsker at bruge dynamisk tildeling af IP-adresser.

1.6.4 Sikkerhed

I et intranet, der er implementeret v.h.a. et normalt, trådet netværk, er trafikken mellem to knuder i netværket normalt ikke krypteret på OSI-lag 3 eller derunder (se figur 1.4 på side 9). Dette åbner i princippet for muligheden for, at personer, der er fysisk i nærheden af netværket

vil have mulighed for at aflytte trafikken på det og måske endda kunne tilgå ressourcer på det (såsom delte drev, printere etc.).

Der vil dog ofte være en form for implicit beskyttelse – f.eks. ved at netværket rent fysisk kun spænder over et område inden for en enkelt bygning, hvortil der kan være adgangskontrol.

Trådløse netværk har ikke nødvendigvis denne form for implicit sikkerhed – hvis et trådløst netværk sættes op i en bygning, vil det i visse tilfælde også kunne nås fra et konkurrende firma i nabobygningen – eller fra en bil på firmaets parkeringsplads. Herved åbner man for, at en knude i nabobygningen eller i en bil på parkeringspladsen kan sættes på netværket og bruges til at aflytte trafik i netværket og eventuelt også til at tilgå ressourcer på det.

Denne problemstilling bliver håndteret af forskellige protokoller. Den nyeste (og den, der i øjeblikket regnes for at være den mest sikre), er "Wi-Fi Protected Access" (WPA, se [WFA03]). WPA sørger for to ting: For det første tillader den kun knuder, der kan autentificeres enten via en centraliseret server (der f.eks. kan tage imod brugernavne og passwords eller digitale signaturer) eller v.h.a. et delt password, at tilgå netværket – dette vil modvirke, at ikke-autoriserede knuder kan komme på netværket og udnytte eventuelle ressourcer derpå. For det andet sørger WPA for, at al trådløs kommunikation i netværket foregår med en stærk kryptering med skiftende nøgler for at modvirke aflytning.

I et MANET skal det overvejes, om man ønsker at have en autentificeringsmekanisme involveret. Det kan udmærket være ånden i et MANET, at alle interesserede knuder skal kunne tilslutte sig og deltage i det. Hvis dette omvendt ikke er meningen, skal en måde, tilladte knuder kan autentificere sig på, findes. Metoden fra WPA med en centraliseret server kan ikke benyttes (p.g.a. MANET's decentraliserede natur), men brugen af et delt password kan muligvis benyttes.

Kryptering i et MANET er til gengæld vanskeligere at implementere. Hvor man i et "normalt", trådløst netværk kun skal kryptere pakketransmissionerne mellem en mobil knude og en centraliseret basestation, og basestationen derfor kan benyttes som centraliseret krypteringsnøgleudsteder, så findes en sådan centraliseret udsteder ikke i et MANET. Hvis en nøgle ikke har kunnet udveksles direkte mellem to knuder, før disse slutter sig til et MANET, men skal udveksles over en multi-hop rute, kan det være vanskeligt at beskytte krypteringen mod "man in the middle"-angreb.

Disse problemer er dog ikke nødvendigvis nogle, den enkelte protokoldesigner skal løse. Bl.a. forfatterne af [WZK05] har arbejdet med at forsøge at lave en protokol-udvidelse, der er designet til at kunne indkluderes i en vilkårlig MANET-protokol, og som vil give denne både kryptering og autentificering. Arbejdet er dog endnu ikke færdigt.

2

Introduktion til Dynamic Source Routing-protokollen

I afsnit 1.3.3 refererede jeg til fire forskellige eksempler på MANET-protokoller – herunder Dynamic Source Routing (DSR). I afsnit 1.3.4 viste jeg, at i to forskellige artikler klarede DSR sig bedst af disse. Med dette som baggrund vil jeg derfor kigge nærmere på DSR i dette kapitel.

2.1 Om DSR-protokollen

Hvis man benytter MANET-klassifikationerne beskrevet i afsnit 1.3, kan man ifølge [JMH05] betegne DSR som en protokol, der er designet til at kunne organisere knuder i et on-demand, link-state (og dermed source-routende) netværk. Protokollen kan ifølge [JMH05] organisere op til 200 mobile knuder i et MANET.

2.1.1 DSR's hovedbestanddele

Da DSR er en source-routende protokol, skal en knude, der gerne vil sende en netværkspakke til en anden knude i et DSR-netværk, medsende den fulde ruteliste over knuder, pakken skal sendes via, for at nå til destinationsknuden. DSR's hovedfunktionalitet (som jeg fremover vil kalde netop "DSR's hovedfunktionalitet") er den del af DSR, der sørger for at sende en netværkspakke til en anden knude givet en sådan rute. Dette beskrives i afsnit 2.2.

Da DSR samtidigt skal lave et fuldstændigt selv-organiserende netværk (jvf. afsnit 1.2), er det en integreret del af protokollen, hvordan denne rute-frembringelse skal foregå – proceduren til dette hedder "Route Discovery" (RD) og beskrives i afsnit 2.3. Bemærk, at da DSR er en distribueret protokol, findes der intet centralt element, de enkelte knuder kan spørge om disse ruter, når de måtte få brug for dem.

I specifikationen [JMH05] behandles "DSR's hovedfunktionalitet" og RD under et, og det er upræcist, hvor grænsen mellem de to går. Jeg vælger i denne gennemgang at være mere stringent og sige, at RD kun inkluderer præcist det at *finde* en rute – ikke at benytte den.

"DSR's hovedfunktionalitet" og RD suppleres af en sidste procedure: "Route Maintenance" (RM). Denne tager sig af det praktiske i forbindelse med selve transmissionen af en

pakke fra en knude til en anden. Dette inkluderer f.eks. at opdage, at en transmission fejler, fordi et link i en rute er blevet forældet, og at meddele dette til den oprindelige afsenderknude af pakken. RM beskrives i afsnit 2.4.

Gennemgangen vil dække hele DSR-protokollen med de udvidelser, der er basale nok til at være en del af specifikationen [JMH05] selv, pånær brugen af eksterne links (links mellem knuder i DSR-netværk og knuder i andre netværk, som endnu kun er rudimentært understøttet i DSR), understøttelse af adskillige netværkskort i de enkelte knuder (til brug for sammensætning af adskillige DSR-delnetværk udnyttende f.eks. forskellige netværkskorttyper) og den valgfrie, omfattende udvidelse "Flow State Extension", der optimerer for ruter, der ofte bruges i et netværk, ved at ophæve kravet om en eksplicit source-rute i headeren for pakker, der benytter disse ruter.

2.1.2 Indplacering i OSI-modellen

Konceptuelt var det ifølge [JMH05] oprindeligt meningen, at protokollen skulle implementeres på OSI-lag 2 (link layer – se figur 1.4 på side 9), så så mange protokoller som muligt kunne benytte den (IPv4, IPv6, IPX etc.). Det blev dog besluttet at specificere DSR, så den passer ind på OSI-lag 3 (network layer) i stedet for, da det var et ønske, at DSR skulle kunne understøtte knuder med adskillige netværkskort af forskellige type, der tilsammen skulle forme et samlet ad hoc-netværk. Realistisk set var dette ifølge [JMH05] den eneste måde, det kunne lade sig gøre på. Indtil videre er DSR dog kun specificeret for IPv4 [JMH05]. Hvis man ser bort fra kravet om adskillige netværkskort af forskellig type, er der dog intet i vejen for, at DSR bliver implementeret på OSI-lag 2 i stedet for (se afsnit 1.5.2).

Jeg har fundet tre eksempler på implementationer af DSR-protokollen: picoNet II ([Son01] – en Linux-implementation af DSR), DSR-UU ([Nor05] – en kombineret ns-2-modellering og Linux- og "LinkSys WRT54G router"-implementation) og Monarch ([RMP00] – en FreeBSD-implementation). I alle disse tre implementationer er DSR-protokollen implementeret på OSI-lag 3 (network layer), altså som en del af IP-protokollen. Desuden er en DSR-model beskrevet i [Dem01], men selve den beskrevne model er ikke tilgængelig, og det benyttede OSI-lag angives ikke i artiklen.

2.1.3 Antagelser

Specifikationen [JMH05] af protokollen stiller en række krav til omgivelserne, protokollen er designet til at fungere under. En af de vigtigste antagelser er, at selvom knuderne i netværket er mobile, bevæger de sig ikke så hurtigt rundt i forhold til netværkshardwarens transmissionshastighed, at flooding er den eneste mulighed for at distribuere pakker rundt i netværket. Dette er i tråd med [CMB96], der netop siger, at efterhånden som knuder bevæger sig hurtigere og hurtigere, må enhver ad hoc-routing-algoritme "give op" – alle algoritmer vil på et tidspunkt blive mindre effektive end blot at floode alle netværkspakkerne rundt i netværket. Det antages desuden, at diameteren (antal hop fra en knude i den ene ende til en knude i den anden ende af et MANET) typisk vil være større end 1 men ofte ikke højere end 5-10. Pakker må gerne tabes eller korrumperes, mens de transmitteres fra en knude til en anden, men det antages, at knuder er i stand til at frasortere netværkspakker, der er blevet korrumperede. Det antages desuden, at alle knuderne i netværket er "venligtsindede", d.v.s. at de vil deltage fuldt i netværket, i forhold til hvad protokollen kræver, f.eks. at de vil videresende pakker for hinanden og ikke vil sende bevidst forkert information af nogen art ud i netværket, f.eks. bekræftelse af ruter, der ikke findes etc. Protokollen gør heller intet for at undgå, at knuder kan aflytte trafik i netværket. De sikkerhedsmæssige aspekter ved brugen af MANET'er er nærmere beskrevet i afsnit 1.6.4.

DSR indeholder ikke en mekanisme til at uddele IP-adresser i netværket. Det antages, at knuderne får disse på en anden måde – enten statisk eller v.h.a. DHCP. Problemstillinger vedrørende brugen af DHCP er nærmere beskrevet i afsnit 1.6.3.

2.2 DSR's hovedfunktionalitet

Et Dynamic Source Routing-netværk er – som navnet angiver – et source-routende netværk. Det betyder, at når en knude A vil sende en netværkspakke til en anden knude B i et DSR-netværk, skal A i pakken selv inkludere den komplette sti af knuder, pakken skal rejse gennem i netværket, for at nå til B. Når en knude modtager en sådan pakke som en del af en videresendelse, skal den blot slå op i pakkens ruteliste for at se, hvem den skal sende pakken videre til, og derefter gøre dette. Dette benævner jeg "DSR's hovedfunktionalitet".

Knuden A kan benytte forskellige metoder til at finde denne rute:

- A kan genbruge en rute, den tidligere har fundet gennem en procedure kaldet "Route Discovery" (som introduceres i afsnit 2.3). Disse ruter gemmes i A's "Route Cache" (som introduceres i afsnit 2.2.4).
- A kan benytte en rute, den har deduceret sig frem til ved at kigge på ruterne i netværkspakker, andre knuder har sendt i systemet, og som A har overhørt (se afsnit 2.2.4). Også disse ruter gemmes i A's "Route Cache".
- A kan igangsætte en "Route Discovery" for at finde en ny rute til B. En "Route Discovery" benyttes kun, hvis A ikke har andre muligheder. Dette beskrives i afsnit 2.3.

Dette afsnit ser på, hvordan DSR fungerer, hvis en sådan rute til en destinationsknude allerede kendes af DSR-laget for den knude, der ønsker at sende en netværkspakke til destinationsknuden.

2.2.1 Opbygning af netværkspakkerne

Når en knude, der indgår i et DSR-netværk, vil sende en netværkspakke ud, modificerer DSR-laget den IP-pakke, den får fra det ovenliggende OSI-lag (se afsnit 2.1.2).

En normal IP-pakke kan konceptuelt beskrives som et tupel jvf. [Sta00, side 57]:

```
(IP-header, brugerdata)
```

"Brugerdata" indeholder de data, en applikation i knuden A gerne vil sende til knuden B, f.eks. i en UDP- eller TCP/IP-pakke (d.v.s. UDP- eller TCP-headeren er her en del af "brugerdata"), mens IP-headeren igen konceptuelt kan beskrives som et tupel jvf. [Com95, side 92]:

```
(id,
 time_to_live,
 IP_source_address,
 IP_target_address,
 andre IP-felter)
```

"Andre IP-felter" dækker her over cirka 10 andre felter, der ikke er interessante for forståelsen af DSR.

I DSR-sammenhænge udvides IP-pakken, så den konceptuelt kan beskrives som følgende tupel i stedet:

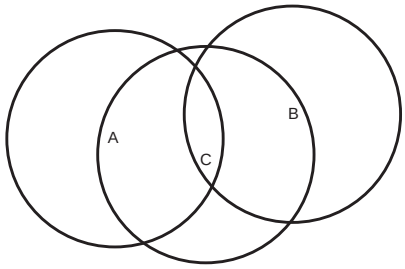
```
(IP-header, DSR-optionsheader, brugerdata)
```

DSR-optionsheaderen består af en liste af "DSR-options", der igen konceptuelt kan ses som tupler:

```
(DSR-optionstype, DSR-optionsdata)
```

DSR-optionstypen bestemmer hvilken type DSR-optionsheader, den her har fat i, mens DSR-optionsdata indeholder data, der er relevante specifikt for hver af disse DSR-optionstyper.

I forbindelse med DSR's hovedfunktionalitet benyttes kun en enkelt optionsheader: "DSR Source Route". Denne beskrives i det efterfølgende afsnit.



Figur 2.1: Tre knuder, A, B og C, med ringe, der angiver deres antennerækkevidder.

2.2.2 DSR-optionsheaders

Den mest brugte DSR-optionstype i DSR-protokollen (og den eneste, der benyttes i forbindelse med "DSR's hovedfunktionalitet") er "DSR Source Route". Optionstypen benyttes til at angive den rute, en pakke skal følge for at nå fra en afsenderknude til en destinationsknude. DSR-optionsdatadelen af en "DSR Source Route"-optionsheader indeholder følgende data:

```
(salvage_counter, segments_left, route)
```

`route` er en liste af knuder, der angiver den rute, pakken skal transmitteres via, eksklusive den oprindelige afsenderknudes og den endelige destinationsknudes adresse – disse adresser sættes i stedet i de tilsvarende felter i IP-headeren. `segments_left` er blot en pointer ind i denne liste, der angiver hvilken knude i listen, der skal modtage pakken næste gang. Dette gør, at videresendelser fra en knude til den næste kan ske uden at ændre ret meget andet i optionsheaderen end pointeren til den næste knude. Initielt sættes `segments_left`-feltet til længden af listen, og den tælles derefter ned ved hver videresendelse. `salvage_counter` er en tæller, der tæller hvor mange gange, pakken har været udsat for en "Salvage Operation" (en type redningsaktioner, der introduceres i afsnit 2.4.6).

Optionstypen kan ikke bruges til at broadcaste pakker med (det må ikke være en broadcast/multicast-adresse, der er destinationen for pakken). Samtidigt skal denne optionstype forekomme i alle pakker i DSR-protokollen (pånær pakker, hvori der findes en såkaldt "Route Request"-optionsheader, da disse netop skal broadcastes rundt – dette beskrives i afsnit 2.3.1). DSR-protokollen understøtter med andre ord ikke broadcasting af andre pakker end "Route Request"-pakker.

2.2.3 Eksempel

Hovedparten af funktionaliteten bag "DSR's hovedfunktionalitet" er nu præsenteret, hvilket gør, at "DSR's hovedfunktionalitet" nu kan illustreres med et eksempel.

Antag, at man har et område med tre deltagende knuder i et DSR-netværk som i figur 2.1. Cirklerne i figuren angiver antennerækkevidderne for de enkelte knuders trådløse netkort. I dette eksempel kan A og C samt C og B altså kommunikere med hinanden trådløst, mens A og B er uden for antennerækkevidde af hinanden. Antag nu, at A gerne vil sende en netværkspakke til B, som den har en rute til i sin "Route Cache". Dette sker på følgende måde:

1. A kigger i sin "Route Cache" og får fra den ruten [A,C,B] til B.

2. A modificerer nu IP-pakken på tre områder:

   - IP-headerens sourceadresse sættes til A's IP-adresse (hvis dette ikke allerede er gjort af det i forhold til DSR-laget ovenliggende OSI-lag).

- IP-headerens destinationsadresse sættes til B's IP-adresse (hvis dette ikke allerede er gjort af det i forhold til DSR-laget ovenliggende OSI-lag).
- En DSR-optionsheaderliste tilføjes IP-pakken. Denne indeholder blot en enkelt DSR-optionsheader:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=1,
                       route=[C]))]
```

  – hvor `C` betegner C's IP-adresse.

3. A sender pakken ud.

4. C modtager pakken fra A og undersøger, om den er modtageren af pakken. Dette gør den ved at kigge på hvad `segments_left`-feltet peger på i rutelisten i "DSR Source Route"-optionsheaderen. Dette giver, at knuden er modtageren af pakken, men ikke den *endelige* modtager af pakken (da `segments_left` ikke er 0). Derfor tæller den `segments_left`-feltet 1 ned (til 0), og sender pakken videre til næste modtager.

5. A modtager pakken fra C men opdager, at den ikke skal være modtager af pakken, og den ignorerer derfor blot pakken.

6. B modtager også pakken fra C og undersøger, om den er modtageren af pakken, ved at kigge på `segments_left`-feltet og rutelisten (eller rettere destinationsfeltet i IP-headeren, da `segments_left`-feltet er 0). Ifølge dette er knuden modtageren af pakken, og da den også er den *endelige* modtager af pakken (da `segments_left`-feltet netop er 0), vil den ikke sende pakken videre.

7. B fjerner DSR-optionsheaderlisten fra IP-pakken og giver pakken til det OSI-lag, der ligger over DSR-laget i knuden.

DSR-protokollen har hermed succesfuldt routet en netværkspakke fra A til B via C.

2.2.4 Foreslåede datastrukturer

I DSR's hovedfunktionalitet benyttes som beskrevet i de foregående afsnit en enkelt datastruktur: En "Route Cache" til opbevaring af ruter til destinationsknuder. Denne gennemgås mere detaljeret her.

"Route Cache"

Når en knude har deduceret sig frem til en rute, f.eks. fordi den skal videresende en pakke, der indeholder f.eks. en "DSR Source Route"-optionsheader, gemmes denne rute i knudens "Route Cache".

Konceptuelt kan ruter gemmes på flere forskellige måder i "Route Cache". I specifikationen [JMH05] foreslås to forskellige strategier:

Komplette ruter: `[(target, [(route, last_used_time)])]`

`route` angiver her en liste af knuder til destinationsknuden. For hver destinationsknude gemmes altså et antal ruter til denne destination. Når nye ruter gemmes i "Route Cache", gemmes ikke blot den komplette rute, men også alle prefixes i rutelisten – for f.eks. rutelisten [A,B,C,D] gemmes altså rutelisterne [A,B,C,D], [A,B,C] og [A,B].

Links i netværket: `[(source_node, destination_node, last_used_time)]`

Et antal links fra en knude til en anden gemmes blot. For rutelisten [A,B,C,D] gemmes linkene (A,B), (B,C) og (C,D). Denne taktik gør det muligt at kombinere rutestumper til nye ruter, selvom en "Route Discovery" i netværket aldrig har returneret denne rute. Det kræver dog mere processorkraft at hente en rute fra "Route Cachen", hvis denne metode benyttes – til dette kan f.eks. Dijkstras shortest-path-algoritme [Dij59] benyttes.

Bemærk, at rutelisten fra "DSR Source Route"-optionsheaders ikke indeholder den oprindelige afsenderknudes eller den endelige destinationsknudes adresse, disse skal henholdsvist sættes før og efter rutelisten, før denne caches i det ovenstående.

Hvis "Route Cachen" bliver forespurgt om en rute til en bestemt destinationsknude, og flere ruter dertil findes i cachen, er det op til implementøren af "Route Cachen" selv at bestemme, hvad den "bedste" rute er – dette kan f.eks. være den korteste rute (målt i antal hops), ruter, den selv har fundet frem til (fremfor ruter, der er fundet v.h.a. videresendte pakker) eller ruter, det er kortest tid siden sidst har været i brug i forbindelse med videresendelse af andre pakker.

Ruter gemmes i "Route Cache" indtil det opdages, at de ikke længere er gyldige – herefter slettes de. Dette kan f.eks. ske som følge af en modtaget (eller overhørt) "Route Error"-pakke (se afsnit 2.4.3) eller som en del af "Route Maintenance" (se afsnit 2.4). Hvis det (som følge af begrænset tilgængelig plads for "Route Cachen") er nødvendigt at slette ruter, er det igen op til implementøren at bestemme, hvad taktikken for denne sletning skal være – det kan f.eks. blot være en Least Recently Used-taktik (i de to ovenstående konceptuelle datastrukturer kan `last_used_time` benyttes til dette), eller det kan være en taktik baseret på, hvad de "dårligste" ruter er (f.eks. længste, ældste etc.).

Når en rute skal findes, har picoNet II [Son01] som strategi valgt, at den "bedste" rute i "Route Cachen" er den korteste rute, knuden har til en destination. De øvrige implementationer er mindre sofistikerede: Monarch [RMP00] vælger som den "bedste" rute den rute, der er fundet sidst – altså den nyeste rute – og kun denne ene rute til destinationen gemmes. DSR-UU [Nor05] gemmer også kun en enkelt rute til hver destination, men når en rute er fundet, overskrives den ikke igen (medmindre den bliver fjernet som en del af RM, se afsnit 2.4). Den "bedste" rute bliver dermed den ældste rute til en destination. Alle tre implementationer benytter "komplette ruter"-strategien til at gemme ruter.

2.2.5 Typer af fysiske netværkssetup i forhold til "Route Cache"

Antag, at en knude C modtager en pakke fra B, hvor rutelisten A→B→C→D→E står i dens "DSR Source Route"-optionsheader. Hvilke ruter, C kan deducere ud fra denne pakke (og derfor kan gemme i sin"Route Cache"), afhænger af, i hvor høj grad det fysiske medie er i stand til at have bidirektionelle links. I DSR-sammenhænge skelner man mellem tre forskellige fysiske netværkssetup.

"Frequently-unidir": Links er "ofte" kun unidirektionelle (d.v.s. ikke-symmetriske jvf. afsnit 1.2), og netværkshardwaren er i stand til at sende pakker over disse ikke-symmetriske links.

Hvis der benyttes "komplette ruter", skal C→D og C→D→E gemmes. Benyttes i stedet en "link"-baseret strategi, skal enkeltlinkene A→B, C→D og D→E gemmes. Der er ingen grund til at gemme B→C, da C ikke kan få brug for en rute, hvor den selv er modtagerknuden.

"Mostly-bidir": Links er kun sjældent unidirektionelle, og netværkshardwaren er i stand til at sende pakker over ikke-symmetriske links.

Ved opdatering af "Route Cachen" skal alle rutelisterne fra "Frequently-unidir"-tilfældet i dette tilfælde gemmes – men her både i den retning, de står i rutelisten, og i den

reverserede (d.v.s. modsatte) retning, da det er sandsynligt, at disse ruter også fungerer. Det betyder, at fra den pakke, C modtager, skal både C→D→E og C→B→A (og præfix-ruter herfra) gemmes, hvis ruter gemmes som “komplette links” i “Route Cache”. Benyttes i stedet den link-baserede strategi, skal A→B, B→A, C→B, C→D, D→E og E→D gemmes, mens C ikke kan få brug for hverken B→C og D→C.

Hvis en knude F overhører pakken, mens C sender den videre til D, kan F (udover de ruter, den kan udtrække af A→B→C→D→E og E→D→C→B→A) deducere sig frem til, at ruten A→B→C→F også må virke, og derfor kan også ruter fra denne og den reverserede af denne gemmes).

“Bidir-only”: Netværkshardwaren er kun i stand til at sende pakker over symmetriske (bidirektionelle) links. Eventuelle ikke-symmetriske links vil ikke blive understøttet af den underliggende hardware, og DSR vil derfor ikke kunne bruge dem.

Ved opdatering af “Route Cachen” skal rutelisten fra “DSR Source Route”-optionsheaderen i dette tilfælde gemmes i begge retninger. I afsnit 2.3.3 beskrives, hvordan “Route Discovery” i denne type netværkssetup kan være ved enten at finde en ny rute eller at teste en ruteliste for bidirektionalitet. Hvis pakken, en knude vil gemme information fra, er en del af en sådan test, skal kun links, som den modtagende knude kan deducere sig frem til, er testet for bidirektionalitet, gemmes.

Ved overhørte pakker skal linket mellem afsenderknuden og knuden, der fik pakken, ligeledes heller ikke gemmes, da også dette link kan være unidirektionelt.

2.3 “Route Discovery”

I det foregående afsnit beskrev jeg, hvordan en pakke kan sendes fra en knude til en anden via en rute. Når en sådan rute skal findes, benytter en afsendende knude proceduren “Route Discovery” (RD).

Kort sagt fungerer RD på følgende måde: En knude broadcaster en pakke med en speciel DSR-optionsheader, en “Route Request”-header, ud i DSR-netværket. Hver knude, der modtager denne pakke, tilføjer sin adresse til en ruteliste heri, og broadcaster pakken videre, indtil destinationsknuden modtager den. Denne returnerer så en “Route Reply”-optionsheader med den opsamlede ruteliste i en pakke til den oprindelige afsender af “Route Request”-pakken.

2.3.1 DSR-optionsheaders

I forbindelse med RD har man ifølge det ovenstående brug for to forskellige DSR-optionstyper (udover “DSR Source Route” præsenteret i afsnit 2.2.2): “Route Request” og “Route Reply”. Disse er ganske simple:

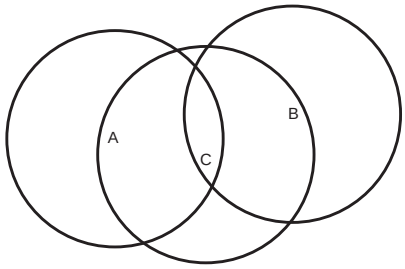
“Route Request”

DSR-optionsdatadelen af en “Route Request”-DSR-option indeholder følgende data:

```
(id, target, route_so_far)
```

– hvor `target` er adressen på den eftersøgte destinationsknude og `route_so_far` er en liste af knuder, der angiver den opsamlede ruteliste indtil videre.

Identifikationsnummeret for optionsheaderen skal gøres så unikt som muligt pr. afsenderknude. Optimalt set bør alle knuderne enten gemme sit sidst brugte identifikationsnummer på en batteridrevet enhed eller benytte et tilfældigt tal som initiel værdi, men DSR er designet til også at fungere på knuder, hvor dette ikke nødvendigvis er muligt, så det er ikke et krav.



Figur 2.2: Tre knuder, A, B og C, med ringe, der angiver deres antennerækkevidder.

"Route Reply"

DSR-optionsdatadelen af en "Route Reply"-DSR-option indeholder blot følgende konceptuelle datastruktur:

```
(route)
```

– hvor `route` er en liste af knuder, der angiver den fundne ruteliste.

Hvordan disse to optionsheaders mere præcist benyttes i en RD, vises her i et eksempel.

2.3.2 Eksempel: "Route Request"-pakker

Antag, at man har et område med tre deltagende knuder i et DSR-netværk (som vist i figur 2.2).

Hvis en knude A igangsætter en RD for at finde en rute til B, sker dette ved at A laver en "Route Request"-pakke, d.v.s. en IP-pakke med en "Route Request"-optionsheader i DSR-optionsheaderlisten. I datadelen af denne DSR-optionsheader sættes et unikt identifikationsnummer, B's IP-adresse og den opsamlede ruteliste indtil videre (som initielt blot er en enkelt adresse: A's IP-adresse).

Pakken propageres nu fra A til B som følger:

1. A broadcaster pakken ud – dette sker blot ved at få det trådløse netkort til at sende pakken som en normal pakketransmission, men med en speciel broadcast-adresse sat i destinationsadressefeltet i IP-headeren af pakken. DSR-optionsheaderen indeholder følgende liste:

   ```
   [("Route Request", (id=5,
                       target=B,
                       route_so_far=[A]))]
   ```

   – hvor A og B betegner henholdsvist A og B's IP-adresser.

2. C modtager pakken fra A og opdager, at den ikke er den ønskede destination for "Route Request"-pakken (så den ikke kan returnere et svar til A), og broadcaster derfor pakken videre med C's IP-adresse tilføjet rutelisten. DSR optionsheaderen indeholder altså her følgende:

   ```
   [("Route Request", (id=5,
                       target=B,
                       route_so_far=[A,C]))]
   ```

3. A modtager pakken fra C, men opdager, at den allerede står i "Route Request"-optionsheaderens ruteliste, og smider derfor pakken ud for at undgå, at en resulterende ruteliste kan komme til at indeholde løkker.

4. B modtager også pakken fra C og opdager, at den *er* den ønskede destination for “Route Request”-pakken. Den laver derfor en “Route Reply”-pakke. Den opsamlede ruteliste fra “Route Request”-DSR-optionsheaderen tilføjet B’s IP-adresse benyttes til dette. Den nye pakke indeholder altså følgende DSR-optionsheader:

```
[("Route Reply", (route=[A,C,B]))]
```

B skal nu sende “Route Reply”-pakken tilbage til A. Hvordan dette gøres afhænger af, hvordan netværkshardwaren virker i detaljer:

2.3.3 Forsendelse af “Route Reply”-pakker

Når B i det ovenstående eksempel har modtaget en “Route Request”-pakke og skal til at sende en “Route Reply”-pakke tilbage, kender den ikke nødvendigvis en rute tilbage til A. Hvis B ikke har en rute til A i sin “Route Cache”, skal den ifølge den beskrevne procedure normalt igangsætte en RD for at finde ruten. Her skal “Route Reply”-optionen dog piggybackes (d.v.s. tilføjes) til den nye “Route Request”-pakke for at undgå en mulig uendelig løkke af “Route Discoveries”.

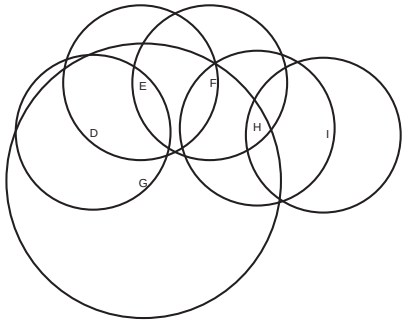
Hvis det underliggende fysiske netværk er af typen “Bidir-only”, d.v.s., at det underliggende lag ikke er i stand til at bruge ikke-symmetriske links (se afsnit 2.2.5), må denne fremgangsmåde dog ikke benyttes – i stedet skal alle rutelister reverseres, og der skal benyttes en såkaldt “Blacklist”-tabel:

“Bidir-only”-netværk: Reverserede rutelister

I tilfældet “Bidir-only” vides det i det ovenstående eksempel, at hvis en rute fra A til B er fundet, kan alle links blot vendes om for at få en rute fra B til A, og B kan derfor undgå at skulle igangsætte en ny RD. Da pakkerne fra A og C i det ovenstående tilfælde oprindeligt blev broadcastet ud, kan B ikke nødvendigvis være sikker på, at linkene A→C og C→B er bidirektionelle, og det er derfor ikke sikkert, at den fundne rute A→C→B vil virke for A. Det er derfor et krav for C, at den benytter den reverserede rute B→C→A for at sende “Route Reply”-pakken tilbage til A – hvis et af linkene viser sig kun at være unidirektionelt, vil denne fremsendelse af pakken fejle, og dels vil C få en fejl tilbage (så den ved, at ruten alligevel ikke kunne benyttes), dels vil A ikke modtage “Route Reply”-pakken, så den begynder ikke at benytte en fejlagtig rute.

Hvis C modtager “Route Request”-pakken broadcastet fra en anden knude også, svarer den endnu engang på pakken ud fra den nye rute, der er i pakken, så ovenstående situation vil ikke nødvendigvis betyde, at A aldrig vil modtage et svar. Alle andre knuder end B må dog kun behandle pakken (d.v.s. broadcaste den videre) en enkelt gang (dette beskrives mere præcist i afsnit 2.3.5). Det betyder, at nogle ruter kan gå tabt, hvis en mellemliggende knude modtager en “Route Request”-pakke fra adskillige andre knuder. Hvis ruten, en mellemliggende knude første gang modtager pakken fra, viser sig ikke at være bidirektionel i alle links, betyder dette altså, at nogle (og potentielt alle) fungerende ruter kan gå tabt i forløbet.

Situationen vises i figur 2.3 på næste side, hvor knuden I modtager den opsamlede rute D→G→H→I, der viser sig at være ikke-symmetrisk (og svar-“Route Reply”-pakken derfor automatisk ikke vil nå frem). I vil aldrig modtage ruten D→E→F→H→I, da H sorterer denne fra, da den allerede har behandlet denne “Route Request”-pakke en gang. Denne rute er ellers symmetrisk og ville have virket. Denne situation kan lyde alvorlig, men den bliver afhjulpet af brugen af en “Blacklist”-tabel, som beskrives i næste afsnit.



Figur 2.3: Seks knuder, D, E, F, G, H og I, med ringe, der angiver deres antennerækkevidder. Bemærk, at linket mellem G og H er ikke-symmetrisk.

"Bidir-only"-netværk: Brug af en "Blacklist"-tabel

Da "Route Request"-pakker jvf. det ovenstående altid bliver broadcastet ud, kan dette (i tilfældet "Bidir-only") betyde, at en del af de fundne ruter er ugyldige. For at minimere dette, vedligeholdes i "Route Maintenance" en "Blacklist"-tabel (som beskrives nærmere i afsnit 2.4.5), hvor indgangene indeholder den tilstand, knuden i øjeblikket har deduceret sig frem til, at linkene til naboknuderne er i.

Hvis det ifølge tabellen er sandsynligt, at linket mellem en naboknude og knuden selv er unidirektionelt ("unidirectionality probable"), og knuden modtager en broadcastet "Route Request"-pakke fra naboknuden, ignoreres pakken helt. Dette vil minimere antallet af afsendte "Route Reply"-pakker, der ender med at fejle.

Hvis linket ifølge tabellen i stedet er i tilstanden "unidirectionality questionable", skal knuden først checke, om tilstanden mellem naboknuden og knuden selv i virkeligheden er unidirektionelt eller bidirektionelt. Dette gøres ved at sende en ny "Route Request"-pakke til naboknuden med time to live-feltet sat til 1. Hvis knuden herefter inden for et vist timeout modtager et svar fra naboknuden, fortsætter knuden med at behandle pakken som normalt, ellers ignoreres pakken (som i "unidirectionality probable"-tilfældet ovenfor). I begge tilfælde opdateres indgangen i "Blacklist"-tabellen.

I situationen fra figur 2.3 vil knuden I have modtaget ruten D→G→H→I og vil returnere denne rute til D. Da den reverserede ruteliste skal benyttes til at sende denne "Route Reply"-pakke, sendes denne gennem ruten I→H→G→D. Linket H→G vil dog fejle, og dette vil få H til at sætte G i sin "Blacklist".

D modtager derfor ikke et svar, og vil efter et vist timeout genudsende "Route Request"-pakken. Den præcise fremgangsmåde til sådanne genudsendelser beskrives i afsnit 2.3.5. Denne gang vil H fjerne pakken med den opsamlede rute D→G→H p.g.a. den opsamlede information i H's "Blacklist". H vil efter dette i stedet videresende pakken med den opsamlede rute D→E→F→H til I som ønskeligt, som endelig kan returnere en fuldt bidirektionel ruteliste til D.

2.3.4 Eksempel: "Route Reply"-pakker

Hvis eksemplet fra afsnit 2.3.2 forsættes, antages det nu, at B har fundet ruten B→C→A (se figur 2.2 på side 24), f.eks. som følge af et opslag i sin "Route Cache" eller ved at reversere den fundne rute (i "Bidir-only"-tilfældet fra foregående afsnit). B skal nu sende en pakke med en "Route Reply"-optionsheader tilbage til A:

5. B tilføjer en "DSR Source Route" med den fundne ruteliste til A til DSR-optionsheaderen, så DSR-optionsheaderen i alt ser ud som følger:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=1,
                       route=[C])),
 ("Route Reply", (route=[A,C,B]))]
```

Her er “DSR Source Route”-optionsheaderens ruteliste, `[C]`, listen af alle knuder, pakken skal transmitteres gennem mellem B og A (jvf. afsnit 2.2.2). B og A vil stå som henholdsvis source- og destinationsadressen i IP-headeren, og derfor behøver de ikke at blive inkluderet i rutelisten.

Bemærk her, at man *ikke* kan forkorte indholdet af rutelisten i “Route Reply” som i “DSR Source Route” ved at skære source- og destinationsadressen væk. To senere udvidelser (beskrevet i afsnit 2.3.6 og 2.4.6) gør, at det ikke altid vil være den samme knude, der afsender en pakke (d.v.s. står i IP-headerens sourceadresse), som der står som det sidste link i “Route Reply”-rutelisten.

6. B sender pakken ud. Svarende til situationen i eksemplet for “DSR’s hovedfunktionalitet” (afsnit 2.2.3) vil pakken nu rejse via C til A, som modtager pakken.

7. Da den modtagne pakke indeholder en “Route Reply” med en rute fra A til B, kan A nu tage den oprindelige pakke, der fik knuden til at igangsætte en RD, og sende den til B. DSR-optionsheaderen indeholder her blot source-ruten:

   ```
   [("DSR Source Route", (salvage_counter=0,
                          segments_left=1,
                          route=[C]))]
   ```

   hvor `[C]` som før er listen af alle knuder, pakken skal rejse gennem mellem A og B (som igen står som henholdsvis source- og destinationsadressen i IP-headeren).

8. Igen foregår dette som i eksemplet for “DSR’s hovedfunktionalitet” (afsnit 2.2.3), så pakken bliver transmitteret via B til C, som nu endelig modtager den oprindelige pakke.

2.3.5 Foreslåede datastrukturer

Til brug for RD-proceduren har hver knude tilknyttet tre forskellige datastrukturer:

“Send Buffer”

Når en knude igangsætter en RD, gemmes den pakke, der fik knuden til at igangsætte RD-algoritmen, i knudens “Send Buffer”. Når en rute til en bestemt knude findes, tages pakken ud af “Send Buffer” og sendes til modtageren. “Send Buffer” indeholder altså kun de netværkspakker, som knuden på et givet tidspunkt ikke kender en rute til.

Konceptuelt gemmes pakkerne i følgende datastruktur:

```
[(target, [(packet, timestamp)])]
```

– d.v.s., at der for hver destinationsadresse gemmes de pakker, der venter på en rute til adressen. Hver pakke er markeret med det tidspunkt, de blev indsat i køen.

Hvis en rute ikke bliver fundet, skal pakkerne efter en vis timeout fjernes fra “Send Buffer” igen uden at blive behandlet yderligere eller give anledning til en fejlmelding. I DSR-implementationerne Monarch [RMP00], DSR-UU [Nor05] og picoNet II [Son01] er timeout’en i alle tilfælde sat til 30 sekunder – hvilket også er det tal, der forslås i specifikationen [JMH05]. For at undgå, at “Send Buffer” bliver overfyldt, kan en FIFO-strategi desuden benyttes til at fjerne netværkspakker fra “Send Buffer”, hvis timeout’en ikke er nok.

"Route Request Table" for internt skabte pakker

For at holde styr på udsendelsen af "Route Request"-pakker for de pakker, der gemmes i en knudes "Send Buffer", benyttes "Route Request Table" for internt skabte pakker. Den består konceptuelt af følgende:

```
[(target, (last_ttl,
           last_route_request_time,
           number_of_route_requests_with_this_ttl,
           timeout_before_next_route_request))]
```

Når en knude igangsætter en "Route Request" for første gang for en destinationsknude, indsættes altså følgende informationer i "Route Request Table":

- Adressen, knuden gerne vil finde en rute til.
- TTL'en, der blev brugt i "Route Request"-pakken.
- Tidspunktet, "Route Request"-pakken blev sendt på.
- Antallet af udsendte "Route Requests" gående på destinationsknuden indtil videre (initielt 1)
- Hvor længe, der skal gå, før en ny "Route Request" må sendes gående på denne destinationsknude.

Herefter udsendes med jævne mellemrum nye "Route Requests" for knuden (sålænge der stadig er pakker til den i "Send Buffer"). Udsendelsen af nye "Route Requests" skal overholde en "backoff"-algoritme – i specifikationen [JMH05] foreslås en algoritme, hvor timeout'en for nye "Route Requests" fordobles (indtil en vis grænse) hver gang, en ny "Route Request"-pakke sendes ud. Den initielle timeout foreslås sat til 500 ms, og må ikke overstige 10 sekunder, når den fordobles. I DSR-implementationerne Monarch [RMP00], DSR-UU [Nor05] og picoNet II [Son01] er denne strategi implementeret, og i Monarch og DSR-UU benyttes disse tal, mens den initielle timeout er sat ned til 250 ms i picoNet II med en øvre timeout-grænse på 1 sekund.

I f.eks. DSR-UU [Nor05] benyttes desuden et *maksimalt* antal "Route Requests". Dette stammer fra en ældre version af specifikationen [JMH05], der foreslår, at der maksimalt må udsendes 16 "Route Requests", hvorefter indgangen fjernes fra "Route Request Table" for internt skabte pakker. Den præcise betydning af dette beskrives i afsnit 3.4.2. I den nyeste version af specifikationen er denne begrænsning ikke længere med – "Route Requests" skal blive ved med at blive udsendt, indtil en rute findes, eller der ikke er flere pakker til destinationen i "Send Buffer" (hvorfra pakker netop foreslås fjernet efter 30 sekunder jvf. foregående afsnit).

"Route Request Table" for eksternt skabte pakker

"Route Request Table" for eksternt skabte pakker benyttes til at filtrere "Route Requests" fra, hvis disse allerede har været modtaget og behandlet én gang af knuden. Da "Route Requests" broadcastes rundt i et system, kan hver knude potentielt modtage en "Route Request"-pakke mange gange, men den skal kun videresendes første gang, den modtages, for at en "Route Request" ikke vil blive videresendt i et netværk for evigt. Konceptuelt gemmes følgende i datastrukturen:

```
[(original_source, [(id, target)])]
```

For hver afsender gemmes altså en liste over identifikationsnummeret fra "Route Request"-pakken (der jvf. afsnit 2.2.2 er unikt pr. afsenderadresse) og den adresse, "Route Request"-pakken forsøger at finde en rute til. Alle tre indgange i tabellen (oprindelig afsenderadresse, identifikationsnummer og destinationsadresse) skal være ens, før to "Route Request"-pakker anses for at være ens, og den sidst ankomne derfor ikke skal videresendes.

Bemærk til gengæld, at tabellen *ikke* bliver konsulteret, hvis knuden vil returnere en "Route Reply"-pakke. Hvis en knude modtager en "Route Request"-pakke adskillige gange, hvor destinationsknuden er den selv, skal den altså svare hver gang.

"Route Request Table" for eksternt skabte pakker må *ikke* være vilkårlig stor, da de oprindelige afsenderknuder af "Route Request"-pakkerne kan blive genstartet og derfor begynde at genbruge identifikationsnumre. Hvis en knude begynder at genbruge et identifikationsnummer, og alle knuderne omkring den stadig husker identifikationsnummeret fra tidligere i deres "Route Request Tables", vil afsenderknuden aldrig få et svar på sin forespørgsel. At sørge for at "Route Request Table" ikke må være vilkårligt stor vil minimere denne ulempe, men vil ikke helt eliminere den.

Specifikationen foreslår, at maksimalt 16 identifikationsnumre gemmes pr. knude for maksimalt 64 knuder, og disse tal benyttes også i DSR-implementationen DSR-UU [Nor05], mens de 16 identifikationsnumre gemmes for højst 16 knuder i Monarch [RMP00] og for højst 12 knuder i picoNet II [Son01]. Da DSR ifølge afsnit 2.1 er beregnet til at kunne benyttes i netværk med op til 200 knuder, og "Route Request"-pakker potentielt kan blive broadcastet ud i hele netværket, kan disse tal virke lave, men da DSR benytter en høj grad af caching af ruteinformation, der modtages i videresendte og i øvrigt overhørte pakker (se afsnit 2.2.5), er det dog muligt, at mange knuder helt kan undgå at igangsætte "Route Discoveries", og derfor kan tabelstørrelserne stadig være realistiske. Ligesom de øvrige variable (f.eks. bufferstørrelser og timeouts) kan denne variabel gøres til genstand for yderligere analyser.

2.3.6 Valgfrie optimeringer

I forbindelse med RD findes der forskellige valgfrie optimeringer:

"Cached Route Reply"

Hvis en knude modtager en "Route Request"-pakke, og knuden har en rute til "Route Request"-pakkens destinationsknude, kan knuden i stedet for at broadcaste pakken videre vælge at sende en "Cached Route Reply"-pakke (CRR) tilbage til "Route Request"-pakkens afsenderknude. Denne pakkes svar-ruteliste vil være stykket sammen af den rute, der er opsamlet indtil videre i "Route Request"-pakken, og den rute, der er fundet til destinationsknuden i knudens "Route Cache". Den nye, sammenstykkede rute må ikke indeholde løkker – hvis den gør det, skal knuden ignorere, at den kender en rute til destinationsknuden og blot broadcaste "Route Request"-pakken videre "som normalt".

"Route Reply Storm Prevention"

Hvis man vælger at implementere CRR (som ovenfor beskrevet), kan netværket omkring en afsender af en "Route Request"-pakke blive oversvømmet af CRR-pakker, hvis mange af knudens omkringliggende knuder kender en rute til den endelige modtagerknude for "Route Request"-pakken. Specifikt kan flere knuder modtage en "Route Request"-pakke fra den samme knude og svare på den samtidigt, hvorved pakkerne kan støde sammen i luften og hermed korrumperes eller tabes helt. For at undgå dette, kan en knude vælge at vente et kort øjeblik, før den sender en CRR til en oprindelig afsenderknude, og kan i mellemtiden forsøge at overhøre, om nogen når at sende en anden, kortere "Route Reply" tilbage til knuden. Hvis nogen gør dette, behøver knuden ikke sende CRR-pakken.

Tiden, der skal ventes, defineres efter følgende formel: $d = H * (h - 1 + r)$, hvor h er antal links i rutelisten, knuden ville have sendt tilbage til den oprindelige afsenderknude, r er et tilfældigt tal mellem 0 og 1, og H er en konstant, der minimum svarer til to gange den tid, det tager at transmittere en pakke trådløst i netværket. På denne måde fås en tilfældig ventetid, hvor knuder med den korteste rute som hovedregel vil svare først.

Videresendelse af piggybacked information

En "Route Request"-pakke kan indeholde piggybacked information – f.eks. en "Route Reply"-optionsheader. Hvis en knude svarer med en CRR-pakke i stedet for at viderebroadcaste pakken, kan dette medføre, at target-knuden for "Route Request"-pakken aldrig får den piggybackede information – dette gælder f.eks., hvis alle ruter fra afsenderknuden til destinationsknuden for "Route Request"-pakken går gennem den knude, der vil svare med en CRR-pakke.

En udvidelse til udvidelsen CRR går på, at hvis en "Route Request", der afføder en CRR, indeholder piggybacked information, skal denne information sendes i en ny pakke direkte til destinationsknuden for "Route Request"-pakken via en source-rute, der svarer til den rute, der blev returneret i CRR.

Bemærk, at hvis der findes flere ruter til destinationsknuden uden om knuden, der vil svare med en CRR-pakke, kan destinationsknuden dog modtage den piggybackede information både direkte fra knuden, der sendte en CRR-pakke, og piggybacket til "Route Request"-pakken.

Da der kan være et antal knuder, der kender en rute til destinationsknuden for en "Route Request"-pakke, kan antallet af pakketransmissioner i netværket ende med at blive forøget kraftigt, idet destinationsknuden vil modtage et antal pakker med videresendte, tidligere piggybackede data via hver sin rute fra forskellige knuder i netværket. Til at minimere dette problem kan samme metode som brugt i "Route Reply Storm Prevention" benyttes.

"Expanding Ring Route Requests"

Der må ifølge specifikationen [JMH05] gerne forekomme variationer i implementationen af udsendelsen af "Route Request"-pakker. F.eks. kan man implementere en "Expanding Ring Route Requests"-algoritme, hvor TTL'en starter på 1 og derefter fordobles, indtil en rute er fundet. På denne måde kan omgivelserne afsøges for den ønskede destinationsknude, uden at afsenderknudens "Route Request"-pakke nødvendigvis når ud til alle knuderne i hele netværket. Dette betyder dog også, at de knuder, der er tættest på afsenderknuden, får (og skal behandle) adskillige versioner af det, der i princippet er den samme "Route Request"-pakke bortset fra TTL-feltet (og identifikationsnummeret).

En mini-version af denne strategi er først at sende en "Route Request"-pakke ud med en TTL på 1 (for at afsøge om destinationsknuden skulle være blandt afsenderknudens naboer, eller en af naboerne skulle kende en rute til destinationsknuden som beskrevet ovenfor) og derefter sende en "Route Request"-pakke ud i netværket med den normale TTL, hvis der ikke modtages noget svar på den første "Route Request"-pakke.

2.4 "Route Maintenance"

Når en del af DSR-protokollen i en knude gerne vil transmittere en pakke via et link til en anden knude (f.eks. som følge af en videresendelse af en pakke til den næste modtager i en ruteliste), benyttes en procedure kaldet "Route Maintenance" (RM). I DSR er hver knude ansvarlig for, at pakker, de har transmitteret til en anden knude via et link, også bliver modtaget af denne naboknude. RM er den del af DSR-protokollen, der sørger for det praktiske i denne forbindelse.

2.4.1 Typer af bekræftelser

Når en knude har transmitteret en pakke over et link og skal være sikker på, at modtagerknuden har modtaget den, gøres dette, ved at knuden får en bekræftelse fra modtagerknuden. Tre forskellige typer bekræftelser findes i DSR-protokollen:

MAC-protokol-bekræftelse: Ofte i trådløse netværk er bekræftelsen på, at en pakke er nået frem til modtageren, en del af den underliggende netværkshardware (som beskrevet i

afsnit 1.5.3). Dette svarer til tilfældet, hvor det fysiske netværkssetup er af typen “Bidir-only” (se afsnit 2.2.5). I dette tilfælde skal RM ikke foretage sig yderligere for at få en bekræftelse på, at en pakke er nået frem – netværkshardwaren giver automatisk en besked tilbage, hvis pakken ikke kunne nå modtagerknuden.

Passiv bekræftelse: Hvis rutelisten i en pakkes “DSR Source Route”-optionsheader indeholder sekvensen af knuder [A,B,C], og A er i stand til at sætte sit netkort i “promiscuous mode” (d.v.s., at knuden instruerer netkortet til at sende *alle* netværkspakker videre til et højereliggende OSI-lag – og ikke som normalt kun de netværkspakker, der ifølge pakkernes respektive headere er tiltænkt knuden), kan A få en passiv bekræftelse af, at forsendelsen til B er lykkedes, ved at overhøre, at B sender pakken videre til C. Bemærk, at hvis C er den endelige modtager af pakken, må B ikke forvente en passiv bekræftelse for pakketransmissionen til C, idet B i dette tilfælde kan deducere sig frem til, at C ikke vil videresende pakken.

Eksplicit bekræftelse: En knude kan bede om en eksplicit bekræftelse ved at inkludere en “Acknowledgement Request”-optionsheader i en pakke. Modtagerknuden skal så sende en pakke med en “Acknowledgement”-optionsheader tilbage til afsenderknuden – enten direkte gennem linket, hvis det er et bidirektionelt link, eller gennem en anden (evt. multi-hop) rute, modtagerknuden finder i sin “Route Cache”.

Det er valgfrit, hvilken strategi en knude vil benytte for at få bekræftet, at en pakke, der blev transmitteret til en naboknude, nåede frem. Specifikationen [JMH05] foreslår en kombinationsstrategi, hvor en pakke gensendes et antal gange (med en vis timeout imellem – i specifikationen foreslået til 100 ms), indtil en passiv bekræftelse er modtaget. Kun hvis ingen passiv bekræftelse modtages, forsøges pakken gensendt et antal gange med en inkluderet “Acknowledgement Request”-optionsheader i sig, indtil et vist maksimalt antal transmissioner er forsøgt. I specifikationen [JMH05] foreslås det mere præcist, at man kun forsøger at få en passiv bekræftelse 1 gang, og derefter forsøger med eksplicitte bekræftelser, indtil der i alt er foretaget 3 pakketransmissioner. Bl.a. DSR-implementationen DSR-UU [Nor05] gør netop dette.

2.4.2 Fejlende links

Hvis en knude ikke er i stand til at bekræfte, at et link til en af knudens naboknuder virker, enten gennem MAC-protokolbekræftelse, passiv bekræftelse eller eksplicit bekræftelse (f.eks. gennem ovenstående kombinationsstrategi af passive og eksplicitte bekræftelser), fjernes alle ruter, der benytter linket, fra “Route Cachen”, og en pakke med en “Route Error”-optionsheader returneres til hver oprindelig afsender af pakker, som denne knude har sendt via linket, siden en bekræftelse sidst blev modtaget. Den oprindelige afsender står som bekendt som source-adressen i IP-headeren. Hvis nogle af pakkerne har samme oprindelige afsender, skal der dog kun sendes en enkelt “Route Error”-pakke tilbage til den.

Hver knude, der modtager en pakke med en “Route Error” i (evt. fordi knuden skal videresende den), fjerner alle de ruter, der benytter det pågældende link, fra deres “Route Cache”. Når den oprindelige afsenderknude til sidst modtager “Route Error”-pakken, forsøger denne ikke (som en del af DSR-protokollen) at gensende pakken. Det er op til et ovenliggende lag i knuden (f.eks. TCP) at opdage, at pakken ikke er nået frem, så den derfor skal gensendes. Hvis det ovenliggende lag vælger at gensende pakken, vil knudens DSR-lag benytte en anden rute til at gensende pakken, enten fordi knuden allerede har en anden rute i sin “Route Cache” eller gennem en ny RD. Hvis en ny RD igangsættes, skal “Route Error”-optionsheaderen, der blev modtaget, piggybackes til “Route Request”-pakken. Dette sker for at sikre, at knuder, der inkluderer udvidelsen CRR (se afsnit 2.3.6) og modtager “Route Request”-pakken, fjerner eventuel “gammel” information om den nu forældede rute fra deres “Route Cache”, før de

undersøger, om de kan returnere en CRR-pakke i stedet for at broadcaste "Route Request"-pakken videre.

Der er ikke i specifikationen [JMH05] specificeret i hvilken datastruktur og hvor længe, disse "Route Error"-optionsheaders skal gemmes, så knuden har dem, når nye "Route Discoveries" skal igangsættes. Dette vender jeg tilbage til i afsnit 3.5.4.

2.4.3 DSR-optionsheaders

Tre nye DSR-optionsheaders er blevet introduceret i det ovenstående:

"Acknowledgement Request"

En "Acknowledgement Request"-optionsheader består kun af et enkelt felt:

`(id)`

Det link, der ønskes bekræftet, er altid det link, pakken denne gang er blevet transmitteret via, og det kan derfor udtrækkes af rutelisten v.h.a. pointeren `segments_left` (se afsnit 2.2.2). Identifikationsnummeret skal være unikt pr. afsenderknude under de samme forudsætninger, som der gælder for "Route Request"-identifikationsnummeret (se afsnit 2.3.1).

"Acknowledgement"

En "Acknowledgement"-optionsheader består af følgende felter:

`(id, ack_src, ack_dest)`

Identifikationsnummeret kopieres fra "Acknowledgement Request"-optionsheaderen, mens bekræftelses-source-adressen (`ack_src`) og bekræftelses-destinations-adressen (`ack_dest`) er adresserne på henholdsvist afsenderen og modtageren af "Acknowledgement"-pakken.

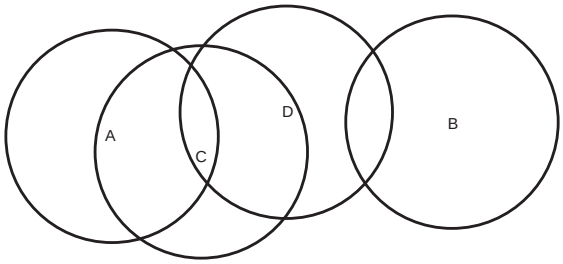
"Route Error"

En "Route Error"-optionsheader består af følgende felter:

`(error_type, salvage_counter, error_src, error_dest, type_spec)`

Der findes forskellige typer fejl, en "Route Error" kan dække over, og fejltyperne (angivet som `error_type`) kan udvides efterhånden som DSR-protokollen udvides. I standard-versionen er det kun nødvendigt at kigge på en enkelt af disse fejltyper:

"Node unreachable": Denne benyttes, når en knude ikke kan nås via et link i en ruteliste. I dette tilfælde vil fejl-source-adressen (`error_src`) være adressen på den knude, der opdager, at et link i en ruteliste ikke længere virker, og fejl-destinations-adressen (`error_dest`) vil være adressen på den knude, der oprindeligt gav den fejlagtige information – altså den knude, der oprindeligt sendte en pakke med en ruteliste, hvori et link ikke længere virker. `salvage_counter`-feltet beskrives nærmere i afsnit 2.4.6. `type_spec` (som kan oversættes til "fejltype-specifik information") indeholder kun et enkelt felt i dette tilfælde: Adressen på den knude, der ikke kunne nås. Modtageren af "Route Error"-pakken kan herefter sætte `error_src` og adressen i `type_spec` sammen for at få det link, der fejlede, og som derfor skal fjernes fra "Route Cache".



Figur 2.4: Fire knuder, A, B, C og D, med ringe, der angiver deres antennerækkevidder.

2.4.4 Eksempel

Antag, at der i et område er fire deltagende knuder i et DSR-netværk (som vist i figur 2.4), og at knuden A har ruten A→C→D→B i sin “Route Cache”, og vil benytte denne rute til at sende en pakke til B. Dette vil i det nedenstående blive brugt til at illustrere brugen af både passive og eksplicitte bekræftelser samt af “Route Error”-pakker.

1. Knuden A sender pakken til B tilføjet den følgende DSR-optionsheaderliste:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=2,
                       route=[C,D]))]
```

2. C modtager pakken og sender den videre til D. Før dette sker, ændres `segments_left`-feltet, så optionsheaderlisten nu ser ud som følger:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=1,
                       route=[C,D]))]
```

3. A overhører denne transmission fra C til D, og kan ud fra rutelisten og `segments_left`-feltet deducere sig til, at den selv var den forrige afsender af pakken. A kan derfor behandle denne pakkemodtagelse som en passiv bekræftelse på, at pakken er modtaget af C.

4. Knuden D hører *ikke* pakketransmissionen fra C (for eksemplets skyld).

5. Efter en passende timeout vil C gensende pakken til D – denne gang med et forsøg på at modtage en eksplicit bekræftelse. Den præcise DSR-optionsheaderliste bliver:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=1,
                       route=[C,D])),
 ("Acknowledgement Request", (id=27))]
```

6. D modtager denne gang pakken, og genererer en ny pakke til C med en eksplicit bekræftelse i. D har på nuværende tidspunkt ruten D→C i sin “Route Cache” (for eksemplets skyld). D vil nu benytte denne rute til at sende pakken til C. DSR-optionsheaderlisten ser ud som følger i den nye pakke:

```
[("DSR Source Route", (salvage_counter=0,
                       segments_left=0,
                       route=[])),
 ("Acknowledgement", (id=27,
                      ack_source=D,
                      ack_dest=C))]
```

7. Udover dette sender D den oprindelige pakke videre til B. Først fjerner den dog den "Acknowledgement Request"-optionsheader, som C tilføjede til pakken. Da pakken denne gang skal sendes over det sidste link i rutelisten, må D dog ikke regne med at modtage en passiv bekræftelse. Den tilføjer derfor sit eget ønske om en eksplicit bekræftelse til DSR-optionsheaderlisten, der nu vil se ud som følger:

   ```
   [("DSR Source Route", (salvage_counter=0,
                          segments_left=0,
                          route=[C,D])),
    ("Acknowledgement Request", (id=18))]
   ```

8. B er imidlertid uden for D's rækkevidde, så efter en passende timeout forsøger D at gensende pakken – med den samme DSR-optionsheaderliste som i sidste forsendelse.

9. D modtager stadig ikke et svar efter en passende timeout, så D forsøger en sidste gang at sende pakken.

10. Da D stadig ikke modtager en bekræftelse, sletter den linket til B fra sin "Route Cache", genererer en "Route Error"-pakke, og sender denne tilbage til den oprindelige afsender af pakken. D har på nuværende tidspunkt ruten D→C→A i sin "Route Cache" (for eksemplets skyld), og den vil nu benytte denne til at sende den nye pakke. DSR-optionsheaderlisten vil altså være:

    ```
    [("DSR Source Route", (salvage_counter=0,
                           segments_left=1,
                           route=[C])),
     ("Route Error", (error_type="Node Unreachable",
                      salvage_counter=0,
                      error_src=D,
                      error_dest=A,
                      type_spec=B))]
    ```

    `error_src` er altså den knude, der genererer "Route Error"-pakken, `error_dest` er den knude, der oprindeligt afsendte en pakke med en ikke-fungerende ruteliste i, og i den fejltype-specifikke information gemmes den knude, der ikke kunne nås via det fejlende link. Præcist hvilket link, der ikke længere fungerer, kan altså udtrækkes af "Route Error"-pakken som `error_src`→`type_spec`.

    "Route Error"-pakken sendes nu fra D til A via C på samme måde som i eksemplet i afsnit 2.2.3, mens den oprindelige pakke blot smides væk af D.

11. C modtager pakken og sender den videre til A som normalt. Som sædvanligt kigger den i optionsheaderne og forsøger at få så meget information fra disse som muligt, hvilket inkluderer, at den skal slette linket D→B fra sin "Route Cache".

12. A modtager "Route Error"-pakken, og fjerner ruten A→C→D→B (og øvrige ruter, der måtte indeholde linket D→B) fra sin "Route Cache", så disse ruter ikke vil blive benyttet (igen), hvis et ovenliggende OSI-lag (f.eks. TCP) opdager, at den oprindelige pakke ikke nåede frem til B, og forsøger at gensende den.

2.4.5 Foreslåede datastrukturer

Til brug for RM-algoritmen har hver knude tilknyttet tre forskellige datastrukturer:

“Network Interface Queue”

“Network Interface Queue” er blot en kø af pakker, der venter på at blive transmitteret af netværkshardwaren. Køens struktur afhænger af det interface, DSR-laget har til det underliggende OSI-lag, da implementationen f.eks. kan vælge at lade det underliggende OSI-lag hente pakker direkte ud af denne kø. Alternativt kan køen være styret helt af DSR-protokollen, så når det underliggende lag er klar til at sende en pakke, tager DSR-implementationen en pakke ud af “Network Interface Queue” og giver den til det underliggende lag.

Af denne grund foreslår specifikationen [JMH05] ikke en standardiseret datastruktur for “Network Interface Queue”, men den kan f.eks. blot være:

```
[(packet)]
```

“Maintenance Buffer”

“Maintenance Buffer” indeholder de pakker, der (via “Network Interface Queue”) er blevet sendt, men som RM endnu ikke har modtaget en bekræftelse tilbage på. Konceptuelt består bufferen af følgende datastruktur:

```
[(next_target, [(packet,
                 number_of_retransmissions,
                 last_retransmission_time)])]
```

Hvis RM, fordi der ikke modtages en bekræftelse jvf. afsnit 2.4.1, finder ud af, at en naboknude ikke kan nås via et link fra denne knude, fjernes alle pakker til denne næste modtagerknude fra både “Maintenance Buffer” og “Network Interface Queue”. For hver pakke sendes en “Route Error”-pakke tilbage til den oprindelige afsenderknude – dog højst én til hver oprindelig afsender.

Bufferen må gerne være begrænset i størrelsen. Hvis den er det, og RM vil gemme en pakke i bufferen, selvom den er fyldt op, slettes pakken uden at give en fejlmelding tilbage. At denne strategi skal benyttes, begrundes ikke i specifikationen [JMH05], men det betyder, at hvis en knude får bufferen fyldt op, koncentrerer den sig om at få gensendt og modtage bekræftelser for de ældste pakker i bufferen, mens nye pakker blot sendes en enkelt gang, uden at knuden nødvendigvis afventer bekræftelser for dem. Den omvendte strategi (FIFO) vil betyde, at *alle* links vil blive smidt ud, inden de når at blive testet færdigt, hvilket vil betyde, at ingen døde links rapporteres tilbage.

“Blacklist”

“Blacklist”-tabellen benyttes kun, når den fysiske netværkstype er “Bidir-only”. Tabellen benyttes til at give hver af en knudes naboknuder en attribut, der betegner, hvilken tilstand (bidirektionalitetsmæssigt set) linket mellem de to knuder efter knudens bedste overbevisning er i – d.v.s., om linket fra eller til en naboknude er unidirektionelt.

Konceptuelt består tabellen af følgende datastruktur:

```
[(neighbour_node, state, confirmed_time)]
```

Tabellens `state` (tilstand) kan have to værdier:

“Unidirectionality probable”: Når en pakke f.eks. ikke har kunnet sendes fra denne knude til en naboknude, selvom en broadcastet pakke tidligere er blevet modtaget fra naboknuden, sættes knuden i tabellen til “unidirectionality probable”.

"Unidirectionality questionable": Når en knude i et stykke tid har haft tilstanden "unidirectionality probable", sættes den i stedet til "unidirectionality questionable", da indgange i tabellen netop har med de grænsetilfælde at gøre, hvor selv små ændringer i knudernes placering kan betyde en ændring i linkets tilstand.

`confirmed_time` er blot det tidspunkt, knuden sidst har kunnet bekræfte, at en pakke ikke nåede frem til en naboknude (altså at tilstanden blev sat til "unidirectionality probable"), eller det tidspunkt, hvor tilstanden ændredes til "unidirectionality questionable".

Som beskrevet i afsnit 2.3.3 skal knuder i visse tilfælde reversere opsamlede rutelister for at finde en rute til en bestemt modtager. Den reverserede ruteliste benyttes til at teste, om alle links i ruten er bidirektionelle. Hvis en knude er i gang med at teste dette og opdager, at det første link i rutelisten ikke er bidirektionelt, indsættes en indgang i "Blacklist" for den knude med tilstanden "unidirectionality probable".

Indgangene i "Blacklist"-tabellen kan herefter benyttes til f.eks. at vurdere, om det er sandsynligt, at modtagne, broadcastede pakker blev modtaget via et bidirektionelt eller et unidirektionelt link. Dette benyttes f.eks. i processeringen af modtagne "Route Requests" (se afsnit 2.3.3).

Hvis knuden opdager, at den kan kommunikere bidirektionelt med en knude i "Blacklist", skal denne knude fjernes fra tabellen. Dette kan f.eks. være fordi knuden overhører, at naboknuden videre-broadcaster en "Route Reply"-pakke, som naboknuden ifølge den opsamlede ruteliste i "Route Reply"-optionsheaderen fik direkte fra denne knude.

Hvis der kun er begrænset plads i tabellen, og den bliver fyldt op, fjernes først de indgange, hvor det er længst tid siden, at indgangens tilstand blev sat til "unidirectionality questionable".

2.4.6 Valgfrie optimeringer

I forbindelse med RM findes der forskellige valgfrie optimeringer:

"Automatic Route Shortening"

Hvis en knude overhører en pakke, og (ved at undersøge "DSR Source Route"-optionsheaderen) opdager, at den selv er nævnt i rutelisten, men først som en senere modtager, kan den deducere sig frem til, at knuderne i netværket har bevæget sig så meget rundt i forhold til hinanden, at rutelisten nu kan forkortes. Hvis rutelisten f.eks. består af knuderne [A,B,C,D,E], og D overhører pakketransmissionen, der sendes fra B til C, kan D deducere sig frem til, at C kan elimineres fra rutelisten.

I dette tilfælde kan D vælge at sende en "Gratuitous Route Reply" tilbage til den oprindelige afsender A. En "Gratuitous Route Reply" er en "Route Reply", hvor rutelisten sættes sammen af den rute, pakken allerede har rejst gennem i netværket (den del af rutelisten, der allerede er "brugt"), og den del af rutelisten, der går fra D til den endelige destinationsknude – i dette tilfælde altså [A,B,D,E].

D skal i dette tilfælde ikke sende pakken videre til den næste knude i rutelisten (eller sende den videre til det ovenliggende OSI-lag, hvis den er den endelige modtager af pakken), som om den havde modtaget den på normal vis – i stedet smides pakken væk, og knuden venter på, at den modtager pakken fra C til videre behandling. Dette forklares ikke i specifikationen [JMH05] eller den tilhørende litteratur, der inkluderer en beskrivelse af "Automatic Route Shortening" ([JM96, JMB01]), men medfører, at antallet af pakketransmissioner i netværket bliver minimeret på bekostning af hastigheden med hvilken, pakken kunne have nået sin endelige modtager. Det kan også være gjort for at forsimple protokollen – en knude kunne i princippet blot huske på hvilke pakker, der har været sendt videre allerede efter en "Gratuitous Route Reply", og undgå at sende disse videre igen. Dette ville dog kræve endnu en datastruktur.

For at undgå at for mange “Gratuitous Route Replies” sendes tilbage til hver oprindelige afsenderknude og dermed oversvømmer disse, benyttes der dog en enkelt ny tabel i forbindelse med udvidelsen: “Gratuitous Route Reply Table” indeholder de “Gratuitous Route Replies”, der er genereret. Tabellen indeholder konceptuelt følgende datastruktur:

```
[(original_source, neighbour_node, timeout)]
```

`original_source` er den oprindelige afsender af den pakke, der fik denne knude til at returnere en “Gratuitous Route Reply”. Det er samtidigt den afsender, der bliver modtageren af “Gratuitous Route Reply”-pakken. Naboknuden er den knude, der transmitterede den pakke, der blev overhørt. Timeout’et angiver, hvornår indgangen igen må fjernes fra tabellen.

I specifikationen [JMH05] foreslås denne timeout sat til 1 sekund, og denne værdi benyttes i f.eks. DSR-implementationen DSR-UU [Nor05], mens “Gratuitous Route Reply” ikke er implementeret i hverken picoNet II [Son01] eller Monarch [RMP00].

Hver gang, en knude gerne vil sende en “Gratuitous Route Reply”, konsulteres tabellen for indgange, hvor den oprindelige afsenderknude og naboknuden matcher, og hvor timeout’en ikke er overskredet – kun hvis en sådan indgang ikke findes, må knuden sende en “Gratuitous Route Reply”-pakke tilbage til den oprindelige afsender.

Hvis netværket er af typen “Bidir-only” (se 2.2.5), skal “Gratuitous Route Reply”-pakken sendes tilbage til den oprindelige afsender ved at reversere den nye ruteliste fra den oprindelige afsender til denne knude. Dette sker som de øvrige steder, hvor dette foregår, for at checke, at alle links i rutelisten er bidirektionelle – er de ikke det, vil den oprindelige afsender ikke modtage pakken med den nye ruteliste, og der vil ikke være sket nogen skade.

Bekræftelser dækker fremtidige pakker

Hvis en knude som en del af RM skal have passiv eller eksplicit bekræftelse tilbage for en pakketransmission til en naboknude, er det tilladt for implementationen af DSR-protokollen at vælge, at bekræftelser ikke er nødvendige inden for en vis timeout, efter der sidst blev modtaget en bekræftelse på, at et link virkede. I specifikationen [JMH05] er denne timeout foreslået til 250 ms. Denne værdi benyttes i DSR-implementationen DSR-UU [Nor05], mens udvidelsen ikke benyttes i hverken picoNet II [Son01] eller Monarch [RMP00].

“Salvage Operations”

En “Salvage Operation” (SO) fungerer på følgende måde: Hvis en knude i forbindelse med RM opdager, at et link i en ruteliste i en pakke, den forsøger at videresende, ikke længere er gyldigt, undersøger knuden, om den har en anden rute til den endelige destinationsknude i sin “Route Cache”. Hvis dette er tilfældet, skal en “Route Error”-pakke sendes tilbage til den oprindelige afsender “som normalt”, men herudover videresendes pakken via den alternative rute som følger:

Hele rutelisten fjernes fra “DSR Source Route”-optionsheaderen og udskiftes med den fundne rute fra “Route Cachen”; mere præcist med listen af knuder fra denne knude (og inklusive denne, da denne knude ikke er den oprindelige afsenderknude) til (men ikke inklusive) den endelige destinationsknude (da adressen på denne knude som normalt står i IP-headeren). Denne fremgangsmåde hjælper knuder, der overhører denne pakke undervejs i dens videre fremsendelse, til ikke at cache forkert information, da de ved at se på værdien af salvage-feltet kan afgøre, om der er et link mellem IP-headerens source adresse og den første adresse i rutelisten eller ej. Udover dette tæller knuden salvage-tælleren i “DSR Source Route”-optionsheaderen op, og hvis det nye totale antal salvages, pakken herefter har været udsat for, ikke overstiger et vist maksimum, sendes pakken videre. I specifikationen [JMH05] foreslås det maksimale antal gange, en pakke må udsættes for en SO, sat til 15.

Hvis en senere knude i den nye ruteliste opdager, at en salvaget pakke igen indeholder en ruteliste med et link, der ikke længere virker, skal der igen sendes en "Route Error"-pakke tilbage, og en SO skal igen forsøges igangsat.

Knuder, der opdager, at et link i rutelisten ikke længere virker, skal p.g.a. dette kigge på pakkens salvage-felt, før de afgør, hvilken adresse, en "Route Error"-pakke skal sendes tilbage til – hvis salvage-feltet er nul, sendes en "Route Error"-pakke tilbage til den oprindelige afsender af pakken (sourceadressen i IP-headeren), ellers sendes "Route Error"-pakken tilbage til den første adresse i rutelisten, hvilket netop vil være adressen på den knude, der sidste gang udskiftede rutelisten i pakken.

Hvis udvidelsen CRR (afsnit 2.3.6) benyttes samtidigt med SO, bliver salvage-feltet også sat i "Route Reply"-pakker, der genereres i forbindelse med en CRR. Feltet sættes til en værdi, der ikke er nul – typisk den maksimale værdi, der findes for feltet, for at undgå, at pakken kan blive udsat for en SO.

At pakken ikke ønskes udsat for en SO i dette tilfælde, er ikke forklaret i specifikationen [JMH05] eller den tilhørende litteratur, hvor udvidelsen CRR i øvrigt behandles ([JMB01]), men kan skyldes, at hvis der viser sig at være en fejl i den ruteliste, knuden, der genererer CRR-pakken, laver, vil der sandsynligvis også være anden ugyldig information i knudens "Route Cache", hvorfor det ikke er ønskeligt, at CRR-pakken når frem til den oprindelige afsender af "Route Request"-pakken og tilføjer sådan mulig ugyldig information til dennes tabeller.

Den protokol, der er præsenteret i dette kapitel, kan man nu undersøge nærmere ved f.eks. at modellere den i et modelleringsværktøj og analysere på denne model. Dette er gjort i næste kapitel og frem.

3

Modellering af DSR-protokollen i Coloured Petri Nets

Jeg har modelleret Dynamic Source Routing-protokollen inklusiv de to udvidelser "Cached Route Reply" (CRR, se afsnit 2.3.6) og "Salvage Operations" (SO, se afsnit 2.4.6) i Coloured Petri Nets. Dette kapitel beskriver denne modellering. Undervejs beskrives fundne problemområder i protokollen, der gjorde det nødvendigt at lave afvigelser i modellen i forhold til specifikationen, præciseringer af specifikationen samt modelleringen af en ekstra optimering af protokollen.

3.1 Introduktion til Coloured Petri Nets

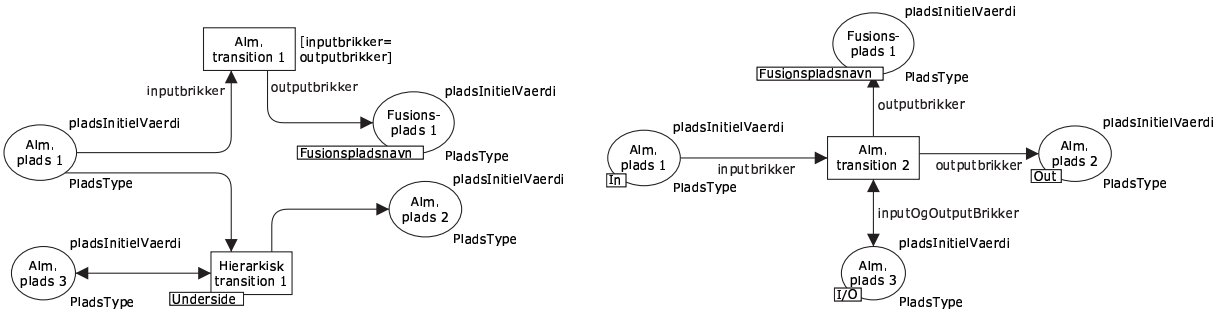
Coloured Petri Nets er kort sagt et grafisk modelleringssprog med en veldefineret semantik, der gør, at et Coloured Petri Nets-værktøj vil være i stand til at foretage en formel analyse af Coloured Petri Net-modeller (eller "CPN-modeller"). Den formelle definition af CPN kan ses i f.eks. [Jen92, Jen94].

En CPN-model består af "pladser" (cirkler) og "transitioner" (firkanter) med pile imellem. Pladserne repræsenterer dataopbevaring ("state"), mens transitionerne repræsenterer handlinger ("action") i modellen.

I tilknytning til den grafiske del (cirkler, firkanter og pile) benyttes det funktionelle programmeringssprog ML [Ull98]. Mere præcist har hver plads en bestemt ML-type, og på pladsen kan et antal ML-objekter ("brikker") af denne type opbevares. Typen kan være så simpel som en integer, eller det kan være en avanceret type sammensat af produkter, records, lister etc. af strenge, integers og booleans.

Hver transition kan være forbundet med et antal pladser. Når en transition vil foretage en handling (vil "fyre"), kan den tage imod input via pile fra pladser og give output tilbage via andre pile til andre (eller de samme) pladser. Både pilene og transitionen selv kan indeholde ML-kode, der angiver, hvornår en transition må fyre, og hvad der skal lægges på outputpladserne – f.eks. i forhold til det, der blev hentet fra inputpladserne.

I en model kan forskellige transitioner være i stand til at fyre samtidigt. Dette betyder, at når man vil "køre" sin model ("udføre en simulering" af den), kan den rækkefølge, hvori transitioner bliver fyret, være tilfældig. Dette gør Coloured Petri Nets god til at modellere pro-



(a) "Hovedside" (b) "Underside"

Figur 3.1: Eksempel på en hierarkisk CPN-model

cesser, der er "samtidige" i natur – f.eks. banksystemer, hvor modellerede kunder kan hæve og indsætte penge uafhængigt af hinanden (og informationen om dette gemmes i en delt database), togsystemer, hvor modellerede toge uafhængigt af hinanden kører fra modelleret station til modelleret station, eller netværksprotokoller, hvori modellerede deltagende maskiner arbejder uafhængigt af hinanden, men alligevel samarbejder om at få et netværk til at fungere.

En CPN-model kan bestå af mere end en "side" med hver sin model på. De forskellige modeller på de forskellige sider kan dog alligevel være sammenhængende, idet modellerne kan indeholde "fusionspladser", som på tværs af siderne altid indeholder den samme "markering" (d.v.s. de samme brikker). Fjernes eller tilføjes en brik fra eller til pladsen på én side, fjernes eller tilføjes brikken også fra eller til pladsen på de øvrige sider.

Alternativt kan CPN-siderne indgå i et hierarki, hvor en transition på én side dækker over en subside med sin egen model. Transitionen på supersiden kan være forbundet med et antal pladser, og disse pladser vil så også være repræsenteret på subsiden og kan manipuleres med transitioner herfra ganske som normalt. Fjernes eller tilføjes en brik fra eller til en af disse pladser på subsiden, fungerer dette præcist som ved fusionspladser: Brikken tilføjes eller fjernes også fra eller til den tilsvarende plads på supersiden.

I figur 3.1(a) kan en CPN-side, der viser, hvordan man grafisk angiver pladsers typer, initielle værdier og evt., at de er fusionspladser, ses. Desuden vises, hvordan man angiver input- og outputbrikker til transitioner, og hvordan man angiver transitioner. Teksten "[inputbrikker=outputbrikker]" er en såkaldt transitionsguard – en boolsk værdi, der skal evaluere til sand, før transitionen kan fyre. Nederst vises en hierarkisk transition, der er forbundet til tre pladser. I figur 3.1(b) vises den underside, den hierarkiske transition dækker over. Her ses hvordan de tre tilknyttede pladser får tilknyttet henholdsvist teksten "In", "Out" eller "I/O" alt efter, hvordan pilene er mellem pladsen og hovedsidetransitionen. Desuden er fusionspladsen genbrugt på undersiden.

Da Coloured Petri Nets har en veldefineret semantik, kan Coloured Petri Nets-værktøjer beregne et state space for en model – et state space er et samlet univers af alle de tilstande, en model kan være i med angivelser af hvilke tilstande, der via en fyring i modellen kan blive til hvilke tilstande. Dette gør, at man med en CPN-model er i stand til automatisk at få beregnet, f.eks. om en model indeholder deadlocks, løkker etc. Antallet af tilstande i en model, der repræsenterer f.eks. en netværksprotokol, vil dog ofte være så højt, at CPN-værktøjet ikke længere vil kunne analysere modellen – analysedelen af værktøjet fungerer kun for trivielt små modeller. Dette vender jeg tilbage til i kapitel 4.

CPN-modelleringssproget beskrives i detaljer i f.eks. [Jen92, Jen94, KCJ98].

Der findes mindst to værktøjer, der tillader en bruger at modellere CPN-modeller: Design/CPN [DC] og efterfølgeren CPN Tools [CT]. Den første version af Design/CPN blev

frigivet i 1989, men vedligeholdes og understøttes ikke længere. Den første version af CPN Tools blev frigivet i 1999, og er i dag både hurtigere til at udføre en simulering og til at beregne et state space end Design/CPN. Visse ting er dog endnu ikke porteret fra Design/CPN til CPN Tools, det gælder f.eks. automatisk generering af en grafisk repræsentation af et state space.

Modellen af DSR, der beskrives i de efterfølgende afsnit, blev oprindeligt modelleret i Design/CPN, men blev porteret til CPN Tools, efterhånden som der blev frigivet versioner af CPN Tools, der både var hurtigere og mere stabile end Design/CPN.

I modelleringen er den nyeste *interne* udviklingsversion af CPN Tools benyttet, da den nyeste *officielle* version af CPN Tools på modelleringstidspunktet (v1.4.0) var lettere ustabil og manglede nogle features, der fandtes i den interne version (f.eks. monitorer, der kunne stoppe en automatisk simulering, når f.eks. en bestemt tilstand blev nået i modellen). Den interne version har naturligvis også indeholdt diverse fejl, som jeg har rapporteret til CPN Tools-programmeringsgruppen. Kun de allermest alvorlige af disse nåede dog at blive rettet, inden modelleringen af DSR-protokollen var overstået, hvilket gør, at modellen, der præsenteres i det efterfølgende, indeholder nogle få "krumspring" på visse af CPN-siderne for at undgå disse fejl – f.eks. m.h.t. initialiseringen af pladser, der skal indeholde tilfældig data (se afsnit 3.4.1). "Krumspringene" kan gøre det sværere at læse CPN-siderne, men de er udformet, så de ikke påvirker funktionaliteten af CPN-modellen.

Jeg valgte at lave en model af DSR frem for at lave en regulær implementation af protokollen, fordi dette ikke ville afskære mig fra at kunne lave en række testsimuleringer/testkørsler, og fordi det ville gå hurtigere end at lave en "rigtig" implementation. I afsnit 3.9 vender jeg tilbage til, om dette var et fornuftigt valg.

Jeg valgte CPN som modelleringssprog, fordi CPN Tools er et etableret modelleringsværktøj, der tidligere har været brugt i forbindelse med modellering af bl.a. kommunikationsprotokoller jvf. bl.a. [KJJ04].

Et alternativt modelleringsværktøj, jeg kunne have valgt, er netværkssimulatoren ns-2 [NS2]. ns-2 er et af de mest brugte værktøjer i forbindelse med undersøgelser af MANET-protokoller [KCC05] og er bygget direkte til at være en simulator for netværksprotokoller. Den understøtter derfor ting, der er relevante i denne forbindelse, og som ikke er understøttet i CPN. I de efterfølgende afsnit er det f.eks. beskrevet, hvordan jeg har bygget CPN-sider, der simulerer effekten af overliggende og underliggende OSI-lag (i forhold til DSR-protokollen, se afsnit 2.1.2 og figur 1.4 på side 9). ns-2 indeholder allerede denne effektsimulering – endda i en mere avanceret version end den, jeg har modelleret, for hvor min model kun kan simulere simpel forsendelse af enkelte pakker, kan man i ns-2 f.eks. simulere brugen af forskellige varianter af TCP på ens modellerede netværksprotokol.

3.2 Oversigt over modellen

3.2.1 Opbygning af modellen

Modellen af DSR-protokollen består af i alt 28 CPN-sider. I dette afsnit vil jeg kigge på, hvordan CPN-siderne overordnet hænger sammen, mens jeg først i de efterfølgende afsnit vil gennemgå de enkelte dele af CPN-modellen i detaljer.

DSR er en netværksprotokol, der er beregnet til at blive implementeret på OSI-lag 3 (se afsnit 2.1.2 og figur 1.4 på side 9). I modellen modelleres det, at DSR-protokollen skydes som den nederste del af OSI-lag 3, så i modellen har netværkspakkebrikkerne allerede fået tilføjet IP-headerne, når DSR-delen af modellen får netværkspakkebrikkerne fra delen, der simulerer de højereliggende OSI-lag. DSR-delen behandler herefter netværkspakkebrikkerne og sender dem videre til den del af modellen, der simulerer de underliggende OSI-lag.

For at kunne modellere, hvad der foregår på lagene over og under modellen af DSR (altså hvordan modellen af de højereliggende lag beder modellen af DSR-protokollen om at sende

OSI-lag 3 og op:
- Hovedside: "OSILayer3AndUp"

DSR-laget mellem OSI-lag 2 og 3:
- Hovedside: "DSR_SendSide" (afsenderdelen af DSR)
  - DSR's hovedfunktionalitet
  - Route Discovery
  - Route Maintenance
- Hovedside: "DSR_RecvSide" (modtagerdelen af DSR)
  - DSR's hovedfunktionalitet
  - Route Discovery
  - Route Maintenance

OSI-lag 2 og ned:
- Hovedside: "OSILayer2AndDown"

Figur 3.2: CPN-modellen af Dynamic Source Routing delt op i OSI-lag – og med DSR-laget delt yderligere op i DSR's hovedbestanddele

netværkspakker, og hvordan modellen af de underliggende lag rent faktisk sender netværkspakkebrikkerne fra en simuleret knude til en anden), findes der yderligere 7 CPN-sider, der modellerer disse for DSR-delen af modellen eksterne omstændigheder.

Der findes i modellen med andre ord tre lag af CPN-sider: De, der simulerer effekten af OSI-lag 3 og op, de, der modellerer DSR-laget, og de, der simulerer effekten af OSI-lag 2 og ned. I figur 3.2 kan denne OSI-opdeling af CPN-siderne ses.

Her kan det desuden ses, at DSR-laget af CPN-modellen konceptuelt består af de tre hoveddele af DSR beskrevet i kapitel 2 – med hver del delt ud i afsender- og modtagerdelen af DSR.

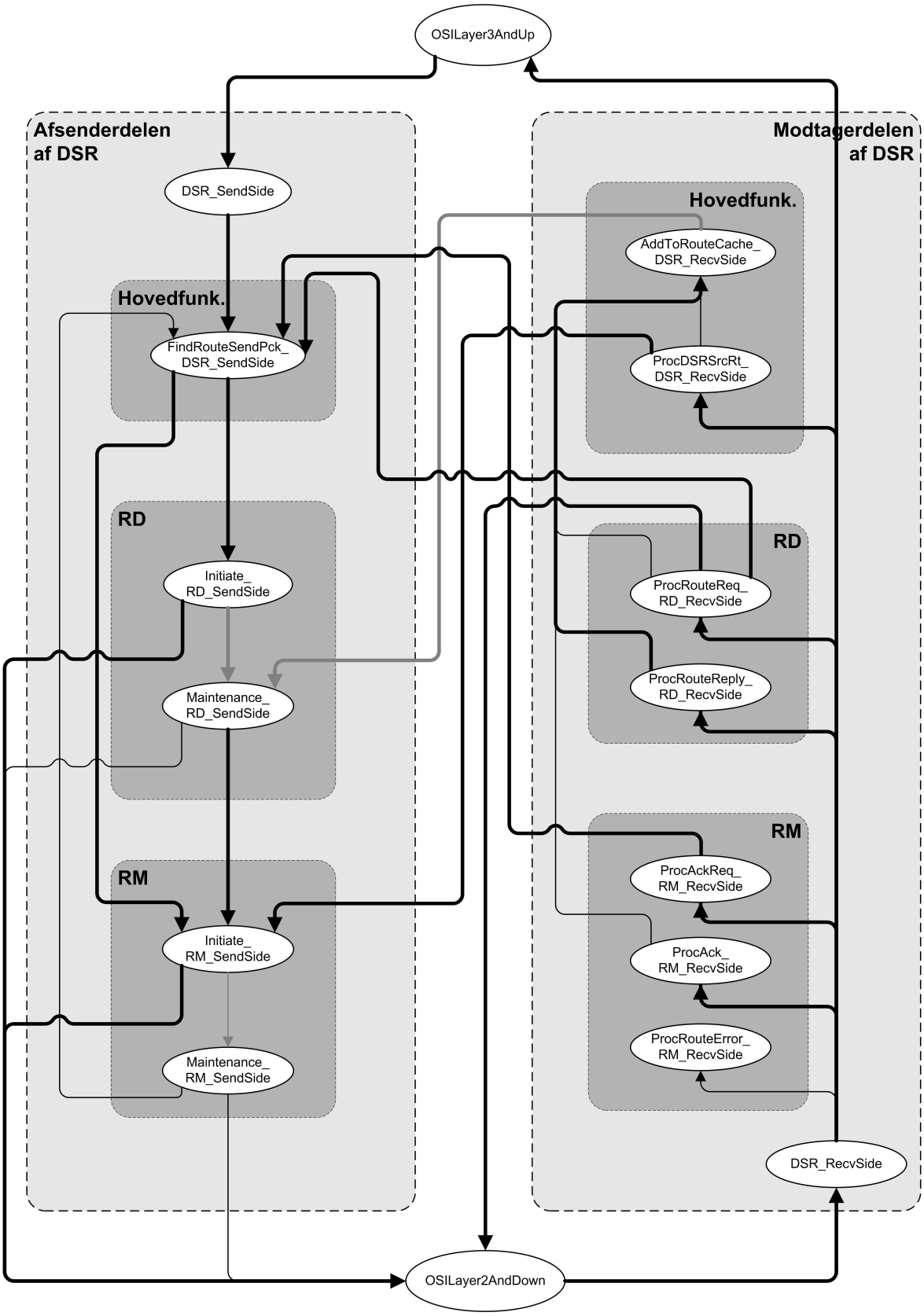
CPN-modellen er kun til dels bygget hierarkisk op. Mange steder er der i stedet brugt såkaldte fusionspladser – f.eks. for alle de datastrukturer, der er beskrevet i kapitel 2. Dette er gjort for at undgå, at der skulle bruges et meget stort antal pile mellem ofte brugte pladser og diverse undersider, hvad der ville have gjort de enkelte CPN-sider meget mere uoverskuelige. Det betyder, at mange sider, der kunne have været undersider af andre sider, i stedet "sværer frit" i CPN-modellen, fordi den eneste sammenhæng, de har med de øvrige CPN-sider, er gennem sådanne fusionspladser.

CPN Tools er i stand til at eksportere en oversigt, der viser den hierarkiske sammenhæng mellem siderne i en model. Som man kan se i figur 3.3 på næste side, er det dog svært at danne sig et overblik over den egentlige sammenhæng i DSR-modellen ud fra denne – bl.a. netop p.g.a brugen af fusionspladser i stedet for undersider.

I stedet findes i figur 3.4 på side 44 en konceptuel oversigt over hovedsiderne i modellen – inkl. hovedflowet mellem disse hovedsider. For at fremme overskueligheden er en del undersider og mindre væsentlige hjælpesider ikke vist i figuren.

Figuren må ikke forveksles med en CPN-model, selvom der benyttes cirkler og pile i den. Den skal udelukkende forstås som en sideoversigt. Pilene angiver et "flow" i modellen – d.v.s., at når en side processerer en modelleret netværkspakke, og der går en pil til en anden side, betyder det, at denne anden side som følge af dette kan blive sat i gang med at processere den samme (eller en anden) modelleret netværkspakke. I figuren angiver tykke pile "hovedflowet", mens tynde pile angiver "mindre vigtigt flow" eller flow, der ikke forekommer så ofte. Dette uddybes i det nedenstående.

Figur 3.3: CPN-sidesammenhænge: DSR-modellen som eksporteret fra CPN Tools



Figur 3.4: CPN-sidesammenhænge: Hovedflowet mellem hovedsiderne i DSR-modellen

Afsenderdelen af modellen

Figuren starter i toppen. Når en simuleret netværkspakke bliver givet til "DSR_SendSide" fra et højereliggende OSI-lag (modelleret i "OSILayer3AndUp"), sender denne pakken videre til "FindRouteSendPck_DSR_SendSide", der undersøger, om den simulerede knude, der skal sende pakken, har en rute til modtageren af pakken. Har den det, gives pakken til "Initiate_-RM_SendSide", der igangsætter en RM – ellers skal der først findes en rute. I det tilfælde gives pakken i stedet til "Initiate_RD_SendSide", der igangsætter en RD.

I praksis kalder "FindRouteSendPck_DSR_SendSide" "Initiate_RD_SendSide" som en hierarkisk underside, mens simulerede pakker til "Initate_RM_SendSide" i stedet gives til denne via en fusionsplads. Når der er flere indgange til en CPN-side, er fusionspladsløsningen typisk valgt, idet dette gør CPN-modellen mindre (d.v.s. med færre sideinstanser) og dermed hurtigere at benytte (betydningen af hvilket, jeg vender tilbage til i afsnit 3.9). Dette er med andre ord et af de steder, hvor man ikke kan se den rette struktur af min model i CPN Tools-eksportoversigten i figur 3.3 på side 43.

"Initiate_RD_SendSide" genererer og giver en simuleret "Route Request"-pakke til "OSI-Layer2AndDown", der bl.a. modellerer afsendelsen i luften til andre modellerede knuder. Pakken selv gemmes i "Send Buffer", hvorfra den læses af "Maintenance_RD_SendSide", når en sådan rute findes. I figuren repræsenteres dette med en grå pil mellem "Initiate-" og "Maintenance_RD_SendSide" – den anden grå pil ind i "Maintenance_RD_SendSide" angiver netop, at en sådan rute er modtaget/overhørt af modtagerdelen af DSR (hvilket jeg vender tilbage til). Pilene er gjort grå, når det ikke er pakkebrikker, der overføres mellem siderne, men flowet går fra en side til en anden, fordi en datastruktur opdateres – i dette tilfælde altså "Send Buffer" for den første grå pil, "Route Cache" for den anden.

Når en rute er fundet, kommer "Initiate_RM_SendSide" i brug: Den simulerede netværkspakke bliver givet til "OSILayer2AndDown", og en kopi af pakken gemmes i "Maintenance Buffer". Hvis den ikke først når at blive fjernet herfra af modtagerdelen (f.eks. når der modtages en passiv bekræftelse), gensendes den herefter fra "Maintenance_RM_SendSide". Også her repræsenteres brugen af en datatruktur ("Maintenance Buffer") af en grå pil mellem "Initiate-" og "Maintenance_RM_SendSide". Denne er samtidigt tynd, idet flowet ikke vil skifte, hvis routingen er succesfuld – i så fald skal der ikke ske genudsendelser. "Maintenance_RM_SendSide" kan desuden generere en "Route Error"-pakke og give denne til "Find-RouteSendPck_DSR_SendSide" til forsendelse til den oprindelige modellerede afsender af den fejlende pakke. Både gensendelsen og returpakkeforsendelsen vil dog ikke ske, hvis pakken rent faktisk når frem (og der f.eks. modtages en passiv bekræftelse), derfor er dette også angivet med tynde pile i modellen.

Modtagerdelen af modellen

Når "OSILayer2AndDown" har fundet ud af, at en pakke skal modtages af en modelleret knude, gives pakken til "DSR_RecvSide", der går alle optionsheaders i den igennem og processerer disse en efter en. Hver type optionsheader bliver behandlet af hver sin CPN-side, som efter endt processering lader flowet gå tilbage til "DSR_RecvSide", så den næste optionsheader kan processeres. Dette returflow er dog ikke vist i figuren af overskuelighedsårsager.

Alle pakker (undtagen "Route Request"-pakker) indeholder en "DSR Source Route"-optionsheader. Findes en sådan, modificerer "ProcDSRSrcRt_DSR_RecvSide" pakken og giver den til "Initiate_RM_SendSide" (altså til afsenderdelen af DSR-modellen) til (simuleret) forsendelse til næste modtager jvf. rutelisten. Samtidigt udtrækkes al den ruteinformation, det er muligt, fra pakken v.h.a. "AddToRouteCache_DSR_RecvSide". Information herfra gemmes i "Route Cache", der konstant bliver overvåget af "Maintenance_RD_SendSide", så flowet kan også her gå tilbage til afsenderdelen af DSR-modellen (via den grå pil).

Hvis en modelleret modtaget pakke indeholder en "Route Request"-optionsheader, kan der ske flere forskellige ting i "ProcRouteReq_RD_RecvSide", der skal processere denne. Hvis

den modtagende knude er den endelige destination for pakken, genereres en modelleret "Route Reply"-pakke, som gives til "FindRouteSendPck_DSR_SendSide". Bemærk her, at figur 3.4 på side 44 udelukkende viser, hvad modellen gør i "Frequently-unidir"- og "Mostly-bidir"-netværk – i "Bidir-only"-netværk skal modellen i stedet give pakken til "Initiate_RM_SendSide", da der her *skal* bruges reverserede rutelister til at transportere pakken ad (jvf. afsnit 2.3.3), og RD derfor ikke må bruges. Af overskuelighedsårsager vises dette ikke i figuren.

Hvis "ProcRouteReq_RD_RecvSide" derimod skal simulere en viderebroadcastning af den modtagne pakke, gør den dette ved blot at modificere pakken og give den til "OSILayer-2AndDown". Også "ProcRouteReq_RD_RecvSide" høster ruteinformation fra optionsheaderen v.h.a. "AddToRouteCache_DSR_SendSide".

Hvis "ProcRouteReply_RD_RecvSide" får en pakke (fordi den indeholder en "Route Reply"-optionsheader), gøres der ikke andet end at høste ruteinformation fra optionsheaderen. Pilen til "AddToRouteCache_DSR_RecvSide" er her gjort tyk, idet den modtagne pakke kan være svar på en tidligere udsendt "Route Request"-pakke og dermed være en del af det, man kan betegne som hovedflowet i DSR-modellen. Det kan netop gøre, at afsenderdelen af RD bliver i stand til at sende en pakke videre til RM.

I modellen prøver afsenderdelen af RM først at sende en pakke og vente på en passiv bekræftelse (pånær i det sidste hop jvf. afsnit 2.4.1). Eksplicitte bekræftelser benyttes derfor ikke kun i specialtilfælde, og tykke pile er brugt til processeringen af to af de tre modellerede RM-optionsheaders. Modtages der en "Acknowledgement Request"-pakke, sender "ProcAckReq_RM_RecvSide" en "Acknowledgement"-pakke til "FindRouteSendPck_DSR_SendSide" (igen kun i ikke-"Bidir-Only"-netværk – i disse vil pakken i stedet blive givet direkte til "Initiate_RM_SendSide"). Modtages en "Acknowledgement"-pakke, fjerner "ProcAck_RM_RecvSide" den tilsvarende pakke fra "Maintenance Buffer", så "Maintenance_RM_SendSide" ikke forsøger at genudsende den (dette er ikke visualiseret i figuren). Modtagelsen af passive bekræftelser er i princippet også en del af RM, men af praktiske grunde foregår dette på "ProcDSRSrcRt_DSR_RecvSide"-siden (da der netop ikke er en seperat optionsheader til passive bekræftelser), men dette fungerer i øvrigt som modtagelsen af en eksplicit bekræftelse.

Som den sidste mulighed kan "DSR_RecvSide" sende den modellerede netværkspakke videre til siden "OSILayer3AndUp", hvis pakken er tiltænkt den knude, der processerer pakken.

I afsnit 3.3 og frem gennemgås de i figur 3.4 på side 44 viste CPN-sider (og deres ikke viste under- og hjælpesider) i flere detaljer.

3.2.2 CPN-modellens repræsentation af data

Før de enkelte CPN-siders virkemåde gennemgås i detaljer, er det relevant at kigge lidt på, hvordan data generelt repræsenteres på siderne. Dette gøres i dette afsnit.

Modellering af knuder og IP-pakker

En "knude" er i DSR-modellen blot modelleret som en streng, der angiver navnet på knuden:

```
colset Node_ = string;
```

(Grundet en bug i den på modelleringstidspunktet nyeste interne version af CPN Tools kan colsettet ikke hedde "Node" (den engelske term for en knude) a.h.t. state space-værktøjet).

I CPN-modellen får knuderne navnene "A", "B", "C" o.s.v. En alternativ navngivningsstrategi kunne have været **colset** Node_ = **product** int∗int∗int∗int, d.v.s. at knuderne blev repræsenteret med IP-adresser. Dette ville have været i tråd med, at jeg (som de næste afsnit viser) forsøger at lave en så nøjagtig model af DSR som muligt, men repræsentationen af IP-adresser hører med til et andet OSI-lag end DSR-protokollen, så her har jeg valgt midlertidigt at gå bort fra dette princip. Der benyttes strenge for at gøre det så nemt som muligt at overskue, hvad der står på de enkelte pladser.

```
colset Identification = int with 1..100000;
colset TTL = int;
colset UserData = union SOMEDATA : STRING + NODATA;

colset DSROptionsHeaderFixed
= union DSRSourceRoute
      + RouteRequest
      + RouteReply
      + AcknowledgementRequest
      + Acknowledgement
      + RouteError;

colset Route = list Node_;

colset RouteRequestOption
= record id: Identification
       * target: Node_
       * routeSoFar: Route;

(* Samme for de øvrige optionsheaders: DSRSourceRouteOption etc. *)

colset DSROptionsHeaderVariable
= union dsrSourceRoute : DSRSourceRouteOption
      + routeRequest : RouteRequestOption
      + routeReply : RouteReplyOption
      + acknowledgementRequest : AcknowledgementRequestOption
      + acknowledgement : AcknowledgementOption
      + routeError : RouteErrorOption;

colset DSROptionsHeader
= product DSROptionsHeaderFixed * DSROptionsHeaderVariable;

colset DSROptionsHeaderList = list DSROptionsHeader;

colset DebugHeader
= record debugId: Identification * ...;
  (* øvrige debug-felter er uvigtige i denne sammenhæng *)

colset IPPacket = record id: Identification
                       * ttl: TTL
                       * src: Node_
                       * dest: Node_
                       * dsr: DSROptionsHeaderList
                       * debug: DebugHeader
                       * data: UserData;
```

Figur 3.5: Modellering af en IP-pakke i CPN-modellen – med detaljerne for en enkelt af DSR-options-headerne ("Route Request") vist

En netværkspakke modelleres som en record, som det kan ses i figur 3.5 på forrige side. Som beskrevet i kapitel 2 er en IP-pakke (vist nederst i figuren) i DSR-sammenhænge blot at regne som en "normal" IP-pakke (jvf. afsnit 2.2.1) med et ekstra felt skudt ind: DSR-optionsheaderfeltet. DSR-optionsheaderfeltet skal ifølge specifikationen [JMH05] deles i to dele: En fast del, der angiver den DSR-optionstype, denne optionsheader indeholder, og en variabel del, der indeholder de optionsafhængige data. Da jeg benytter unions (se figur 3.5 på forrige side) til den variable del, og disse også navngiver de enkelte optionstyper, er det strengt taget ikke nødvendigt at have den faste del med, men for at lægge mig så tæt op af specifikationen [JMH05] som muligt, har jeg valgt at gøre det alligevel.

En enkelt af optionsheaderne ("Route Request") er specificeret fuldt ud i figur 3.5 på forrige side. Denne (og de øvrige) indeholder præcist de informationer, der er beskrevet i kapitel 2 – den præcise specifikation kan ses i appendix B.2.

Ud over DSR-optionsheader-IP-feltet tilføjer jeg et andet, ekstra felt til modellen af en IP-pakke: Et "debug"-felt. Dette felt benyttes hovedsageligt til at kunne gemme informationer om, hvad der rent faktisk sker i modellen med diverse netværkspakkebrikker – f.eks. hvilken knude, pakkebrikken bliver simuleret sendt fra og til. Når modellen af OSI-lag 3 og op beder DSR-lagsmodellen om at sende en pakke (d.v.s. den giver en pakkebrik til DSR-laget af CPN-modellen), gemmer OSI-lag 3-modeldelen altid et unikt debug-id i denne debug-header. DSR-laget manipulerer aldrig med debug-headeren, men når nye pakker bliver genereret i DSR-laget (f.eks. nye "Route Request"-pakker), kopieres denne debugheader med over i den nygenererede pakke. Det betyder, at man i debugheaderen på en vilkårlig netværkspakkebrik under en simulering altid kan se, hvilken oprindelig netværkspakkebrik en pakkebrik "udsprang" fra. Dette benyttes hovedsageligt i kapitel 5 til at udtrække informationer om antal pakketransmissioner for hver genereret pakkebrik etc.

Repræsentation af data specifikt for en knude

Konceptuelt skal samlingen af CPN-sider repræsentere et antal knuders opførsel i et DSR-netværk. Da antallet af knuder i DSR-netværket er variabelt, kan man dog ikke have et sæt CPN-sider for hver knude. Derfor benyttes hver enkelt CPN-side til at modellere alle knudernes opførsel i en del af protokollen på én gang. Dette gøres således, at når en transition fyrer, fyrer den kun for en enkelt knude ad gangen, og alle data, der ligger på pladserne, er splittet op i seperate multisets for hver knude. Typisk forekommer to forskellige typer for pladserne:

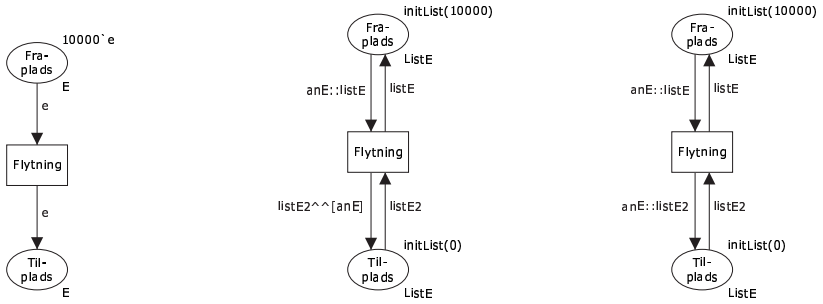
`(knude,data)`: Hvert stykke data, der ønskes gemt på en plads for en knude, repræsenteres som en enkelt brik.

`(knude,[data])`: Alle stykker data, der ønskes gemt på en plads for en knude, gemmes i en samlet liste.

Når jeg i de efterfølgende afsnit skriver, at en modelleret knude gemmer noget data i f.eks. "sin Route Cache", menes der netop, at det gemmes som en `(knude,data)`-brik, hvis der benyttes enkeltbrikker, og i `data`-listen i brikken tilknyttet `knude`, hvis der benyttes lister.

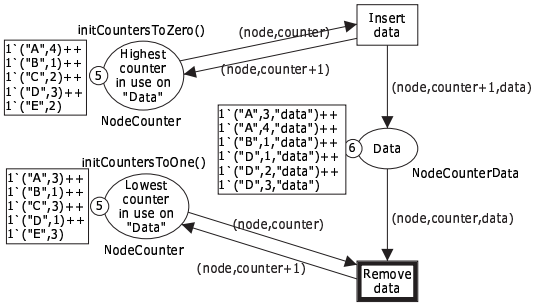
Listemetoden benyttes på langt den overvejende del af pladserne for implicit at bibeholde rækkefølgen på de dataklumper, der er tilknyttet knuden på pladsen. Uheldigvis har metoden et problem: CPN Tools er langsommere til at processere pladser med lange lister, end den er til at processere pladser med mange brikker.

Dette kan illustreres med en test: Først forsøger jeg at flytte 10000 enhedsbrikker en efter en fra en plads til en anden plads. Figur 3.6(a) på næste side viser en eksempelmodel, der kan gøre netop dette. Gentagne tests viste, at udførelsen af 10000 steps i denne model på en 3 GHz maskine tager ca. 1 sekund. Herefter forsøgte jeg at flytte 10000 listeenhedselementer en efter en fra en liste på en plads til en liste på en anden plads som illustreret i figur 3.6(b) på næste side. (`initList(`n`)` er her blot en funktion, der laver en liste bestående af n enhedselementer). Gentagne tests viste, at denne model tager ca. 14 sekunder at udføre.



(a) 10000 enhedsbrikker (b) 10000 listeenhedselementer (c) 10000 listeenhedselementer (med reversering af rækkefølgen)

Figur 3.6: Eksempelmodeller, der illustrerer flytningen af 10000 elementer



Figur 3.7: Eksempel på brug af subpladser til at understøtte eksplicit lagring af rækkefølgen af brikker på pladsen "Data"

En logisk årsag til dette kan være, at lister generelt repræsenteres som linkede lister i ML (jvf. [Ull98, side 85]), altså som par af en pointer til et element og en pointer til det næste par. Dette gør, at indsættelse af et element først i en liste (eller udtagelse af et element fra først i en liste) er en operation, der tager konstant tid, mens indsættelse af et element *sidst* i en liste tager tid proportionalt med længden af listen. I figur 3.6(b) bliver elementer netop indsat i slutningen af listen på pladsen "Til-plads". Hvis dette også gælder CPN-værktøjet, er denne (for at flytte 10000 listeelementer) derfor nødt til at bruge i omegnen af $\sum_{i=1}^{10000} i \approx 50.000.000$ operationer i stedet for blot i omegnen af 10000 som ved enhedsflytningerne vist i figur 3.6(a). Figur 3.6(c) er en modificering af figur 3.6(b), hvor der kun indsættes og udtages fra starten af listerne. 10000 steps i denne model viser sig dog konsekvent at tage ca. 12 sekunder, så forskellen på de to måder at flytte listeelementer på er minimal i forhold til forskellen på at benytte brikker frem for lister. Problemet kan derfor ikke løses ved blot at lade hver anden plads i DSR-modellen have reverserede lister.

En anden løsningsmodel er at ændre pladstyperne til formen `(knude,tæller,data)`, hvor tælleren giver en eksplicit rækkefølge af de stykker data, der ligger på pladsen. For at dette skal kunne fungere, er det dog nødvendigt, at transitioner, der lægger noget på pladsen, kender den foreløbige maksimale tællerværdi, der er benyttet (så knuden kan forhøje værdien og benytte den nye værdi til den nye brik), og at transitioner, der vil fjerne brikker fra pladsen, kan finde ud af hvilken brik, der indeholder den mindste tæller. Dette er ikke muligt uden at gemme disse to værdier særskilt – f.eks. i to subpladser tilknyttet transitionerne, der vil hente eller gemme data på en sådan plads. Dette er illustreret i figur 3.7. Da dette skal gøres for alle transitioner, der er knyttet til denne type pladser, vil det gøre CPN-modellen unødigt kompliceret, og jeg har derfor valgt kun at gøre det et enkelt sted (se afsnit 3.6.4).

```
(* Modellering af et timestamp *)
colset Timestamp = int;

(* En knude gemmer i sin Send Buffer for hver destination information
   om pakker til denne destination og indsættelsestidspunktet: *)
colset IPPacketTime = product IPPacket * Timestamp;
colset IPPacketTimeList = list IPPacketTime;

(* En knudes SendBuffer-indgang gående på en bestemt destination: *)
colset NodeIPPacketTimeList = product Node_ * IPPacketTimeList;

(* En SendBuffer for en knude: *)
colset SendBuffer = product Node_ * NodeIPPacketTimeList;
```

Figur 3.8: Eksempel på modellering af en af datastrukturerne: "SendBuffer"

Datastrukturer

I DSR-protokollen er en række datastrukturer defineret som beskrevet i afsnit 2.2.4, 2.3.5 og 2.4.5. Fælles for langt de fleste af disse er, at de ifølge specifikationen [JMH05] skal indexeres efter en destinationsknude. F.eks. skal ruterne i "Route Cache" (når disse gemmes som komplette ruter) sorteres efter hvilken destinationsknude, de er tiltænkt, så når en rute til en bestemt destinationsknude skal findes af en knude, kan en algoritme hurtigt finde ud af, om den har en eller flere ruter til destinationsknuden, og kan vurdere disse gemte ruter til destinationen i forhold til hinanden.

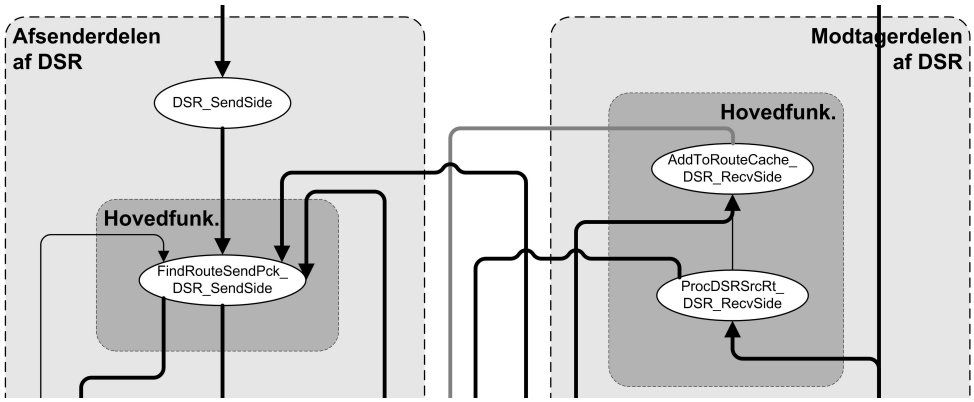
For at modellere dette benyttes der for disse datastrukturer pladstyper af formen `(knude, destination, [data])`. Pladserne har initielt brikker af typen `(knude, destination, [])` for alle par af knuder i systemet. Dette betyder, at jeg i modellen bryder en enkelt regel i forhold til DSR-protokollens forudsætninger: Jeg går ud fra, at alle modellens knuder ved præcist hvilke andre knuder, der er i modellen. Hvis man ikke opretter alle brikker fra starten, er det ikke i CPN Tools muligt at forespørge, om en brik for et par af `(knude, destination)` findes eller ej.

En mere korrekt modellering ville være at lade pladstyperne være på formen `(knude, [(destination, [data])])`. Dette ville give den fordel, at en knude ikke skulle vide noget om andre knuder på forhånd, og at den stadig kunne søge igennem en liste for at finde ud af, om den har data om en destination eller ej. For at minimere de føromtalte tidsproblemer med lister i CPN Tools, har jeg dog valgt ikke at benytte denne type pladser – jeg vurderede, at det var vigtigere at have en model, det var praktisk muligt at benytte, end at opfylde kravet om, at en modelleret knude ikke på forhånd kender til hvilke andre knuder, der findes i modellen.

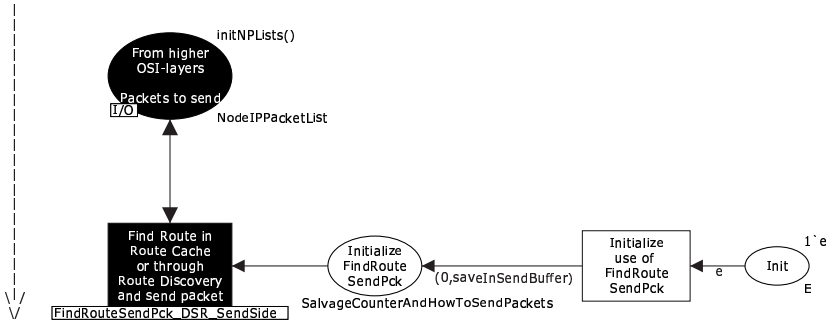
I figur 3.8 kan et eksempel på en modelleret datastruktur ("SendBuffer") ses. Denne er (som det kan ses) modelleret, så den svarer til den foreslåede datastruktur beskrevet i afsnit 2.3.5. Dette gælder også de øvrige datastrukturer beskrevet i kapitel 2 pånær "Network Interface Queue", som ikke har en foreslået datastruktur (se afsnit 3.5.1 angående dennes modellering), og "Route Cache", der har flere forskellige foreslåede datastrukturer (se afsnit 3.3.1 angående dennes modellering). Den præcise definition af alle DSR-modellens datastrukturer kan ses i appendix B.2.

3.3 Modellering af DSR's hovedfunktionalitet

De enkelte sider i CPN-modellen af DSR-protokollen kan nu gennemgås mere detaljeret. Dette afsnit vil beskæftige sig med den første af de tre hoveddele af DSR-protokollen, selve "DSR's



Figur 3.9: CPN-sidesammenhænge: Hovedflowet mellem hovedsiderne i "DSR's hovedfunktionalitet" (udsnit af figur 3.4 på side 44)



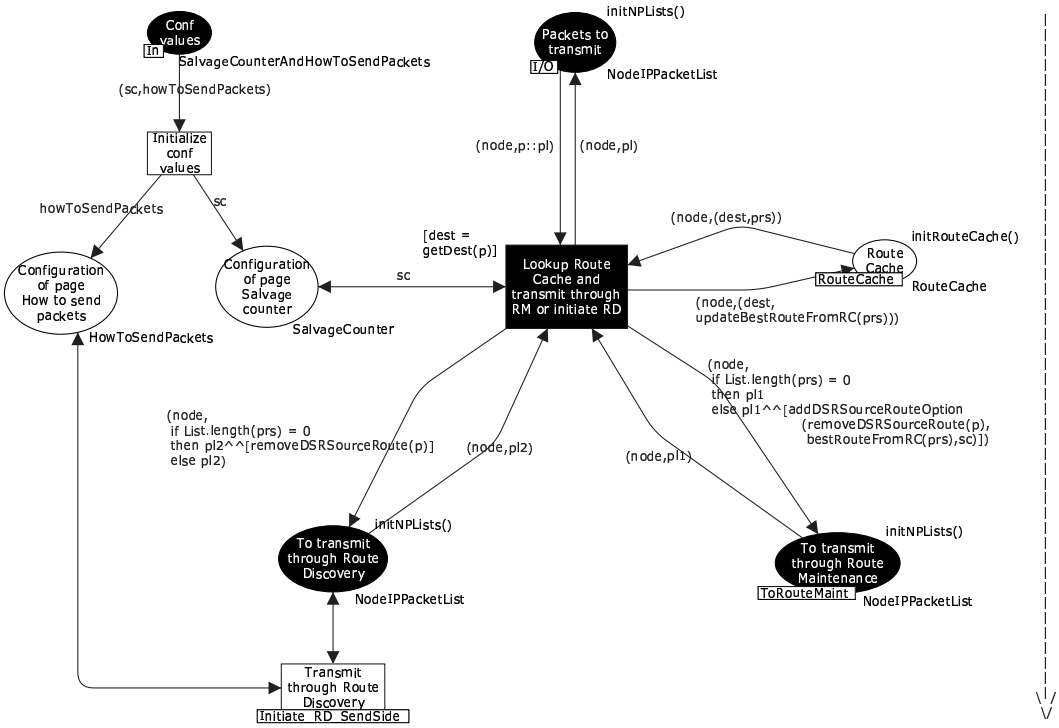
Figur 3.10: DSR-modellens CPN-side "DSR_SendSide"

hovedfunktionalitet" (som beskrevet i afsnit 2.2). RD og RM vil blive behandlet i henholdsvist afsnit 3.4 og 3.5. I figur 3.4 på side 44 kan de vigtigste CPN-sider knyttet til "DSR's hovedfunktionalitet" ses i oversigt. De relevante sider er genvist i figur 3.9 – men uden sammenkædningen med resten af CPN-modellen, som den oprindelige figur viser. Udover siderne i denne figur består "DSR's hovedfunktionalitet" af en enkelt hjælpeside mere: "MaintainRouteCache_DSR", der vedligeholder "Route Cache"-datastrukturen. Funktionaliteten i alle disse sider vil blive gennemgået i det nedenstående.

3.3.1 Modelleret afsendelse af pakker i "DSR's hovedfunktionalitet"

DSR's afsenderhovedside kan ses i figur 3.10. Flowet på siden er ganske simpelt: Når det modellerede højereliggende OSI-lag giver en netværkspakkebrik til DSR-laget for at modellere, at en knude skal sende en pakke til en anden knude, får DSR-protokollen i modellen pakkebrikken via pladsen "From higher OSI-layers – Packets to send". Disse pakkebrikker bliver blot sendt videre til undersiden "FindRouteSendPck_DSR_SendSide", der kan ses i figur 3.11 på næste side. Allerede her kan to specielle ting for CPN-siderne i DSR-modellen ses:

Stiplet pil: På CPN-siden kan en stiplet pil ses yderst til venstre. Denne pil angiver "flowet" på siden – d.v.s., at på "DSR_SendSide" vil (pakke-)brikker generelt gå nedad i figuren. Denne pil er medtaget på de fleste af CPN-side-figurerne, da brikkerne ofte gemmes på pladserne i lister, og "de normale" pile på siderne derfor ofte vil gå i begge retninger mellem en plads og en transition (f.eks. som en pil, der henter en liste ud af en plads til



Figur 3.11: DSR-modellens CPN-side "FindRouteSendPck_DSR_SendSide"

en transition, og en pil, der bringer listen tilbage til pladsen pånær det første element i den), og det derfor kan være svært at overskue flowet på siderne ud fra disse. Generelt vender flowet på en side samme vej, som det indikeres i "hovedflow"-figuren (figur 3.4 på side 44).

Sorte transitioner og pladser: Visse af pladserne og transitioner er gjort sorte med hvid tekst i stedet for hvide med sort tekst. Dette har ingen betydning for funktionaliten af modelsiden, det er udelukkende for at fremhæve visse af transitionerne eller pladserne – f.eks. fordi de angiver et hovedflow på siden. I figur 3.10 på forrige side er pladserne og transitionerne med hvid baggrund f.eks. udelukkende brugt til konfigurering af "FindRouteSendPck_DSR_SendSide"-undersiden – dette beskrives nærmere i det nedenstående.

"FindRouteSendPck_DSR_SendSide" får altså pakkebrikker, knuden modelleret skal sende via DSR. Siden (som kan ses i figur 3.11) fungerer som følger: I toppen modtages pakkebrikker, der ønskes behandlet, på pladsen "Packets to transmit". Først undersøges, om en rute til destinationen allerede findes i "Route Cache".

"Route Cachen" er i CPN-modellen modelleret som en samling af komplette ruter til hver destinationsknude (som foreslået i afsnit 2.2.4). Når en rute skal bruges, benyttes strategien, at den "bedste rute" til en destinationsknude er den rute, der for kortest tid siden er tilføjet "Route Cachen". Dette betyder, at så gyldige ruter som muligt altid bør blive returneret fra "Route Cachen" – også selvom en kortere, men ældre rute tidligere er indsat i "Route Cachen". Strategien er dog muligvis ikke optimal, hvis modellens deltagende knuder benytter CRR (se afsnit 2.3.6), da dette kan gøre, at ældre, ugyldige ruter kan blive sendt rundt i modellen af netværket (og blive tilføjet de enkelte modellerede knuders "Route Cache" som den nyeste rute). Det ville være relevant at tilføje en konfigureringsmetode af CPN-modellen, så man kan

skifte mellem forskellige "Route Cache"-strategier i CPN-modellen, men dette har jeg valgt ikke at prioritere i forhold til at modellere eksempler på udvidelser af CPN-modellen, som det beskrives senere i dette kapitel.

Hvis den modellerede knude har en sådan rute til destinationsknuden i sin modellerede "Route Cache" allerede, tilføjes en "DSR Source Route"-optionsheader med den fundne, "bedste" rute til pakkebrikken, og denne anbringes på fusionspladsen "To transmit through Route Maintenance" (hvorfra den bliver fjernet af "Initiate_RM_SendSide"-siden og sendt via RM, som det beskrives i afsnit 3.5.1). Hvis den modellerede knude *ikke* har en sådan rute, gives pakkebrikken i stedet til undersiden "Initiate_RD_SendSide", der er en del af RD-delen af DSR-modellen, og som beskrives i afsnit 3.4.1.

At hovedparten af DSR's afsenderside-hovedfunktionalitet er flyttet fra "DSR_SendSide"-siden til en underside ("FindRouteSendPck_DSR_SendSide") kan virke mærkeligt, men det skyldes, at undersiden "FindRouteSendPck_DSR_SendSide" benyttes flere forskellige steder i modellen, og siden skal opføre sig en anelse forskelligt alt efter hvilken side, der benytter den som underside. To ting kan konfigureres på siden:

"HowToSendPackets": De pakkebrikker, der bliver sendt til RD-delen af modellen, skal enten altid inkluderes i pakken (værdien "includeInPacket" benyttes) eller altid gemmes på "Send Buffer"-pladsen, mens en rute til destinationsknuden findes (værdien "saveInSendBuffer" benyttes). Dette vender jeg tilbage til i afsnit 3.4.1.

"SalvageCounter": Siden tilføjer en "DSR Source Route"-optionsheader til pakkebrikker, der simuleres sendt via RM. Disse kan indeholde forskellige værdier på `salvage_counter`-feltet.

Når "FindRouteSendPck_DSR_SendSide" kaldes fra "DSR_SendSide" (figur 3.10 på side 51), sættes `salvage_counter` til 0 (da pakkebrikken netop repræsenterer en ny, indtil videre usendt pakke), og RD konfigureres til at gemme pakkebrikken på "Send Buffer"-pladsen, mens en evt. rute til destinationsknuden forsøges fundet. Dette er præcist, som specifikationen [JMH05] siger, det skal gøres.

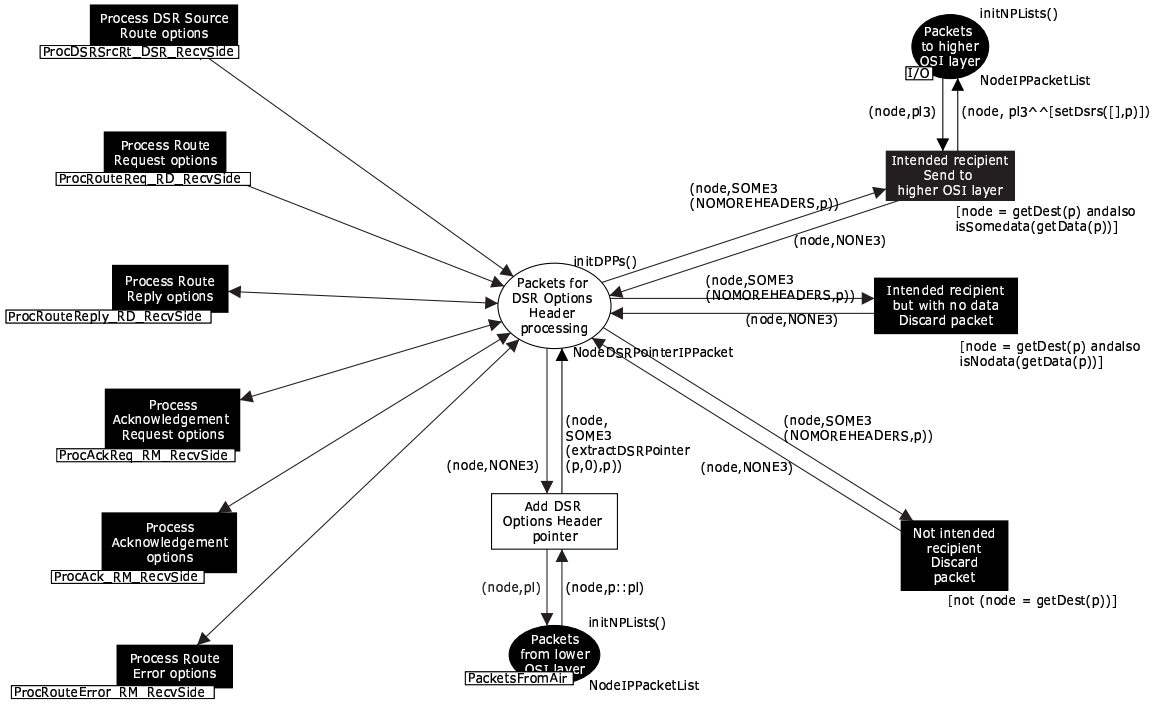
Det er værd at nævne, at brugen af "FindRouteSendPck_DSR_SendSide" ikke nødvendigvis behøvede at være modelleret som underside-sideinstanser. Konfigurationen kunne i stedet have fulgt med den pakkebrik, der gives til siden, så "Packet to transmit"-pladsen i stedet for at have typen "NodeIPPacketList" (der har typen `(Node,[IPPacket])`, altså en pakkeliste pr. knude i systemet) kunne have været en fusionsplads, der havde en type svarende til `(Node,[(HowToSendPackets,SalvageCounter,IPPacket)])`. Jeg vurderede dog, at dette ville gøre siden mere uoverskuelig, og derfor er denne metode ikke benyttet.

3.3.2 Modelleret modtagelse af pakker i "DSR's hovedfunktionalitet"

Processering af optionsheaderlisten

Når en pakkebrik modtages fra det modellerede underliggende OSI-lag (d.v.s. konceptuelt fra netværket), siger specifikationen, at hver DSR-optionsheader i pakkebrikken skal processeres i rækkefølge. V.h.a. en række undersider sørger CPN-siden "DSR_RecvSide" (se figur 3.12 på næste side) for dette.

Flowet på siden er som følger: Pakkebrikkerne modtages på pladsen "Packets from lower OSI-layer". En pointer ind i DSR-optionsheaderlisten tilføjes, så det er muligt at holde styr på, hvor langt processeringen af optionsheaderne er kommet. Herefter processeres hver optionsheader en efter en, og til sidst vil pakkebrikken få fjernet DSR-optionsheaderlisten og blive givet til det modellerede højereliggende OSI-lag, hvis den modellerede knude er den endelige modtager af pakkebrikken, og pakkebrikken indeholder datatrafik (jvf. definitionen i afsnit 1.2). I modsat tilfælde vil pakkebrikken blive smidt ud.



Figur 3.12: DSR-modellens CPN-side "DSR_RecvSide"

Inden pakkebrikken sendes op eller smides ud, vil den allerede f.eks. være blevet sendt videre, hvis den indeholder en "DSR Source Route"-optionsheader. Dette sker på den første af undersiderne, der behandler optionsheadererne, som beskrevet i det nedenstående.

"DSR Source Route"-optionsheaderen

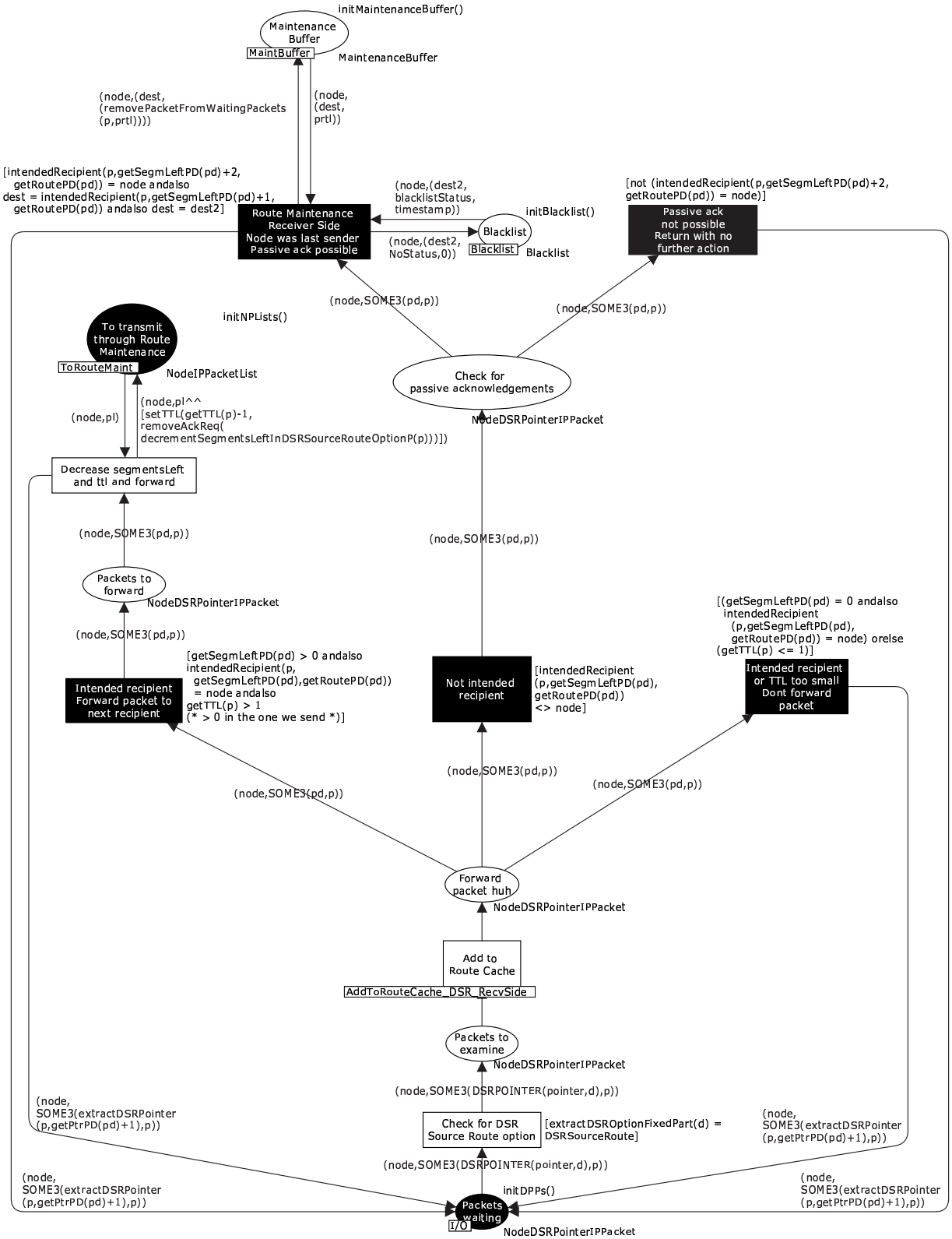
CPN-siden, der håndterer modtagne pakkebrikker med "DSR Source Route"-optionsheaders, kan ses i figur 3.13 på næste side.

Siden er relativt simpel: Pakkebrikkerne modtages på pladsen "Packets waiting" i bunden af figuren. For at sikre, at det netop kun er brikker med en pointer, der peger på en "DSR Source Route"-optionsheader i en pakke, er denne plads kun forbundet med en enkelt transition, og denne transition kan kun fyre, hvis pointeren peger på en "DSR Source Route"-optionsheader. I modsat fald vil det være en af de øvrige undersider til "DSR_RecvSide" (se afsnit 3.3.2), der indeholder en indgangstransition, der er i stand til at fyre, da der netop findes én underside for hver optionsheadertype, der er medtaget i DSR-modellen.

Det første, der sker med optionsheaderen, er, at den gives til undersiden "AddToRouteCache_DSR_SendSide"-siden, der sørger for at udtrække ruteinformation fra den. Dette beskrives nærmere i næste afsnit. Siden kan herefter være i en af tre situationer:

Modtaget til videresendelse (transitionen til venstre på siden): Den modellerede, modtagende knude kan have modtaget pakkebrikken som specificeret i pakkebrikkens ruteliste undervejs i dens forsendelse til destinationen. I dette tilfældes skal pakkebrikken videresendes. Pakkebrikken modificeres og gives til RM-delen af DSR-modellen, der modelleret vil stå for den faktiske forsendelse af pakkebrikken til næste modtager.

Modtaget som endelig destination (transitionen til højre på siden): Den modellerede modtagende knude kan være den endelige destination for pakkebrikken. I dette tilfælde ignoreres optionsheaderen, idet pakkebrikken blot skal tilbage til "DSR_RecvSide"-siden



Figur 3.13: DSR-modellens CPN-side "ProcDSRSrcRt_DSR_RecvSide"

(figur 3.12 på side 54) og gives videre til det modellerede højereliggende OSI-lag. Der kan også være andre grunde til, at optionsheaderen skal ignoreres, såsom at indholdet `TTL`-feltet er blevet for småt til at pakkebrikken må modelleres videresendt.

Overhørt pakke (transitionen i midten af siden): Pakkebrikken kan repræsentere en *overhørt* pakke, som egentlig ikke var tiltænkt den knude, der i modellen er i gang med at behandle pakkebrikken.

Her skal pakkebrikken normalt blot ignoreres, medmindre den modellerede, modtagende knude kan have overhørt pakkebrikken fra en naboknude, som knuden netop har simuleret en videresendelse af pakkebrikken til. Dette fungerer i så tilfælde som en passiv bekræftelse (se afsnit 2.4.1), og bevirker, at den kopi af pakkebrikken, der ligger på "Maintenance Buffer"-pladsen (hvorpå den blev lagt af RM, som det beskrives i afsnit 3.5) tages ud, og at en eventuel indgang på "Blacklist"-tabel-pladsen for denne naboknude kan nulstilles (da dette netop fungerer som en bekræftelse på, at man både kan sende til og modtage fra den modellerede naboknude jvf. afsnit 2.4.5).

I alle tilfældene hentes pakkebrikkens næste optionsheader til sidst ud, og denne lægges sammen med pakken selv tilbage på "Packets waiting". Dermed er pakkebrikken klargjort til, at også den næste optionsheader kan blive behandlet af "DSR_RecvSide" (figur 3.12 på side 54).

Ved brugen af passive bekræftelser under "overhørt pakke" ovenfor er det nødvendigt for en modelleret knude at kunne genkende, at to pakkebrikker er ens (modulo `segments_left`- og `TTL`-feltet). En "DSR Source Route"-optionsheader (eller den faste del af DSR-headeren i øvrigt) indeholder dog ikke en unik identifikation, man kan benytte til dette – hvis en modelleret knude afsender to ens pakker til den samme destinationsknude via den samme rute, kan knuder undervejs på ruten mellem de to ikke skelne mellem pakkebrikkerne v.h.a. DSR-headeren.

Ifølge [JMH05] skal også IP-headeren bruges til at sammenligne, om to pakker er ens. Mere præcist skal source-adresse-feltet, destinations-adresse-feltet, identifikations-feltet og fragment offset-feltet være ens, før to pakker regnes for at være den samme. Da IP-pakkefragmentering ikke er modelleret med i DSR-modellen, kigger modellen kun på de tre første af disse, når den skal sammenligne pakker. Den ovenstående sammenligning vil fungere, fordi DSR-modellen alle steder, der genereres nye IP-pakker, sørger for, at IP-headerens identifikationsnummer er unikt pr. modelleret afsenderknude, startende med et tilfældigt tal. Den måde, identifikationsnumre skal bruges i DSR-optionsheaders, er altså her udvidet til også at dække IP-headeren (se afsnit 2.3.1).

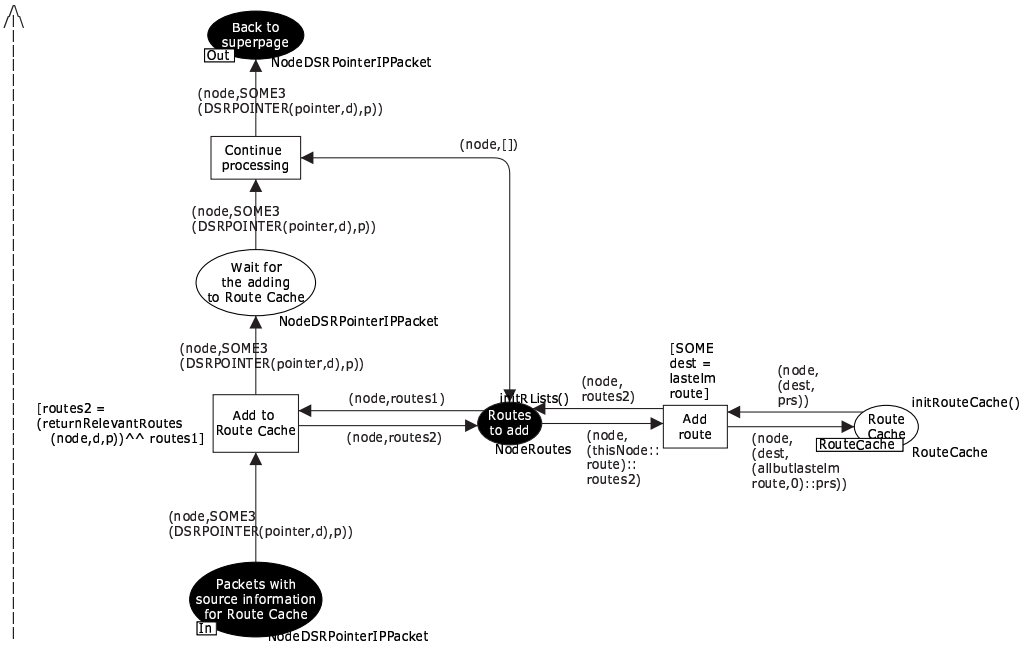
Tilføjelser til "Route Cache"

Som nævnt i det ovenstående benytter flere af de CPN-sider, der afkoder optionsheaders fra pakkebrikker, en underside, der er i stand til at uddrage ruteinformationer fra disse optionsheaders og tilføje disse informationer til "Route Cachen". Figur 3.14 på næste side viser denne side.

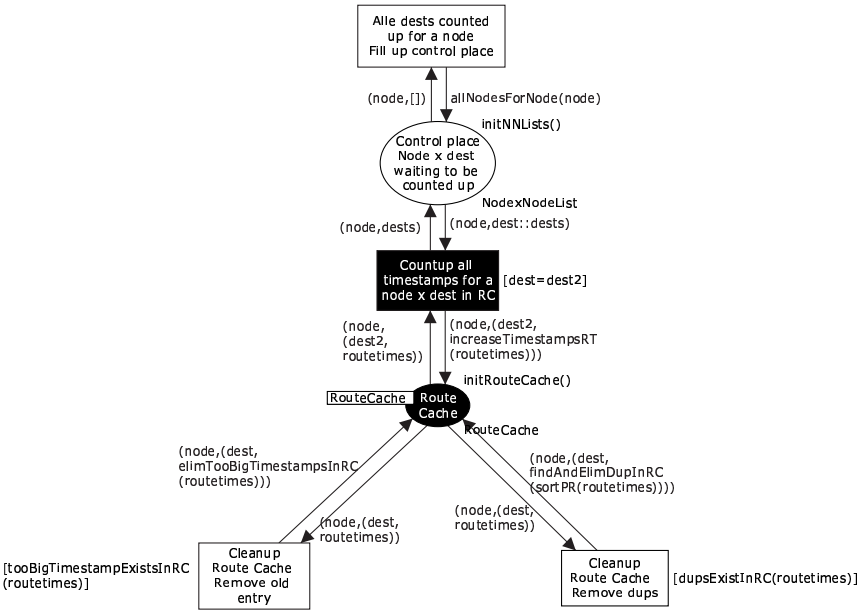
Siden benytter de regler, der er beskrevet i afsnit 2.2.5, til at uddrage ruter fra en optionsheader – og returnerer pakkebrikken til supersiden igen, når "Route Cache" er opdateret med disse nye ruter.

"Route Cachen" ryddes løbende op i. Dette sker v.h.a. hjælpesiden "MaintainRouteCache_DSR" (figur 3.15 på næste side), der dels sørger for at tælle timeout-værdierne tilknyttet hver indgang i "Route Cachen" for en knude ned, dels fjerner indgange, når denne timeoutværdi når nul. Indgange i "Route Cachen" foreslås i specifikationen [JMH05] fjernet med en timeout-værdi på 300 sekunder. Hvordan dette mere præcist modelleres, beskrives nærmere under "Brugen af tid i modellen" nedenfor.

Da de samme pakkebrikker kan blive modtaget (d.v.s. "overhørt") af den samme modellerede knude flere gange, kan de samme ruter blive udtrukket og tilføjet "Route Cachen" for



Figur 3.14: DSR-modellens CPN-side "AddToRouteCache_DSR_RecvSide"



Figur 3.15: DSR-modellens CPN-side "MaintainRouteCache_DSR"

den samme modellerede knude adskillige gange. For at gøre indholdet af "Route Cachen" så overskuelig som muligt under simuleringer, sørger "MaintainRouteCache_DSR" også for at fjerne sådanne duplikerede ruter deri, selvom det egentlig ikke er nødvendigt.

Brugen af tid i modellen

I stil med mange af datastrukturerne, der senere gennemgås, er hver indgang i "Route Cachen" tilknyttet en timeout, der bestemmer, hvornår indgangen skal fjernes. CPN Tools har et indbygget tidsbegreb, der ville kunne bruges til dette. En timer ([KCJ98]) kan benyttes, hvor brikker tilknyttes en tidsangivelse, som denne timer skal nå op på, før brikken må benyttes i fyringen af en transition. Hvis ingen transitioner i modellen kan fyre, forøges timeren til den mindste værdi, der tillader CPN-værktøjet at fyre en transition.

Problemet med denne fremgangsmåde er, at timeren er *global*. Når timeren bliver forøget for transitionerne modellerende en enkelt knude, bliver den også forøget for de øvrige. Det er dermed ikke muligt at lave en simulering, hvor to knuder fungerer i forskelligt tempo, uden at modellere det specifikt.

I stedet laver jeg en distribueret model, hvor hver indgang i en tabel tilknyttes en timeout eller et timestamp. "Route Cache"-pladsen er f.eks. generaliseret set på formen:

```
(knude,(destination,[(tabelindgang,timestamp)]))
```

En transition er tilknyttet datastrukturen, og denne transition vil tælle alle timestamps tilknyttet indgangene i listen op for et par af `(knude,destination)` – se f.eks. figur 3.15 på forrige side. For at sikre, at en knudes indgange til én destination ikke bliver talt hurtigere op end til en anden, styres transitionen af en anden plads, der indeholder en brik for de par af `(knude, destination)`, der endnu ikke er blevet talt 1 op. Når alle indgange til en knude er blevet talt 1 op, bliver styringspladsen "fyldt op" igen, så timestampene kontinuerligt kan blive talt op. Hermed sikres det, at alle en knudes indgange tælles lige hurtigt op (selvom det sker i løbet af et antal fyringer).

For at sikre, at modellen er "rigtigt" distribueret, er der derimod intet gjort for at sikre, at alle indgangene i "Route Cache" bliver talt op lige hurtigt på tværs af knuderne. Dette kan netop gøres, da der ikke bruges en global timer.

Der bliver heller ikke gjort noget for at sikre, at alle indgangene i de forskellige datastrukturer for den samme knude (f.eks. en knudes "Maintenance Buffer" og "Route Cache") bliver talt lige hurtigt op. Jeg har vurderet, at dette ikke var vigtigt, idet timeouts netop typisk blot skal være store nok til at fange, at en begivenhed *ikke* fandt sted, eller til at markere, at data ikke længere er relevante – f.eks. at en netværkspakke ikke nåede frem til destinationen, eller en indgang i en tabel ikke længere er gyldig. Da CPN Tools tilfældigt vælger en timeout-transition, der skal fyre, vil der blive talt op *cirka* lige hurtigt i de forskellige datastrukturer – hvis det giver en forskel i funktionaliteten, er problemet snarere, at timeout-værdierne ikke sat højt nok. Ingen af dataene i datastrukturerne afhænger af hinanden på tværs af datastrukturerne, så dette giver ikke et problem i denne forbindelse.

I tilfældet med "Route Cache" siger specifikationen [JMH05], at når et timestamp når 300000 millisekunder, skal den tilknyttede indgang fjernes. Hvordan kan man oversætte dette til DSR-modellen? Skal man også her blot lade timestampene nå 300000? – d.v.s., at timestamptransitionen skal fyre 300000 gange for en kombination af `(node,destination)`, før indgangen skal fjernes? Det virker urealistisk, at man på denne måde blot kan oversætte millisekunder direkte til transitionsfyringer.

For at undersøge dette nærmere, vælger jeg at arbejde ud fra, hvor hurtigt en pakke skal gensendes i modellen (altså hvor hurtigt, den skal forældes fra "Maintenance Buffer" jvf. afsnit 2.4.5), idet dette har stor betydning for funktionaliten af modellen. Hvis en pakke ikke kan nå frem, inden den forsøges gensendt, er timeout-værdierne sat for lavt.

CPN Tools fungerer på den måde, at når en simulering er i gang, og værktøjet skal finde en transition, der skal fyre, vælges der en tilfældig transition blandt alle i modellen (jvf. [CTS]). Herefter undersøges det, om denne transition kan fyre. Hvis den ikke kan det, vælger værktøjet en anden transition og undersøger denne i stedet. Sådan fortsættes indtil en fyringsklar transition er fundet. Hvis den fundne transition kan fyre på flere forskellige måder (f.eks. fordi en inputplads har to brikker, men transitionen kun skal bruge en af dem), vælges *herefter* en tilfældig af disse – denne transition vil altså ikke have højere sandsynlighed for at blive valgt end en anden.

Med dette i mente kan man nu se nærmere på, hvor tit de forskellige transitioner vil fyre i forhold til hinanden. Mere præcist bør man se på forholdet mellem antal fyringer af en timeout-transition og antal fyringer af pakkebrik-transitioner:

- Hvor mange gange skal en timeout-transitionen for "Maintenance Buffer" fyre, før timeout-værdien tilknyttet en indgang når nul?
- Hvor mange gange skal de transitioner, der er involveret i at bringe en pakkebrik fra en knude A til en knude B, *til sammen* fyre, før pakkebrikken når frem til B?

"Maintenance Buffer" har samme opbygning som "Route Cache" beskrevet ovenfor. D.v.s., at for hver fyring opdateres $\frac{1}{(\text{antal knuder})^2}$ af indgangene på "Maintenance Buffer"-pladsen.

I alt skal "Maintenance Buffer"-timeout-transitionen altså fyre cirka $(\text{antal knuder})^2 *$ timeout gange før en nyindsat indgang når nul. Tallet er et cirka-tal, da det netop er tilfældigt, hvilke indgange, der bliver talt ned pr. fyring jvf. ovenstående.

Dette skal holdes op imod, at hvis der er n transitioner, der skal fyre, fra en tilstand, hvor "DSR-SendSide" modtager en pakkebrik fra det modellerede højereliggende OSI-lag (se figur 3.4 på side 44) (konceptuelt som knuden A) til en tilstand, hvor "DSR-RecvSide" kan aflevere den til det modellerede højereliggende OSI-lag (konceptuelt som knuden B), skal *disse* transitioner foretage n fyringer undervejs i pakkebriktransporten. Dette gælder dog kun, hvis pakkebrikken er den eneste, der skal transporteres. Hvis der er to pakkebrikker, der skal transporteres nogenlunde samtidigt, kræver dette ca. $2n$ fyringer af pakketransporttransitionerne, før den ene (og dermed også den anden) pakkebrik er fremme. Generelt skal transporttransitionerne altså fyre $(\text{antal transitioner}) * (\text{antal samtidige pakker})$ gange, før en enkelt pakkebrik er fremme.

Hvis hver knude hele tiden konceptuelt behandler én pakke, bliver forholdet mellem de to altså:

$$(\text{antal transitioner}) * (\text{antal samtidige pakker}) \mathbf{:} (\text{antal knuder})^2 * \text{timeout}$$

svarende til:

$$(\text{antal transitioner}) * (\text{antal knuder}) \mathbf{:} (\text{antal knuder})^2 * \text{timeout}$$

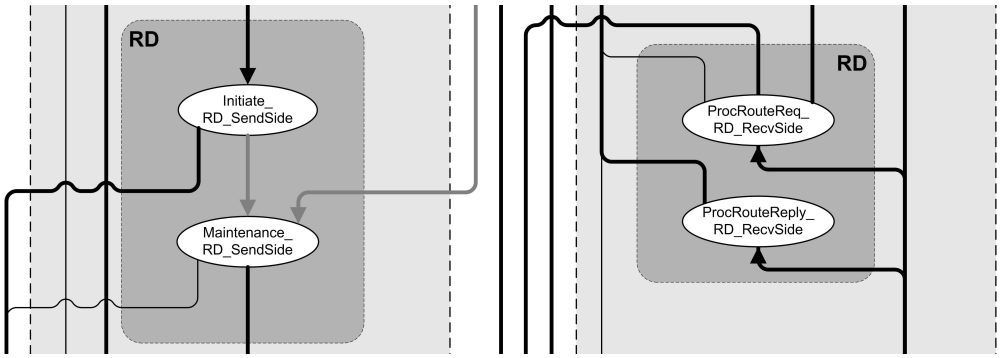
svarende til:

$$(\text{antal transitioner}) \mathbf{:} (\text{antal knuder}) * \text{timeout}$$

Timeout-værdierne vil altså blive talt langsommere ned i forhold til hvor hurtigt, pakketransporten foregår, jo flere knuder, der modelleres i modellen.

Har hver knude i stedet to pakker i behandling, fås i stedet forholdet $2 * (\text{antal transitioner}) \mathbf{:} (\text{antal knuder}) * \text{timeout}$. Timeout-værdiens effekt vil altså også falde, jo flere pakker, hver enkelt knude behandler.

Alt dette betyder, at for at få en funktionel model burde det være nok at finde ud af, hvad timeout-værdierne skal være, for at man kan transportere et minimalt antal pakker gennem modellen med det maksimale antal knuder, der bliver benyttet. Det maksimale antal knuder, der blive blive brugt i forbindelse med analyser eller demonstrationer i dette speciale, er 7 (i afsnit 3.8.2), og dette kan altså holdes op imod en enkelt pakketransport.



Figur 3.16: CPN-sidesammenhænge: Hovedflowet mellem hovedsiderne i "Route Discovery" (udsnit af figur 3.4 på side 44). Afsenderdelen er til venstre i figuren, modtagerdelen til højre.

En række tests har vist, at hvis alle modeldatastruktur-timeouts sættes til halvdelen af de værdier, der er foreslået i specifikationen [JMH05] i millisekunder, fås den ønskede effekt: Protokollen bliver f.eks. i stand til at modellere, at en pakke kan blive transmitteret fra en knude til en anden og få en passiv bekræftelse tilbage, før de satte timeout-værdier i "Maintenance Buffer" når nul og får modellen af protokollen til at foretage en gentransmission af pakken. De øvrige datastrukturer opfører sig også som forventet. En yderligere reduktion af faktoren gør, at det ikke længere konsekvent lykkes at gennemføre en pakketransmission uden gentransmissioner, mens en forøgelse af faktoren gør, at simulationer bliver unødigt lange, hvilket kan blive et problem i kapitel 5, hvor et antal simuleringer skal foretages.

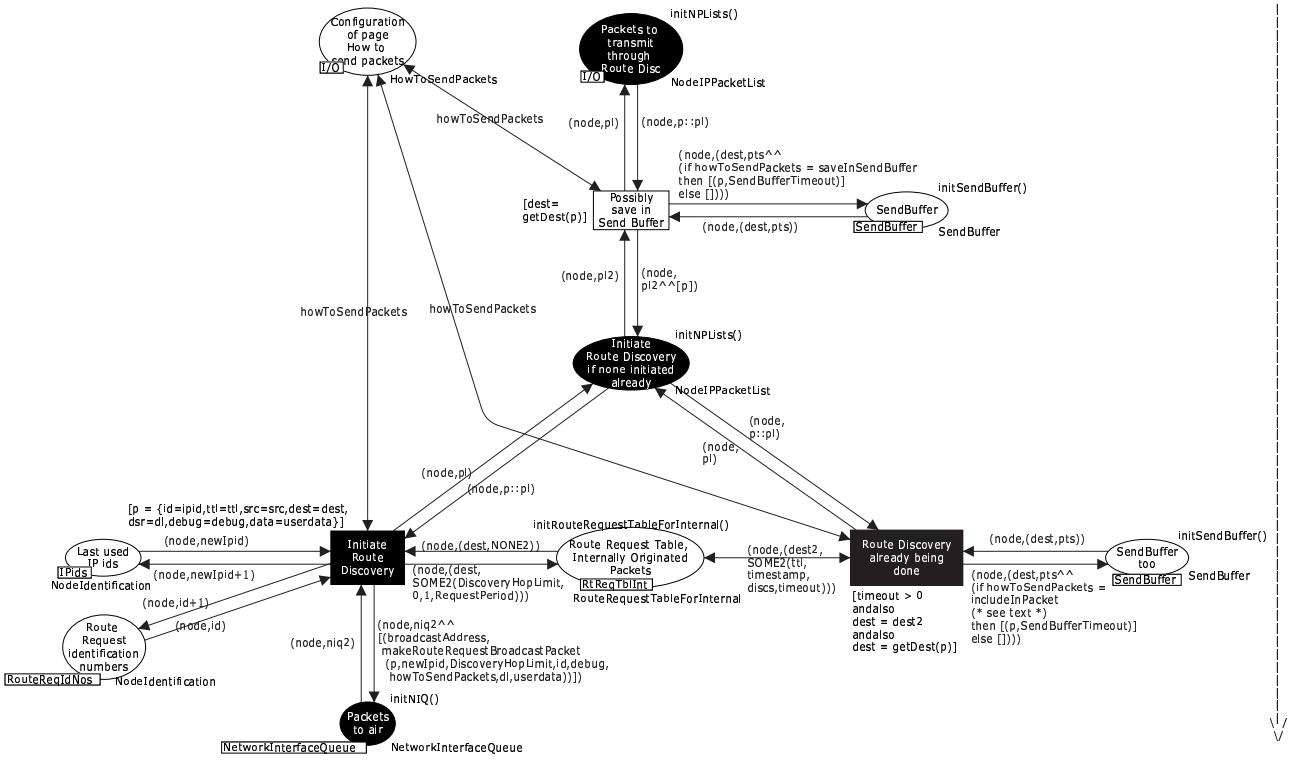
Men samtidigt er det værd at indse, at det er en svaghed ved modellen, at timeoutværdiernes effekt ikke er ens, alt efter hvor mange knuder, der modelleres, og hvor mange pakker, der samtidigt sendes.

Problemet kunne f.eks. blive delvist løst ved lave datastrukturerne om fra multiset af `(knude,(destination,[indgange]))` til multiset af `(knude,[(destination,[indgange])])` – altså med en brik pr. knude i stedet for (som nu) en brik pr. `(knude,destination)`. Imidlertid kan dette ikke lade sig gøre i praksis p.g.a. CPN Tools' svaghed over for lister (som beskrevet i afsnit 3.2.2). Alternativt kunne jeg have ladet timeout-transitionerne virke på et *antal* brikker pr. fyring.

I retrospekt ville det muligvis have været bedre at have benyttet CPN Tools' timer-begreb i stedet, da ulemperne ved det ovenstående kan overskygge ulemperne ved, at der ikke længere ville findes en distribueret nedtællingsmekanisme. Modellen skal dog til en vis grad gentænkes, før dette kan lade sig gøre. F.eks. er det i "Route Cache"-tilfældet lige nu sådan, at hver pakkebrik er tilknyttet en liste med et antal indgange (med hver sin timeout-værdi jvf. ovenstående). Med CPN Tools' timer-begreb er det kun muligt at have én timer-værdi pr. pakkebrik, så det er altså nødvendigt at gemme ting anderledes i "Route Cache". Det er ikke nok at dele liste-pakkebrikkerne op i enkelt-pakkebrikker, da det f.eks. skal være muligt at finde den pakkebrik, der har den korteste rute (for et par af `(knude,destination)`) – og en sådan selektiv udtagning af pakkebrikker er p.t. ikke mulig i CPN Tools (jvf. [CTS]).

3.4 Modellering af "Route Discovery"

Den næste del af DSR-modellen er "Route Discovery" (RD, som beskrevet i afsnit 2.3). En logisk oversigt over hovedsiderne, der hører til RD, kan ses i figur 3.4 på side 44. De relevante sider er genvist i figur 3.16 – men uden sammenkædningen med resten af CPN-modellen, som den oprindelige figur viser. Ud over disse sider findes der en række undersider til "ProcRouteReq_RD_RecvSide" (i afsnit 3.4.3 beskrives det hvilke, det drejer sig om, og hvordan de-



Figur 3.17: DSR-modellens CPN-side “Initiate_RD_SendSide”

res indbyrdes sammenhæng er) samt en række hjælpesider: “InitRouteReqIdNos_RD_RecvSide”, der initialiserer en datastruktur, samt “MaintainSendBuffer_RD_SendSide”, “MaintainRouteReqTableInt_RD_SendSide” og “MaintainRouteReqTableExt_RD_SendSide”, der vedligeholder hver deres datastruktur.

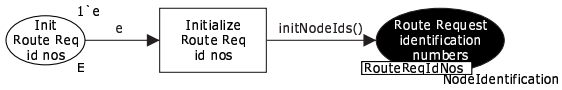
Alle disse sider (hovedsiderne, undersiderne og hjælpesiderne) vil blive beskrevet i afsnit 3.4.1-3.4.3, mens afsnit 3.4.4-3.4.6 beskriver fundne problemområder o.lign. i den officielle specifikation [JMH05] indenfor RD.

3.4.1 Modelleret afsendelse af pakker i “Route Discovery”

Afsenderdelen af RD-delen af DSR-modellen kan ses i figur 3.17. Når pakkebrikker bliver sendt til RD-delen af DSR-modellen, vides det allerede, at en rute til pakkebrikkens destinationsknude ikke findes på den modellerede knudes “Route Cache”-plads. Derfor er siden ganske simpel, idet den blot sørger for at generere en “Route Request”-pakke gående på destinationsknuden for pakken og give denne til det modellerede underliggende OSI-lag, så pakkebrikken kan blive simuleret “broadcastet ud”.

Som beskrevet i afsnit 3.3.1 bliver siden konfigureret af en superside til enten at gemme de pakkebrikker, der skal genereres “Route Requests” for, på “Send Buffer”-pladsen, mens ruten forsøges fundet, eller til at piggybacke indholdet af pakkebrikken til “Route Request”-pakkebrikken. Det sidste bruges f.eks. af “SendRouteReply_RD_RecvSide” (som beskrives i afsnit 3.4.3), hvor en “Route Reply” netop skal piggybackes til en genereret “Route Request”-pakkebrik for at undgå mulige uendelige RD-løkker.

Før en ny “Route Request”-pakkebrik laves, undersøges det dog, om en RD allerede er igangsat gående på denne destinationsknude. Når en RD igangsættes, gemmes information om



Figur 3.18: DSR-modellens CPN-side "InitRouteReqIdNos_RD_SendSide"

den på "Route Request Table, Internally Originated Packets"-tabel-pladsen (jvf. afsnit 2.3.5), og det er blot denne tabel, der konsulteres, før en "Route Request"-pakkebrik laves. Hvis dette afslører, at en RD allerede er igangsat, må en ny "Route Request"-pakke ikke laves, da udsendelsen af disse skal følge en backoff-algoritme. I stedet bliver den oprindelige pakkebrik gemt på "Send Buffer"-pladsen (hvis den ikke allerede er blevet det). Dette vender jeg tilbage til i afsnit 3.4.6.

Hver "Route Request"-pakkebrik, der genereres, skal indeholde et identifikationsnummer, der som beskrevet i afsnit 2.3.1 skal forsøges gjort så unikt som muligt. Ifølge specifikationen [JMH05] er en mulig måde at gøre dette på, at en knude benytter et tilfældigt tal som det første identifikationsnummer. Denne metode benyttes i DSR-modellen, og tallet gemmes for hver modelleret knude på fusionspladsen "RouteReqIdNos". Funktionen "initNodeIds()" returnerer en mængde brikker af typen `(node,identification)`, hvor `identification` er sat til et tilfældigt tal. Normalt ville denne funktion blot blive brugt som initialisering på "RouteReqIdNos"-pladsen, men her fandt jeg en fejl i den på modelleringstidspunktet nyeste version af CPN Tools: En fusionsplads kan ikke have en initiel, *tilfældig* værdi. Derfor benyttes i stedet en transition på en ny side, "InitRouteReqIdNos_RD_SendSide" (se figur 3.18), til at initialisere pladsen med.

3.4.2 Modelleret vedligeholdelse af afsenderdelen i "Route Discovery"

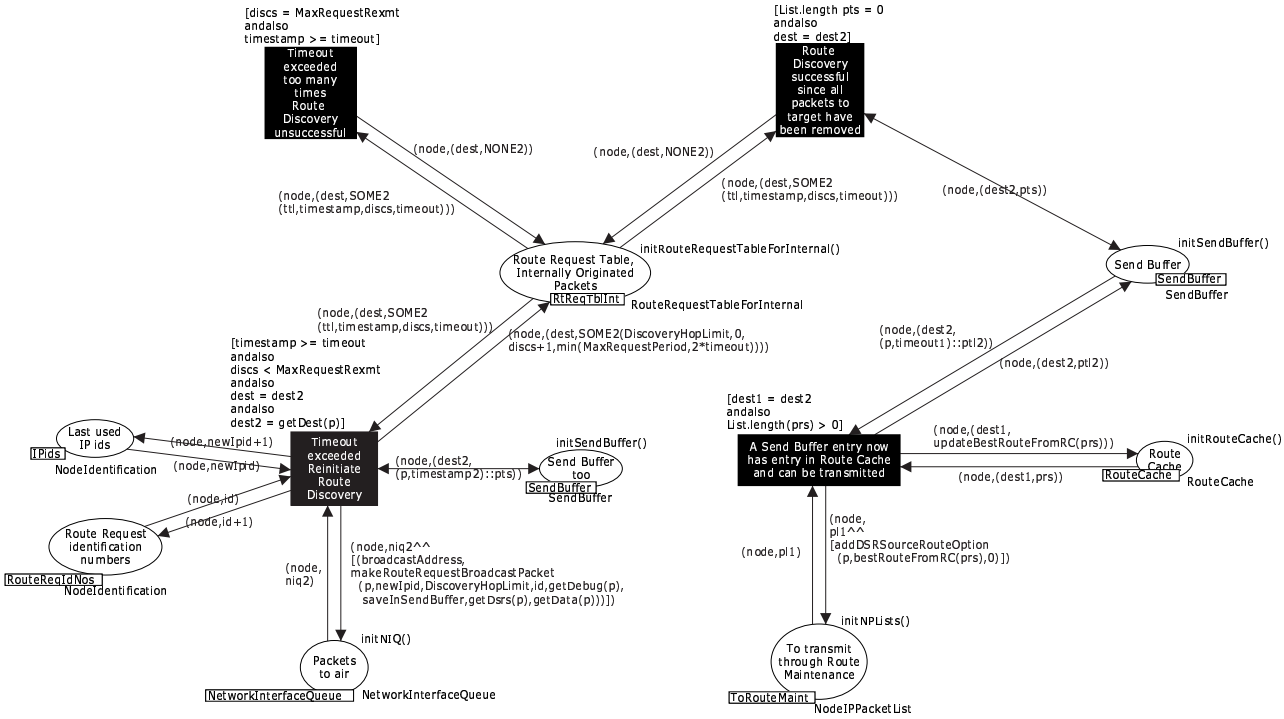
I figur 3.19 på næste side kan de fire ting, modellen kan foretage sig som en del af RD, efter en RD er igangsat, ses. Det drejer sig om følgende:

Rute fundet, pakke sendes (transitionen nederst til højre): Den modellerede knude kan være i stand til at matche en pakke i "Send Buffer" og en rute i "Route Cache". Dette kan f.eks. ske, hvis knuden har fået svar på en udsendt "Route Request"-pakkebrik gående på destinationen for pakkebrikken på "Send Buffer"-pladsen. I dette tilfælde kan man blot tage pakkebrikken ud af "Send Buffer"-pladsen, tilføje den fundne rute, og give brikken til RM via fusionspladsen "ToRouteMaint".

Oprydning efter alle pakker sendt (transitionen øverst til højre): Hvis den modellerede knude har fundet en rute som beskrevet i det forrige punkt, og alle pakkebrikker til destinationsknuden som følge af dette er fjernet fra "Send Buffer"-pladsen, kan knuden fjerne indgangen fra "Route Request Table, Internally Originated Packets"-pladsen. Dette vil betyde, at der ikke bliver udsendt flere "Route Request"-pakker.

Genudsendelse af "Route Requests" (transitionen nederst til venstre): De enkelte indgange i "Route Request Table, Internally Originated Packets" indeholder en timestamp-værdi, der tælles op v.h.a. CPN-siden "MaintainRouteReqTableInt_RD_SendSide". Denne ligner til forveksling optællingen af "Route Cache" (figur 3.15 på side 57), og vises derfor ikke her – den kan ses i appendix B.1 som figur B.1 på side 140. Hver indgang indeholder desuden en timeout-værdi, som på "Initiate_RD_SendSide"-siden (figur 3.17 på forrige side) initialiseres til værdien af "DiscoveryHopLimit" (se afsnit 3.3.2 om den generelle brug af timeouts i DSR-modellen).

"Timeout exceeded, Reinitiate Route Discovery"-transitionen på figur 3.19 på næste side sørger for, at når timestamp-værdien når op på denne timeout-værdi, laves der en



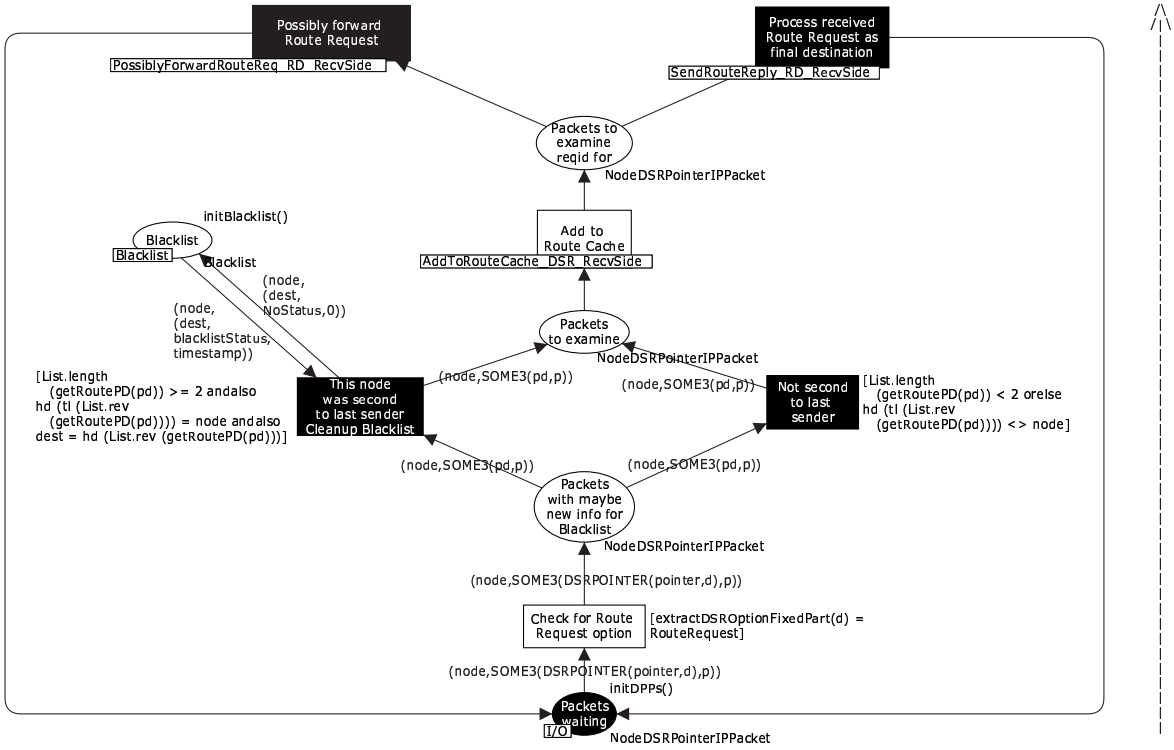
Figur 3.19: DSR-modellens CPN-side “Maintenance_RD_SendSide”

ny “Route Request”-pakkebrik, og denne gives til modellen af det underliggende OSI-lag, som så kan simulere en rebroadcasting af pakkebrikken. Herefter nulstillles timestamp-værdien, og timeout-værdien fordobles (indtil værdien når et vist maskimum, “MaxRequestPeriod”). Dette modellerer altså en “backoff”-algoritme som beskrevet i afsnit 2.3.5.

Maksimalt antal genudsendelser nået (transitionen øverst til venstre): Som beskrevet i afsnit 2.3.5 beskrev en ældre version af specifikationen, at der højst må udsendes “Route Request”-pakker et bestemt antal gange (benævnt “MaxRequestRexmt”). Uheldigvis er dette blevet modelleret med i DSR-modellen, selvom det ikke er en del af den nuværende specifikation (omend en foreslået værdi til konstanten stadig findes deri).

Så i modellen bliver indgange i “Route Request Table, Internally Originated Packets” fjernet efter “MaxRequestRexmt” udsendelser af “Route Request”-pakker. Pakkerne i “Send Buffer” (d.v.s. de pakker, der venter på en rute til en destination) påvirkes dog ikke af dette, disse fjernes stadig efter “SendBufferTimeout”, som det er meningen, at det skal ske. Dette sker på siden “MaintainSendBuffer_RD_SendSide”, der heller ikke er gengivet i dette kapitel, da også denne til forveksling ligner “MaintainRouteCache_-DSR” (figur 3.15 på side 57) – den kan ses i appendix B.1 som figur B.2 på side 141.

Den forældede funktionalitet i DSR-modellen i forhold til DSR-protokollen har dog en praktisk betydning: Hvis en knude A vil sende en strøm af pakkebrikker til en destination B, der ikke kan nås, gemmes disse i A’s “SendBuffer”. Er strømmen hurtig nok, vil der altid være mindst én pakkebrik til B i “SendBuffer”. Er det tilfældet, vil der være en periode, fra “MaxRequestRexmt” “Route Request”-pakkebrikker er blevet udsendt, til den næste pakkebrik til B gemmes i “SendBuffer”, hvor der ikke udsendes nye “Route Request”-pakkebrikker gående på B. Herefter begynder backoff-algoritmen blot “forfra”. Dette blev først opdaget, efter analyserne i kapitel 5 blev gennemført (og derfor er



Figur 3.20: DSR-modellens CPN-side "ProcRouteReq_RD_RecvSide"

funktionaliteten ikke rettet i figur 3.19 på forrige side), men ingen af simuleringerne af modellen i resten af dette speciale er dog lange nok til, at det har kunnet betyde noget for resultaterne i resten af specialet.

3.4.3 Modelleret modtagelse af pakker i "Route Discovery"

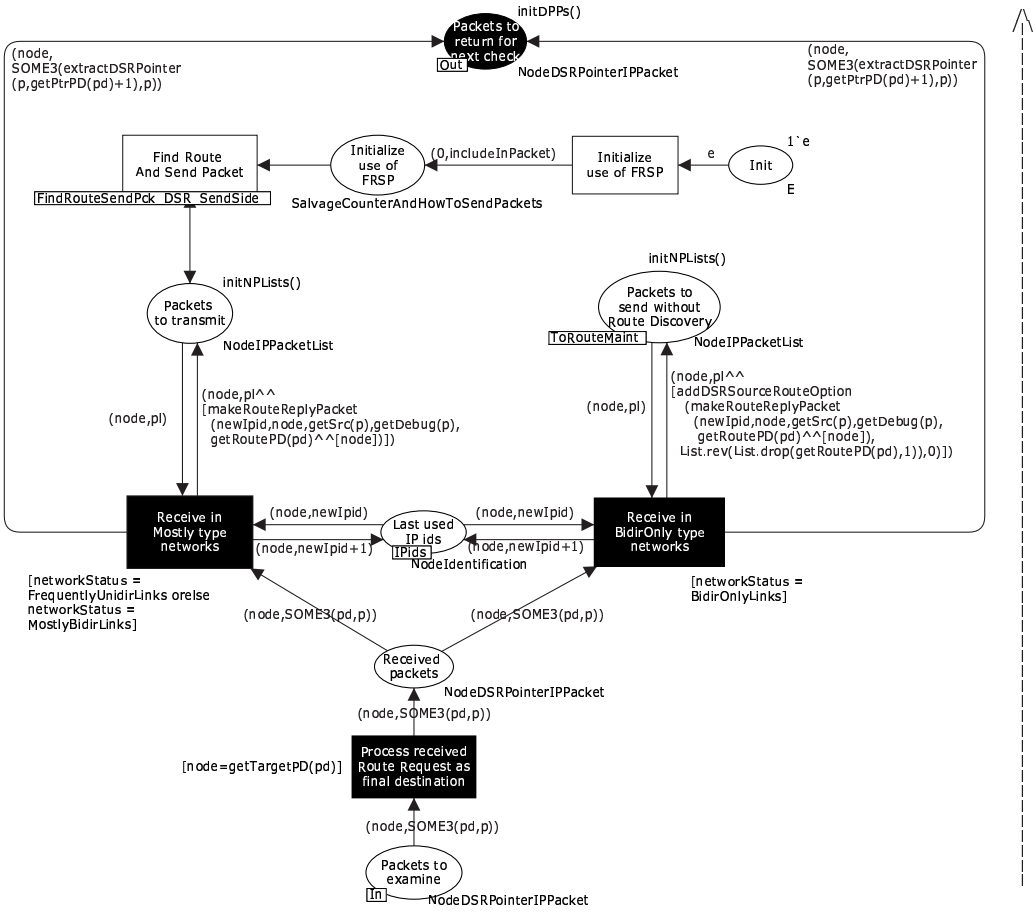
Som beskrevet i afsnit 3.3.2 sørger CPN-siden "DSR_RecvSide" for at kalde undersider, der sørger for at behandle de enkelte optionsheaders i en pakkebrik, når en modelleret knude modtager en pakkebrik fra det modellerede underliggende OSI-lag. I forbindelse med RD findes der to typer optionsheaders, der kan modtages: "Route Request"- og "Route Reply"-optionsheaders.

"Route Request"-optionsheaderen

CPN-siden, der håndterer modtagne "Route Request"-optionsheaders, kan ses i figur 3.20. Siden består af en række undersider, hierarkiet over disse er som følger:

- "ProcRouteReq_RD_RecvSide" (figur 3.20) har 3 undersider:
  - "AddToRouteCache_DSR_RecvSide" (allerede gennemgået i afsnit 3.3.2)
  - "SendRouteReply_RD_RecvSide" (figur 3.21 på næste side)
  - "PossiblyForwardRouteReq_RD_RecvSide" (figur 3.22 på side 66), der har yderligere 2 undersider:
    - "CheckBlacklistBeforeForward_RD_RecvSide" (figur 3.23 på side 69)
    - "SendCachedRouteReply_RD_RecvSide" (figur B.4 på side 142 i appendix B.1)

Processeringen af "Route Request"-optionsheaderens hovedside (som vist i figur 3.20) benytter kun optionsheaderen til "vedligeholdelse" af knudens tabeller: Først ryddes der op



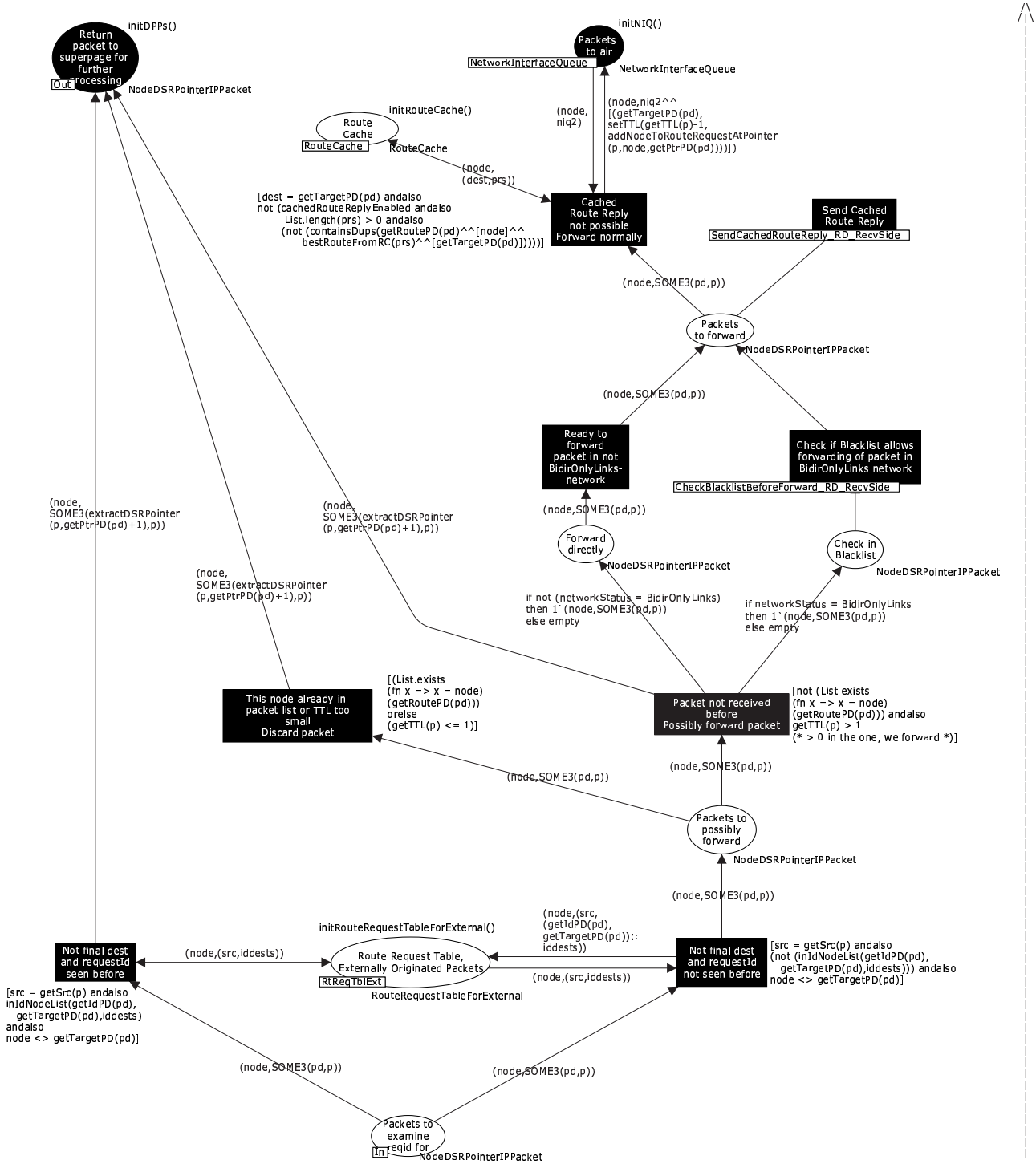
Figur 3.21: DSR-modellens CPN-side "SendRouteReply_RD_RecvSide"

på "Blacklist"-tabel-pladsen, hvis dette er muligt, og undersiden "AddToRouteCache_DSR_-RecvSide" (se afsnit 3.3.2) benyttes til at trække ruteinformation ud af optionsheaderen efter reglerne i afsnit 2.2.5.

Jvf. afsnit 2.3 skal pakken nu enten videresendes eller besvares:

Generering af en "Route Reply"-pakkebrik: Tilfældet, hvor en "Route Reply"-pakkebrik skal genereres, behandles på undersiden "SendRouteReply_RD_RecvSide" (figur 3.21). Siden er ganske simpel: Hvis DSR-modellen er konfigureret til at modellere et rent bidirektionelt netværk, skal en "Route Reply"-pakkebrik simuleres sendt tilbage over den reverserede ruteliste (for at teste bidirektionaliteten af den opsamlede rute i "Route Request"-pakkebrikken) som beskrevet i afsnit 2.3.3. Hvis modellen er konfigureret til at simulere en anden type netværk, benyttes i stedet "FindRouteSendPck_DSR_SendSide" (beskrevet i afsnit 3.3.1), der enten vil benytte RM direkte til at simulere forsendelse af pakkebrikken, hvis den modellerede knude allerede kender en rute dertil, eller i modsat fald benytte RD til at finde en sådan rute. I det sidste tilfælde instrueres undersiden til at piggybacke indholdet af "Route Reply"-pakkebrikken i "Route Request"-pakkebrikken for at undgå uendelige RD-løkker.

Viderebroadcasting af "Route Request"-pakkebrik: Tilfældet, hvor "Route Request"-pakkebrikken (under visse forudsætninger) skal simuleres viderebroadcastet, behandles på



Figur 3.22: DSR-modellens CPN-side “PossiblyForwardRouteReq_RD_RecvSide”

undersiden "PossiblyForwardRouteReq_RD_RecvSide" (figur 3.22 på forrige side).

Siden starter med at foretage forskellige checks på den modtagne pakkebrik for at finde ud af, om den må simulere en sådan viderebroadcasting af pakkebrikken:

Check på identifikationsnummer: Alle "Route Request"-optionsheaders indeholder et identifikationsnummer. Dette identifikationsnummer gemmes hos de enkelte knuder på pladsen "Route Request Table, Externally Originated Packets" (se afsnit 2.3.5). Hvis en modelleret knude kan genkende identifikationsnummeret, har "Route Request"-pakkebrikken været behandlet af knuden før, og resten af optionsheaderen ignoreres, pointeren opdateres til at pege på den næste optionsheader, og pakken returneres til "DSR_RecvSide"-siden, så denne kan sørge for at behandle eventuelle andre optionsheaders i pakkebrikken.

Man kan argumentere for, at hele pakkebrikken i dette tilfælde bør fjernes, men dette er ikke klart specificeret i protokolspecifikationen [JMH05] (her står kun, at "Route Requestet" skal ignoreres uden at henvise til, om det drejer sig om optionsheaderen eller hele pakken). Da optionsheaders kan blive piggybacket til "Route Request"-pakkebrikker i modellen, har jeg valgt kun at ignorere "Route Request"-*optionsheaderen*, selvom jeg i den nuværende modellering kun piggybacker optionsheaders på pakke-genererings-tidspunktet (og altså ikke undervejs i pakketransporten, så pakker kan ændre indhold efter de allerede har været transmitteret én gang).

Check på TTL: Hvis `TTL`-feltets værdi er for lille, må pakkebrikken heller ikke simuleres viderebroadcastet.

Check for allerede modtaget pakke: Den modellerede knude kan også opdage, at den allerede står i rutelisten i "Route Request"-optionsheaderen i pakkebrikken, og så vil der opstå en løkke heri, hvis knuden føjer sin adresse til rutelisten. I dette tilfælde må knuden ikke simulere en viderebroadcast af pakkebrikken.

At den modellerede knude allerede optræder i rutelisten kan ske, hvis knuden allerede har simuleret en viderebroadcasting af pakkebrikken én gang, men derefter har behandlet så mange andre "Route Request"-pakkebrikker fra samme afsenderknude, inden den relevante pakkebrik igen blev modtaget, at indgangen på "Route Request Table, Externally Originated Packets"-pladsen er blevet slettet igen, idet tabellen ikke må være vilkårlig stor pr. afsenderknude jvf. afsnit 2.3.5.

Oprydningen i "Route Request Table, Externally Originated Packets" foregår på CPN-siden "MaintainRouteReqTableExt_RD_RecvSide", som ligner oprydningen af "Route Cache" (nederste del af figur 3.15 på side 57), og vises derfor ikke her – den kan ses i appendix B.1 som figur B.3 på side 141.

"Blacklist"-check: Hvis DSR-modellen er konfigureret til at modellere et "Bidir-only"-netværk (se afsnit 2.2.5), skal den modellerede naboknude, pakkebrikken blev modtaget fra, slås op i tabellen "Blacklist". Dette sker på undersiden "CheckBlacklistBeforeForward_RD_RecvSide" (figur 3.23 på side 69). Alt efter hvilken tilstand naboknuden har i tabellen, kan en af tre ting ske på denne side (som beskrevet i afsnit 2.3.3):

"NoStatus": Naboknuden er ikke nævnt i "Blacklist" (hvilket i modellen er modelleret som, at den har status "NoStatus" jvf. afsnit 3.2.2): "Route Request"-optionsheaderen sendes blot tilbage til "PossiblyForwardRouteReq_RouteDisc_RecvSide" til videre behandling.

"Unidirectionality probable": Det er sandsynligt, at benyttelsen af rutelister, der inkluderer en direkte pakketransmission fra naboknuden til denne knude højst sandsynligt vil fejle. I dette tilfælde lader den modellerede knude som om, den ikke har modtaget optionsheaderen.

"Unidirectionality questionable": Den modellerede knude skal simulere en broadcasting af en "Route Request"-pakke med naboknuden som destinationsknude, og med `TTL`-feltet sat til 1. Dette betyder, at kun hvis de to knuder er inden for simuleret antennerækkevidde af hinanden (som det beskrives i afsnit 3.6.4), vil naboknuden kunne sende en "Route Reply"-pakkebrik tilbage. Før dette sker, fjernes eventuelle ruter af længde 0 (målt som antal knuder mellem afsender- og modtagerknuden) fra "Route Cache"-pladsen, og der checkes herefter for, om et svar modtages inden for en vis timeout. Hvis et sådant svar modtages, vil dette opdatere "Route Cachen" jvf. afsnit 3.3.1, så denne blot overvåges af en transition for, om ruter af længde 0 til naboknuden opstår, for at afgøre, om et svar på "Route Request"-pakkebrikken blev modtaget. Dette er ikke helt i overensstemmelse med specifikationen [JMH05], hvilket jeg vender tilbage til i afsnit 3.4.5.

Alt efter hvilket resultat, testen gav, opdateres den relevante indgang på "Blacklist"-tabel-pladsen.

I et senere afsnit (afsnit 3.7) beskrives det, at modellen ikke er færdigmodelleret for "Bidir-only"-netværk. En del af denne manglende færdigmodellering gælder netop "Blacklist"-tabel-checket, der kun er modelleret på "PossiblyForwardRouteReq_RD_RecvSide" (figur 3.22 på side 66) – d.v.s. for "Route Request"-pakker, der skal simuleres viderebroadcastet. I en endelig model skal også pakkebrikker, der modtages af den *endelige* destinationsknude (og som dermed skal resultere i en "Route Reply"-pakkebrik, se figur 3.21 på side 65), først gennemgå ovenstående "Blacklist"-filtrering. Desuden foregår checket på "PossiblyForwardRouteReq_RD_RecvSide" i den forkerte rækkefølge nu – jvf. specifikationen [JMH05] skal "Blacklist"-checket ske *før* den modtagne pakke føjes til "Route Request Table, Externally Originated Packets" – i modellen sker det efter. Dette gør, at eksemplet beskrevet i afsnit 2.3.3 f.eks. ikke vil blive reddet af den måde, "PossiblyForwardRouteReq_RD_RecvSide" er bygget op på nu i "Bidir-only"-netværk. Dette vil dog ikke have betydning for de gennemførte analyser i kapitel 5, da disse udelukkende benytter modellerede "Mostly-Bidir"-netværk.

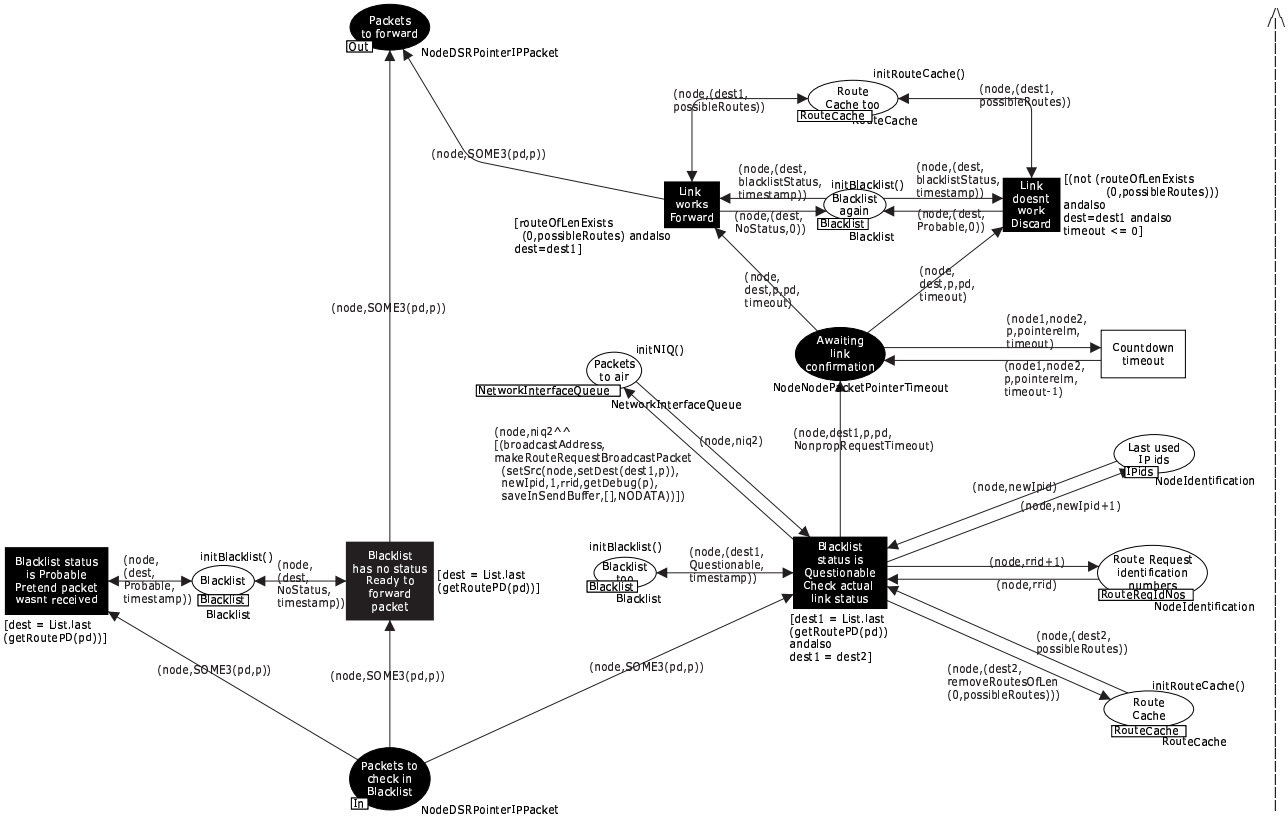
Hvis ingen af disse ovenstående checks gør, at pakkebrikken ikke skal behandles videre af "PossiblyForwardRouteReq_RD_RecvSide" (figur 3.22 på side 66), vil pakkebrikken nu være nået til transitionen "Packets to forward" øverst til højre i figuren.

På dette sted skal den modellerede knude (hvis CRR-udvidelsen er slået til i konfigurationen af DSR-modellen) checke for, om den på sin "Route Cache"-plads har en rute til "Route Request"-optionsheaderens destinationsknude. Har den ikke det, tilføjes knudens adresse til "Route Request"-optionsheaderens opsamlede ruteliste, og pakkebrikken simuleres broadcastet videre – direkte til det underliggende OSI-lag (udenom RM, da det netop er en broadcast, modellen har simulerer).

Hvis den modellerede knude derimod har mulighed for at returnere en CRR (jvf. afsnit 2.3.6), får undersiden "SendCachedRouteReply_RD_RecvSide" pakkebrikken. Her genereres en "Route Reply"-pakkebrik med den sammensatte ruteliste fra den opsamlede ruteliste i "Route Request"-optionsheaderen og den fundne rute fra knudens "Route Cache". "SendCachedRouteReply_RD_RecvSide" ligner til forveksling "SendRouteReply_RD_RecvSide" (figur 3.21 på side 65), og er derfor ikke gengivet her – den kan evt. ses i appendix B.1 som figur B.4 på side 142. Den eneste reelle forskel på de to sider ligger i *hvilken* rute, der benyttes i den genererede "Route Reply"-pakkebrik.

"Route Reply"-optionsheaderen

Behandlingen af en "Route Reply" er meget simplere end en "Route Request". Det eneste, modellen gør, når den skal behandle en "Route Reply"-pakke, er at udtrække ruteinformation



Figur 3.23: DSR-modellens CPN-side “CheckBlacklistBeforeForward_RD_RecvSide”

fra den via undersiden “AddToRouteCache_DSR_RecvSice” (figur 3.14 på side 57) og returnere pakken til “DSR_RecvSide”. Siden “ProcRouteReply_RD_RecvSide”, der gør dette, er derfor ikke vist her, men kan ses som figur B.5 på side 142 i appendix B.1.

At siden kan være så simpel, skyldes, at de steder, hvor der ventes på ruter i CPN-modellen, bliver transitioner automatisk fyringsklare, når en relevant rute bliver tilføjet “Route Cache”-pladsen. Dette gælder f.eks. siden “Maintenance_RD_SendSide” (figur 3.19 på side 63).

3.4.4 Specifikationspræcisering: Piggybacking generelt

RD-delen af DSR-modellen opfører sig på et par områder anderledes end specifikationen [JMH05] foreskriver. Det første område beskæftiger sig med piggybacking af optionsheaders.

Når “Initiate_RD_SendSide” (figur 3.17 på side 61) konfigureres med “includeInPacket”-værdien, skal en pakkebriks DSR-optionsheaders piggybackes til den nye “Route Request”-pakkebrik. Dette benyttes ved returnering af “Route Reply”-pakkebrikker (for at undgå en uendelige løkke i den simulerede udsendelse af “Route Request”-pakker) og inklusion af “Route Error”-pakkebrikker (for at undgå, at andre knuder returnerer ukurant information, hvis de vil returnere en CRR-pakkebrik som svar på en “Route Request”-pakkebrik).

Normalt vil man ved “piggybacking” af information til noget andet information forstå, at den nye information tilknyttes den oprindelige information – i “halen” af denne.

Men når DSR-modellen modtager en pakkebrik fra modellen af det underliggende OSI-lag, kræver specifikationen, at DSR-optionsheaderne afkodes og behandles i den rækkefølge, som de står i netværkspakken (som beskrevet i afsnit 3.3.2). Hvis informationen blot piggybackes til “Route Request”-pakkebrikken, bliver f.eks. en “Route Reply”-pakkebrik genereret

og simuleret sendt ud som svar på en "Route Request"-optionsheader, inden den piggybackede "Route Reply"- eller "Route Error"-optionsheader bliver behandlet, og dermed bliver den piggybackede information ikke brugt.

Derfor bliver optionsheaders, der skal piggybackes til en pakke, sat ind *først* i DSR-optionsheaderliste i min model. At det skal gøres således, er ikke beskrevet nogen steder i DSR-specifikationen [JMH05] eller den tilhørende litteratur, der beskæftiger sig med piggybacking i DSR [JM96, JMB01].

3.4.5 Specifikationsfejlretning: Blacklist-check ved modtagne "Route Request"-pakker

Når CPN-modellen er konfigureret til at simulere et "Bidir-only"-netværk, og en modelleret knude skal behandle en modtaget "Route Request"-pakke, kan den som beskrevet i punktet "Unidirectionality questionable" under "Blacklist-check" på side 68 i afsnit 3.4.3 komme ud for, at indgangen til naboknuden, som den modtog "Route Request"-pakken fra, i knudens "Blacklist"-tabel er mærket med "unidirectionality questionable". Dette betyder, at knuden skal undersøge, hvorvidt linket til naboknuden er bidirektionelt eller ej.

Specifikationen siger, at knuden i dette tilfælde skal udsende en "Route Request"-pakke til naboknuden og herefter vente på en "Route Reply"-pakke som svar derpå. For at undgå, at "Route Request"-pakken routes igennem en anden knude, sættes `TTL`-feltet til 1, og desuden *unicastes* pakken direkte til destinationsknuden (d.v.s. at den indeholder en "DSR Source Route"-optionsheader). Modtages en "Route Reply"-pakke nu, anses det for verificeret, at linket mellem de to knuder er symmetrisk. Dette er imidlertid ikke nødvendigvis en rigtig fortolkning: Hvis A unicaster en "Route Request"-pakke til B med `TTL` sat til 1, kan en knude C i nærheden af A stadig modtage pakken, hvis den har slået "promiscuous mode" i sit netkort til. Hvis C kender en rute til B og desuden har slået udvidelsen CRR til, kan den nu svare A med denne rute. Dette specialtilfælde er ikke medtaget i specifikationen [JMH05].

Problemet kan løses på forskellige måder. Den måde, der benyttes i DSR-modellen, er at checke, at en af de "Route Reply"-pakker, der modtages, har en rute af længde 0 (ekskl. afsender- og modtagerknude). Dette vil indikere, at linket er bidirektionelt, da naboknuden så har modtaget afsenderknudens "Route Request"-pakke og har svaret med en "Route Reply"-pakke (hvilket i bidirektionelt konfigurerede netværk altid vil ske via den reverserede ruteliste). Hvis en knude C som i ovenstående tilfælde også svarer på "Route Request"-pakken, vil det ikke forstyrre dette, da disse ruter altid vil have en rutelængde på mindst 1.

Alternativt kunne man blot have kigget på, om afsenderen af "Route Reply"-pakken svarer til destinationen for "Route Request"-pakken. Dette ville dog kræve, at den del af modellen, der checker "Blacklist"-tabellen (figur 3.23 på forrige side), skulle modtage "Route Reply"-pakkerne direkte. Dette ville komplicere modellen, og er dette ikke valgt her. Den valgte løsning kræver blot, at "Route Cache"-pladsen først renses for ruter af længde 0 til destinationsknuden og derefter overvåges for sådanne ruter.

Desuden kan modellen forsimples på et andet område: Det er ikke længere nødvendigt at *unicaste* "Route Request"-pakkerne; de må gerne broadcastes ud. "Route Reply"-pakkerne vil stadig blive unicastet, så linket checkes stadig for bidirektionalitet. Det er derfor ikke nødvendigt, at en "DSR Source Route"-optionsheader tilføjes en "Route Request"-pakke. Dette ville være det eneste sted i modellen, dette ville have skullet ske, så også dette har gjort modellen mindre kompleks.

3.4.6 Specifikationsproblemområde: Piggybacking i allerede igangsatte "Route Discoveries"

Under normale omstændigheder skal en pakkebrik, der ønskes modelleret sendt til en destinationsknude, som en modelleret afsenderknude ikke har en rute til, gemmes på "Send Buffer"-

pladsen, og en RD skal igangsættes, hvis en sådan ikke allerede er i gang. I visse tilfælde (ved returnering af "Route Reply"- og "Route Error"-pakker som beskrevet i afsnit 3.4.4) skal pakken dog ikke gemmes i "Send Buffer", men skal piggybackes til den nye "Route Request"-pakkebrik. Hvis en RD allerede er igangsat, må der ikke genereres en ny "Route Request"-pakkebrik straks, og derfor er der ikke nogen ny pakkebrik at piggybacke indholdet af "Route Reply"- eller "Route Error"-optionsheaderen til. DSR-specifikationen [JMH05] og den tilhørende litteratur, der inkluderer en beskrivelse af kravet om en backoff-algoritme ([JMB01]), nævner ikke, hvad der skal gøres i dette specialtilfælde, men der findes forskellige løsninger:

Pakkebrikken kan smides væk: Dette er naturligvis ikke nogen optimal løsning, idet informationen i den ønskede afsendte pakkebrik i så fald mistes, men bemærk, at information piggybacket til "Route Request"-pakker i forvejen ikke er garanteret fremkomst, idet protokollen ikke specificerer nogen form for bekræftelser undervejs, og "Route Request"-pakker ikke genudsendes, idet ingen information her gemmes i "Send Bufferen".

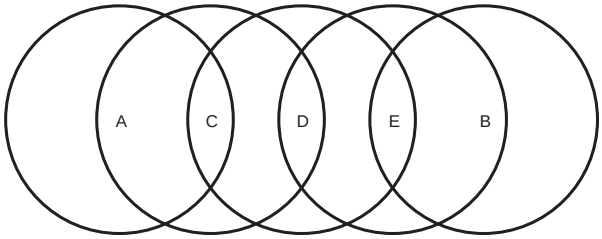
Men det, at informationen konsekvent mistes, kan give et problem, hvis to knuder samtidigt går i gang med en RD gående på hinanden, idet begge knuder så vil begynde at smide "Route Reply"-pakker til den anden knude væk, indtil de får ruteinformation angående den anden knude. Om dette kan give en reel deadlock-situation i netværket, eller opsamlingen af ruteinformation undervejs fra videresendte og overhørte netværkspakker (se afsnit 2.2.5) sørger for, at dette altid undgås, undersøger jeg nærmere nedenfor.

Piggybacking til næste udsendelse: Pakkebrikkens optionsheader kan piggybackes til den "Route Request"-pakkebrik, der allerede forsøges sendt, så informationen bliver inkluderet næste gang, "Route Request"-pakkebrikken simuleres genudsendt. Hvis en rute modtages af knuden inden næste gang, "Route Request"-pakkebrikken simuleres genudsendt, mistes informationen dog ligesom ved løsning, hvor pakkebrikken smides væk. Også dette betyder, at en "Route Reply"-pakkebrikken ikke nødvendigvis vil blive sendt ud, som den skal. Desuden kan sandsynligheden for, at man modtager ugyldig ruteinformation (som følge af, at en "Route Error"-pakkebrik, en knude gerne ville have sendt, ikke er blevet sendt), blive forstørret.

Pakkebrikken, der forsøges sendt, og dennes optionsheaders gemmes dog ifølge specifikationen ikke nogen steder – kun information om den sidst brugte TTL, antal pakker sendt indtil videre etc. gemmes. Dette foregår i "Route Request Table, Internally Originated Packets", som man derfor er nødt til at udvide, hvis denne løsning skal benyttes. Herefter kan optionsheaderen tilføjes fremtidige udsendelser af "Route Request"-pakken.

Dette skal sammenlignes med, at når en knude i prokollen i det "normale" tilfælde (d.v.s. tilfældet, hvor en RD ikke allerede er igangsat) udsender en "Route Request"-pakke piggybacket med f.eks. en "Route Reply"-optionsheader, vil denne piggybackede optionsheader kun blive inkluderet i den første "Route Request"-pakke. Hvis der ikke modtages noget svar på "Route Request"-pakken, genudsendes "Route Request"-pakken *uden* den piggybackede optionsheader. Det står ikke direkte i specifikationen [JMH05], at det skal foregå på denne måde, men man må udlede dette ud fra det faktum, at optionsheaders, der ønskes piggybacket til "Route Request"-pakker, ikke gemmes i nogen datastruktur.

Gemme på "Send Buffer": Pakkebrikken kan gemmes på "Send Buffer"-pladsen på trods af, at den egentlig skulle piggybackes til en "Route Request"-pakkebrik. Dette sikrer, at når en rute til destinationsknuden findes, bliver informationen i pakkebrikken sendt til destinationsknuden – ingen information mistes. Dog vil information ikke blive sendt, før der findes en rute til destinationsknuden, så denne løsning kan være udsat for samme problemstilling som løsningen, hvor pakkebrikken smides væk.



Figur 3.24: Setup af knuder i et simuleret netværk i CPN-modellen til undersøgelse af mulig deadlock i modellen. Ringene angiver knudernes antennerækkevidder.

Kombination: En kombination af de to sidste muligheder kan benyttes: Informationen piggybackes til den "Route Request"-pakkebrik, der allerede forsøges gensendt, men hvis en rute findes, før pakkebrikken bliver gensendt, sendes informationen i stedet direkte via den nyfundne rute. Også denne løsning kræver en udvidelse af de datastrukturer, der er specificeret i DSR-specifikationen [JMH05] – ud over udvidelsen fra piggybackingløsningen vil den f.eks. kræve, at man udvider "Route Request Table, Internally Originated Packets" med en pointer, der viser, hvilke dele af pakkebrikkernes optionsheaders, der indtil videre er forsøgt sendt.

Udsende straks alligevel: "Route Request"-pakken udsendes blot med det samme – og den "Route Reply"- eller "Route Error"-optionsheader, der ønskes sendt til destinationen, piggybackes dertil. Dette vil løse alle de ovenstående problemer, men specifikationen [JMH05] er meget specifik omkring, at dette *ikke* må gøres, da udsendelsen *skal* følge en "backoff"-algoritme.

Da jeg har ønsket at lægge mig så tæt op ad datastrukturerne fra specifikationen [JMH05] i modellen som muligt, for at have en så præcis model som muligt i forhold til specifikationen, har jeg valgt at fravælge piggybackingløsningen og kombinationsløsningen. Desuden har jeg fravalgt straks-udsendelsen af en ny "Route Request"-pakke, da dette også strider mod specifikationen. I stedet har jeg i modellen valgt at benytte løsningen, hvor pakken gemmes på "Send Buffer". Yderligere analyser, der ligger uden for dette speciale, er dog nødvendige for at se præcist hvilken løsning, der vil give den bedste effekt – f.eks. om kombinationsløsningen som beskrevet vil kunne løse de eventuelle problemer i de tre første løsninger.

Jeg vil i stedet her undersøge, om løsningerne, der ikke kræver en udvidelse af datastrukturerne, og ikke direkte forbydes af specifikationen, kan resultere i en deadlock i netværket som antydet ovenfor.

Jeg har kørt en række simuleringer for at undersøge denne situation nærmere. DSR-modellen er konfigureret, så et netværkssetup svarende til figur 3.24 er brugt, hvor de modellerede knuder A og B samtidigt giver en pakkebrik til DSR-laget med hinanden som destinationsknude for pakkebrikken. Dette vil udløse en simulering af den ovenstående situation. I appendix C.1 findes en kommandoliste, der konfigurerer modellen til at simulere denne situation – brugen af disse kommandolister vender jeg tilbage til i afsnit 3.6.3. DSR-modellen kan desuden konfigureres til at simulere, at den arbejder under en af de tre netværkstyper, der blev introduceret i afsnit 2.2.5 ("Frequently-unidir", "Mostly-bidir" og "Bidir-only"), og hver af de modellerede udvidelser "Salvage Operations" (SO) og "Cached Route Reply" (CRR) kan slås til eller fra. I appendix C.2 kan resultaterne med modellen konfigureret med forskellige kombinationer af de ovenstående konfigurationer ses. Resultaterne er opfanget v.h.a. en type dataopsamling, der introduceres i afsnit 3.6.4. Resultaterne af disse kørsler kan ses i tabel 3.25 på næste side.

I "Frequently-unidir"-tilfældet fås præcist det resultat, der blev forudset i det ovenstående: En deadlock-situation opstår, og ingen pakkebrikker når frem. I både "Mostly-bidir"- og

Netværkstype	DSR-udv. brugt	Resultat fra simuleringer
"Frequently-unidir"	Ingen	Ingen pakker nåede frem
"Frequently-unidir"	CRR og SO	Ingen pakker nåede frem
"Mostly-bidir"	Ingen	Begge pakker nåede frem
"Mostly-bidir"	CRR og SO	Begge pakker nåede frem
"Bidir-only"	Ingen	Begge pakker nåede frem
"Bidir-only"	CRR og SO	Begge pakker nåede frem

Tabel 3.25: Undersøgelse af mulig deadlock-situation, når to knuder igangsætter "Route Discoveries" gående på hinanden samtidigt. De præcise pakketransmissioner kan ses i appendix C.2.

"Bidir-only"-tilfældene når pakkebrikkerne til gengæld frem alligevel.

I "Mostly-bidir"-tilfældet (uden nogen DSR-udvidelser slået til) bliver pakkebrikken fra den modellerede knude A sendt lige så snart, "Route Request"-pakkebrikken fra knude B når frem – og omvendt. Der ventes ikke på "Route Reply"-pakkebrikker (omend disse også bliver returneret fra B til A og omvendt i netværket). Dette skyldes, at situationen bliver reddet af funktionaliteten i "AddToRouteCache_DSR_RecvSide" (se afsnit 3.3.2): Når B's "Route Request"-pakkebrik når frem til A, gemmer A den reverserede rute til B i sin "Route Cache" – og kan umiddelbart begynde at bruge den til både at sende en "Route Reply"-pakkebrik tilbage til B og sende pakkebrikken, der er gemt på A's "Send Buffer"-plads, til B.

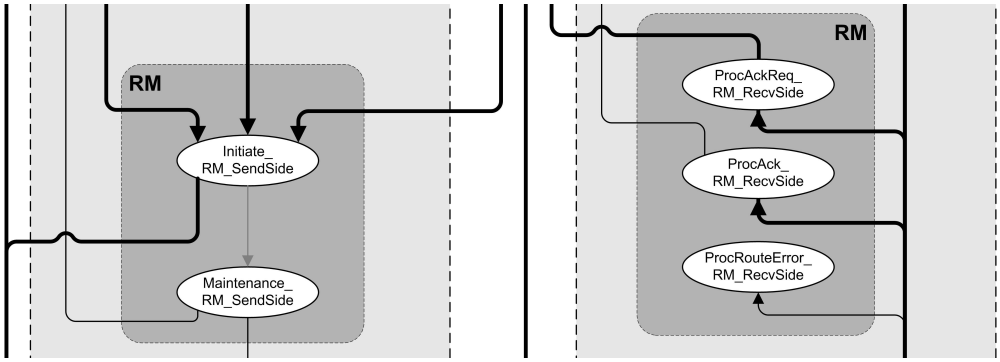
Slås CRR til i "Mostly-bidir"-tilfældet, sker der det samme som i ovenstående tilfælde, bortset fra, at det bliver de modellerede mellemliggende knuder, der svarer på "Route Request"-pakkebrikkerne ud fra den reverserede information i deres "Route Cache". Når f.eks. den modellerede knude D modtager "Route Request"-pakkebrikken fra A, har den allerede processeret "Route Request"-pakkebrikken fra B og kender derfor resten af ruten til B – og kan generere en CRR. Som beskrevet i afsnit 2.2.5 caches noget mindre information i "Frequently-unidir"-netværk. Derfor gav CRR ikke samme effekt for denne.

Tilfældet, hvor DSR-modellen er konfigureret med et "Bidir-only"-netværk, er kun medtaget for fuldstændighedens skyld, idet returnering af en "Route Reply"-pakkebrik aldrig må resultere i en igangsat RD i "Bidir-only"-netværk, da disse altid skal returneres over den reverserede rute fra den opsamlede ruteliste i "Route Request"-pakkebrikken. Dermed udløses en simulering af ovennævnte problemsituation aldrig i denne type netværk.

Alt i alt kan man ud fra dette konkludere, at der i hvert fald er et seriøst problem med DSR-modellen (og dermed potentielt også med DSR-protokollen), når den benyttes i et "Frequently-unidir"-netværk. I retrospekt ville det have været formålstjenstligt at modellere kombinationspunktet på forrige side (der benytter en løsning, der direkte modsiger opbygningen af datastrukturerne i specifikationen [JMH05]) som en variation af DSR-modellen og undersøge, om denne løser disse problemer. Dette måtte dog desværre vige til fordel for modelleringen og undersøgelserne af CRR og SO.

3.5 Modellering af "Route Maintenance"

Den sidste del af CPN-modellens DSR-lag er "Route Maintenance" (RM, som beskrevet i afsnit 2.4). En logisk oversigt over hovedsiderne, der hører til RM, kan ses i figur 3.4 på side 44. De relevante sider er genvist i figur 3.26 på næste side – men uden sammenkædningen med resten af CPN-modellen, som den oprindelige figur viser. Ud over disse sider findes der en række hjælpesider til "Maintenance_RM_SendSide" og et par datastrukturvedligeholdelsessider: "MaintainMaintBuffer_RM_SendSide" og "MaintainBlacklist_RM_SendSide". Alle disse sider vil blive gennemgået i afsnit 3.5.1-3.5.3, mens afsnit 3.5.4-3.5.5 beskriver fundne problemområder o.lign. i den officielle specifikationer indenfor RM.



Figur 3.26: CPN-sidesammenhænge: Hovedflowet mellem hovedsiderne i "Route Maintenance" (udsnit af figur 3.4 på side 44). Afsenderdelen er til venstre i figuren, modtagerdelen til højre.

3.5.1 Modelleret afsendelse af pakker i "Route Maintenance"

RM-afsenderdelen af DSR-modellen fungerer som beskrevet i afsnit 2.4. Den står for simuleringen af forsendelsen eller videresendelsen af en pakke via den rute, der står i pakkens "DSR Source Route"-optionsheader. Afsenderdelens igangsætterdel kan ses i figur 3.27 på næste side. Flowet på siden er ganske simpelt: Pakkebrikker, der ikke indeholder en "Acknowledgement"-optionsheader, skal gemmes i kopi på "Maintenance Buffer"-pladsen (så den kan blive simuleret gensendt, indtil en aktiv eller passiv bekræftelse modtages, som det beskrives i næste afsnit). Hvis pakkebrikken skal simuleres videresendt via det sidste hop i rutelisten i pakkens "DSR Source Route"-optionsheader, tilføjes der en "Acknowledgement Request"-optionsheader til pakkebrikkens optionsheader, da modtagelsen af en passiv bekræftelse her ikke er mulig. Herefter gives pakkebrikken til det modellerede underliggende OSI-lag, der vil sørge for simuleringen af den rent faktiske forsendelse af pakkebrikken (se afsnit 3.6.4).

Hvis pakkebrikken allerede indeholder en "Acknowledgement"-optionsheader, siger specifikationen [JMH05], at en kopi af pakkebrikken ikke må gemmes i "Maintenance Buffer" (hvorfra pakken ellers ville blive forsøgt gensendt som beskrevet i næste afsnit). En pakkebrik kan allerede indeholde en sådan "Acknowledgement"-optionsheader, hvis modellen f.eks. er konfigureret til at være af netværkstypen "Bidir-only" (se afsnit 2.2.5), og det underliggende modellerede OSI-lag tillader en modelleret knude A at simulere forsendelser af pakkebrikker til en anden modelleret knude B, mens laget ikke tillader B at simulere forsendelser til A direkte; men kun via en tredie modelleret knude C. En illustration af dette kan ses i figur 3.28 på næste side. Hvis A simulerer en pakkebrikforsendelse til B med et "AcknowledgementRequest" i optionsheaderen, vil modellen netop komme i den situation, at C skal simulere en pakkebrikforsendelse fra B til A med en "Acknowledgement"-optionsheader, og disse må jvf. specifikationen [JMH05] ikke sendes mere end én gang, og må ikke få tilføjet en "AcknowledgementRequest"-optionsheader.

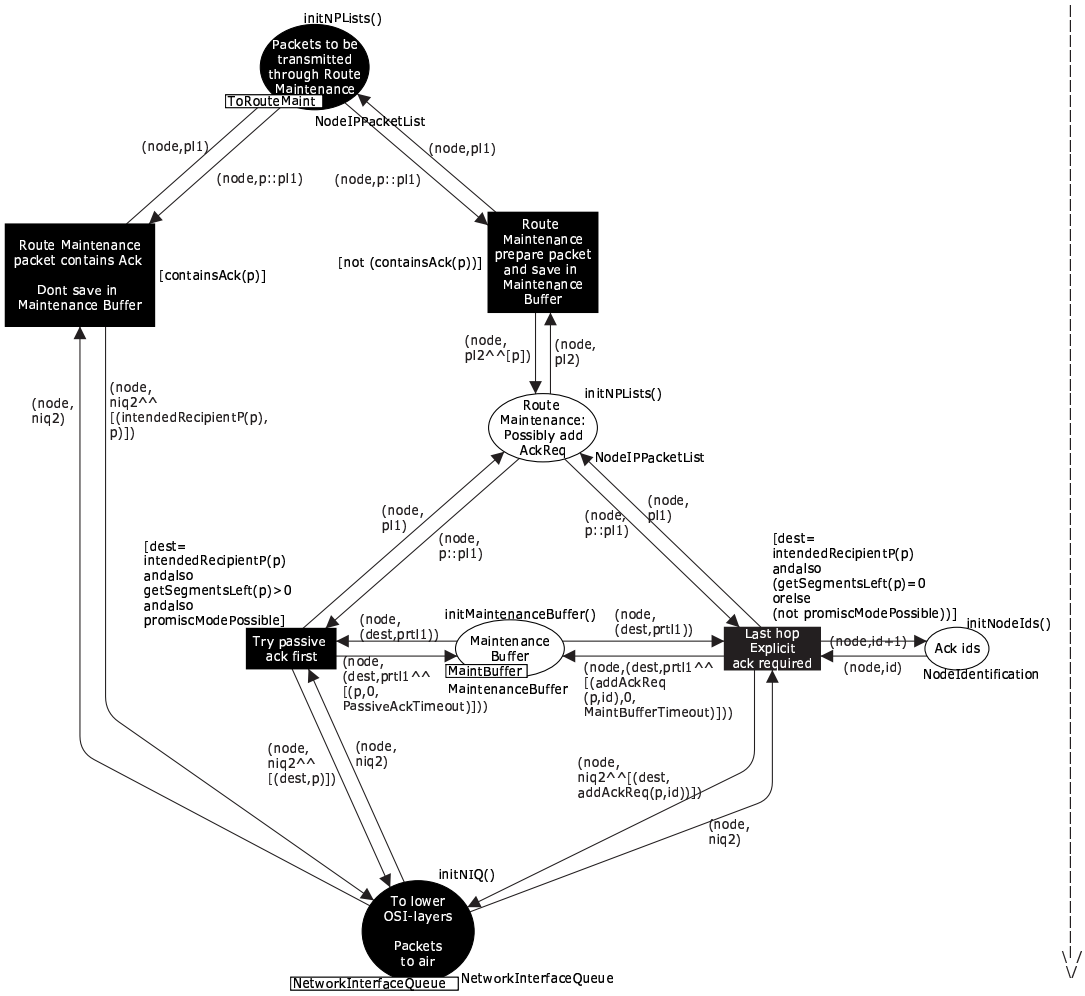
På siden simuleres forsendelse af pakkebrikker ved at gemme dem på "Network Interface Queue"-pladsen (hvorfra en CPN-side tilhørende det underliggende OSI-lag vil hente dem – se afsnit 3.6.4).

Specielt om modelleringen af "Network Interface Queue"

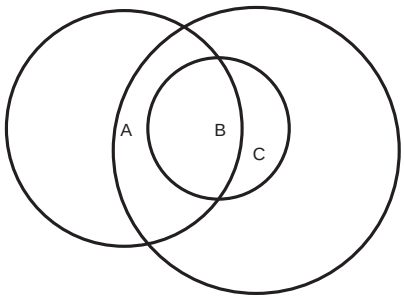
"Network Interface Queue" har som den eneste datastruktur ikke en foreslået type i specifikationen [JMH05]. Dette skyldes, at datastrukturen benyttes til at sende netværkspakker videre til underliggende OSI-lag, og derfor ofte vil være bestemt af disse.

Jeg har valgt følgende datastruktur for "Network Interface Queue" for en knude:

```
[(target, packet)]
```



Figur 3.27: DSR-modellens CPN-side “Initiate_RM_SendSide”



Figur 3.28: Tre knuder, A, B og C, med ringe, der angiver deres antennerækkevidder. Bemærk, at linket fra A til B ikke-symmetrisk.

– altså en liste af pakker, der hver er knyttet til den næste naboknude, de skal sendes til. Ved "Route Request"-pakker er `target` en broadcast-adresse. Angivelsen af destinationsknuden vil kunne blive benyttet af det underliggende OSI-lag i netværkssetuppet "Bidir-only" til at finde ud af hvilken adresse, den skal handshake med (jvf. afsnit 1.5.3). I de øvrige netværkstyper skal destinationsknude-feltet ikke aflæses, idet pakken blot bliver sendt ud i simuleringen af det trådløse netværk (og derfor modtages af alle modellerede knuder, der i modellen er simuleret inden for antennerækkevidde af den afsendende knude, som det beskrives i afsnit 3.6.4).

Ovenstående er ikke ensbetydende med, at knuder i "Bidir-only" ikke kan gå i "promiscuous mode", som det udnyttes flere steder i DSR-protokollen (se kapitel 2) – et netværkskort kan godt være i stand til at overhøre pakker, der egentlig ikke er tiltænkt dens knude, selvom pakkerne er afsendt via en handshake-protokol. Derfor kan modellen også konfigureres til at benyttes promiscuous mode, så den ovenstående broadcast-agtige funktionalitet også kan bruges, selvom modellen er konfigureret til at benytte et "Bidir-only"-netværk.

Den ovenstående løsning står i kontrast til en anden fremgangsmåde, der kan benyttes, hvor datastrukturen for "Network Interface Queue" for en knude blot er:

```
([packet])
```

– og hvor modellen af det underliggende OSI-lag så kigger i pakkebrikken for at finde destinationsknuden. Denne metode svarer til, at det underliggende OSI-lag er i stand til at forstå og udtrække information af DSR-optionheaders – specifikt "DSR Source Route" og "Route Request". Jeg har fravalgt denne løsning, eftersom den giver en sammenblanding af OSI-lagene, men den benyttes f.eks. i DSR-implementationen DSR-UU [Nor05].

3.5.2 Modelleret vedligeholdelse af afsenderdelen i "Route Maintenance"

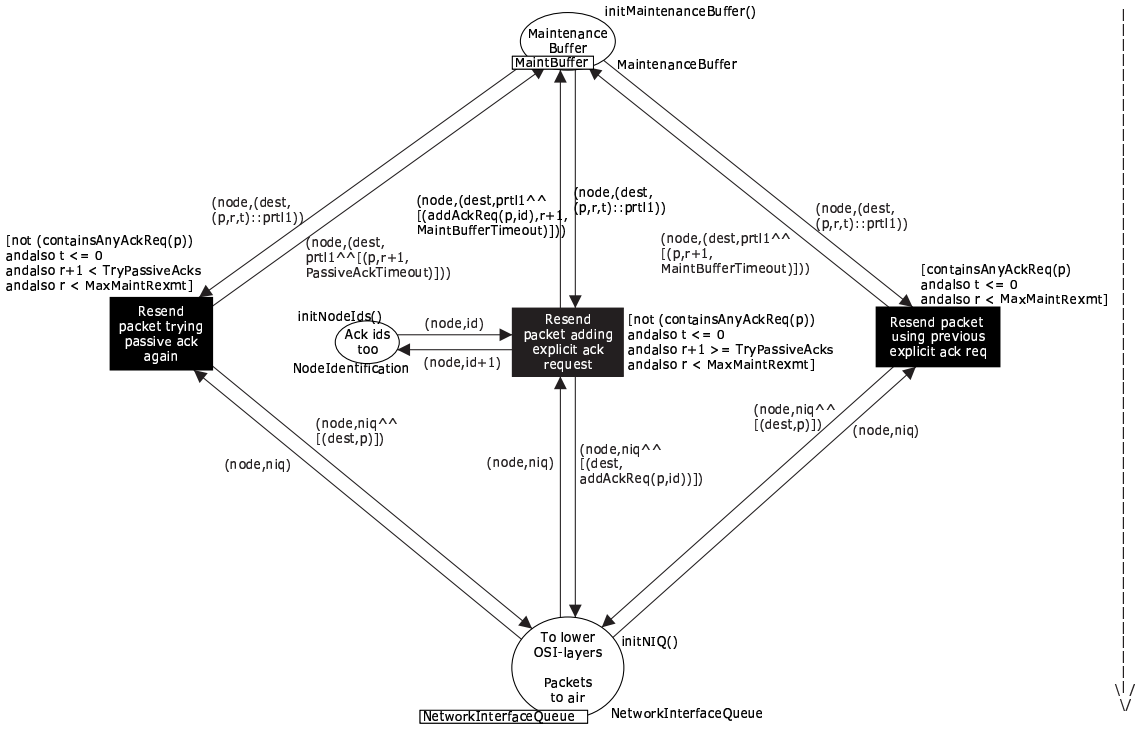
Selve det at sørge for, at pakkebrikker gennemgår en simuleret gentransmission, hvis en passiv eller eksplicit bekræftelse ikke er modtaget, klares af CPN-siden "Maintenance_RM_SendSide" (vist i figur 3.29 på næste side).

I afsnit 2.4.1 blev forskellige typer bekræftelser beskrevet. Som foreslået i specifikationen [JMH05] benyttes de i modellen på følgende måde:

- Først forsøges pakkebrikken simuleret sendt et antal gange, mens der ventes på en passiv bekræftelse. CPN-siden "ProcDSRSrcRt_DSR_RecvSide" (afsnit 3.3.2) sørger for at fjerne kopien af pakkebrikken fra "Maintenance Buffer"-pladsen igen, hvis der modtages en passiv bekræftelse.

- Herefter forsøges pakkebrikken simuleret sendt et antal gange med anmodning om en eksplicit bekræftelse, altså med en "Acknowledgement Request"-optionsheader tilføjet pakkebrikken. Her er det CPN-siden "ProcAck_RM_RecvSide" (se afsnit 3.5.3), der sørger for at fjerne kopien af pakkebrikken fra "Maintenance Buffer"-pladsen igen, hvis en eksplicit bekræftelse modtages.

Som udgangspunkt har jeg valgt, at der i modellen kun én gang skal forsøges med passive bekræftelser, hvorefter der to gange forsøges at få en eksplicit bekræftelse, da det er disse værdier, der foreslås benyttet i DSR-specifikationen [JMH05]. Som i specifikationen er dette dog modelleret på den måde, at passive bekræftelser forsøges benyttet 1 gang, og eksplicitte bekræftelser derefter forsøges benyttet, indtil pakkebrikken i alt er simuleret *gen*udsendt 2 gange. Forskellen er, at når en modelleret knude simulerer en forsendelse af en pakke via det sidste link i rutelisten – d.v.s. til den endelige destinationsknude – må afsenderknuden ikke forsøge at få en passiv bekræftelse først (da modtagerknuden netop ikke vil simulere en videreforsendelse af pakkebrikken), og den ovenstående forskel vil så betyde, at protokollen skal forsøge at få eksplicitte bekræftelser fra modtagerknuden 3 gange (i stedet for kun 2).



Figur 3.29: DSR-modellens CPN-side "Maintenance_RM_SendSide"

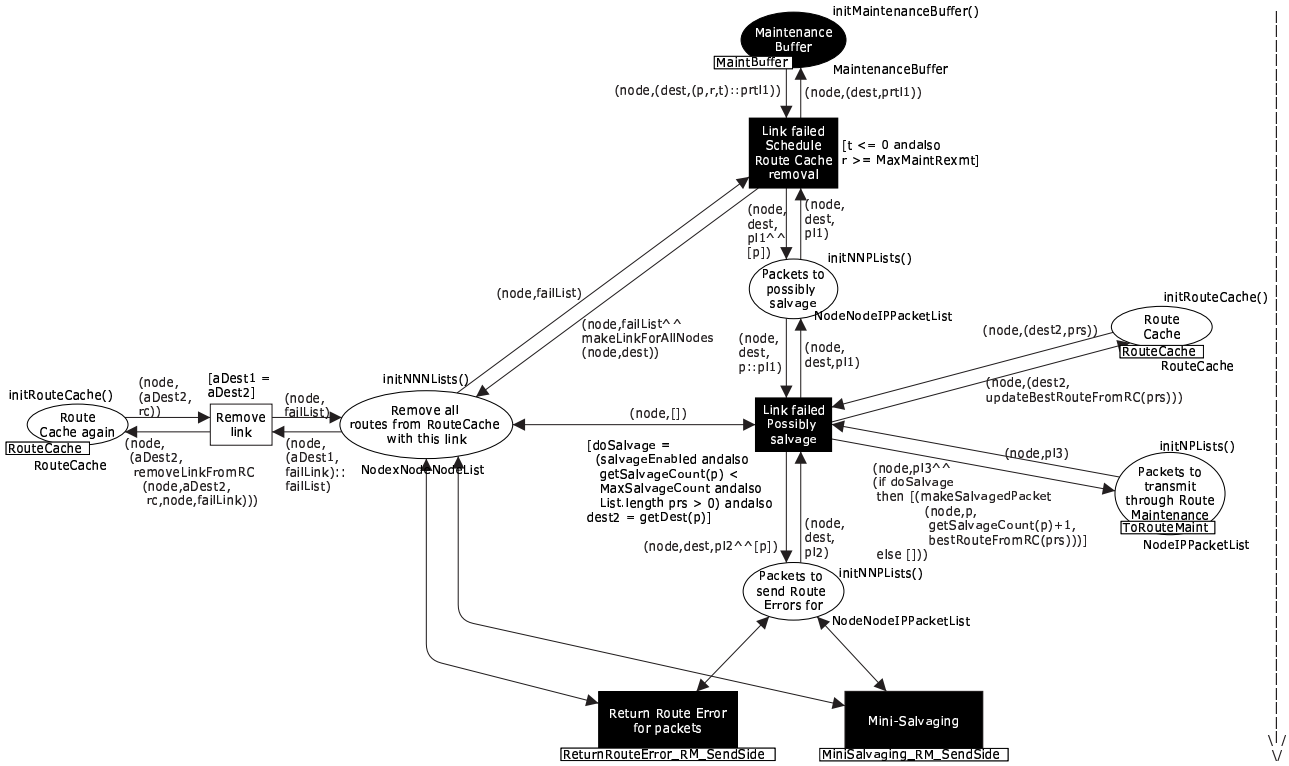
Gentransmissionerne af pakkerne sker efter et vist timeout – "PassiveAckTimeout" for pakkebrikker, som en modelleret knude forsøger at få en passiv bekræftelse tilbage for, og "MaintBufferTimeout" for pakkebrikker, som en modelleret knude forsøger at få en eksplicit bekræftelse tilbage for.

Specifikationen [JMH05] siger, at man enten kan benytte en fast værdi eller en adaptiv algoritme til at fastslå, hvor meget timeout'en bør være, når man skal have en bekræftelse tilbage fra en nabo-knude. Et eksempel på en sådan adaptiv algoritme gives, men denne kræver en mere detaljeret tidsmodellering, end der benyttes i modellen. En fast værdi, der kan benyttes, hvis man vælger ikke at benytte en adaptiv algoritme, foreslås kun for indgange, der venter på en passiv bekræftelse – denne timeout er på 100 ms.

I modellen har jeg sat en fast timeout på de øvrige indgange i "Maintenance Buffer" til 500 ms ud fra den betragtning, at hvor passive bekræftelser altid modtages direkte tilbage fra den modellerede naboknude, som en modelleret knude simulerer en pakkeforsendelse til, så kan besvarelsen af eksplicit bekræftelsesanmodning risikere at skulle tage en anden rute end over det direkte link (hvis det direkte link f.eks. er unidirektionelt, eller naboknuden blot ikke har det direkte link i sin "Route Cache"). Timeout'en bør afspejle, at denne type bekræftelser derfor kan tage længere tid.

Selve timeout-nedtællingen for "Maintenance Buffer" foregår på CPN-siden "Maintain-MaintBuffer_RM_SendSide"-siden. Denne er ikke gengivet her, da den til forveksling ligner f.eks. optællerdelen af "MaintainRouteCache_DSR" (figur 3.15 på side 57) bortset fra, at der i den figur tælles op i stedet for ned. Siden kan evt. ses i appendix B.1 som figur B.6 på side 143.

"Maintenance_RM_SendSide" (figur 3.29) håndterer dog ikke alle tilfælde, der kan forekomme i vedligeholdelsen af en pakke, der forsøges transmitteret af RM – et enkelt tilfælde mangler, nemlig det, hvor det *ikke* lykkes modellen at simulere en forsendelse eller videresendelse af en pakkebrik via et link. Denne situation håndteres af hjælpesiden "HandleFai-



Figur 3.30: DSR-modellens CPN-side "HandleFailingLinks_RM_SendSide"

lingLinks_RM_SendSide" i figur 3.30. Siden er ikke en underside til "Maintenance_RM_-SendSide", men svæver (som "Maintenance_RM_RecvSide" selv) frit i CPN-modellen (se figur 3.3 på side 43). Dette skyldes som tidligere beskrevet, at alle input- og output-pladser på disse sider er gjort til fusionspladser for at gøre CPN-siderne mere overskuelige.

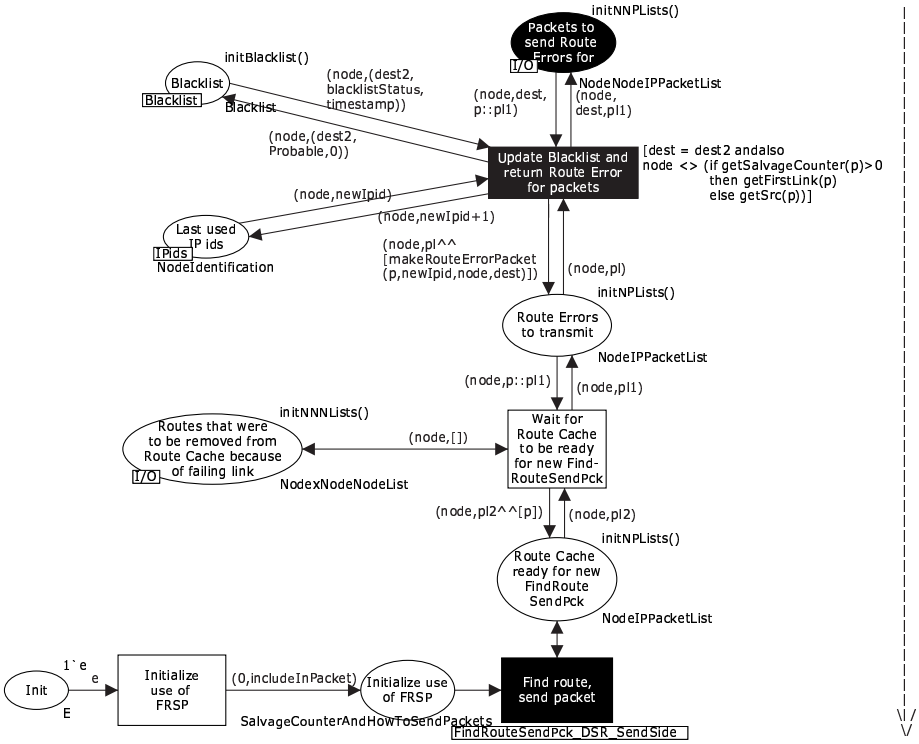
Det første, der sker, når et link er fejlet, er at alle ruter indeholdende det fejlende link fjernes fra "Route Cache"-pladsen, så en modelleret knude ikke kommer til at benytte ukurant information i forbindelse med f.eks. SO.

Der kan nu ske tre forskellige ting:

"Salvage Operation": Hvis SO er slået til i konfigurationen af DSR-modellen, og den modellerede knude kender en anden rute til destinationsknuden, "salvages" pakken som beskrevet i afsnit 2.4.6. I praksis skal der kun ske simple operationer i modellen: Pakkebrikkens ruteliste skal udskiftes, `salvage_counter` skal tælles op, og pakkebrikken skal simuleres gensendt gennem RM.

Returnering af "Route Error": Uanset om pakkebrikken bliver simuleret forsøgt videresendt via en anden rute eller ej i det foregående punkt, skal modellen generere en "Route Error"-pakkebrik til den modellerede knude, der oprindeligt simulerede afsendelselsen af pakkebrikken. Dette foregår på undersiden "ReturnRouteError_RM_SendSide" (figur 3.31 på næste side), som følger et ganske simpelt flow: Først indser modellen, at den er i en situation, hvor den modellerede knude ikke kan simulere forsendelse af pakkebrikken til en modelleret naboknude (men naboknuden er muligvis stadig i stand til at simulere forsendelse af pakkebrikker til knuden). Derfor indsættes naboknuden i knudens "Blacklist"-tabel-plads som beskrevet i afsnit 2.4.5.

Herefter simulerer modellen, at der sendes en "Route Error"-pakkebrik tilbage til den oprindelige afsenderknude v.h.a. CPN-siden "FindRouteSendPck_DSR_SendSide" –



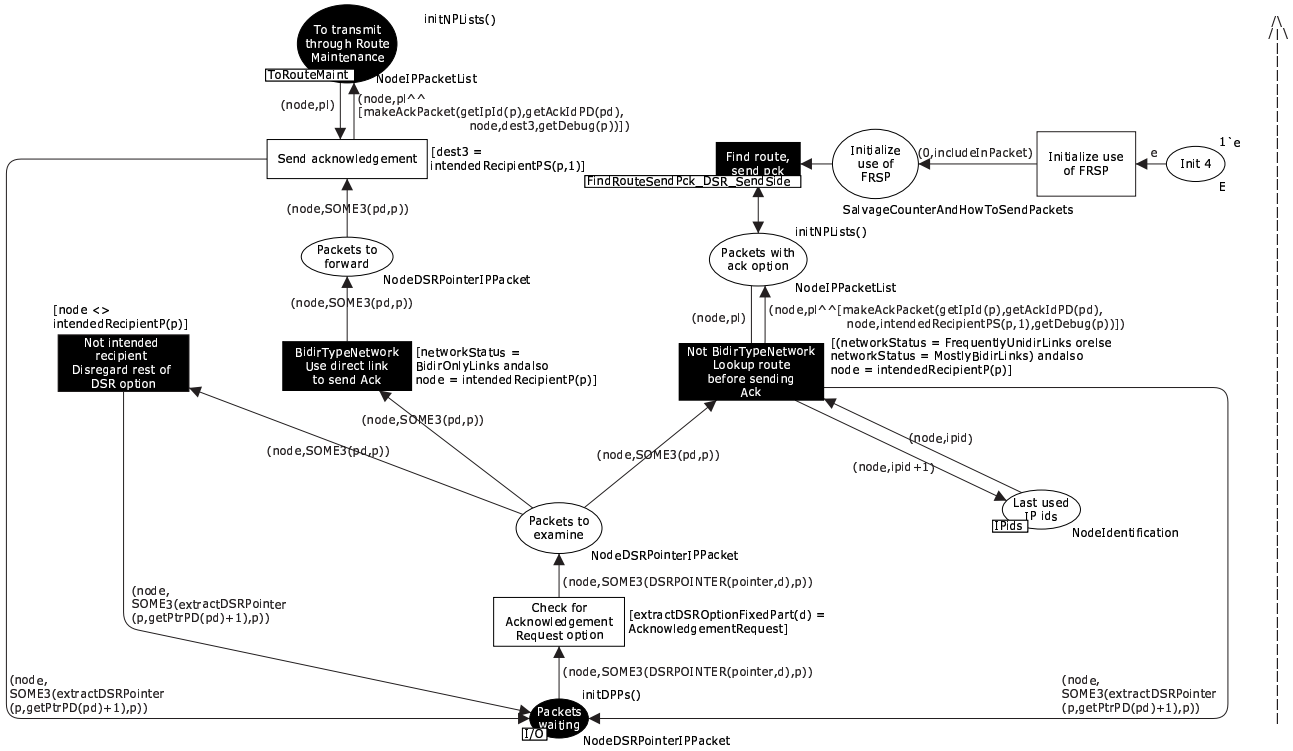
Figur 3.31: DSR-modellens CPN-side "ReturnRouteError_RM_SendSide"

og (som beskrevet i afsnit 3.3.1) konfigureres siden til at piggybacke "Route Error"-pakkebrikken i en evt. "Route Request"-pakke, hvis det bliver nødvendigt at igangsætte en ny RD. Dette gøres, så andre modellerede knuder ikke eventuelt kan komme til at sende en CRR (se afsnit 2.3.6) tilbage indeholdende en rute med det fejlende link.

"Mini-Salvage Operation": Det foregående punkt dækker dog også over et specialtilfælde: Hvis den modellerede knude, der i modellen oprindeligt genererede pakkebrikken, også er den knude, der opdagede, at et link i rutelisten i pakkebrikkens "DSR Source Route"-optionsheader fejlede, d.v.s. at det var det første link i rutelisten, der fejlede, skal det ifølge specifikationen modelleres således, at knuden sender en "Route Error"-pakke til sig selv. Dette tilfælde er udvidet i modellen, og behandles i stedet af undersiden "MiniSalvaging_RM_SendSide", som beskrives i afsnit 3.5.5.

De indgange, der indsættes i "Blacklist"-tabellen i det ovenstående, får knyttet tilstanden "Unidirectionality Probable" på sig. Efter en timeout skal tilstanden ændres til "Unidirectionality Questionable", og efter et yderligere timeout skal indgangen fjernes helt (som i DSR-modellen modelleres ved, at den får tilstanden "NoStatus"). Dette foregår på CPN-siden "Maintain-Blacklist_RM_SendSide", som følger samme princip som timeout-behandlingen og fjernelsen af elementer fra "Route Cache" (figur 3.15 på side 57). Siden er derfor ikke gengivet her, men kan ses i appendix B.1 som figur B.7 på side 143.

Der er ikke specificeret en eksempel-timeout til brug i "Blacklist"-tabellen – hverken til hvor hurtigt tilstanden skal ændres fra "probable" til "questionable", eller hvor hurtigt indgange med tilstanden "questionable" skal fjernes fra tabellen. Jeg har i modellen sat begge disse timeouts til 10 sekunder (= 10000 ms). Det er gjort ud fra en betragtning om, at når en knude indsætter en naboknude i sin "Blacklist", sker det, fordi linket dertil ikke er bidirektionelt, men befinder sig i et grænsetilfælde, hvor en knude kan sende til den anden, men ikke



Figur 3.32: DSR-modellens CPN-side “ProcAckReq_RM_RecvSide”

omvendt. Hvis knudernes antenner er ens (som de er i min model, hvilket jeg vender tilbage til i afsnit 3.7.2), vil dette ofte skyldes, at knuderne bevæger sig på kanten af hinandens antennerækkevidder, og timeout’en bør derfor ikke være alt for stor, da forholdene hurtigt kan ændre sig. Hvis man kan gå ud fra, at man har forskellige antennerækkevidder i et netværkssetup, bør man naturligvis vælge en anden strategi til fastlæggelse af disse timeouts.

3.5.3 Modelleret modtagelse af pakker i “Route Maintenance”

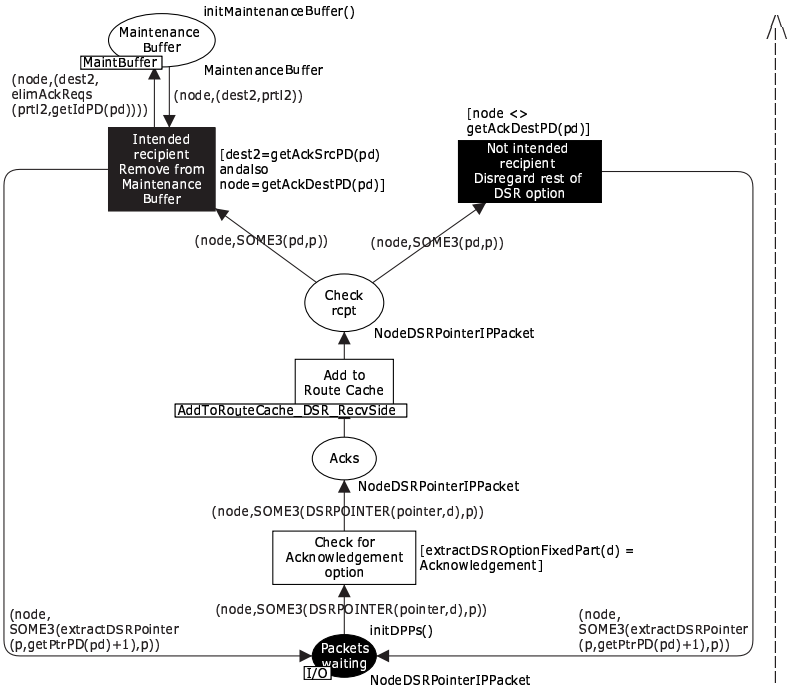
Som tidligere beskrevet i afsnit 3.3.2 sørger CPN-siden “DSR_RecvSide” (figur 3.12 på side 54) for at kalde undersider, der sørger for at behandle de enkelte optionsheaders i modtagne pakkebrikker. I forbindelse med RM kan der modtages tre typer optionsheaders: “Acknowledgement Request”, “Acknowledgement” og “Route Error”.

“Acknowledgement Request”-optionsheaderen

CPN-siden, der håndterer modtagne “Acknowledgement Request”-optionsheaders, kan ses i figur 3.32. Flowet på siden er som følger: Pakkebrikken modtages som på de øvrige undersider tilknyttet “DSR_RecvSide”-siden på “Packets waiting”-pladsen.

Hvis den modtagende knude er den endelige modtager af pakken, genereres en “Acknowledgement”-pakke, og denne simuleres sendt tilbage til afsenderen som beskrevet i afsnit 2.4.1, d.v.s. enten, hvis modellen er konfigureret som et “Bidir-only”-netværk, direkte via et link til den naboknude, der sendte pakken, eller, hvis modellen er konfigureret som et “Frequently-unidir”- eller “Mostly-bidir”-netværk, v.h.a. en rute fundet i “Route Cache” eller v.h.a. RD.

Hvis modellens netværkskonfiguration er af subtypen “Mostly-bidir”, vil “Route Cachen” her allerede indeholde en rute tilbage til afsenderknuden, hvor ruten ikke går via andre knuder,



Figur 3.33: DSR-modellens CPN-side “ProcAck_RM_RecvSide”

idet den reverserede rute vil være blevet gemt i “Route Cachen” under processeringen af “DSR Source Route”-optionsheaderen jvf. afsnit 2.2.5. Kun i tilfældet “Frequently-unidir” gemmes reverserede ruter ikke, og kun her er det derfor muligt, at en RD skal igangsættes.

“Acknowledgement”-optionsheaderen

CPN-siden, der håndterer modtagne “Acknowledgement”-optionsheaders, kan ses i figur 3.33. Flowet på siden er som følger: Hvis en “Acknowledgement”-optionsheader modtages på siden i en pakkebrik, der er sendt til den modtagende knude som endelig destinationsknude, fjernes kopien af den relevante pakkebrik fra knudens “Maintenance Buffer”-plads. Som beskrevet i afsnit 3.5.2 betyder dette, at RM ikke længere vil forsøge at gensende pakkebrikken, og bekræftelsen her dermed haft sin virkning i modellen.

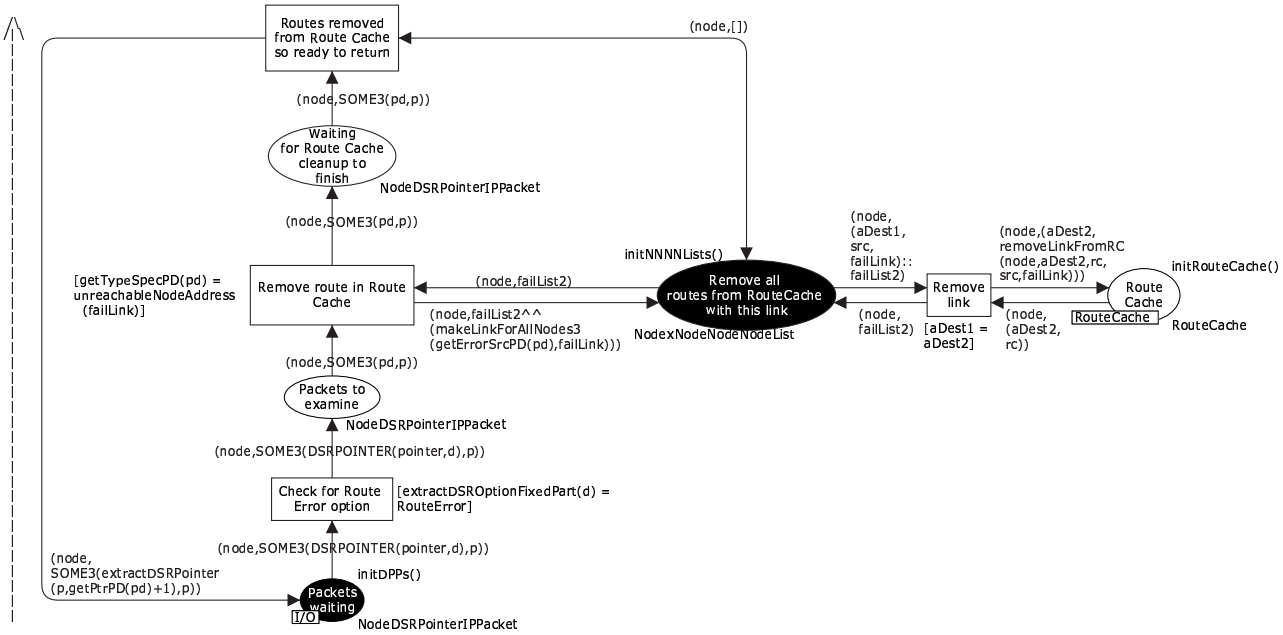
“Route Error”-optionsheaderen

CPN-siden, der håndterer modtagne “Route Error”-optionsheaders, kan ses i figur 3.34 på næste side. Når en “Route Error”-pakkebrik modtages på siden via “Packets waiting”, fjernes alle ruter indeholdende det fejlende link fra knudens “Route Cache”.

Da jeg i CPN-modellen har valgt at benytte en datastruktur, der består af lister af “komplette ruter” til hver destinationsknude (jvf. afsnit 2.2.4), bliver alle ruter til alle destinationer gennemgået af siden for at sikre, at alle ruter med det fejlende link bliver fjernet.

3.5.4 Specifikationsproblemområde: Piggybacking af “Route Errors”

RM-delen af DSR-modellen opfører sig på et par områder anderledes end specifikationen [JMH05]. Det første tilfælde er i brugen af piggybackede “Route Error”-optionsheaders:



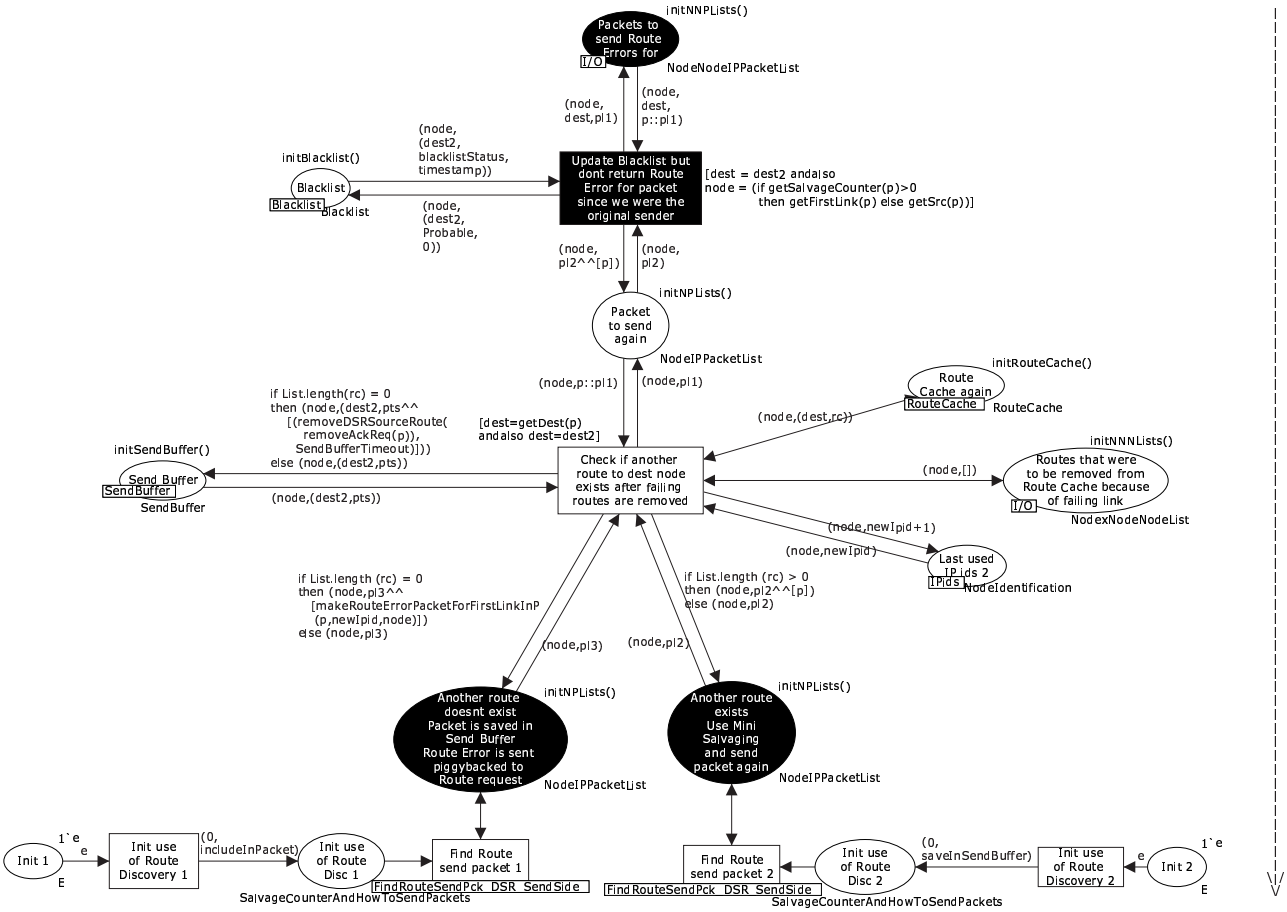
Figur 3.34: DSR-modellens CPN-side “ProcRouteError_RM_RecvSide”

Hvis en knude har forsøgt at sende en pakke til en anden knude, og har fået en “Route Error”-pakke tilbage som svar, skal afsenderknuden fra sin “Route Cache” slette ruten, der forårsagede “Route Error”-pakken. Hvis knuden herefter vil sende endnu en pakke til den samme destinationsknude (f.eks. fordi et TCP-lag i knuden har opdaget, at den første pakke ikke nåede frem, og forsøger at gensende denne), kan dette naturligt resultere i, at knuden starter en RD gående på destinationsknuden, fordi den nu ikke længere har flere ruter dertil.

Specifikationen [JMH05] siger, at i så tilfælde skal den modtagne “Route Error”-options-header piggybackes til den “Route Request”-pakke, der bliver udsendt som en del af RD-proceduren. Dette gøres for at undgå, at knuder mellem afsenderknuden og destinationsknuden kan komme til at sende en ugyldig ruteliste med det fejlende link tilbage til afsenderknuden (se afsnit 2.4.2), hvis disse benytter sig af CRR.

Da DSR-protokollen er designet til at være en on-demand-algoritme (som karakteriseret i afsnit 1.3.2), vil en ny RD ikke automatisk blive igangsat, når en knude modtager en “Route Error”-pakke, og der er derfor ikke i modellens DSR-lag en logisk forbindelse mellem det, at en modelleret knude modtager en “Route Error”-pakkebrik, og det, at den igangsætter en ny RD. Derfor betyder kravet om, at “Route Error”-pakker piggybackes til nye “Route Request”-pakker, at modtagne “Route Error”-pakkebrikker skal gemmes i en datastruktur – men en sådan datastruktur er ikke defineret i specifikationen [JMH05] ligesom de timeoutværdier, der skal benyttes hertil, ikke er angivet. Da jeg har ønsket at lægge mig så tæt op ad datastrukturerne i specifikationen som muligt, har jeg derfor valgt ikke at modellere denne piggybacking.

Specifikationen [JMH05] angiver dog også, at hvis en rute bliver fjernet som følge af en modtaget “Route Error”-pakke, og der ikke er flere ruter til denne destinationsknude i knudens “Route Cache”, må knuden godt igangsætte en RD øjeblikkeligt alligevel, hvis den har en tydelig indikation af, at der vil blive brug for denne rute igen (f.eks. fordi knuden stadig har en åben TCP-forbindelse til destinationsknuden). Dette vil i de tilfælde, hvor knuden får lov til at igangsætte en ny RD, løse det ovenstående problem. Da det modellerede, ovenliggende OSI-lag i modellen ikke har et sådant koncept (som det kan ses i afsnit 3.6.2), er denne funktionalitet ikke medtaget i modellen.



Figur 3.35: DSR-modellens CPN-side "MiniSalvaging_RM_SendSide"

3.5.5 Specifikationsoptimeringsforslag: Indførelse af "Mini-Salvage Operations"

Som beskrevet i afsnit 3.5.2 simulerer modellen i praksis kun, at en modelleret knude sender en "Route Error"-pakkebrik retur, hvis knuden ikke selv er den oprindelige afsenderknude – er den derimod det, behandles pakkebrikken i stedet af CPN-siden "MiniSalvaging_RM_SendSide" (figur 3.35).

Det tilfælde, der er dækket af siden, er ikke dækket af specifikationen [JMH05], som tværtimod siger, at der altid skal sendes en "Route Error" tilbage – i princippet også, selvom den skal sendes til knuden selv. Desuden siger specifikationen, at hvis en knude får en "Route Error" tilbage angående en pakke, den har forsøgt at sende ud, skal den fjerne den fejlende rute fra sin "Route Cache", men den skal ikke forsøge at genudsende pakken – det er op til et højereliggende OSI-lag at sørge for dette. Dette krav i specifikationen kan skyldes, at det ikke i DSR-protokollen kræves, at afsenderknuder gemmer afsendte pakker, efter at forsendelsen via det første link er verificeret, for ikke unødigt at komplicere protokollen – et sådant krav kunne afføde, at det ville blive relevant for den endelige destinationsknude at sende en fuld-rute-bekræftelse tilbage til den oprindelige afsenderknude for at tilkendegive, at afsenderknuden ikke længere behøver gemme pakken.

Hvis "MiniSalvaging_RM_SendSide" modtager pakkebrikken, har den modellerede, oprindelige afsenderknude dog stadig pakkebrikken, idet det netop er det første link, der fejler,

	"Salvage Operations"	**"Mini-Salvage Operations"**
Knuden, der opdager et fejlende link, er...	En knude i rutelisten (*efter* afsenderknuden)	Den oprindelige afsenderknude
Pakken modelleres videresendt via ny rute?	Hvis modellen er konfigureret til SO, og en alternativ rute haves	Altid
`salvage_counter`...	Tælles op i videresendt pakke	Tælles ikke op i gensendt pakke
"Route Error"-pakke...	Genereres og returneres	Genereres ikke (der er ingen knude at returnere den til)
Optimeringen er specificeret i [JMH05]?	Ja	Nej

Tabel 3.36: Forskel på SO og Mini-SO i CPN-modellen

og det gør det muligt at lave en yderst simpel optimering.

"MiniSalvaging_RM_SendSide" opdaterer "Blacklist"-tabellen som i tilfældet på CPN-siden "ReturnRouteError_RM_SendSide" (figur 3.31 på side 79), og vil herefter (når det er sikret, at det fejlende link er fjernet fra "Route Cache"-pladsen) undersøge, om der findes en anden rute til pakkenbrikkens destinationsknude i knudens "Route Cache". Findes der en sådan rute, benyttes "FindRouteSendPck_DSR_SendSide" (se afsnit 3.3.1) blot til at simulere en genforsendelse af pakkebrikken – denne gang via denne nye rute.

Findes en anden rute ikke i knudens "Route Cache", startes en ny RD gående på destinationsknuden. Imens gemmes pakkebrikken på "Send Buffer"-pladsen, og indholdet af den "Route Error"-pakkebrik, der på "ReturnRouteError_RM_SendSide" (figur 3.31 på side 79) ville have været simuleret sendt til den oprindeligt afsendende knude, piggybackes til "Route Request"-pakkebrikken for at undgå, at en anden knude i modellen kan komme til at returnerer en CRR-pakkebrik med ukurant information som svar på "Route Request"-pakkebrikken.

I virkeligheden foregår der en form for "Mini-Salvage Operation" (Mini-SO), der dog adskiller sig fra en normal SO på et enkelt punkt: Rutelisten i pakkebrikken, der simuleres gensendt i det ovenstående, er stadig den komplette ruteliste fra sourceadressen til destinationsadressen i IP-headeren. Dette betyder, at når andre modellerede knuder, der behandler pakkebrikken på dens vej til destinationsknuden, vil uddrage ruteinformation fra den, kan de medtage sourceadressen som en del af rutelisten (i modsætning til pakker, der har været udsatte for "rigtige" SO, hvor der normalt ikke er et link mellem sourceadressen og den første knude i rutelisten jvf. afsnit 2.4.6).

Jeg har valgt at prioritere denne inklusion af sourceadresser højere end andre knuders behov for at vide, at den første rute, der blev forsøgt, var fejlagtig, og derfor sætter jeg ikke `salvage_counter`-feltet til 1 i disse pakker – feltet sættes stadig til 0. Ulempen er, at hvis pakkebrikken også bliver "salvaged" af andre modellerede knuder i DSR-modellen, vil `salvage_counter` senere end normalt nå op til `MAX_SALVAGE_COUNT`.

Det er værd at bemærke, at modellen pr. definition ikke vil komme i en situation, hvor den udfører en Mini-SO på en pakkebrik så mange gange, at det i praksis ødelægger simuleringen af DSR. Dette skyldes, at det som beskrevet i afsnit 2.1.3 ifølge specifikationen [JMH05] er en forudsætning for benyttelsen af DSR-protokollen, at knuderne ikke bevæger sig så meget rundt, at en rute ikke kan findes og benyttes.

De præcise forskelle på SO og Mini-SO kan ses i tabel 3.36. I øjeblikket er optimeringen altid slået til i modellen. I kapitel 5 undersøges effekten af optimeringerne CRR og SO. På samme måde, som når disse undersøges, kan effekten af Mini-SO undersøges, og i en fremtidig model bør det af denne grund naturligvis være konfigurerbart, om optimeringen er slået til eller ej. Jeg valgte dog at prioritere undersøgelserne af CRR og SO højere end undersøgelsen af

Mini-SO. I retrospekt var dette en fejl – det er sandsynligt, at det kunne have givet interessant information at undersøge også effekten af denne – f.eks. om den modvirkes af andre dele af protokollen eller modvirker sådanne.

En demonstration af Mini-SO kan ses i afsnit 3.8.5.

3.5.6 Begrænsninger i forhold til specifikationen

På et enkelt andet punkt adskiller modellen sig fra specifikationen [JMH05]: Specifikationen siger, at når RM har opdaget, at et link i en ruteliste ikke længere fungerer, skal alle pakkebrikker, der i øjeblikket simuleres forsøgt sendt over dette link, (d.v.s. som modellen har en kopi af i "Maintenance Buffer" eller "Network Interface Queue"), tages ud af "Maintenance Buffer" og "Network Interface Queue", og "Route Error"-pakkebrikker skal genereres og modelleres returneret til de oprindelige afsendere for hver af disse pakkebrikker – dog kun med én "Route Error"-pakkebrik genereret pr. oprindelig afsender jvf. afsnit 2.4.2. Dette vil sørge for, at hvis en knude genererer en række pakkebrikker til den samme modtager via den samme rute med et ugyldigt link i, vil afsenderknuden kun få den første "Route Error"-pakkebrik tilbage.

At kun én pakke skal returneres er ikke modelleret med, men effekten af dette er relativt lille, da det kræver, at en knude sender mere end en pakke til en anden knude (og fejler). Jeg vender i afsnit 5.1.2 tilbage til, at man kun kan udføre simuleringer af modellen med relativt få pakketransmissioner, så dette blev prioriteret ned i modelleringen af protokollen i forhold til at få andre områder til at fungere korrekt i modellen. Begrænsningen bør naturligvis fjernes, hvis en mere færdig model over DSR-protokollen skal bygges på et tidspunkt.

3.6 Modellering af simulering af de omkringliggende OSI-lags funktionalitet

For at kunne bruge modellen af DSR-protokollen (beskrevet i de foregående afsnit i dette kapitel), har jeg modelleret en simulering af effekten af de højereliggende og lavereliggende OSI-lag. De højereliggende lag benyttes til at simulere en ønsket afsendelse af en netværkspakke fra en knude til en anden, mens de underliggende lag benyttes til at simulere, hvordan det trådløse netværk vil opføre sig, når det bliver bedt om at transmittere en pakke. Begge dele er beskrevet nærmere i de efterfølgende afsnit.

3.6.1 Omgivelsesdata og generering af tilfældige begivenheder

Først er det nødvendigt at se på, hvad en CPN-model er nødt til at inkludere for at kunne understøtte en DSR-model. En DSR-model kan jvf. de foregående afsnit bl.a. håndtere følgende situationer:

- En rute kan findes blandt modellerede knuder spredt ud over et område, og en netværkspakkebrik kan simuleres sendt via denne rute.
- Hvis en rute invalideres, kan dette opdages, og en ny rute kan findes.

Disse to punkter medfører, at man i modellen er nødt til at kunne modellere mindst følgende:

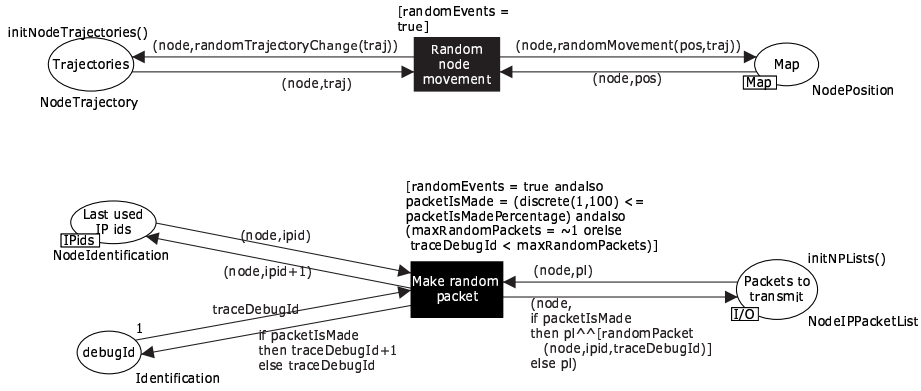
- Modellen skal modellere et "område".
- Modellen skal modellere et antal knuder i dette område.
- Modellen skal modellere, at knuderne kan bevæge sig i området.
- Modellen skal modellere, at knuderne kan sende pakker til hinanden.

```
(* Konfigurerbare variable *)
val mapSize            = 2000;
val maxSpeed           = 10;
val maxSpeedChange     = 2;

(* Definering af knuders placering i et område *)
colset Coor            = int with 1..mapSize;
colset MapPosition     = product Coor * Coor;
colset NodePosition    = product Node_ * MapPosition;

(* Definering af knuders retning og hastighed *)
colset Speed           = int with ~maxSpeed..maxSpeed;
colset SpeedChange     = int with ~maxSpeedChange..maxSpeedChange;
colset Trajectory      = product Speed * Speed;
colset NodeTrajectory  = product Node_ * Trajectory;
```

Figur 3.37: Modellering af typer tilhørende pladsen "Map" og typer brugt til repræsentation og manipulation af knuders placering i denne



Figur 3.38: DSR-modellens CPN-side "RandomEvents_OSILayer3AndUp"

Konceptet "knuder" er allerede benyttet i DSR-laget (definitionen kan ses i afsnit 3.2.2), men konceptet et "område" er nyt. Et område defineres som et kort, hvis størrelse konfigureres i DSR-modellen – det kan f.eks. være $500 * 500$ enheder (konceptuelt f.eks. meter) stort.

Herefter defineres det blot, at hver knude i DSR-modellen har en position inden for dette kort – hver knude får simpelthen et koordinatsæt i området. Den præcise ML-definition kan ses i figur 3.37.

Hver knudes bevægelse inden for området simuleres v.h.a. CPN-siden "RandomEvents_OSILayer3AndUp", som kan ses i figur 3.38. Knuderne får en tilfældig hastighed (inden for en maksimal grænse, f.eks. 10 enheder pr. gang, der flyttes) i enten positiv eller negativ X- og Y-retning. Da hastigheden bestemmes pr. X- og Y-retning, bestemmer dette også knudernes retning. Hver gang, knuderne flyttes (som sker, hver gang en speciel fyringstransition fyrer), ændres hastighederne (og dermed retningen) tilfældigt (inden for en maksimal grænse, f.eks. 2 enheder/fyring positivt eller negativt i hver retning).

Dette svarer til en forsimplet udgave af "Boundless Simulation Area Mobility Model" ("BSAMM" som beskrevet i [CBD02]). I min model vil det modellerede område dog ikke være "Boundless" i den betydning, der normalt bruges i BSAMM, hvor en mapning af en torus ned på en flade bruges til at bestemme, at hvis en knude forlader området et sted, vil den omgående blive teleporteret til et andet sted på randen af området med en ny retning,

der går ind i området igen. Når en knude når randen af området i DSR-modellen, ændres retningen blot, så den følger kanten, til den tilfældigt får en ny retning, der bevæger den ind i området igen. Jeg anser ikke dette for at være en perfekt, men dog mere korrekt opførsel – en sådan omgående teleportering vil ikke kunne finde sted i virkeligheden, så designerne af en MANET-protokol har sandsynligvis ikke forberedt protokollen på dette, hvilket kan få protokollen til at opføre sig unødigt suboptimalt i en simulering af en model i forhold til i en implementation, der benyttes i virkeligheden.

Jeg har valgt denne mobilitetsmodel frem for to andre modeller, der ellers er de mest populære i simulationer af MANET-protokoller (jvf. [CBD02]): "Random Walk", der for hver knude vælger en ny retning og hastighed hver gang, de har bevæget sig (hvilket gør, at knuder bevæger sig i et yderst urealistisk mønster jvf. [CBD02]), og "Random Waypoint", der for hver knude vælger en destination og en gennemsnitlig hastighed, som knuderne bevæger sig mod destinationen med. Når de når destinationen, holder de pause dér, indtil en ny destination vælges. Den sidste model ville også være god til formålet, men da jeg er nødsaget til at have korte simuleringer (se afsnit 5.1.2), valgte jeg en model uden pauser. Dette kunne naturligvis også have været gjort ved at vælge den sidste model og have elimineret pauserne fra den.

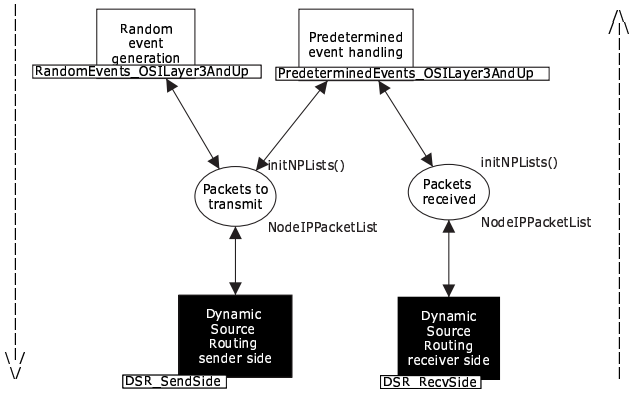
At knudernes hastighed bestemmes pr. X- og Y-retning betyder, at en knudes maksimale hastighed bestemmes af hvilken retning, den har – hvis den f.eks. bevæger sig stik nord på kortet, vil den maksimale hastighed være n – men hvis den bevæger sig stik nord-øst på kortet, vil den maksimale hastighed i stedet være $\sqrt{2n^2}$. Jeg har valgt at prioritere, at få modellen af DSR-protokollen til at fungere ordentligt, i stedet for at tilrette denne type inkonsistenser. I en "normal" "Boundless Simulation Area Mobility Model" benyttes polære koordinater til at undgå denne type problemer.

I figur 3.37 på forrige side kan det ses, at kortet er modelleret som et *diskret* kort (i stedet for et kontinuert kort – altså med koordinater angivet i heltal i stedet for reelle tal). Dette giver et problem i forhold til knudeflytninger: Hvis en knude skal flyttes f.eks. 20 enheder stik nord-øst, svarer dette til, at knuden skal flyttes $\sqrt{\frac{20^2}{2}} \approx 14{,}14$ enheder i både X- og Y-retningen på kortet. Dette kan ikke lade sig gøre på et kort repræsenteret med heltal. Som beskrevet i det ovenstående angives knudeflytninger dog slet ikke på denne måde i modellen, i stedet angives de netop direkte i antal enheder i henholdsvist X- og Y-retningen.

I retrospekt havde det dog nok været mere korrekt at lade kortet være kontinuert, da det dels ville give mulighed for at angive knudeflytningslængder på den anden måde, dels ville have modelleret virkeligheden mere korrekt. Metoden med et diskret kort benyttes også i f.eks. en DSR-model beskrevet i [Dem01], mens et kontinuert kort benyttes i f.eks. NS-2-simuleringer jvf. [AJ03].

På CPN-siden "RandomEvents_OSILayer3AndUp" (figur 3.38 på forrige side) kan det også ses, hvordan DSR-modellen simulerer, at en knude vil sende en pakke (med tilfældigt indhold) til en anden knude. For at undgå, at for mange pakker genereres inden for kort tid, kan en variabel "packetIsMadePercentage" sættes til en vis procentdel, f.eks. 20%. Dette vil betyde, at hver gang, CPN-modellen gerne vil fyre transitionen "Make random packet" er der kun 20% sandsynlighed for, at der rent faktisk produceres en pakke. Dette er en nem måde at styre på, hvor oversvømmet et netværk skal være af pakker. En anden variabel, "maxRandom-Packets", bestemmer hvor mange tilfældige pakker, der skal sendes ud i netværket – dette kan evt. være uendeligt mange (hvis variablen sættes til -1).

Siden benytter sig af tre forskellige pladser, der skal have tilfældige start-værdier: "Trajectories" (der styrer den retning og hastighed, de enkelte modellerede knuder har), "Map" (der styrer de modellerede knuders placering i modellen af et kort) og "Last used IP ids" (der styrer hvilket identifikationsnummer, de modellerede knuder sidst har brugt i genererede pakkebrikker, jvf. afsnit 3.3.2). De to sidste er fusionspladser (idet indholdet af pladserne også bliver brugt på andre sider), og grundet en bug i CPN Tools (beskrevet i afsnit 3.4.1) kan disse ikke initialiseres direkte med tilfældig data. Derfor har jeg lavet yderligere to CPN-sider, der



Figur 3.39: DSR-modellens CPN-side "OSILayer3AndUp"

står for initialiseringen af hver af disse to fusionspladser: "InitIPids_OSILayer3AndUp" og "InitMap_OSILayer3AndUp". Disse benytter samme fremgangsmåde som "InitRouteReqId-Nos_RD_SendSide" (figur 3.18 på side 62) og vises derfor ikke her. De kan evt. ses i appendix B.1 som figur B.8-B.9 på side 143.

3.6.2 OSI-lag 3 og op

Det er nu på tide at se, hvordan laget over DSR-laget i CPN-modellen er modelleret jvf. sammenhængen vist i figur 3.4 på side 44. Dette kan ses i figur 3.39. Figuren benytter de to DSR-undersider, der er beskrevet i de foregående afsnit, "DSR_SendSide" og "DSR_RecvSide" til henholdsvis at sende pakker gennem og modtage pakker gennem.

Figuren har (som den eneste figur i DSR-modellen) *to* stiplede linier. Dette angiver, at pakkeflowet både går nedad fra "RandomEvents_OSILayer3AndUp" og "PredeterminedEvents_-OSILayer3AndUp" til "DSR_SendSide" og opad fra "DSR_RecvSide" tilbage til "PredeterminedEvents_OSILayer3AndUp".

Siden har, som det kan ses i figuren, yderligere to undersider øverst. Disse er i stand til henholdsvist at generere tilfældige begivenheder og afspille en fastlagt begivenhedsrækkefølge. Den første af disse var den side, der blev beskrevet i afsnit 3.6.1, mens den anden beskrives i næste afsnit.

3.6.3 Simulering af reproducerbare omgivelser

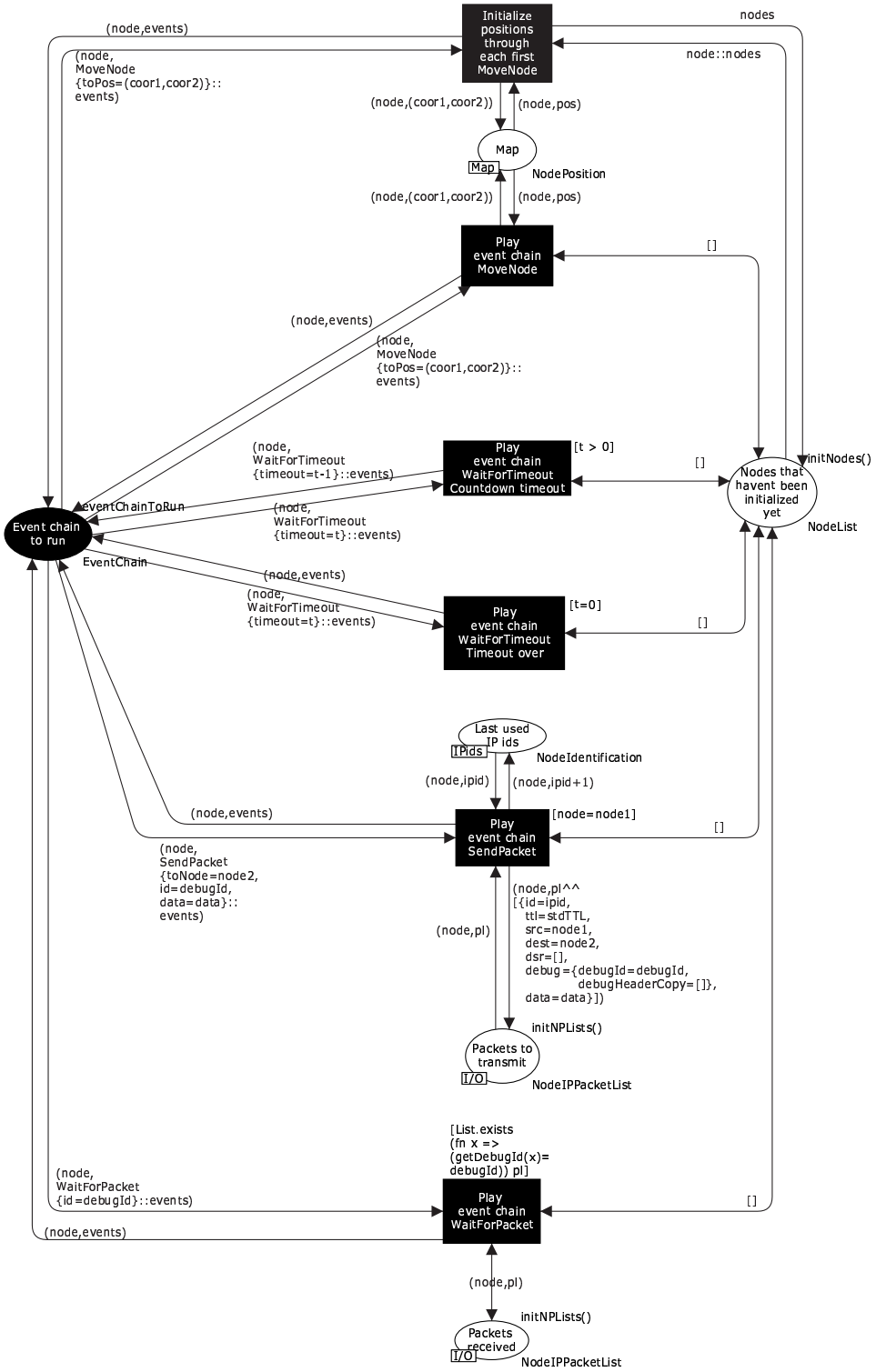
Den i afsnit 3.6.1 nævnte type tilfældig pakkebrikgenerering og tilfældige bevægelse af modellerede knuder gør det svært at lave testeksempler, der skal illustrere forskellige problemstillinger eller teste de samme situationer under forskellige konfigurationer af modellen (f.eks. med en optimering slået til eller fra).

Derfor kan man i CPN-modellen i stedet for den tilfældige model benytte et sprog til at angive, hvordan pakkebrikker skal genereres, og modellerede knuder skal bevæge sig. For hver modelleret knude kan man angive en liste af kommandoer, der skal udføres. Kommandoerne gemmes på en plads på CPN-siden "PredeterminedEvents_OSILayer3AndUp" (figur 3.40 på næste side).

Kommandosproget har jeg selv defineret. I sproget findes følgende kommandoer:

`MoveNode{toPos=(`*x*`,`*y*`)}:` Flytter en modelleret knudes position til (x, y).

`SendPacket{toNode=`*knude*`,id=`*id*`,data=`*tekst*`}:` Genererer en pakke med en bestemt knude som destination og med en debug-id sat i pakkeheaderen (som beskrevet i af-



Figur 3.40: DSR-modellens CPN-side “PredeterminedEvents_OSILayer3AndUp”

snit 3.2.2, så pakken kan spores rundt i netværket), og med userdata-delen af pakken sat til *tekst*.

`WaitForPacket{id=`*id*`}`: Sørger for, at modellen ikke går videre i en modelleret knudes kommandoliste, før knuden har modtaget en pakkebrik med debug-id *id*.

`WaitForTimeout{timeout=`*counter*`}`: Sørger for, at modellen ikke går videre i en modelleret knudes kommandoliste, før en timeouttransition har fyret *counter* gange for knuden.

`WaitForTimeout` kan virke overflødig, hvis man gerne vil lave en kommandoliste, hvor f.eks. en modelleret knude A sender en pakke til en anden modelleret knude B, som derefter sender en pakke tilbage til knude A. Imidlertid lover DSR-protokollen ikke, at pakker altid vil nå frem, da der ikke altid findes en fremkommelig rute mellem to knuder, så hvis man gerne vil undgå, at alle pakkebrikker skal simuleres sendt samtidigt rundt i netværket, er en kommando i stil med `WaitForTimeout` ønskelig.

En simpel kommandoliste kan f.eks. være den følgende:

```
1`("A",[MoveNode{toPos=(1,200)},
        SendPacket{toNode="B",id=1,data=SOMEDATA
                                   ("pakke 1 fra A til B via C")},
        WaitForPacket{id=2}]) ++
1`("B",[MoveNode{toPos=(1,700)},
        WaitForPacket{id=1},
        SendPacket{toNode="A",id=2,data=SOMEDATA
                                   ("pakke 2 fra B til A via C")}]) ++
1`("C",[MoveNode{toPos=(1,450)}])
```

Som det kan ses, er der en kommandoliste for hver knude i stedet for en samlet kommandoliste for alle knuderne. Hvis der i stedet var en samlet kommandoliste for alle knuderne, ville der kun være en enkelt liste, og alle kommandoer ville indeholde en knude-reference, d.v.s. f.eks. `MoveNode{node=`*knude*`,toPos=`(x,y)`}` (pånær `WaitForTimeout`, der netop fungerer på kommandoliste-niveau i stedet for knude-niveau).

Der er flere grunde til, at jeg ikke har valgt en samlet kommandoliste. Den vigtigste er, at et niveau af parallelisme mistes, hvis en sådan benyttes. En kommandoliste kunne f.eks. være:

```
[...,
 SendPacket{fromNode="A",toNode="B",id=1,data=...},
 SendPacket{fromNode="C",toNode="D",id=2,data=...},
 WaitForPacket{atNode="B",id=1},
 SendPacket{fromNode="B",toNode="A",id=3,data=...},
 WaitForPacket{atNode="D",id=2},
 SendPacket{fromNode="D",toNode="C",id=4,data=...},
 ...]
```

I dette tilfælde vil det simuleres, at A vil sende en pakke til B, og C vil sende en pakke til D. Herefter vil B vente på, at pakken fra A kommer frem, og vil derefter sende en pakke tilbage til A, og D vil vente på, at pakken fra C kommer frem, og vil derefter sende en pakke tilbage til C.

Der er ingen måde at modellere, at de sidste pakkeforsendelser skal ske parallelt, altså at D må sende et svar tilbage til C, når den modtager pakken fra C, uanset om B har modtaget pakken fra A eller ej (uden at man får det modsatte problem). Dette undgås ved at benytte forskellige kommandolister for hver knude:

```
1`("A", [...,
         SendPacket{toNode="B",id=1,data=...},
         ...]) ++
```

```
1`("B", [...,
         WaitForPacket{id=1},
         SendPacket{toNode="A",id=3,data=...},
         ...]) ++
1`("C", [...,
         SendPacket{toNode="D",id=2,data=...},
         ...]) ++
1`("D", [...,
         WaitForPacket{id=2},
         SendPacket{toNode="C",id=4,data=...},
         ...])
```

En anden årsag til, at knudeopdelte kommandolister er valgt, er, at i et af analyse-kapitlerne benyttes kommandolisterne til at lave simuleringer med (kapitel 5), og heri kan kommandolisterne blive meget lange. Hvis mange pakker skal modelleres sendt i modellen, kan de komme op på at bestå af adskillige tusinde kommandoer. Som beskrevet i afsnit 3.2.2 har CPN Tools en svaghed m.h.t. dette, og forsøg med samlede kommandolister viste, at en eksekvering af en kommandoliste med f.eks. 2500 kommandoer ikke kunne gennemføres inden for et halvt døgn. Hastigheden blev mærkbart bedre, efter listerne blev splittet ud på knudeniveau, og det blev p.g.a. dette muligt at gennemføre simulationerne i analyse-kapitlet.

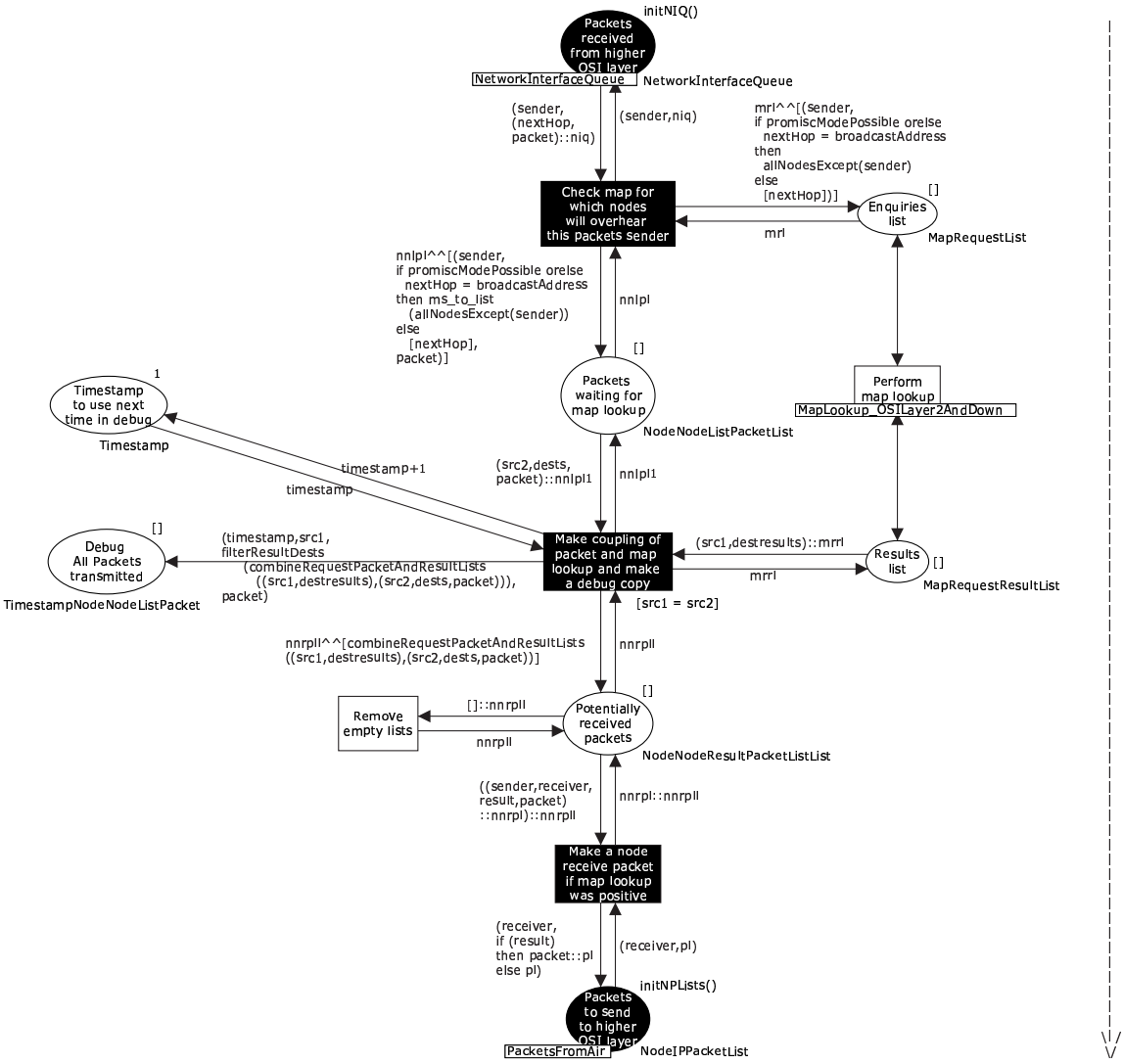
Da kommandolister stadig kan blive meget lange, kunne dette antyde, at det ville give endnu en hastighedsforbedring, hvis man fjernede listerne helt og brugte enkelt-brikker for hvert enkelt element i listerne ud fra strategien i afsnit 3.2.2.

En test, hvor jeg forsøgte dette, har dog afsløret et andet problem i CPN Tools: I afsnit 5.2.1 beskrives det, hvordan jeg v.h.a. et script automatisk genererer tilfældige kommandolister ud fra en række parametre. Scriptet gemmer kommandolisterne i en fil, som så hentes ind i modellen v.h.a. en simpel `use"kommandoliste.sml"`-kommando. Dette er den eneste måde, man i praksis kan benytte disse lange kommandolister på.

Hvis man lader CPN Tools hente en samlet kommandoliste på 500 elementer delt mellem 7 brikker (d.v.s. man vil have modellen til at simulere 7 knuders virkemåde), tager det maksimalt et par sekunder på en 3 GHz maskine med 1 GB RAM. Hvis man derimod laver kommandolisterne om til 500 enkelt-brikker, tager det CPN Tools cirka 20 minutter at hente de samme data. Dette afslører en svaghed i indlæsningen af brikker i forhold til lister i CPN Tools, og jeg har derfor valgt at bibeholde "mellemløsningen", hvor den samlede liste kun deles op i én liste pr. modelleret knude i CPN-modellen.

På "PredeterminedEvents_OSILayer3AndUp" (figur 3.40 på side 89) kan det også ses, at når en simulering af DSR-modellen starter, får hver modelleret knude en tilfældig startposition i netværket. For at undgå, at dette skal indvirke på en eksekvering af en kommandoliste, vil modellen altid sørge for at udføre den første kommando i hver knudes kommandoliste, før det er op til CPN-værktøjet tilfældigt at bestemme fra hvilken knudes kommandoliste, der skal udføres en kommando. Denne første kommando i hver kommandoliste skal være en `MoveNode`. Dette vil gøre det muligt for kommandolisten at "sætte scenen", før den egentlige pakkebrikgenerering og flytning af modellerede knuder går i gang.

En mulig udvidelse til kommandosproget kunne være en `WaitForPacketOrTimeout`-kommando, der ville få en knude til at vente med at udføre flere kommandoer, indtil en angivet pakke er modtaget – medmindre et vist timeout nås først. På denne måde kan problemet med `WaitForPacket` undgås, hvis det er usikkert, om en pakke vil nå frem til en modtager eller ej. Sproget, der beskrives i dette afsnit, skal imidlertid hovedsageligt bruges i et af analyse-kapitlerne (kapitel 5). Her er der behov for, at timingen mellem de pakkeforsendelser og de flytninger, der sker i netværket, er så ens som muligt hver gang, for at sikre så ens omstændigheder som muligt til at undersøge effekterne af forskellige optimeringer under. Derfor er denne kommando ikke implementeret, ligesom `WaitForPacket`-kommandoen heller ikke benyttes i analyserne i kapitlet.



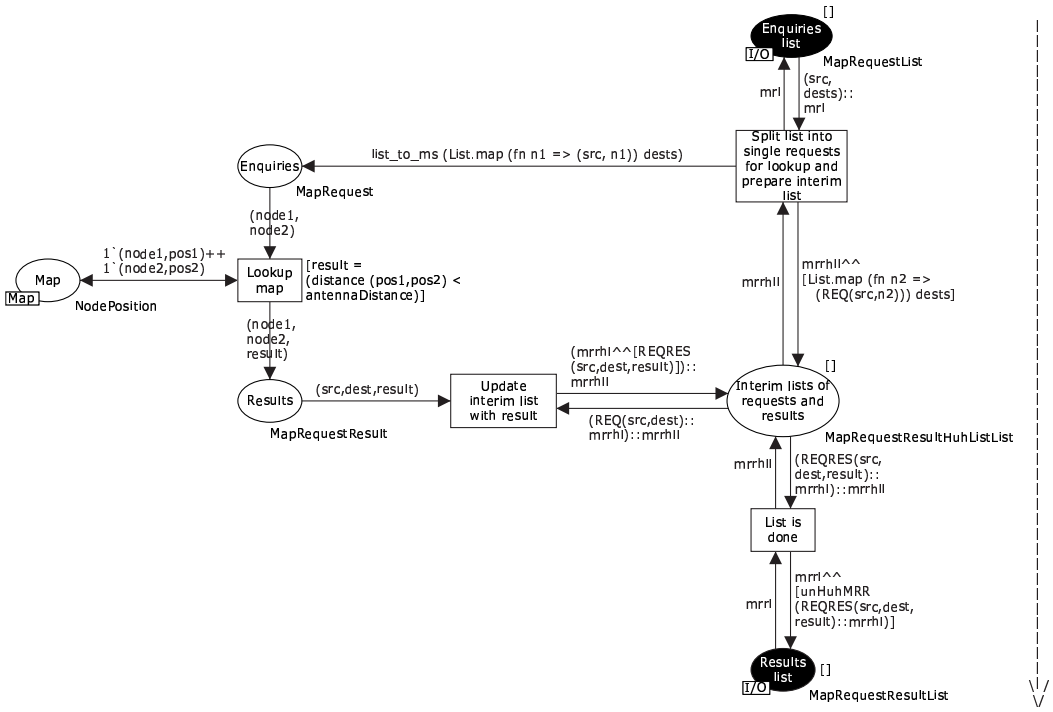
Figur 3.41: DSR-modellens CPN-side “OSILayer2AndDown”

3.6.4 OSI-lag 2 og ned

Den overordnede side, der styrer simuleringen af effekten af et trådløst netværk, kan ses i figur 3.41. Dens sammenhæng med resten af modellen kan ses i figur 3.4 på side 44. Siden sørger overordnet for at finde ud af hvilke knuder, der skal modtage en pakkebrik, hvis en knude simuleret transmitteret sender en sådan.

Siden tager imod pakkebrikker på pladsen “Packets received from higher OSI layer”, og slår for hver enkelt pakkebrik op hvilke andre modellerede knuder i netværket, der på forsendelsestidspunktet vil være i stand til at “overhøre” pakkebrikken, hvis knuden sender den ud, d.v.s. der i modellen er tilknyttet et koordinatsæt, der er tæt nok på koordinatsættet for den afsendende knude i forhold til modellens konfigurerede antennerækkevidde. En kopi af pakkebrikken føjes herefter til listerne, der er knyttet til disse modellerede, modtagende knuder på pladsen “Packets to send to higher OSI layer”.

Når siden skal finde ud af, præcist hvilke modellerede knuder, der simuleret vil overhøre det, når en knude udsender en pakke, gøres det ved at lave en liste af “forespørgsler” for



Figur 3.42: DSR-modellens CPN-side "MapLookup_OSILayer2AndDown"

hver enkelt potentiel modtagerknude i DSR-modellen. En forespørgsel består blot af par af knuder `(afsenderknude,modtagerknude)`. En underside, "MapLookup_OSILayer2-AndDown" (figur 3.42) sørger for at splitte disse lister af forespørgsler op i enkeltforespørgsler og undersøge, hvor langt hvert par af knuder er fra hinanden i forhold til knudernes modellerede antennerækkevidder (som blot er en afstandsvariabel, der kan konfigureres i DSR-modellen). Undersiden genererer herefter svar på formen `(afsenderknude,modtagerknude,boolean)`, som igen samles til lister. Supersiden "OSILayer2AndDown" kobler herefter disse listesvar sammen med en liste med kopier af pakkebrikken, der forårsagede forespørgslerne, hvilket tilsammen vil give en resultatpakkebrik til de modtagerknuder, der simuleret var "inden for antennerækkevidde" af afsenderknuden.

Pakketab er ikke modelleret i CPN-modellen, da jeg vurderede, at det ikke gav noget ekstra til forståelsen af den samlede DSR-del af modellen – det vil højst kunne teste modelleringen af gentransmissioner i RM. Modellen vil dog være nem senere at udvide med dette, hvis det skulle være ønskeligt, netop p.g.a. ovenstående opbygning.

Når pakker bliver transmitteret på siden, foretages en dataopsamling, der f.eks. benyttes i kapitel 5 til at opfange antal pakketransmissioner under forskellige konfigureringer af DSR-modellen. I et multiset gemmes brikker med følgende data:

```
(tæller, afsenderknude, [destinationsknude], pakke)
```

Det gemmes altså (for hver pakketransmission) hvem, der afsendte pakken, hvilke knuder, der modtog pakken, og præcist hvilken pakke, der blev afsendt. Den første indgang, der gemmes, er en tæller, der sikrer, at man bagefter kender rækkefølgen af pakketransmissionerne.

Hver pakkebrik er tilknyttet et debug-id (se afsnit 3.6.3), der beskriver hvilken startpakkebrik, den enkelte pakkebrik er udsprunget af – en "Route Request"-pakkebrik vil altså have et debug-id, der er magen til debug-id'et på den pakkebrik, der forårsagede udsendelsen af "Route Request"-pakkebrikken. Da hele pakkebrikken gemmes i det ovenstående multiset, gemmes

altså også dette debug-id, og dermed er det også muligt at udtrække informationer om f.eks. antal pakketransmissioner, der bliver affødt af den enkelte pakkeforsendelse.

Oprindeligt blev denne plads modelleret som en liste for implicit at bibeholde rækkefølgen af de foretagne pakketransmissioner i simuleringer, men som beskrevet i afsnit 3.2.2 kan brugen af en liste gøre simuleringen af CPN Tools meget langsom at arbejde med, og specielt kan der på denne plads forekomme store mængder data i forbindelse med store simuleringer. Derfor har jeg valgt at benytte metoden med transmissionsnumre (som også blev introduceret i afsnit 3.2.2), og dette gjorde, at det blev praktisk muligt at simulere større kommandolister.

3.7 Begrænsninger i modellen

CPN-modellen af DSR-protokollen beskrevet i de foregående afsnit afspejler en fungerende basis-protokol med de to udvidelser CRR og SO. Der er dog som allerede nævnt visse begrænsninger i hvad, der er medtaget i modelleringen af protokollen.

3.7.1 Netværkstype

DSR-protokollen er designet til at fungere under tre forskellige netværkssetup: "Frequently-unidir", "Mostly-bidir" og "Bidir-only" (se afsnit 2.2.5). Som beskrevet i kapitel 2 er der i protokollen dele, hvis kompleksitet afhænger af hvilket netværkssetup, protokollen opererer under.

I "Frequently-unidir" og "Mostly-bidir" benyttes der f.eks. passive/eksplicitte bekræftelser (som protokollen tager sig af at styre), mens man ofte kan nøjes med at benytte bekræftelser på MAC-niveau (som et underliggende OSI-lag tager sig af) i "Bidir-only" (se afsnit 2.4.1).

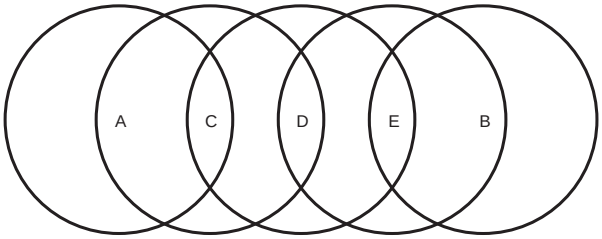
I den modellerede version af DSR er kun "Frequently-unidir" og "Mostly-bidir" fuldt modelleret, da disse setups repræsenterer den største kompleksitet af de to. Visse ting i "Bidir-only" er dog også modelleret, f.eks. at rutelister skal reverseres ved besvarelser af "Route Request"-pakker, og at der skal benyttes en "Blacklist" – men bekræftelser på MAC-niveau er f.eks. ikke modelleret.

Områder af protokollen, der lægger sig tæt op af det faktum, at protokollen beskæftiger sig med trådløse netværk, er heller ikke implementeret. Det drejer sig hovedsageligt om brugen af variablen "BroadcastJitter", som de enkelte knuder kan benytte som grundlag for en tilfældig forsinkelse, som de derefter kan vente, før de videresender "Route Request"-pakker, for at undgå pakkekollissioner i luften (se afsnit 1.6.2). Det vil kræve et ekstra kompleksitetslag i CPN-modellen at se på dette problem (hovedsageligt m.h.t. en global tid i modellen, se afsnit 3.3.2), og jeg har derfor valgt at prioritere modelleringen af andre områder af protokollen højere end dette.

3.7.2 Antennerækkevidder

I modellen er antennerækkevidden fast for alle modellerede knuder. Dette er en uprecis måde at simulere et trådløst netværk på, for selvom alle trådløse netkort i et netværk kan være ens, kan batteriforhold, områdets kupering etc. betyde, at antennerne ikke fungerer lige langt i forhold til hinanden (se afsnit 1.2). Når alle antennerækkevidder er lige lange i modellen, vil man i praksis være i netværkstypen "Mostly-bidir" eller "Bidir-only", da de fleste (eller rettere alle) links vil virke i begge retninger, hvis de blot virker i den ene. Da den sidste netværkstype ("Bidir-only") ikke er fuldt modelleret (som ovenfor beskrevet), kan simuleringen af modellen af DSR-protokollen altså hovedsageligt benyttes til at teste netværkstypen "Mostly-bidir".

Specielt betyder dette, at modellerede knuder i langt de fleste tilfælde kun får brug for eksplicitte bekræftelser ved videresendelser via det sidste link i rutelister i netværkspakker. I de øvrige vil der ske det, at når en modelleret knude A videresender til en anden modelleret



Figur 3.43: Setup af netværket til demonstration af "Route Discovery". Ringene angiver knudernes antennerækkevidder.

knude B, og B herefter videresender til en tredie modelleret knude C, vil A *altid* modtage en passiv bekræftelse fra B, idet de altid er inden for gensidig antennerækkevidde af hinanden (medmindre de i samme øjeblik flytter sig uden for denne).

En mere komplet modellering af DSR-protokollen vil inkludere en variabel antennerækkevidde, så det bliver muligt at teste det ovenstående samt give en mere realistisk model af en "Frequently-unidir"-netværkstype, men jeg har valgt at prioritere modelleringen af diverse optimeringer af DSR-protokollen (CRR og SO) højere end dette.

3.8 Demonstrationer af modellen

I dette afsnit præsenteres et antal tests, der dels viser mere praktisk, hvordan protokollen Dynamic Source Routing fungerer, dels demonstrerer, at modellen i disse tests opfører sig korrekt i forhold til specifikationen (under forbehold for de ændringer, der blev beskrevet i de foregående afsnit).

I disse demonstrationer benyttes et kort, der er $2000 * 2000$ enheder stort med en antennerækkevidde på 300 enheder i et "Mostly-bidir"-netværk (jvf. afsnit 3.7.1). De forskellige variable timeouts etc. i DSR er sat til standardværdierne givet i de foregående afsnit og i [JMH05], ganget med en faktor på $\frac{1}{2}$ som beskrevet i afsnit 3.3.2.

3.8.1 "DSR's hovedfunktionalitet" og "Route Discovery"

Det første eksempel demonstrerer "DSR's hovedfunktionalitet" og "Route Discovery" under et: Et netværkssetup som i figur 3.43 benyttes, og v.h.a. dette sendes en pakke fra knuden A til knuden B. Hvis modellen opfører sig korrekt, vil A først igangsætte en RD, finde ruten A→C→D→E→B, og derefter sende pakken via denne rute som en del af "DSR's hovedfunktionalitet".

Kommandolisten er som følger:

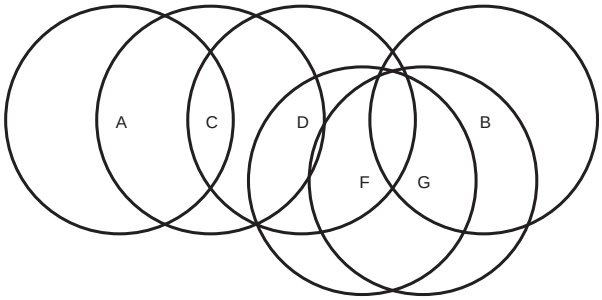
```
1`("A",[MoveNode{toPos=(200,200)},
        SendPacket{toNode="B",id=1,data=SOMEDATA
                                        ("pakke 1 fra A til B")}]) ++

1`("C",[MoveNode{toPos=(200,400)}]) ++

1`("D",[MoveNode{toPos=(200,600)}]) ++

1`("E",[MoveNode{toPos=(200,800)}]) ++

1`("B",[MoveNode{toPos=(200,1000)},
        WaitForPacket{id=1}])
```



Figur 3.44: Setup af netværket til demonstration af "Route Maintenance" med knude E flyttet ud og knuderne F og G flyttet ind i stedet for. Ringene angiver knudernes antennerækkevidder.

Husk, at den første kommando for hver knude angiver det initielle setup af knudernes placering i simulationen. Outputtet på dataopsamlingspladsen på "OSILayer2AndDown"-siden (se afsnit 3.6.4) kan ses i appendix D.1.

Demonstrationen tager 2.000-3.000 steps i CPN Tools-simulatoren at fuldføre, hvilket tager ca. 5 sekunder at simulere på en 3 GHz maskine med 1 GB RAM, og resulterer i alt i 14 pakketransmissioner. Først broadcastes en "Route Request"-pakke ud fra A gennem C, D og E, og B modtager den og sender en "Route Reply"-pakke tilbage til A gennem den reverserede ruteliste, som den har uddraget af den opsamlede ruteliste i "Route Request"-pakken. Dette skal den gøre, idet den er i et "Mostly-bidir"-netværk. Dette er altså RD-delen af demonstrationen og står for i alt 9 af pakketransmissionerne.

A modtager rutelisten og benytter den til at sende pakken til B, som modtager den efter i alt 13 pakketransmissioner (den 14. transmission er den bekræftelse, A sender tilbage til C på, at den har modtaget pakken jvf. afsnit 2.4.1). Selve forsendelsen af pakken, efter A har modtaget en rute til B (d.v.s. demonstrationen af "DSR's hovedfunktionalitet"), benytter altså blot 5 pakketransmissioner – 4 pakkeforsendelser og en enkelt bekræftelse.

Ifølge demonstrationen opfører modellen sig, som den skal m.h.t. dette eksempel. Den ønskede pakke når frem til den endelige destinationsknude, og der bliver ikke udsendt for mange pakker i netværket – hver knude videresender f.eks. kun "Route Request"-pakken én enkelt gang, selvom de modtager den adskillige gange.

3.8.2 "Route Maintenance"

Det følgende eksempel demonstrerer "Route Maintenance": Igen benyttes et netværkssetup som vist i figur 3.43 på forrige side, og v.h.a. dette sendes en pakke fra knuden A til knuden B gennem C, D og E som i det foregående eksempel. Dette sikrer, at A har en rute til B i sin "Route Cache".

Herefter flyttes knuden E væk, og knuderne F og G flyttes ind i stedet for som vist i figur 3.44, hvilket gør A's rute til B ugyldig. En ny pakke sendes fra A til B. Denne pakke bør ikke nå frem, da knuden D's RM vil fejle i forsøget på at få linket D→E til at virke og vil sende en "Route Error" tilbage. Det er herefter op til et højereliggende OSI-lag i A at gensende pakken til B jvf. afsnit 2.4.2. Endnu en pakke sendes efter et timeout, hvilket vil resultere i en ny RD og endelig fremkomsten af pakken hos B.

En kommandoliste, der kan teste dette, er vist herunder. For at sikre, at knuderne flyttes imellem de to første pakkeudsendelser, flyttes E, F og G efter en timeout på 2000, og A udsender pakke nr. 2 efter en timeout på 4000. Pakke nr. 3 udsendes først efter en timeout på 50000 efter pakke nr. 2, da knuden D skal timeout'e adskillige gange på brugen af linket til knude E i rutelisten i pakke nr. 2 – først en gang, mens den forsøger at få en passiv bekræftelse, derefter to gange, hvor den forsøger at få en eksplicit bekræftelse.

```
1`("A",[MoveNode{toPos=(200,200)},
        SendPacket{toNode="B",id=1,data=SOMEDATA("pakke 1")},
        WaitForTimeout{timeout=4000},
        SendPacket{toNode="B",id=2,data=SOMEDATA("pakke 2")},
        WaitForTimeout{timeout=50000},
        SendPacket{toNode="B",id=3,data=SOMEDATA
                   ("pakke 3 (genudsendelse af pakke 2)")}]) ++

1`("C",[MoveNode{toPos=(200,400)}]) ++

1`("D",[MoveNode{toPos=(200,600)}]) ++

1`("E",[MoveNode{toPos=(200,800)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(2000,2000)}]) ++

1`("B",[MoveNode{toPos=(200,1000)},
        WaitForPacket{id=1},
        (* Pakke 2 vil ikke nå frem *)
        WaitForPacket{id=3}]) ++

1`("F",[MoveNode{toPos=(2000,2000)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(350,700)}]) ++

1`("G",[MoveNode{toPos=(2000,2000)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(350,900)}])
```

Outputtet på dataopsamlingspladsen på "OSILayer2AndDown"-siden (se afsnit 3.6.4) kan ses i appendix D.2. Demonstrationen tager 300.000-400.000 steps i CPN Tools-simulatoren at fuldføre (p.g.a. de mange timeouts både i kommandolisten, og når knuden D forsøger at benytte linket til E). Dette tager cirka 16-20 minutter at simulere på en 3 GHz maskine med 1 GB RAM.

I outputtet kan det ses, at kommandolisten i alt afføder 39 pakketransmissioner. Modellen foretager sig følgende under eksekveringen af kommandolisten: 14 af pakketransmissionerne går som i demonstrationen i afsnit 3.8.1 med, at A først sender en "Route Request" ud, B modtager den (efter den har været videresendt gennem C, D og E), B sender en "Route Reply" tilbage til A, og A til sidst sender selve pakken til B.

Allerede her demonstreres RM, idet ingen pakker bliver gensendt – knuder, der sender eller videresender pakker, får en passiv bekræftelse tilbage, når de hører, at knuden, de sender pakken til, sender pakken videre. Dette gælder for alle videresendelser, pånær når "Route Reply'en" modtages af A (via linket fra C), og når den endelige pakke modtages af B (via linket fra E). I begge disse tilfælde opdager C hhv. E, at de er i gang med at sende en pakke over det sidste link i rutelisten, og beder derfor om en eksplicit bekræftelse med det samme – som A hhv. B sørger for at sende til dem.

Når pakke 2 sendes (via den ved udsendelsen af pakke 1 fundne rute), forsøger knuden D at sende pakken videre til E, men da den ikke får en passiv bekræftelse tilbage, forsøger den at sende pakken igen to gange med en eksplicit bekræftelsesanmodning i. Heller ikke disse bliver besvaret, så D sender herefter en "Route Error"-pakke gående på linket mellem D og E tilbage til A. I alt bruges der 8 pakketransmissioner på dette.

"Route Error"-pakken bliver korrekt behandlet af A, idet den ugyldige rute ikke benyttes næste gang, A vil sende en pakke til B, men en ny RD i stedet igangsættes, og denne returnerer den nye rute A→C→D→F→G→B, som A herefter benytter til at sende pakke 3 (en genudsendelse af pakke 2) til B.

Bemærk, at i eksemplet valgte jeg at lade knuden E forsvinde – hvis jeg havde ladet knuden C forsvinde i stedet, ville det være A, der opdagede at et link ikke længere eksisterede – og da A er den oprindelige afsenderknude for pakken, kommer simuleringen hermed i Mini-SO-situationen beskrevet i afsnit 3.5.5, og den normale funktion af RM ville ikke blive demonstreret med eksemplet.

I alt afføder den brudte rute, at der foretages 25 pakketransmissioner i netværket for at få pakke 2 frem fra knude A til knude B (inkl. genudsendelsen som pakke 3) – og desuden skal et højereliggende OSI-lag i A opdage, at første gang, den forsøgte at sende pakken, nåede den ikke frem, så dette højereliggende lag skal gensende pakken.

3.8.3 "Cached Route Reply"

Den første optimering, der er modelleret med i CPN-modellen af DSR, og kan slås til og fra, er "Cached Route Reply" (CRR). Som beskrevet i afsnit 2.3.6 går en CRR ud på, at en knude i stedet for at sende en "Route Request"-pakke videre kan vælge at sende en "Route Reply"-pakke tilbage baseret på information i dens egen "Route Cache".

For at demonstrere effekten af optimeringen, laves der igen et netværkssetup som vist i figur 3.43 på side 95. Knuden C skal derefter sende en pakke til knuden E, hvilket vil få knuden til at igangsætte en RD gående på E. C vil herefter få ruten C→D→E i sin "Route Cache".

A skal herefter sende en pakke til E. A vil også igangsætte en RD gående på E, men C vil svare tilbage til A med sin cachede rute til E i stedet for at sende forespørgslen videre.

Den benyttede kommandoliste kan ses i appendix D.3.1. Outputtet på dataopsamlingspladsen på "OSILayer2AndDown"-siden (se afsnit 3.6.4) kan ses i appendix D.3.2 for tilfældet, hvor optimeringen er slået fra, og i appendix D.3.3 for tilfældet, hvor optimeringen er slået til.

Som det kan ses, resulterer dette i alt i 24 pakketransmissioner i den uoptimerede model, 10 for RD og forsendelsen af pakke 1 og 14 for RD og forsendelsen af pakke 2.

Når optimeringen slås til, foregår der i alt 20 pakketransmissioner, som før 10 for RD og forsendelsen pakke 1, men nu kun 10 for RD og forsendelsen af pakke 2.

At antal pakketransmissioner i forbindelse med pakke 2 er faldet med 4, svarer til, at "Route Request"-pakken ikke længere broadcastes til knuderne E og B, og at en "Route Reply"-pakke ikke længere skal sendes tilbage gennem disse to knuder.

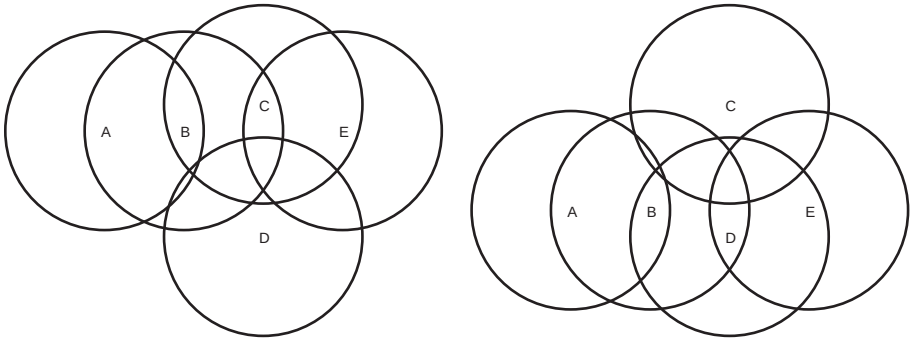
Det burde være klart, at det også er muligt at lave en demonstration af CRR, hvor det samlede antal pakketransmissioner stiger, hvis flere knuder svarer samtidigt med en CRR-pakke. Det er derfor interessant at lave en mere generel analyse af effekten af CRR – en sådan kan ses i afsnit 5.3.3.

3.8.4 "Salvage Operations"

Den anden optimering, der er modelleret med i CPN-modellen af DSR, og kan slås til og fra, er "Salvage Operations" (SO). Som beskrevet i afsnit 2.4.6 går SO ud på, at en knude i forbindelse med en videresendelse af en pakke opdager, at linket, pakken skal videresendes via, ikke længere fungerer. Knuden har dog en anden rute til destinationsknuden, og kan nu vælge at sende pakken videre via denne nye rute. Når knuden gør dette, kaldes det en "Salvage Operation" (SO).

For at demonstrere effekten af optimeringen, laves der et netværkssetup som vist i figur 3.45(a) på næste side. Knuden B skal sende en pakke til knuden E, hvilket vil få knuden til at igangsætte en RD gående på E. B vil derved få ruten B→C→E i sin "Route Cache".

En ny knude D sættes nu ind mellem B og E. A skal nu have ruten A→B→D→E (og kun den ene rute til E) i sin "Route Cache". Dette gøres ved at lade A sende en pakke til D, hvilket vil få A til at igangsætte en RD. For at sikre, at A ikke samtidigt får ruten A→B→C→E, flyttes knude C samtidigt væk fra de øvrige knuder. Dette kan ses i figur 3.45(b) på næste side.



(a) Initielt setup

(b) Knuden C er flyttet ud, og knuden D er flyttet ind mellem B og E

Figur 3.45: Setup af netværket til demonstration af SO. Ringene angiver knudernes antennerækkevidder.

Efter pakken er sendt, fjernes D igen (så A's rute til E bliver ugyldig), og C genindsættes på sin gamle plads – netværkssetuppet fra figur 3.45(a) er altså i brug igen. A sættes igen til at sende en pakke til E. B vil nu opdage, at linket til D ikke længere fungerer, og vil derfor returnere en "Route Error"-pakke til A – men den opdager samtidigt, at den kan igangsætte en SO og sende pakken videre langs den anden rute, den kender til E (B→C→E). Pakken bør derfor nå frem alligevel, hvor den ellers ville skulle genudsendes af et højereliggende OSI-lag i knuden A, hvis udvidelsen SO ikke var slået til.

Den benyttede kommandoliste kan ses i appendix D.4.1. Outputtet på dataopsamlingspladsen på "OSILayer2AndDown"-siden (se afsnit 3.6.4) kan ses i appendix D.4.2 for kommandolisten kørt med SO slået fra og i appendix D.4.3 med SO slået til.

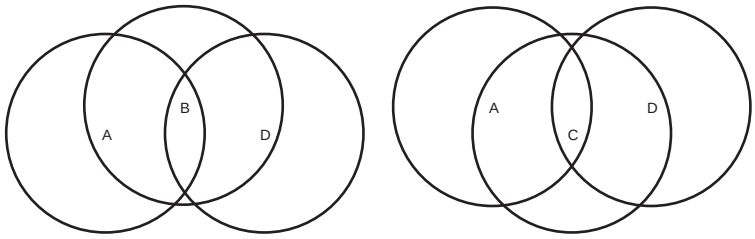
Når optimeringen er slået fra, forekommer der 26 pakketransmissioner i netværket, og den sidste pakke fra A til E vil ikke nå frem. Når optimeringen er slået til, forekommer der 29 pakketransmissioner i netværket, og pakken når frem, uden at et højereliggende OSI-lag i knuden A skal foretage sig mere. De 3 ekstra pakketransmissioner er de 2 "nye" videresendelser af pakken gennem ruten B→C→E og den obligatoriske eksplicitte bekræftelse af det sidste link i denne rute.

For at den sidste pakke rent faktisk skal nå frem fra knude A til knude E i den uoptimerede model, skal et højereliggende lag sende pakken igen. Da A ikke har andre ruter til E, inkluderer dette, at A skal igangsætte en RD, modtage et svar, og først derefter sende pakken. Fra demonstrationen i afsnit 3.8.2 vides, at dette vil tage 11 ekstra pakketransmissioner (3 transmissioner af en "Route Request"-pakke fra A til E, 3 transmissioner af en "Route Reply"-pakke fra E til A, 3 transmissioner af pakken selv samt 2 "Acknowledgement"-pakker (for de sidste to pakker)).

Man kan derfor udlede, at der, for at pakke 3 også skal nå frem, i den uoptimerede version vil være 37 pakketransmissioner, mens der kun vil være 29 pakketransmissioner i den optimerede version. I den uoptimerede version vil knuden A dog kende en rute til knuden E efter de 37 transmissioner, det vil den ikke i den optimerede version efter de 29 transmissioner. Præcist hvilken effekt, optimeringen derfor har, kræver en mere grundig analyse, end dette konstruerede eksempel kan give. SO undersøges nærmere i afsnit 5.3.4.

3.8.5 "Mini-Salvage Operations"

Som beskrevet i afsnit 3.5.5 er der i DSR-modellen lavet et ekstra forslag til en DSR-optimering: En "Mini-Salvage Operation" (Mini-SO). I modellen kan denne optimering p.t. ikke slås til og fra, da analyser af effektiviteten af denne er fravalgt i kapitel 5 til fordel for analyser af



(a) Initielt setup

(b) Knuden B er flyttet ud, og knuden C er flyttet ind mellem A og D

Figur 3.46: Setup af netværket til demonstration af Mini-SO. Ringene angiver knudernes antennerækkevidder.

CRR og SO. Som jeg har beskrevet i afsnit 3.5.5, og som jeg vender tilbage til i afsnit 5.4, var dette i retrospekt nok en fejl. Det er dog stadig muligt at demonstrere effekten af Mini-SO. Det vil jeg gøre i dette afsnit.

I et netværkssetup som det i figur 3.46(a) vil A sende en pakke til D. Dette vil resultere i, at A igangsætter en RD og får ruten A→B→D tilbage, som herefter bruges til at sende pakken.

B flyttes nu væk fra A og D, og C flyttes ind i mellem de to knuder i stedet for, som det kan ses i figur 3.46(b). A forsøger igen at sende en pakke til D, men finder ud af, at linket A→B ikke længere fungerer. A vil nu altid fjerne linket fra sin "Route Cache", men hvad, der sker herefter, afhænger af de brugte udvidelser:

Ingen: I en model, hvor hverken Mini-SO eller SO er modelleret med (eller hvor de begge kan slås fra og er blevet det), vil A blot give op og smide pakken til D ud, uanset om A måtte have andre ruter til D eller ej.

Kun SO: Hvis SO (som den er defineret i specifikationen [JMH05]) er slået til (igen i en model, hvor Mini-SO enten ikke er modelleret med eller er slået fra), vil A forsøge at sende pakken via en alternativ rute, hvis en sådan haves i knudens "Route Cache" – men hvis en sådan ikke haves, giver A også her op.

Mini-SO og evt. SO: I DSR-modellen, som den er beskrevet i dette kapitel, vil Mini-SO træde i kraft i stedet for SO (selvom SO måtte være slået til), og vil (ligesom SO) forsøge at sende pakken via en alternativ rute, hvis en sådan haves i knudens "Route Cache". Hvis en sådan ikke haves, igangsætter A i stedet en RD for at finde en rute til D. Set fra de andre knuder i netværkets synspunkt foregår der ikke nogen form for SO i forbindelse med dette, for det er den oprindelige afsenderknude, der forsøger at sende pakken igen via en ny rute – præcist den samme effekt ville være forekommet, hvis et ovenliggende OSI-lag i A havde opdaget, at pakken ikke var nået frem, og havde forsøgt at gensende den – det vil bare ske hurtigere med Mini-SO (fordi en timeout i det ovenliggende OSI-lag ikke også skal nås).

En kommandoliste, der demonstrerer dette, kan ses i appendix D.5.1. Resultatet af en simulation af denne kommandoliste, d.v.s. outputtet på dataopsamlingspladsen på "OSILayer2And-Down"-siden (se afsnit 3.6.4), kan ses i appendix D.5.2.

Som det kan ses, giver simulationen præcist det forventede resultat: Ved forsendelsen af pakke nr. 2 forsøger A at sende en pakke via linket A→B tre gange (en, hvor den forsøger med passiv bekræftelse, og to, hvor den beder om en eksplicit bekræftelse). Herefter igangsætter den en ny RD, finder ruten A→C→D, og sender pakken via denne. Pakken går altså ikke tabt, selvom A i denne demonstration ikke på tidspunktet, hvor pakke nr. 2 blev afsendt, havde en alternativ rute til D. Det ville den have gjort, hvis "kun" den almindelige SO var blevet modelleret med i DSR-modellen.

3.9 Brugen af en model frem for en implementation, samt relateret arbejde

I dette kapitel har jeg præsenteret en model af Dynamic Source Routing. Et relevant spørgsmål at stille sig selv i denne forbindelse er: Hvorfor overhovedet bygge en sådan model? Ville det ikke have været mere relevant at gå direkte til en implementation af modellen?

[Sel03] argumenterer for, at det er nemmere at specificere, forstå og vedligeholde modeller end implementationer, idet disse kan benytte sig af modelleringskoncepter, der er tættere på problemdomænet og længere fra implementationsdetaljer. Koncepterne giver et abstraktionsniveau, der kan hjælpe én til at forstå problemstillinger bedre og finde potentielle løsninger derpå. Desuden kan modelleringsværktøjer, der er i stand til at køre en simulering af modellerede systemer (f.eks. en protokol), give tidlige erfaringer med det valgte design. Fra [Sel03]:

> "One of the fundamental ways that we learn is through experimentation – that is, through model execution."

Der er dog også argumenter imod modellering af systemer. Man kan anse modelleringsprocessen som et uønsket dobbelt-arbejde af den grund, at når man har lavet en model af et system, skal man bagefter begynde forfra med at lave en "rigtig" implementation. Desuden kan man under implementationen finde detaljer, der gør, at man er nødt til at lave om i designet af systemet i forhold til den tidligere model. Dette gør, at modellen og implementationen ikke nødvendigvis længere passer sammen, og værdien af modellen som illustration af systemets virkemåde bliver væsentligt forringet, hvis den ikke hver gang bliver tilpasset.

Om man skal bygge en model eller gå direkte til en "rigtig" implementation kan derfor afhænge af, hvordan man vægter disse ting i forhold til hinanden. [Sel03] opstiller en række faktorer, der vil påvirke det resultat, man vil kunne få ud af at bygge en model i stedet for at gå direkte til en implementation. De tre første faktorer beskæftiger sig med modellen selv:

Abstraktion: For det første skal man i modelleringsprocessen holde sig for øje, at irrelevante detaljer skal skjules eller fjernes fra modellen. Hermed forstås essensen af det modellerede system bedre i modellen.

Forståelig: Det, der ikke abstraheres væk, skal præsenteres i en form, som man intuitivt forstår. Dette er en models force frem for en implementation – selv skrevet i høj-niveau sprog kræver en implementation parsning af tekst for at forstå den.

Nøjagtig: Modellen skal være præcis inden for det interessante område for at sikre et korrekt resultat af f.eks. simuleringer.

Jeg har lavet en model, hvor det intuitive flow (f.eks. hvordan en knude skal behandle den enkelte netværkspakke) forsøges fremhævet. Dette er direkte gjort for at fremme det forståelsesmæssige aspekt af modellen. Men spørgsmålet er, om siderne altid er overskuelige nok til, at dette lykkes. Da det er prioriteret, at modellen netop også skal være så nøjagtig som mulig (i DSR-laget) a.h.t. den fremtidige simulering, har dette arbejde hovedsageligt inkluderet en afvejning af hvad, der skulle modelleres grafisk (og dermed vises intuitivt), og hvad, der skulle gemmes væk i SML-kode ved siden af modellen. Ikke så meget af DSR-funktionaliten kunne abstraheres helt væk fra modellen/SML-koden.

Derimod er de omkringliggende OSI-lag (afsnit 3.6) kraftigt barberet – man kunne f.eks. have valgt at inkludere en TCP-protokol, så det ville være muligt at se den samlede effekt af TCP/IP og DSR-protokollen.

Dette er i skarp kontrast til arbejdet i f.eks. [KJJ04]. I denne artikel præsenteres en række CPN-modeller, der sørger for at abstrahere alt andet end en enkelt problemstilling væk. Dette kan f.eks. være en låsemekanisme i et BeoLink-system eller integrationen af MANET'er med et "normalt", trådet netværk.

Det sidste af de to eksempler beskæftiger sig mere præcist med problemstillingen med routingen af IPv6-pakker mellem et "normalt", trådet netværk og fire MANET'er. I modellen kan mobile knuder frit bevæge sig fra et MANET til et andet, og kan i en periode i forbindelse med dette enten være medlem af begge MANET'er eller helt miste forbindelsen til dem begge. Pakker kan simuleres routet i netværksmodellen – men præcist hvilken MANET-protokol, der benyttes i de fire modellerede MANET'er, er abstraheret væk, da det ikke er vigtigt for problemstillingen. Dette kan senere udfyldes, hvis det skulle blive nødvendigt.

Også den fysiske placering af de enkelte knuder er abstraheret væk. I stedet for at have et "kort" (som i afsnit 3.6.1) har forfatterne valgt blot at modellere sammenhængen mellem knuder som et antal links – og knuders bevægelse modelleres, som at et nyt link spontant kan opstå mellem to knuder, eller et gammelt link kan blive fjernet. Dette kan i princippet give simuleringer, hvor man ud fra de links, der er mellem modellerede knuder, ikke kan placere knuderne fysisk i et kort (så alle krav om tilstedeværelse eller fravær af links bliver opfyldt), men dette er ikke vigtigt for problemstillingen.

I princippet kunne det samme have været gjort i CPN-modellen af DSR. Jeg valgte dog ikke at gøre det, idet simuleringer af MANET-protokoller netop kan ses som en simulering af, at en netværkspakke skal bevæge sig over en fysisk afstand via et antal knuder for at nå til sin destination, og disse knuder vil netop typisk bevæge sig i en "retning" under dette forløb (jvf. afsnit 3.6.1). I retrospekt er det dog ikke sikkert, at forskellen ville have været så stor, da DSR-protokollen ikke internt benytter sig af geografiske data (såsom f.eks. MANET-protokollen "Location-Aided Routing" [KV98] gør det), så det er muligt, de samme resultater i de næste afsnit ville have kunnet nås med en mere simpel kort-model. Dette ville f.eks. have gjort CPN-siderne i figur 3.40-3.42 på side 89–93 meget mere overskuelige.

Ud fra samme betragtning kan man argumentere for, at jeg ikke burde have modelleret både RD og RM – de to dele af DSR-protokollen repræsenterer to forskellige problemstillinger og burde i princippet blive behandlet hver for sig. Dette ville have givet en CPN-model, der ville være noget mere overskuelig, og som man stadig ville have kunnet benytte til at analysere på visse egenskaber for DSR-protokollen, som jeg gør det i kapitel 5. Størrelsen og den følgende uoverskuelighed udgør nok det største problem med den DSR-model, der er blevet præsenteret i dette kapitel.

Det er dog værd at bemærke, at idet både RD og RM er valgt, kan man netop se effekten af den *samlede* model. Hvis RM f.eks. modvirker en del af RD, vil dette formodentligt blive demonstreret i en fuld model. Jeg har dog ikke fundet en sådan negativ effekt mellem de to under modelleringen.

Man kunne også have skåret f.eks. modelleringen af "Blacklist" (der jvf. afsnit 3.4.3 alligevel ikke er modelleret færdig) væk, men på et tidspunkt var jeg nødt til at gå igang med at udføre simuleringerne, og derefter kunne jeg ikke tillade mig at lave strukturelle ændringer i DSR-modellen (dette vender jeg tilbage til i afsnit 5.2.2).

At der er foretaget så kraftige abstraktioner i CPN-modellerne i [KJJ04] som beskrevet i det ovenstående, har ud over overskueligheden en anden umiddelbar fordel: Man bliver i stand til at foretage mere formelle analyser på modellerne end dem, der foretages i resten af dette speciale. Mere præcist bringer det os til det næste krav fra [Sel03]:

Automatisk verifikation: En model skal kunne bruges til at påvise interessante (men ikke nødvendigvis åbenlyse) egenskaber, som en endelig implementation af det modellerede system vil have. Dette kan f.eks. ske gennem formel matematisk analyse eller gennem simuleringer (empirisk verifikation).

I kapitel 4 underbygger jeg, at det ikke kan lade sig gøre at lave en formel analyse af DSR-modellen v.h.a. CPN Tools, da der er alt for mange tilstande i modellen. I kapitel 5 benytter jeg til gengæld simuleringer til at undersøge forskellige egenskaber (f.eks. hvor mange pakker, der kan afleveres i et simuleret netværk, og antallet af pakketransmissioner i forbindelse hermed), og hvordan disse egenskaber påvirkes, når man slår valgfrie DSR-optimeringer til eller fra. Så selvom jeg ikke kan gennemføre en formel analyse, opfyldes kriteriet alligevel.

Billig: En model skal være billigere at lave og analysere end en rigtig implementation.

Opfyldelsen af dette kriterie afhænger af flere forskellige ting. Hvor lang tid vil det tage at lave en model? Hvor lang tid vil det tage at lave en "rigtig" implementation? Typisk er dette først noget, man kan estimere, når man har analyseret det system, der skal modelleres eller implementeres, til bunds – og det er netop det, en modellering kan hjælpe med.

I dette tilfælde vil jeg vurdere, at modelleringen tog cirka halvt så lang tid, som det ville have taget at lave end en "rigtig" implementation, men det er ikke noget, jeg kan verificere, da jeg netop valgte at bygge en model først.

Selve modelleringsværktøjet, man benytter (i dette tilfælde CPN Tools) betyder jvf. [Sel03] også meget for modelleringstiden (og dermed prisen for modellen). Både tiden til at lave en fuld og en inkrementel kompilering er vigtig. De inkrementelle kompileringer er vigtigst: Som oftest foretages små ændringer af systemet meget oftere end fulde genindlæsninger af modellen. Fra [Sel03]:

> "[. . .]if a small, localized change requires regenerating a disproportionately large part of the code, development can slow to an unacceptable pace."

Derfor kræves det, at et modelleringsværktøj er i stand til at analysere hvad, der skal genkompileres, når noget ændres i en model, og dette kan CPN Tools netop.

Når modellen hentes ind i CPN Tools på en 3 GHz maskine med 1 GB RAM, tager det ca. 15 minutter, før modellen er brugbar. Dette virker som lang tid, men indlæsningen skal typisk kun udføres, når CPN Tools skal startes op (efter en reboot eller et tool-crash). Når man har lavet en enkelt ændring i modellen, tager det fra 3 sekunder til 6 minutter, før man kan eksperimentere med ændringen. De fleste ændringer ligger dog tættest på de 3 sekunder, hvilket må siges at være rimeligt.

I alt brugte jeg ca. 2 mandemåneder på at designe og bygge modellen. Med til dette hører dog, at jeg i kapitel 5 viser, at jeg ikke var i stand til at udføre en stor mængde simulationer for at få et stort statistisk materiale, som det var ønskeligt. Dette skyldes, at det tager *noget* længere tid at gennemføre en modelsimulering i CPN Tools, end det ville have taget at eksekvere en implementation. Dette kan derfor til dels neutralisere den ovennævnte påstand om, at "prisen" (d.v.s. tiden) ved en modellering var mindre, end den ville have været ved en implementation.

Automatisk kodegenerering: For at få fuldt udbytte af modellen, skal man være i stand til gradvist at forfine den, indtil man kan trække den *komplette*, færdige implementation ud af den. Det er ikke nok, at man kan trække skeletter/fragmenter ud af modellen. V.h.a. et komplet udtræk undgås dobbeltarbejde i processen. Når dette punkt er opfyldt, garanteres nøjagtigheden af modellen også, fordi modellen til sidst vil blive til det system, den skal modellere.

Her taber CPN-modellen en meget stor del af sin værdi, eftersom ovenstående p.t. er ikke muligt med CPN Tools. Som nævnt ovenfor, betyder dette ud over dobbeltarbejdet også, at man under implementationen kan blive opmærksom på andre problemer, der medfører, at noget af designet i modellen skal laves om. Hermed kommer modellen og implementationen ikke længere til at passe sammen, og enten skal modellen så også laves om, eller man må acceptere, at modellen ikke længere vil have den fulde effekt af at være en intuitiv oversigt over systemet.

Det skal bemærkes, at det tidligere har været muligt at lave automatisk kodegenerering ud fra CPN-modeller ([Mor00]) som en udbygning på Design/CPN. Det er derfor muligt, at dette på et tidspunkt vil blive porteret til CPN Tools. Havde jeg valgt ns-2 [NS2] til at bygge DSR-protokollen i i stedet, ville den automatiske kodegenerering ikke have været et problem: I ns-2 skriver man "modellen" direkte i C++, og denne kan så benyttes i både ns-2 og f.eks. en Linux-implementation. Det er det, man har gjort i f.eks. DSR-UU [Nor05].

Til gengæld gør dette naturligvis, at selve modellen i ns-2 er tekstbaseret (i stedet for grafisk som i en CPN-model), og det kan derfor være sværere at give den intuitive forståelse af en protokol heri. Simuleringerne af en model kan dog stadig vises grafisk i ns-2.

4

Analyse af DSR-modellen v.h.a. state space-beregninger

I kapitel 3 præsenterede jeg en CPN-model af DSR-protokollen. I afsnit 3.9 omtalte jeg to metoder til analyse af modeller: Formel analyse og analyse v.h.a. simuleringer.

De indbyggede analyseværktøjer i CPN Tools kan benyttes til at lave en formel analyse af CPN-modeller, og derigennem kan man få oplysninger om f.eks. deadlocks i dem. Analyseværktøjerne er dog kun i stand til at analysere relativt små modeller – min model af DSR er alt for stor til, at man kan benytte analyseværktøjerne på den. Denne påstand vil jeg underbygge i dette kapitel.

4.1 Brug af de indbyggede analyseværktøjer i CPN Tools

CPN Tools indeholder et state space-værktøj, der gør det muligt f.eks. at detektere løkker i brikkernes placering i modellen, deadlocks etc. State space-værktøjet fungerer på den måde, at det laver en graf over alle tilstande i modellen og benytter denne til videre analyser. Ifølge [JCK02, side 11] var state space-værktøjet i år 2002-versionen af CPN Tools i stand til at håndtere op til 20.000-200.000 forskellige tilstande afhængigt af mængden af RAM i den brugte maskine. Jvf. [CT] er der ikke siden da sket forbedringer på dette område.

I min model af DSR er der væsentligt flere tilstande end dette, hvilket gør det umuligt at benytte state space-værktøjet. Denne påstand underbygges i resten af dette afsnit ved at undersøge forholdet mellem den nyeste version af CPN Tools' formåen og antallet af tilstande i DSR-modellen.

Som en indledende øvelse forsøgte jeg at benytte CPN Tools til at beregne state space for DSR-modellen. Som forventet lykkedes dette ikke inden for et halvt døgn på en 3 GHz maskine med 1 GB RAM.

Antallet af tilstande i min model bestemmes bl.a. af antallet af knuder, modellen er konfigureret til at simulere effekten af, da mange af pladserne (som beskrevet i afsnit 3.2.2) indeholder en brik for hver knude i systemet. Det mindste antal knuder, man kan tillade sig at konfigurere modellen til at simulere, er 3 – dette vil kunne simulere situationer, hvor en pakke rejser gennem det simulerede netværk fra en afsenderknude via en videresendende knude til en modtagende knude. Dermed kommer de transitioner, der beskæftiger sig med den basale si-

mulering af afsendelse, videresendelse og modtagelse i brug. I RD vil både viderebroadcasting og besvarelse af "Route Request"-pakker kunne blive simuleret, og i RM vil både passive og eksplicitte bekræftelser kunne blive simuleret. "Cached Route Reply" (CRR – knude A igangsætter en RD på knude C, og knude B svarer jvf. afsnit 2.3.6) vil også kunne blive simuleret, mens "Salvage Operations" (SO) vil kræve mindst 4 knuder (knude A sender en pakke til knude C via knude B, knude B opdager at linket til knude C ikke fungerer, men en rute gennem knude D findes jvf. afsnit 2.4.6). Sættes antallet af knuder ned til 2 vil heller ikke brugen af videresendende knuder, af passive bekræftelser eller af CRR kunne simuleres.

I modellen er en af de faktorer, der sætter antallet af tilstande op, timeout-værdierne. F.eks. bliver hver indgang i en "Send Buffer"-tabel tilknyttet en timeout, der starter på 15.000 og bliver talt ned en ad gangen, mens hver indgang i en "Route Cache"-tabel bliver tilknyttet en timeout, der starter på 150.000, og ligeledes bliver talt ned en ad gangen. Da hver knude kan have et antal indgange i hver af disse to tabeller, kan dette give et meget stort antal tilstande i DSR-modellen, selvom man begrænser sig til at fokusere på disse to tabeller. Med f.eks. 3 knuder, 2 indgange i "Route Cache"-tabellen i hver knude og 2 indgange i "Send Buffer"-tabellen i hver knude, vil blot dette give $(150000^2 * 15000^2)^3$ svarende til cirka $1,3 * 10^{56}$ forskellige tilstande i tilstandsgrafen.

Dette tal virker højt i forhold til de ovenstående 20.000-200.000 mulige tilstande, CPN Tools kan håndtere, men tallet kan nemt reduceres: Man kan blot eliminere al timeoutfunktionalitet fra systemet, så alle timeout-bestemte handlinger i modellen nu kan udføres uden ventetid – f.eks. kan en indgang i "Send Buffer" straks fjernes som forældet, selvom den blev indsat i tabellen i skridtet før i modellen.

I simuleringssammenhænge ville dette betyde, at det i praksis ville blive næsten umuligt at simulere en forsendelse af en pakke fra en knude til en anden, idet alle tabel-indgange straks ville blive fjernet. I state space-sammenhænge betyder dette intet, idet *alle* tilstande beregnes – inkl. dem, hvor en pakke i "Send Buffer" ikke fjernes, før en rute til pakkens destination findes v.h.a. RD (og pakken derfor kan sendes).

Jeg lavede derfor et forsøg med DSR-modellen, hvor alle timeouts på denne måde blev fjernet, og forsøgte at benytte denne til igen at beregne state space. Heller ikke dette lykkedes inden for et halvt døgn.

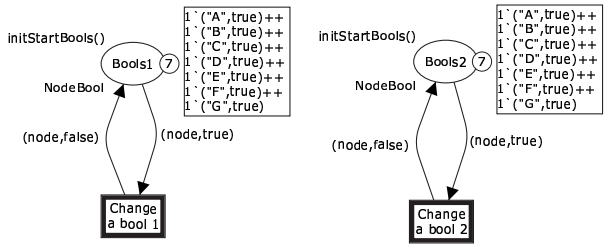
Også dette var dog forventeligt p.g.a. den måde, jeg gemmer data i modellen: Jeg har et stort antal pladser i modellen, hvor der på hver plads er individuelle brikker for hver knude i systemet (jvf. afsnit 3.2.2). Dette gælder næsten alle pladser i figur 3.10-3.35 på side 51–83 i kapitel 3 og figur B.1-B.7 på side 140–143 i appendix B.1 – over 100 pladser i alt.

Hvis man antager, at der er 3 knuder i en konfiguration af modellen, og at hver plads i modellen kan have 2 forskellige tilstande for hver knude (for eksemplets skyld – i praksis vil næsten alle pladser i modellen kunne have flere tilstande end det), så vil dette give $(2^3)^{100}$ forskellige tilstande – svarende til cirka $2,0 * 10^{91}$ forskellige tilstande.

Det er klart, at dette antal tilstande er for højt til at man kan udregne en tilstandsgraf for dem inden for nogen form for rimelig tid. For at illustrere dette, vil jeg i det nedenstående estimere hvor lang tid, en sådan beregning vil tage, v.h.a. en eksempelmodel.

Hvis man laver en eksempelmodel (som den, der er vist i figur 4.1 på næste side), hvor man har blot 2 pladser, der hver indeholder 7 brikker, der igen kan tage 2 værdier, vil der være $(2^7)^2 = 16384$ mulige tilstande i denne model. Med den på testtidspunktet nyeste version af CPN Tools (version 1.5.16) kunne state space for denne eksempelmodel beregnes på 18 minutter på en 3 GHz maskine med 1 GB RAM.

For at finde ud af, om man kan vurdere, hvor lang tid det vil tage at beregne state space for den fulde model (uden timeout-værdier), undersøger jeg, om man kan udtrække noget information om beregningstiden ud fra antal tilstande. Til dette formål har jeg lavet en række forsøg, hvor jeg varierede både antallet af knuder og antallet af pladser. I tabel 4.2 på næste side kan resultattiderne for state space-beregningen på en 3 GHz maskine med 1 GB RAM ses. I tabellen bliver både antallet af knuder og antallet af pladser varieret for at sikre, at det



Figur 4.1: Eksempelmodel med 2 pladser med hver 7 brikker, der hver kan have 2 værdier, hvilket tilsammen giver 16384 mulige tilstande – her vist med den initielle tilstand

Antal brikker pr. plads	Antal pladser	Antal tilstande i modellen	Ca. beregningstid
3	3	$(2^3)^3 = 512$	< 1 sek.
5	2	$(2^5)^2 = 1024$	3 sek.
6	2	$(2^6)^2 = 4096$	53 sek.
4	3	$(2^4)^3 = 4096$	53 sek.
7	2	$(2^7)^2 = 16384$	1080 sek. (= 18 min.)
5	3	$(2^5)^3 = 32768$	5160 sek. (= 86 min.)

Tabel 4.2: Tider for at beregne state space for en eksempelmodel i CPN Tools

er antallet af tilstande, der er hovedafgørende for tiden, og ikke at f.eks. antallet af pladser i modellen er mere betydende for tiden end f.eks. antallet af tilstande på den enkelte plads.

En graf over beregningstiderne kan ses i figur 4.3(a) på næste side. Som det kan ses, er beregningstiden *ikke* lineær i antallet af tilstande. En mulig årsag til dette kan være, at når et analyseværktøj skal beregne en tilstandsgraf, kan dette bl.a. gøres ved, at hver gang en tilstand findes i modellen, søger værktøjet gennem alle tidligere fundne tilstande for at finde ud af, om den fundne tilstand er ny, eller den er magen til en tidligere fundet tilstand. Dette gør, at man kan forvente, at beregningstiden vil vokse eksponentielt med antallet af tilstande.

I figur 4.3(b) på næste side er de samme beregningstider derfor vist i en graf, hvor begge akser er logaritmiske, og her er linien tilnærmelsesvist retlinet. Jvf. f.eks. [MU90, side 15] betyder dette netop, at der er en eksponentiel sammenhæng mellem antal tilstande og beregningstiden – mere præcist at beregningstiden kan estimeres som følger:

$$\text{beregningstid} = a * (\text{antal tilstande})^n$$

Man kan nu finde et estimat for beregningstidsfunktionen. Fra [MU90, side 15] får man, at n vil være hældningen på det dobbelt-logaritmiske papir, og at a er værdien på den lige linie, hvor den skærer antal tilstande $= 1$. Fra figuren kan man hermed estimere a og n:

$$n \approx \frac{\ln(\text{beregningstid}_1) - \ln(\text{beregningstid}_0)}{\ln(\text{antal tilstande}_1) - \ln(\text{antal tilstance}_0)} = \frac{\ln(5160) - \ln(3)}{\ln(32768) - \ln(1024)} \approx 2.15$$

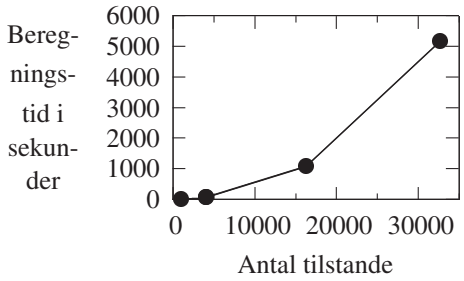
og

$$a \approx 1 * 10^{-6}$$

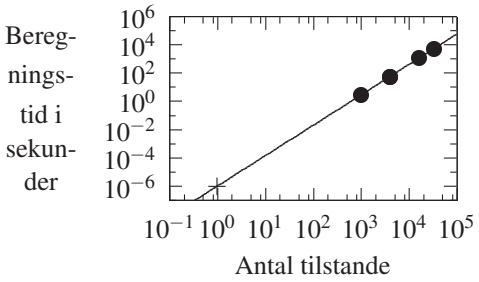
Ialt kan beregningstiden altså estimeres med funktionen:

$$\text{beregningstid} \approx 10^{-6} * (\text{antal tilstande})^{2,15} \text{ sekunder}$$

Da man som tidligere beregnet har mindst $2,0 * 10^{91}$ tilstande i modellen uden timeout-værdier, giver dette en approksimeret beregningstid på mindst $10^{-6} * (2,0 * 10^{91})^{2,15}$ sekunder, hvilket svarer til $4,6 * 10^{180}$ år.



(a) Plot på normal graf



(b) Plot på en graf med logaritmiske skalaer – med en retlinet linie tegnet gennem de plottede punkter og med et mærke sat, hvor linien skærer antal tilstande $= 10^0 (= 1)$

Figur 4.3: Beregningstiderne fra tabel 4.2 på forrige side plottet

Som tidligere beskrevet er dette tal baseret på, at modellen konfigureres til at simulere effekten af 3 knuder. Sættes antallet af knuder ned til 2 (hvilket som tidligere forklaret kraftigt vil begrænse hvilke områder af modellen, der bliver analyseret), vil man i stedet få en approksimeret beregningstid på $8,8 * 10^{115}$ år. Sættes antallet af knuder yderligere ned til 1 (hvilket i praksis vil betyde, at næsten ingen områder af DSR-modellen bliver analyseret), bliver den approksimerede beregningstid $1,7 * 10^{51}$ år.

Det skal bemærkes, at de her udledte tal naturligvis kan variere, hvis man bruger en anden model til at udtrække tider med end den, jeg konstruerede i figur 4.1 på forrige side. Forskellen på de tal, man får, og de tal, det vil være realistisk at arbejde videre med, er dog så stor, at jeg hermed anser påstanden om, at man ikke vil kunne bruge state space-analyser til at analysere modellen beskrevet i kapitel 3, for underbygget.

4.2 Relateret arbejde

Som beskrevet i afsnit 3.9 findes der adskillige eksempler på systemer, man har bygget en CPN-model over, og derefter brugt denne model til at lave en formel analyse af – f.eks. for at uddrage information om mulige deadlocks i modellerne og dermed eventuelt de modellerede systemer selv. Dette har man kunnet gøre, fordi man har fokuseret på en enkelt problemstilling i systemerne ad gangen og dermed har kunnet skære CPN-modellerne så kraftigt til, at antallet af tilstande i modellerne kunne holdes under det, CPN-værktøjerne kan håndtere.

Der findes også andre eksempler på relateret arbejde. I [BGH04] beskrives en fremgangsmåde for, hvordan man formelt kan verificere, at en protokol fungerer som forventet. Dette gøres ved at bygge en "service"-CPN-model i stedet for kun at bygge en "protokol"-CPN-model. Denne angiver de kald, en bruger af protokollen (f.eks. en ovenliggende protokol) kan benytte i protokollen. Hver af disse angives som en atomar handling i modellen – d.v.s. en enkelt transition. Fra disse to modeller kan man nu uddrage tilstandsgrafer, der kan omdannes til endelige automater, som herefter kan sammenlignes mere eller mindre automatisk. Hvis de er ens, opfylder "protokol"-CPN-modellen kravene fra "service"-CPN-modellen – hvis ikke, kan dette afsløre problemer i en af modellerne og dermed måske i protokollen.

Også dette kræver dog, at antallet af tilstande i modellen er lille nok til, at man inden for rimelig tid kan få f.eks. CPN Tools til at danne denne tilstandsgraf, og i artiklen gøres det af samme grund kun for henholdsvist en stop-and-wait-protokol, som benytter et sekvensnummer, der kun må være 0 eller 1, og for et udsnit af forbindelsesdelen af TCP-protokollen. At genereringen af de endelige automater kræver, at der også her laves tilstandsgrafer, gør dog, at man heller ikke kan benytte denne metode til at analysere min DSR-CPN-model.

5 Analyse af DSR-modellen v.h.a. simuleringer

At benytte de indbyggede analyseværktøjer i CPN Tools (som beskrevet i afsnit 4.1) er ikke den eneste måde, man kan undersøge en CPN-model på. I dette afsnit vil jeg lave mere pragmatiske analyser af DSR-modellen ved at køre en række simuleringer af den.

I denne forbindelse skal det først defineres, hvordan en testkørsel skal se ud, og hvordan DSR-protokollen skal konfigureres i forbindelse med en sådan kørsel – specielt m.h.t. hvilke parametre, man varierer heri, og hvad det forventede resultat af dette vil være. Dette gøres i afsnit 5.1. Her beskrives det desuden, at tiden, det tager at udføre en enkelt simulering gør, at jeg har været nødt til at begrænse mig til at variere på tre parametre: Knudernes hastighed, om "Cached Route Reply" (CRR, se afsnit 2.3.6) er slået til eller fra, og om "Salvage Operations" (SO, se afsnit 2.4.6) er slået til eller fra. Herefter skal simuleringskørslerne rent faktisk foretages, og en række måledata skal udtrækkes herfra. Dette beskrives i afsnit 5.2. Disse data kan så indgå i en række statistiske beregninger, som jeg vil benytte til at forsøge at drage konklusioner om egenskaberne af modellen af DSR – de statistiske metoder ridses kort op i appendix G og benyttes i afsnit 5.3. Her kan man også se, at disse egenskaber ændrer sig, når man "skruer" på en af de tre valgte parametre. Jeg finder bl.a., at CRR normalt har en positiv effekt på DSR-modellens effektivitet, men at effekten bliver negativ, hvis knudernes hastighed er meget høj.

5.1 Konfigurationer til brug for testsimuleringer

I dette afsnit vil jeg opstille en række testkonfigurationer, der kan bruges til simuleringer af DSR-modellen i afsnit 5.2.

5.1.1 Valg af parametre, der kan varieres i testkonfigurationerne

Der er en lang række mulige parametre, man kan "skrue" på i opstillingen af testkonfigurationer. Det drejer sig om f.eks.:

1. Hvor mange simulerede knuder er der i DSR-modellen?
2. Hvor stort er det simulerede område, knuderne kan bevæge sig rundt i?

3. Hvad er de simulerede knuders tæthed (d.v.s. hvor langt er knuderne fra hinanden)?
4. Hvor stor simuleres antennerækkevidden til at være?
5. Hvor hurtigt bevæger knuderne sig (simuleret) rundt i området?
6. Hvor tit simuleres det, at knuderne sender pakker til hinanden?
7. Er udvidelsen CRR (se afsnit 2.3.6) slået til eller ej i DSR-modellen?
8. Er udvidelsen SO (se afsnit 2.4.6) slået til eller ej i DSR-modellen?

Listen kan i princippet udvides med flere andre parametre, men disse understøttes endnu ikke fuldt af modellen i den nuværende version, så dem vil jeg se bort fra, da de ovenstående allerede giver nok at arbejde med i det nedenstående. De ekstra punkter, der kan udvides med, inkluderer bl.a. hvilken netværkstype, modellen konfigureres til at simulere ("Frequently-unidir", "Mostly-bidir" eller "Bidir-only" – den første og den sidste er ikke fuldt modelleret jvf. afsnit 3.7.2 og 3.7.1), og hvilken "Route Cache"-strategi, der benyttes i modellen (dette er i øjeblikket fastsat til en "den bedste rute er den sidst modtagne eller brugte"-strategi jvf. afsnit 3.3.1). Som beskrevet i afsnit 2.2.4 findes der mange andre "Route Cache"-strategier, men en måde at konfigurere modellen til at skifte mellem disse er ikke inkluderet.

Som beskrevet i afsnit 3.6.3 benytter jeg et egenudviklet sprog til at styre, hvordan en simulering i DSR-modellen skal foregå – mere præcist hvordan knuderne skal bevæge sig rundt i forhold til hinanden og hvornår og til hvem, de skal sende pakker. Visse af de ovenstående parametre bestemmes derfor implicit af hvilke kommandoer, man giver til DSR-modellen i dette sprog, mens andre af parametrene styres gennem en direkte konfigurering af DSR-modellen. Mere præcist kan de ovenstående parametre deles op i tre grupperinger:

A. Parametre, der styres udelukkende gennem en konfigurering af DSR-modellen: Parameter 4, 7 og 8.

B. Parametre, der styres udelukkende gennem de givne kommandoer skrevet i det ovenfor nævnte sprog: Parameter 3, 5 og 6.

C. Parametre, der styres gennem en konfigurering af DSR-modellen, og som kommandoerne derefter skal overholde: Parameter 1 og 2.

Hvis man gerne vil teste, hvordan DSR-modellen opfører sig under forskellige forhold, bør man altså sammensætte testkonfigurationer, hvor alle kombinationer af disse parametre benyttes til et antal simuleringskørsler. Men er dette muligt i praksis?

Hvor punkt 7-8 hver har to forskellige indstillinger ("slået fra"/"slået til"), så kan de øvrige (punkt 1-6) have et vilkårligt stort antal indstillinger. Selv hvis man finder 3 indstillinger, som man kan argumentere for, dækker spektret af indstillingsmuligheder for hvert af disse punkter, giver dette $3*3*3*3*3*3*2*2 = 2916$ forskellige testkonfigurationer – som der hver bør udføres et antal simuleringer under (dette beskrives nærmere i afsnit 5.2.1). Efter hver simulering skal data hentes ud af modellen, modellen skal "resettes" (d.v.s. hver plads i modellen skal sættes til sin initielle værdi), konfigurationen skal ændres til nye parameterværdier og en ny kommandoliste, hvorefter en ny simulering skal startes. Ifølge CPN Tools-programmørerne (via CPN Tools-support-mailinglisten [CTS]) er dette i praksis umuligt at gøre som en automatiseret proces, og derfor skal hver af disse tests igangsættes manuelt. Dette vil gøre testprocessen uoverskueligt stor.

Imidlertid kan man argumentere for, at det hovedsageligt er hvordan, Dynamic Source Routing-modellens måledata påvirkes af ændringer i de første seks parametre hver for sig, der er interessant, inkl. om ændringerne er anderledes, når man slår de to udvidelser til eller fra.

Man kan derfor minimere antallet af testkonfigurationer ved at se på hver af de første seks parametre hver for sig, d.v.s. f.eks. ved at lade antallet af knuder variere, mens de øvrige fem parametre holdes fast på et defineret middelniveau.

Dette vil naturligvis være langt fra lige så udtryksfuldt som den fulde analyse, idet en række sammensætninger af parametrenes yderområder ikke bliver belyst. Hvis de fastsatte parametre er valgt med omhu, kan det dog alligevel fortælle en del om protokollens virkemåde. Dette vil nedbringe antallet af testkonfigurationer, så det i alt bliver $(3+3+3+3+3+3)*2*2=72$.

For yderligere at nedbringe antallet af testkonfigurationer, kan man argumentere for, at visse af de første seks punkter i høj grad dækker hinanden:

- Antennerækkevidden i min model er ens for alle knuder (jvf. afsnit 3.7.2). På grund af dette kan den gennemsnitlige tæthed i et område og knudernes antennerækkevidde ses som to sider af den samme sag: Hvis man har en CPN-model, hvor et antal knuder er placeret ens i forhold til hinanden i et kort, men afstanden mellem dem alle forøges (f.eks. ved at kortet gøres større, og knuderne "følger med" i ekspansionen), så ville den samme situation opstå, hvis man lod kortstørrelsen (og knudernes position) være den samme, men blot formindskede knudernes antennerækkevidde. P.g.a. dette kan man i analysesammenhænge faktorere antennerækkevidden ud af mængden af parametre uden at ændre på udtryksfuldheden af denne.

- Hvis man ser på knudernes tæthed som en gennemsnitlig værdi (i modsætning til f.eks. en minimum- eller maksimum-afstand mellem knuder), kan tætheden i et område beskrives som antal knuder divideret med områdestørrelsen. Begge disse variable er parametre, der allerede findes i listen. Derfor kan man i analysesammenhænge også faktorere tætheden ud af mængden af parametre uden at ændre væsentligt på udtryksfuldheden af denne.

Man kan hermed skære den ovenstående parameterliste ned til følgende:

1. Hvor mange simulerede knuder er der i DSR-modellen?
2. Hvor stort er det simulerede område, knuderne kan bevæge sig rundt i?
5. Hvor hurtigt bevæger knuderne sig (simuleret) rundt i området?
6. Hvor tit simuleres det, at knuderne sender pakker til hinanden?
7. Er udvidelsen CRR (se afsnit 2.3.6) slået til eller ej i DSR-modellen?
8. Er udvidelsen SO (se afsnit 2.4.6) slået til eller ej i DSR-modellen?

Dette giver nu $(3+3+3+3)*2*2=48$ forskellige testkonfigurationer, der hver bør benyttes i et antal simuleringskørsler, som det beskrives i afsnit 5.2.1.

I næste afsnit vil jeg vise, hvordan selv en minimal testkørsel tager halvanden time at gennemføre, og i afsnit 5.2.1 vil jeg argumentere for, at der skal gennemføres mindst 5 testkørsler pr. konfiguration. Sammenholder man dette med det ovenstående kan man se, at man af praktiske årsager vil være nødt til at skære ned på det antal parametre, man vælger at variere i de forskellige testkonfigurationer, hvis man skal være i stand til at gennemføre analyserne.

For at finde ud af hvilke parametre, der skal overleve denne nedskæring, er det værd at kigge på, hvilke resultater man ville kunne få ud af at variere de enkelte parametre:

1. Variation af antal knuder: Kan bl.a. bruges til at teste, om der vil blive flere invaliderede ruter i netværket p.g.a. forhøjede ventetider, hvis knudernes processorkraft bliver udnyttet fuldt ud, eller det trådløse netværk bliver overfyldt. Da de to valgte DSR-model-udvidelser CRR og SO bl.a. forsøger at nedsætte antallet af pakketransmissioner i

netværket, kan variation af denne parameter også benyttes til at teste, hvorvidt udvidelserne vil modvirke en for hurtig fuld udnyttelse af knudernes processorkraft eller kapaciteten i det trådløse netværk. Parameteren kan også benyttes til at teste, om der kan opnås samarbejdsfordele, jo flere knuder, der deltager i et netværk – f.eks. p.g.a. genbrug af cachede data.

Dette betyder, at der potentielt skal modelleres afsendelse af mange pakker i et netværk for at kunne teste de ovenfor beskrevne punkter, og derfor kan simulationerne blive meget lange og dermed meget tidskrævende.

2. Variation af områdestørrelsen: Kan hovedsageligt bruges til at teste hvor hurtigt, en rute ikke længere kan findes mellem to knuder, fordi knuderne kommer til at ligge for spredt i forhold til hinanden.

Dette er et interessant problem, men det er et problem, der er generelt for MANET-protokoller, og som ikke burde give specifikke resultater for modeller af DSR-protokollen – specielt da DSR ligesom f.eks. AODV (se appendix A.3) benytter en generel flooding-algoritme til at finde ruter med (som beskrevet i f.eks. [Per05]). En generel undersøgelse af flooding-algoritmers effektivitet i forhold til områdestørrelsen kan f.eks. ses i "Simulation Study A" i [YGK03].

5. Variation af knudernes bevægelseshastighed: Kan hovedsageligt bruges til at teste, om DSR-modellens evne til at aflevere pakker i et netværk falder, jo hurtigere knuderne bevæger sig rundt i forhold til hinanden. Desuden forsøger de to valgte DSR-model-udvidelser SO og CRR netop at påvirke, hvad der sker, når ruter invalideres, eller nye ruter skal findes, så da forøgelse af knudernes hastighed kan betyde en forhøjet ruteinvalidering i netværk, kan effekten af udvidelserne måske tydeligere vises selv under korte testkørsler.

6. Variation af knudernes pakkeforsendelsesrate: Også denne kan hovedsageligt bruges til at teste, om der vil blive flere invaliderede ruter i netværket p.g.a. forhøjede ventetider, hvis knudernes processorkraft bliver udnyttet fuldt ud, eller det trådløse netværk bliver overfyldt. Desuden kan parameteren bruges til at teste, hvorvidt der kan opnås "stordriftsfordele", jo flere pakker, der sendes rundt i netværket – f.eks. p.g.a. genbrug af cachede data.

Variation af denne parameter vil med andre ord kunne vise en variation over de samme ting som parameter 1, men har også de samme ulemper.

7. Variation af, om CRR er slået til eller fra: Kan bruges til at teste, om man kan argumentere for eller imod, at det er værd at medtage udvidelsen i en senere implementation af protokollen. Dette kan gøres i kombination med variation af vilkårlige af de ovenstående parametre.

8. Variation af, om SO er slået til eller fra: Kan bruges til at teste, om man kan argumentere for eller imod, at det er værd at medtage udvidelsen i en senere implementation af protokollen. Dette kan gøres i kombination med variation af vilkårlige af de ovenstående parametre.

Ud fra denne gennemgang vælger jeg at fokusere på tre parametre: Parameter 5: Hvor hurtigt knuderne (simuleret) bevæger sig rundt i området, parameter 7: Om CRR er slået til eller fra og parameter 8: Om SO er slået til eller fra. Dette giver i alt $3 * 2 * 2 = 12$ testkonfigurationer, hvilket man skulle være i stand til at gennemføre nok testkørsler for selv under de ovenfor nævnte betingelser for tid og antal testkørsler.

5.1.2 Valg af testkonfigurationernes længde og vigtigste parametre

Som beskrevet i forrige afsnit er noget af det, jeg gerne vil undersøge, effekten af f.eks. udvidelsen CRR på DSR-modellen. Da denne først viser sin effekt efter et antal "Route Discoveries" er blevet igangsat gående på den samme knude fra forskellige knuder (jvf. afsnit 2.3.6), vil jeg gå ud fra, at det vil have en positiv effekt på resultaterne af simuleringerne, hvis de kørte kommandolister (der som beskrevet i afsnit 3.6.3 igangsætter simulerede knudebevægelser og pakkeafsendelser) er så lange som muligt. Jeg vil også gerne undersøge effekten af SO på DSR-modellen, og her gælder det samme: Den viser først sin effekt efter et antal "Route Discoveries" allerede har været igangsat, og knuderne efterfølgende har bevæget sig rundt blandt hinanden (jvf. afsnit 2.4.6). Derfor vil jeg gå ud fra, at også resultaterne for undersøgelserne af SO vil være mere tydelige, jo længere kommandlisterne er i testene.

I afsnit 4.1 blev en minimal testkonfiguration af modellen defineret som en konfiguration bestående af 3 knuder. Hvis man skal være sikker på, at udvidelserne kommer i brug, er man dog nødt til at sætte dette tal op. I demonstrationerne af udvidelserne i afsnit 3.8.3 og 3.8.4 vises det, at hvis man håndbygger knudernes placering og bevægelse i modellen, kan udvidelserne komme i brug, hvis modellen konfigureres til at benytte 4-5 knuder. I testkørslerne vil knudernes placering og bevægelse ikke være håndbygget, og derfor vil samme effekt muligvis ikke ses, medmindre man sætter antallet af knuder yderligere op.

På den anden side er jeg begrænset af rent praktiske forhold: At køre bare en enkelt simulering af en vis størrelse tager tid. Jeg har testet modellen med brug af et forskelligt antal knuder, forskellige kommandolistelængder og et forskelligt antal pakker afsendt pr. knude.

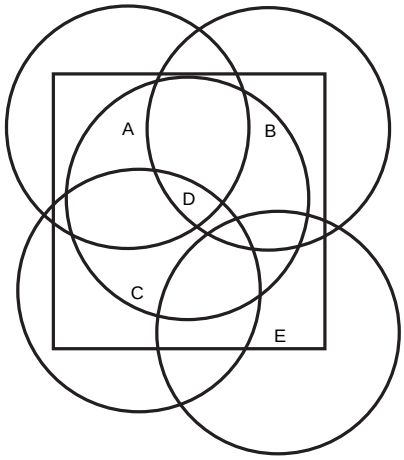
Her har jeg bl.a. fundet, at det tager cirka halvanden time at gennemføre en testkørsel af en testkonfiguration med 5 knuder, der for hver knude i løbet af 25000 skridt (altså *timeout*-skridt, som beskrevet i afsnit 3.6.3 – ikke at forveksle med "steps" i CPN Tools) sender en pakke gennemsnitligt for hvert 12500 skridt, d.v.s. i snit 2 pakker pr. knude eller 10 pakker i alt. Dette er målt på en 3 GHz maskine med 1 GB RAM og inkluderer tid til at gennemføre en rekonfigurering af modellen, simulering af modellen og udtrækning af resultatdata fra modellen. Forøges kommandolistelængerne (f.eks. ved at sende flere pakker), forøges tiden, simuleringen tager, eksponentielt (hvilket falder i tråd med diskussionen i afsnit 3.2.2, da kommandolisterne repræsenteres som lister i modellen). Forøges antallet af knuder, forøges tiden kun proportionalt hermed.

I afsnit 5.2.1 beskriver jeg, at man skal lave et vist antal testkørsler for hver konfiguration af modellen. Derfor har jeg valgt ikke at gøre størrelsen af testkonfigurationerne større end de ovenstående 5 knuder. Jvf. demonstrationerne i afsnit 3.8.3 og 3.8.4 burde denne størrelse være stor nok til, at i det mindste nogle af testkørslerne vil vise en effekt af de to udvidelser, selvom dette jvf. ovenstående ikke er sikkert. Samtidigt anser jeg også testkonfigurationens størrelse for at være minimal – formindskes den yderligere (m.h.t. antal knuder eller antal afsendte pakker), formindskes muligheden for, at en af de to udvidelser kommer i brug i de tilsvarende testkørsler, yderligere.

Artiklen [KCC05] har gennemgået en række artikler fra årene 2000-2005, der alle involverer simuleringer af MANET-protokoller. [KCC05] viser, at antallet af knuder i artiklerne svinger mellem 10 og 30000, så her er mit antal altså sat temmeligt lavt. Jeg har dog set mig nødsaget til at gøre det alligevel a.h.t. tiden, de enkelte testkørsler ellers vil ende med at tage. I afsnit 5.4 vender jeg tilbage til, om dette i retrospekt var et godt valg.

5.1.3 Valg af testkonfigurationernes øvrige parametre

Modellen konfigureres, så området i modellen laves $500 * 500$ enheder stort (jvf. afsnit 3.6.1) og antennerækkevidden sættes til 300. Dette gør, at der er en relativt stor sandsynlighed for, at en rute mellem to knuder findes, da antennerækkevidden er relativt stor i forhold til områdestørrelsen, se figur 5.1 på næste side.



Figur 5.1: Eksempel på tilfældig placering af 5 knuder i et simuleret $500 * 500$ enheder stort område, hvor hver knude har en antennerækkevidde på 300 enheder. I dette eksempelnetværk kan f.eks. A og C nå hinanden via D, mens f.eks. E ikke kan nå nogen af de øvrige knuder i netværket. Ringene angiver de enkelte knuders antennerækkevidder, mens firkanten angiver det $500 * 500$ enheder store område.

De ovenstående tal skal sammenlignes med, at [KCC05] finder frem til, at andre artikler har benyttet en områdestørrelse på mellem $25 * 25$ og $5000 * 5000$ enheder med en antennerækkevidde på mellem 3 og 1061 enheder – med kombinationer, der både giver en større og en mindre antennerækkevidde pr. områdestørrelse-ratio end mit valg.

Værdierne for parametrene fundet i afsnit 5.1.1 kan nu defineres mere præcist:

1. Antal simulerede knuder i DSR-modellen: 5

2. Områdestørrelsen: $500 * 500$ enheder

3. Knudernes tæthed: Jvf. [KCC05] udregnes denne som $\frac{\text{antal knuder}}{\text{områdestørrelse}}$ = 0,00002. Til sammenligning refererer artiklen til andre simuleringer med både en højere og lavere tæthed end denne.

4. Antennerækkevidde: 300 enheder – d.v.s., at hvis knude A og knude B har koordinaterne (x_A, y_A) og (x_B, y_B), så skal $\sqrt{(x_A - x_B)^2 + (y_A - y_B)^2} \leq 300$ for at pakker sendt fra den ene knude vil blive overhørt af den anden.

Endnu et interessant forhold findes her; det mellem tætheden og antennerækkevidden. Også dette forhold har [KCC05] en formel for, og også her refereres der i [KCC05] til artikler med både et højere og et lavere forhold end den ovenstående konfigurering giver.

De valgte parametre gør også, at selvom der sagtens kan forekomme konfigurationer, hvor ingen af knuderne nogensinde kan nå hinanden, så vil der i de fleste tilfælde kunne findes en rute mellem to knuder, hvilket betyder, at de fleste pakker rent faktisk burde nå frem til den endelige modtager af pakken (se figur 5.1).

5. Knudernes bevægelseshastighed: Knuderne bevæger sig fra -40 til 40 enheder i hver af X- og Y-retningerne (dog justeret, så de holder sig inden for det definerede område fra parameter 2) efterfulgt af en pause, hvorefter der sker en ny bevægelse. Denne pause er den parameter, der varieres i de forskellige forsøg. Gennemsnitligt skal en knude vente 12500 skridt (fremover kaldet "slow"-konfigurationen), 7500 skridt (fremover kaldet "medium"-konfigurationen) eller 2500 skridt (fremover kaldet "fast"-konfigurationen), før den bevæger sig. Som tidligere nævnt skal "skridt" her ikke forveksles med "steps"

i CPN Tools. Med samme timeout ændres den bevægelse, de enkelte knuder er i gang med, med mellem -5 og 5 i hver af X- og Y-retningerne, så en knude f.eks. ikke først vil gå 40 enheder den ene vej og derefter gå 20 skridt tilbage igen – hvis man ser på de enkelte knuder bevægelsesmønster, vil det se ud, som om de har en "retning" jvf. afsnit 3.6.1.

6. **Knudernes pakkeforsendelsesrate:** En knude sender gennemsnitligt en pakke for hver 12500 skridt. Der bliver kun sendt pakker i løbet af de første 25000 skridt, d.v.s., at hver knude i gennemsnit når at sende to pakker i løbet af en testkørsel. Denne værdi er valgt, da det (med det antal knuder, der er, og den hastighed, det tager for en knude af behandle en videresendelse af en pakke i DSR-modellen) betyder, at afsendte pakker i de fleste tilfælde når at nå frem til modtageren, før den næste pakke afsendes – og hermed kan information fundet i en RD som regel udnyttes, når den næste pakke skal sendes – men ikke hver gang, da de 12500 skridt er en gennemsnitlig værdi.

   Grunden til, at det er værd at forsøge at minimere antallet af "Route Discoveries" på denne måde, er udelukkende for at forsøge at minimere tiden, der skal benyttes til at køre simuleringerne. Dette vil naturligvis påvirke de måledata, man får ud af simuleringerne af testkørslerne, men det vil ske på en ensartet måde, og det vil derfor ikke f.eks. invalidere eventuelle konklusioner, man i afsnit 5.3 kan uddrage om, hvad der sker, når man f.eks. går fra en konfiguration uden en bestemt udvidelse til en konfiguration *med* udvidelsen.

Hastigheden af knuderne kan virke meget høj. En knude vil kunne bevæge sig med op til 40 enheder i hver af X- og Y-retningerne i området, og vil gøre det gennemsnitligt 1 (i "slow"-konfigurationen") til 5 (i "fast"-konfigurationen) gange så tit, som den afsender en pakke. Dette vil altså svare til, at en knude i den hurtigste konfiguration i snit kan komme helt op på at bevæge sig $5 * \sqrt{40^2 + 40^2} \approx 283$ enheder hver gang, en pakke sendes ud – eller lige under antennerækkevidderadiussen på 300 enheder.

Dette er gjort med vilje, idet der kun er 5 knuder i systemet, og hver knude i snit kun sender 2 pakker ud pr. simuleringskørsel. Hvis man skal kunne nå at se en effekt af de to udvidelser CRR og SO (der hovedsageligt virker, når ruter bliver invalideret, eller nye ruter skal findes til knuder, som en naboknude kender en rute til jvf. afsnit 2.3.6 og 2.4.6), skal ruter kunne nå at blive invalideret inden for den periode, det tager at sende disse i snit 10 pakker. Ud fra andre artiklers undersøgelser ([BMJ+98, CK04]) får man, at man kan forvente en pakkefremkomst for DSR på mindst $75 - 95\%$. Hvis man skal fremprovokere nogle fejl (som herefter kan optimeres af de valgfrie udvidelser) inden for det begrænsede antal pakker, der simuleret bliver sendt i netværket, er knudernes hastighed nødt til at blive sat op. Derfor er hastigheden af knuderne sat så højt, som den er.

5.2 Testkørsler og fundne måledata

5.2.1 Testkørslerne i praksis

Noget af det vigtigste for succesgraden af brugen af statistiske metoder på måledata fra testkørsler er *antallet* af kørte testkørsler. I appendix G præsenteres kort en række statistiske metoder og herunder en række formler, der for et nøgletal vil give et område, hvori middelværdien af dette nøgletal med høj sikkerhed ("konfidens") vil ligge. "Middelværdi" betyder her nøgletalsgennemsnittet fordelt over *alle* mulige testkørsler, ikke kun de kørte testkørsler. Formlerne er indrettet, så jo færre testkørsler, der køres (og dermed jo færre måledata, der haves), jo bredere (og dermed mindre præcise) bliver de fundne områder ("konfidensintervaller").

I [AF97, Sør98] gives der forskellige formler, alt efter om man har over eller under 30 observationer, men derudover giver litteraturen ingen hjælp m.h.t. hvor langt ned i antal observationer, man kan gå. Man er nødt til at prøve at udføre et sæt testkørsler, udtrække måledata

herfra, behandle disse med de statistiske metoder præsenteret i appendix G og se, om de resulterende konfidensintervaller bliver smalle nok til, at man kan udlede resultater af dem.

Af praktiske årsager var der kun begrænset tid til rådighed til at udføre testkørslerne. Antallet af testkørsler kom derfor til at afhænge heraf. Hvis man benytter adskillige maskiner til at teste på, vil dette naturligvis nedsætte den krævede simuleringstid. Desværre kørte CPN Tools på testtidspunktet kun på Windows XP med specielle krav til grafikkortshardwaren, og ekstremt få af de offentlige maskiner på Datalogisk Institut på Aarhus Universitet understøttede på testtidspunktet disse krav. Derfor blev testene udført på en række private maskiner, hvoraf kun den hurtigste (en 3 GHz-maskine med 1 GB RAM) kunne gennemføre en test på under to timer. Den langsomste, en 650 MHz-maskine med 256 MB RAM, brugte til sammenligning ca. 8 timer på hver testkørsel. I alt var jeg i stand til at gennemføre 84 testkørsler på tiden, der var til rådighed, svarende til 7 testkørsler af hver af de 12 testkonfigurationer. Til sammenligning vises i [Sør98] eksempler på analyser med ned til 5 testkørsler.

De benyttede kommandolister, der skulle bruges i de 84 tests, blev lavet v.h.a. et Perlscript, der kan ses i appendix E.1. Dette script tager som input diverse data som antal knuder, områdestørrelsen, knudernes maksimale hastighed, tiden, der skal ventes mellem hver knudeflytning, og tiden, der skal ventes mellem hver pakkeforsendelse for hver knude, og genererer herudfra en sml-fil, som CPN-modellen af DSR kan indlæse. At starte en ny simuleringskørsel blev herefter et spørgsmål om at rette i CPN-modellen hvilken kommandolistefil, der skulle indlæses og køres, og hvilke af DSR-udvidelserne, der skulle benyttes i kørslen. Scriptet lader den underliggende opførsel af de modellerede knuder følge opførslen beskrevet i afsnit 3.6.1, altså som en modificeret version af "Boundless Simulation Area Mobility Model" [CBD02].

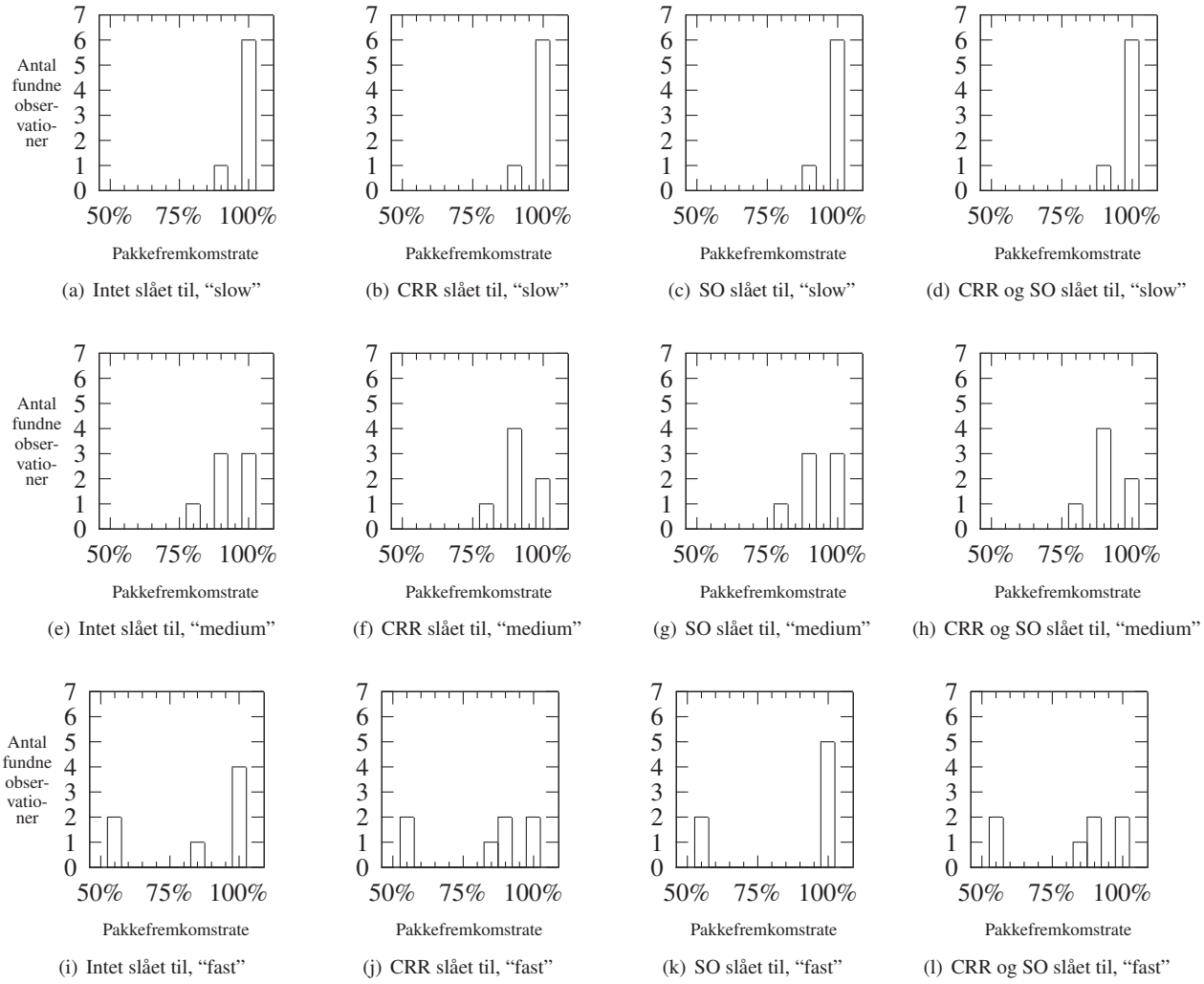
Detaljerne for testkørslerne er valgt ud fra, at det i så høj grad som muligt skal være muligt at benytte statistiske metoder på dem. Detaljerne omkring dette står i appendix G, men det er værd at fremhæve, at der er forskel på *afhængige* og *uafhængige* testkørsler. To *uafhængige* testkørsler er to kørsler, der ikke har noget fælles input – to *afhængige* testkørsler vil derimod f.eks. benytte samme kommandoliste som input.

Som beskrevet i afsnit 5.1.3 opererer jeg med 3 forskellige konfigurationer ("slow", "medium" og "fast"), og det er i hver af disse, jeg har foretaget simuleringerne af 7 forskellige (og dermed uafhængige) testkørsler. For hver af tabellerne med måledata (tabel F.1-F.3 på side 212–213) giver dette i princippet 21 uafhængige observationer at arbejde med.

Hver af disse observationer er imidlertid igen underopdelt i fire observationer, der er *afhængige* af hinanden: Hver kommandoliste er simuleret fire gange, hvor DSR-modellen henholdsvis er konfigureret til ikke at slå nogle af de modellerede udvidelser af DSR til, til at slå "Cached Route Reply" (CRR) til, til at slå "Salvage Operations" (SO) til og endelig til at slå begge udvidelser til. Dette giver i alt 84 observationer.

I mine analyser vil jeg gerne arbejde med at sammenligne, hvad der sker med observationerne, når jeg ændrer en konfiguration fra f.eks. "slow" til "fast", eller når jeg slår en udvidelse til eller fra. At de fire observationer på tværs af udvidelserne er gjort afhængige af hinanden, d.v.s. at det er den samme testkørsel, der er brugt til hver af simuleringerne, kan virke som en hæmsko i det videre arbejde, idet forskellige beregningsmetoder skal benyttes, alt efter om man ønsker at sammenligne på tværs af hastighed eller udvidelser, men det giver som beskrevet i appendix G en fordel, når man sammenligner observationer fra forskellige udvidelseskonfigurationer: Standardafvigelsen bliver mindre, og de fundne konfidensintervaller bliver derfor mere præcise.

Det ville derfor have givet mening også at lade observationerne på tværs af hastigheden være afhængige af hinanden, men dette var umiddelbart sværere at gøre, idet det ville betyde, at kommandolisten (se afsnit 3.6.3) skulle gentænkes, så `MoveNode`-kommandoer og `SendPacket`-kommandoer ikke længere foregår i samme tempo hver gang – altså så `MoveNode`-kommandoerne for en knude ville kunne "afspilles" f.eks. fem gange så hurtigt som knudens `SendPacket`-kommandoer. Derfor er kun observationer på tværs af benyttede udvidelser gjort afhængige af hinanden.



(a) Intet slået til, "slow" (b) CRR slået til, "slow" (c) SO slået til, "slow" (d) CRR og SO slået til, "slow"

(e) Intet slået til, "medium" (f) CRR slået til, "medium" (g) SO slået til, "medium" (h) CRR og SO slået til, "medium"

(i) Intet slået til, "fast" (j) CRR slået til, "fast" (k) SO slået til, "fast" (l) CRR og SO slået til, "fast"

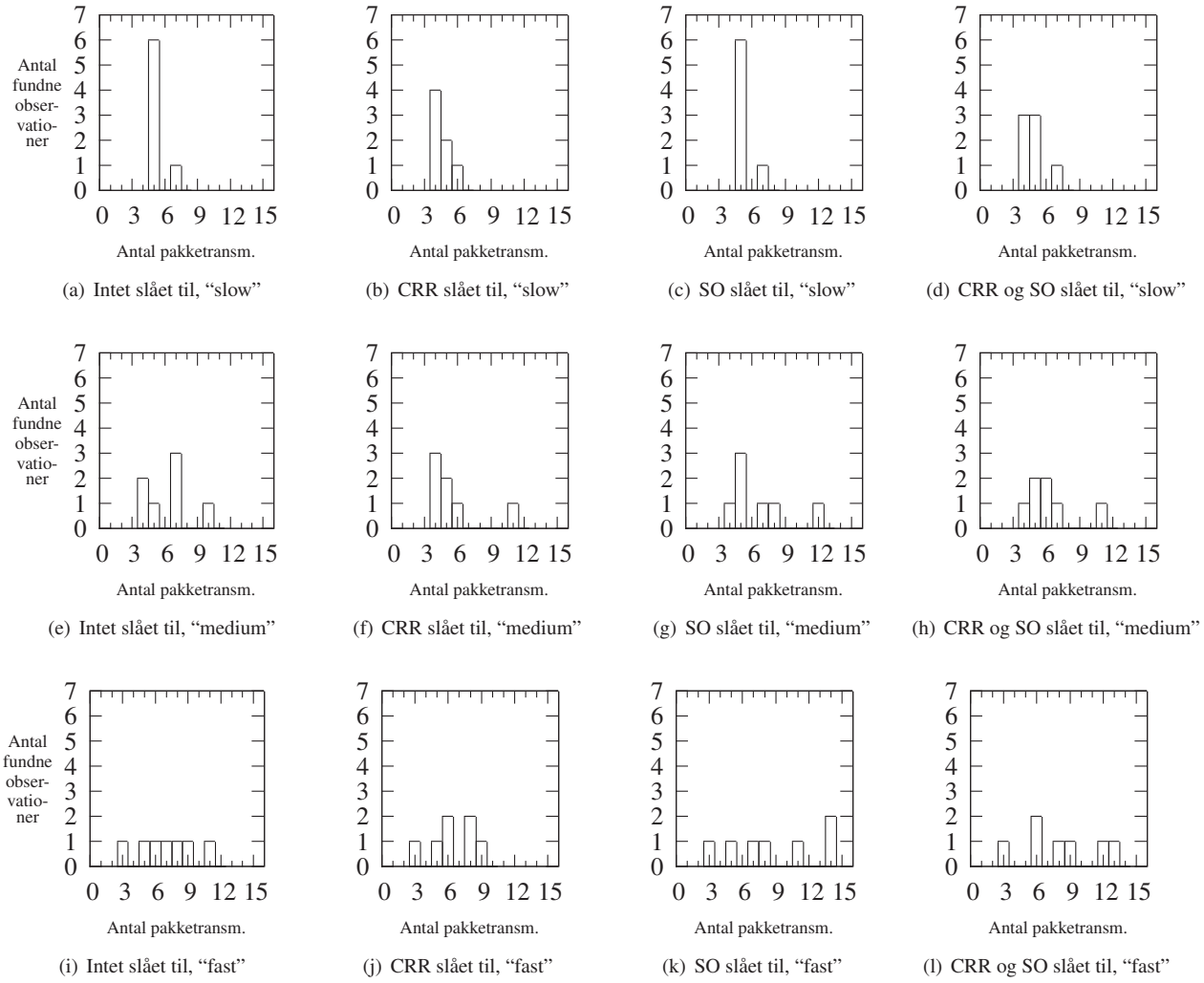
Figur 5.2: Histogram over måledata: Procentdel af de afsendte pakker, der nåede frem til den endelige modtager. De rå måledata kan ses i figur F.1 på side 212 i appendix F.1.

Output fra kørslerne blev manuelt "cut'n'pasted" fra dataopsamlingspladsen beskrevet i afsnit 3.6.4 efter hver kørsel og gemt i en række filer. Disse filer kunne herefter analyseres af et andet Perl-script, der kan ses i appendix E.2 – dette script er bl.a. i stand til at udtrække informationer som antal pakker, der nåede frem til den endelige modtager, antal pakketransmissioner m.v. Det er disse tal, jeg vil behandle i de følgende afsnit.

5.2.2 Fundne måledata fra testkørslerne

Fra testkørslerne i forrige afsnit har jeg valgt at fokusere på tre nøgletal fra hver kørsel:

Pakkefremkomst: Procentdel af de afsendte pakker, der nåede frem til den endelige modtager. Disse tal kan bl.a. benyttes til at undersøge, om udvidelserne CRR og SO sørger for, at flere pakker kommer frem til den endelige modtager. Måledataene kan ses i histogramform i figur 5.2. De rå tal kan ses i figur F.1 på side 212 i appendix F.1.



(a) Intet slået til, "slow" (b) CRR slået til, "slow" (c) SO slået til, "slow" (d) CRR og SO slået til, "slow"

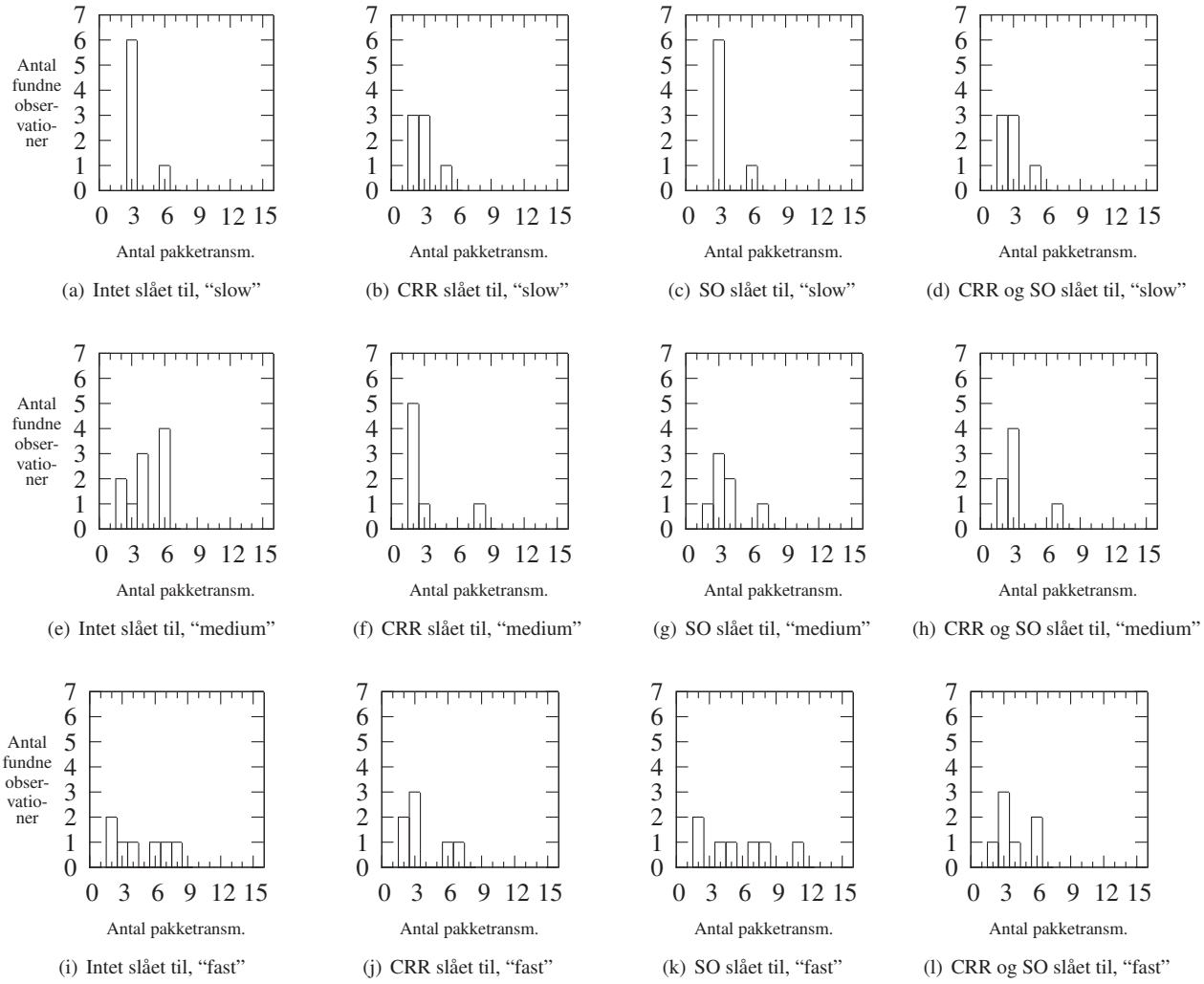
(e) Intet slået til, "medium" (f) CRR slået til, "medium" (g) SO slået til, "medium" (h) CRR og SO slået til, "medium"

(i) Intet slået til, "fast" (j) CRR slået til, "fast" (k) SO slået til, "fast" (l) CRR og SO slået til, "fast"

Figur 5.3: Histogram over måledata: Gennemsnitligt antal pakketransmissioner i netværket pr. afsendt pakke. De rå måledata kan ses i figur F.2 på side 213 i appendix F.1.

Antal pakketransmissioner: Gennemsnitligt antal pakketransmissioner i netværket pr. afsendt pakke. Også disse tal kan benyttes i forbindelse med udvidelserne for at se, om de skulle have en nedsættende effekt på antallet af pakketransmissioner i netværket, hvilket f.eks. kan være gavnligt for de enkelte knuders batteriforhold (se afsnit 1.6.1) etc. Desuden kan man f.eks. se, om antallet af pakketransmissioner stiger, jo hurtigere knuderne bevæger sig rundt i forhold til hinanden. Måledataene kan ses i histogramform i figur 5.3. De rå tal kan ses i figur F.2 på side 213 i appendix F.1.

Antal pakketransmissioner før pakkefremkomst: Gennemsnitligt antal pakketransmissioner i netværket før en pakke når frem til modtageren pr. succesfuldt modtaget pakke. Disse tal kan bruges til at se, hvor lang tid der går, før en pakke er nået frem til den endelige modtager (målt i antal pakketransmissioner for pakken) – også her kan man f.eks. se, om udvidelserne har en effekt på disse tal. Bemærk, at tallet dog ikke afspejler, hvor lang tid (i f.eks. sekunder) det tager, før en pakke kommer frem – en pakke kan f.eks. ligge og vente i en buffer hos en knude, mens knuden v.h.a. RM er i gang med at



(a) Intet slået til, "slow" (b) CRR slået til, "slow" (c) SO slået til, "slow" (d) CRR og SO slået til, "slow"

(e) Intet slået til, "medium" (f) CRR slået til, "medium" (g) SO slået til, "medium" (h) CRR og SO slået til, "medium"

(i) Intet slået til, "fast" (j) CRR slået til, "fast" (k) SO slået til, "fast" (l) CRR og SO slået til, "fast"

Figur 5.4: Histogram over måledata: Gennemsnitligt antal pakketransmissioner før en pakke når frem til modtager for succesfuldt modtagede pakker. De rå måledata kan ses i figur F.3 på side 213 i appendix F.1.

finde ud af, om linket til en naboknude stadig fungerer, og dette vil ikke blive afspejlet af tallet. Måledataene kan ses i histogramform i figur 5.4. De rå tal kan ses i figur F.3 på side 213 i appendix F.1.

I udtrækket af måledata bryder jeg med et enkelt princip: I [KCC05] fastslås det, at man i testkørsler bør lade simuleringer køre indtil indholdet i diverse datastrukturer (f.eks. de enkelte knuders "Route Cache") har nået et stabilt niveau før man begynder at gemme data, der senere skal behandles v.h.a. statisiske metoder. Dette siger den ud fra den betragtning, at knuder vil opføre sig anderledes, når de har data i deres datastrukturer, end når de ikke har. Dette har jeg måttet se bort fra i mit udtræk af måledata, idet det ville gøre simuleringerne for lange til, at jeg i praksis kunne gennemføre dem (jvf. afsnit 5.1.2). Ifølge [KCC05] er det dog ikke unormalt at fravige princippet, omend det naturligvis ikke anbefales.

Bemærk desuden, at testene blev foretaget på en tidligere version af DSR-modellen end den, der er vist i kapitel 3. Efter simuleringskørslerne blev udført, er der dog kun foretaget ændringer af forståelsesmæssig karakter. Dette er sket efter aftale med vejlederen af specialet.

Testkonfiguration		Ingen udv. ($u = \emptyset$)	CRR ($u = \{\mathrm{c}\}$)	SO ($u = \{\mathrm{s}\}$)	CRR og SO ($u = \{\mathrm{c},\mathrm{s}\}$)
"Slow"	Antal obs. ($n_{\mathrm{s},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{s},u}$.):	98,81	98,81	98,81	98,81
	Standardafvigelse ($s_{\mathrm{s},u}$):	3,15	3,15	3,15	3,15
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{s},u}$):	**[95,90;100,00]**	**[95,90;100,00]**	**[95,90;100,00]**	**[95,90;100,00]**
"Medium"	Antal obs. ($n_{\mathrm{m},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{m},u}$.):	92,26	90,83	92,26	90,83
	Standardafvigelse ($s_{\mathrm{m},u}$):	8,40	7,69	8,40	7,69
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{m},u}$):	**[84,49;100,00]**	**[83,72;97,95]**	**[84,49;100,00]**	**[83,72;97,95]**
"Fast"	Antal obs. ($n_{\mathrm{f},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{f},u}$.):	85,34	82,13	87,38	82,13
	Standardafvigelse ($s_{\mathrm{f},u}$):	20,83	18,83	21,56	18,83
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{f},u}$):	**[66,08;100,00]**	**[64,71;99,55]**	**[67,44;100,00]**	**[64,71;99,55]**

Tabel 5.5: Parameterinferens ud fra måledataene i figur 5.2 på side 116 (procentdel af de afsendte pakker, der nåede frem til den endelige modtager)

5.3 Resultater udtrukket af testsimuleringer

5.3.1 Parameterinferens

Analyserne af tallene startes ved at beregne en række nøgletal for hver af observationsgrupperne. Formlerne til beregning af disse nøgletal præsenteres i appendix G.3. For hver af de tre sæt måledata (vist i histogramform i figur 5.2-5.4 på side 116–118, de rå tal kan ses i appendix F.1) kan de infererede parametre ses i henholdsvis tabel 5.5, 5.6 på næste side og 5.7 på næste side. ("Inferens" betyder "forudsigelser om populationen på basis af observationer").

Fra tabel 5.5, som handler om det antal pakker, der nåede frem til den endelige modtager, kan man udlede følgende:

- Hvis man ser på gennemsnittet af procentdelen af fremkomne pakker over *alle* de mulige testkørsler, der i sammensætning svarer til "slow"-konfigurationen, og hvor de deltagende knuder opfører sig som i denne konfiguration, kan man med 95% konfidens sige, at dette gennemsnit er på mindst 95,90%. Dette gælder uanset om CRR og/eller SO er slået til eller fra. For en enkelt "slow"-kørsel i et enkelt netværk kan procentdelen sagtens være mindre; det ovenstående tal gælder som sagt gennemsnittet over *alle* sådanne kørsler i denne type netværk.

- I "medium"-netværk kan man tilsvarende med 95% konfidens sige, at gennemsnits-pakkefremkomst-procentdelen er mindst 84,49%. Gennemsnittet kan dog gå ned til 83,72%, hvis CRR slåes til, og i dette tilfælde kan man (med den samme 95% konfidens) sige, at gennemsnittet højst vil være 97,95%.

- Tilsvarende kan man i "fast"-netværk med 95% konfidens sige, at gennemsnits-pakkefremkomst-procentdelen ligger mellem 64,71% og 67,44% til 99,55%-100% alt efter hvilke udvidelser, der er slået til.

Testkonfiguration		Ingen udv. ($u = \emptyset$)	CRR ($u = \{\mathrm{c}\}$)	SO ($u = \{\mathrm{s}\}$)	CRR og SO ($u = \{\mathrm{c},\mathrm{s}\}$)
"Slow"	Antal obs. ($n_{\mathrm{s},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{s},u}$.):	5,06	4,41	5,08	4,55
	Standardafvigelse ($s_{\mathrm{s},u}$):	1,03	0,98	0,96	0,98
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{s},u}$):	**[4,10;6,01]**	**[3,50;5,31]**	**[4,20;5,97]**	**[3,64;5,45]**
"Medium"	Antal obs. ($n_{\mathrm{m},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{m},u}$.):	6,11	5,67	6,42	6,05
	Standardafvigelse ($s_{\mathrm{m},u}$):	2,08	2,54	2,70	2,27
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{m},u}$):	**[4,19;8,04]**	**[3,33;8,02]**	**[3,92;8,92]**	**[3,95;8,15]**
"Fast"	Antal obs. ($n_{\mathrm{f},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{f},u}$.):	7,09	6,49	8,80	8,30
	Standardafvigelse ($s_{\mathrm{f},u}$):	2,58	2,12	4,19	3,61
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{f},u}$):	**[4,70;9,48]**	**[4,53;8,45]**	**[4,93;12,68]**	**[4,96;11,63]**

Tabel 5.6: Parameterinferens ud fra måledataene i figur 5.3 på side 117 (gennemsnitligt antal pakketransmissioner i netværket pr. afsendt pakke)

Testkonfiguration		Ingen udv. ($u = \emptyset$)	CRR ($u = \{\mathrm{c}\}$)	SO ($u = \{\mathrm{s}\}$)	CRR og SO ($u = \{\mathrm{c},\mathrm{s}\}$)
"Slow"	Antal obs. ($n_{\mathrm{s},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{s},u}$.):	3,42	2,87	3,47	2,86
	Standardafvigelse ($s_{\mathrm{s},u}$):	1,01	1,03	1,03	0,94
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{s},u}$):	**[2,49;4,35]**	**[1,92;3,82]**	**[2,52;4,42]**	**[2,00;3,73]**
"Medium"	Antal obs. ($n_{\mathrm{m},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{m},u}$.):	3,58	3,01	3,59	3,20
	Standardafvigelse ($s_{\mathrm{m},u}$):	1,53	2,07	1,71	1,73
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{m},u}$):	**[2,16;5,00]**	**[1,10;4,93]**	**[2,01;5,17]**	**[1,60;4,80]**
"Fast"	Antal obs. ($n_{\mathrm{f},u}$):	7	7	7	7
	Obs.middelværdi ($\bar{x}_{\mathrm{f},u}$.):	4,56	3,70	5,64	3,87
	Standardafvigelse ($s_{\mathrm{f},u}$):	2,47	1,82	3,31	1,45
	95% konfidensinterval for middelværdien ($\mu_{\mathrm{f},u}$):	**[2,28;6,84]**	**[2,01;5,39]**	**[2,58;8,71]**	**[2,53;5,21]**

Tabel 5.7: Parameterinferens ud fra måledataene i figur 5.4 på side 118 (gennemsnitligt antal pakketransmissioner før en pakke når frem til modtager for succesfuldt modtagede pakker)

Men hvad kan man så udlede af det? Tallene lader til at lægge op til, at gennemsnits-pakkefremkomst-procentdelen falder jo hurtigere, knuderne bevæger sig rundt imellem hinanden. Dette kan man imidlertid *ikke* udlede af disse tal. At man har et 95% konfidensinterval for hvor middelværdien ligger betyder blot, at "den rigtige" middelværdi med 95% konfidens har en værdi, der ligger i konfidensintervallet for den gruppe, man kigger på. Denne værdi kan f.eks. være 97,2% for alle kombinationer af de tre hastigheder og de fire udvidelseskombinationer, hvorved disse altså ikke vil have nogen betydning for fremkomst-procentdelen – eller den kan være 96% for "slow"-netværk, 97% for "medium"-netværk og 98% for "fast"-netværk, d.v.s. stigende, jo hurtigere knuderne bevæger sig. Hvis man skal udlede noget om, hvorvidt middelværdien falder, når man sætter hastigheden op, skal man i stedet sammenligne grupperne direkte. Dette gør jeg derfor i afsnit 5.3.2.

Det samme gælder, hvis man sammenligner middelværdierne for pakkefremkomst-procentdelen på tværs af, om CRR og SO er slået til eller fra. Denne kunne godt se ud til at falde, når man slår CRR til – men igen kan man *ikke* udlede dette ud fra disse tal, hvorfor jeg i stedet analyserer mere direkte på dette i afsnit 5.3.3 og 5.3.4.

Også i tabel 5.6 på forrige side og 5.7 på forrige side kan 95% konfidensintervaller for middelværdierne over *alle* testkørsler i de enkelte konfigurationsgrupper ses, og igen er man som ved tabel 5.5 på side 119 nødt til at sammenligne grupperne mere direkte for at kunne uddrage konklusioner om hvad knudernes hastighed og hvilke udvidelser, der er slået til i DSR-modellen, gør ved middelværdierne.

Bemærk, at ifølge tabel 5.5 på side 119 ligger middelværdien for f.eks. "slow"-netværk uden udvidelser slået til med 95% konfidens i intervallet [95,9%;100,00%]. I virkeligheden blev konfidensintervallet her udregnet til [95,9%;101,72%], men i praksis kan procentdelen naturligvis ikke gå over 100% pakkefremkomst. Jvf. [SW99, side 213] skal man i disse tilfælde blot indskrænke konfidensintervallerne, så de ikke går over 100%, og dette er netop gjort i tabel 5.5 på side 119.

5.3.2 Påvirkes DSR-modellens effektivitet af knudernes hastighed?

I dette afsnit vil jeg undersøge, om de måledata, der blev fundet i figur 5.2-5.4 på side 116–118, påvirkes af knuders hastighed. Dette sker v.h.a. et antal statistiske formler, der præsenteres i appendix G.4.

Påvirkes pakkefremkomsten af knudernes hastighed?

Som nævnt i afsnit 5.3.1 er det med den gennemgang, der har været indtil videre fuldt ud muligt, at knudernes hastighed ikke har nogen betydning for den rigtige gennemsnitlige fremkomst-procentdel. Jeg vil derfor i dette afsnit undersøge nærmere, om det er sandsynligt, at knudernes hastighed *har* en betydning for den gennemsnitlige fremkomst-procentdel.

Jeg vælger at sammenligne observationerne fra "slow"-konfigurationen og "fast"-konfigurationen. Jeg benytter ikke måledataene fra "medium"-konfigurationen i disse analyser, idet jeg på denne måde forsøger at maksimere muligheden for at få et resultat ved at kigge på yderområderne af parametrene. En mere vidtgående analyse bør naturligvis også sammenligne måledataene fra simulationerne af testkørslerne fra "slow"- og "medium"-konfigurationerne og fra "medium"- og "fast"-konfigurationer. Dette foregår præcist, som når jeg i det efterfølgende sammenligner måledataene fra simulationerne af testkørslerne fra "slow"- og "fast"-konfigurationerne.

I tabel 5.8 på næste side kan 95%-konfidensintervallet for forskellen på "slow"- og "fast"-konfigurationernes middelværdi ses. I alle fire kombinationer af til- og fraslåede udvidelser dækker konfidensintervallet værdien 0. Man kan altså ikke (med 95% konfidens) fastslå, at middelværdien for "fast"-konfigurationen er mindre end middelværdien for "slow"-konfigurationen for hver af de fire udvidelseskombinationer.

Konfidens	Udvidelser	Konfidensinterval for $\mu_f - \mu_s$
95%	Ingen:	[-32,77;5,84]
	CRR:	[-34,14;0,78]
	SO:	[-31,40;8,55]
	CRR og SO:	[-34,14;0,78]
90%	Ingen:	[-28,83;1,90]
	CRR:	**[-30,59;-2,78]**
	SO:	[-27,32;4,47]
	CRR og SO:	**[-30,59;-2,78]**
80%	Ingen:	**[-24,88;-2,06]**
	CRR:	**[-27,01;-6,35]**
	SO:	[-23,23;0,38]
	CRR og SO:	**[-27,01;-6,35]**

Tabel 5.8: 95%, 90% og 80% konfidensintervaller for forskellen på "slow"- og "fast"-konfigurationernes pakkefremkomstmiddelværdi. Bemærk at tallene er i procentpoint.

Hvis man slækker på konfidensen for, at konfidensintervallet dækker den rigtige forskel på de to middelværdier, får man dog et bedre resultat: Med 90% konfidens har man, at konfidensintervallet for middelværdien for de konfigurationer, hvor CRR er slået til, ikke længere indeholder 0. Med 90% konfidens kan man altså fastslå, at middelværdien for fremkomne pakker vil være mindre, når CRR er slået til, og man sætter knudernes hastighed op fra hastigheden i "slow"-konfigurationen til hastigheden i "fast"-konfigurationen (og alle andre parametre i øvrigt er ens i de to konfigurationer).

Sætter man konfidensen yderligere ned til 80%, får man samme resultat for tilfældet, hvor hverken SO eller CRR er slået til.

Dette betyder, at man ud fra dette (med en konfidens på 80%) kan konkludere, at middelværdien for pakkefremkomst-procentdel i DSR-modellen falder for alle kombinationer af inkluderede udvidelser pånær den, hvor kun SO er slået til, hvis knudernes hastighed sættes op fra "slow"- til "fast"-konfigurationens hastighed. Dette giver mening, da antallet af benyttede ruter, der invalideres, må forventes at blive højere, jo hurtigere knuderne bevæger sig – og hermed vil færre pakker i snit nå frem.

Påvirkes pakketransmissionantallet af knudernes hastighed?

I tabel 5.9 på næste side er et antal statistiske formler, der præsenteres i appendix G.4, benyttet på tallene fra figur 5.3 på side 117 for at finde ud af, om middelværdien for antal pakketransmissioner pr. afsendt pakke falder eller stiger, når knudernes hastighed sættes op.

Her kan man med 95% konfidens konkludere, at når CRR er slået til, er middelværdien for pakketransmissionsraten altid højere i "fast"-konfigurationer end i "slow"-konfigurationer. Hvis man sætter konfidensen ned til 90%, gælder denne konklusion alle kombinationer af udvidelser.

Også her giver dette fint mening i forhold til hvad, man kunne forvente: Når knudernes hastighed sættes op, vil ruter oftere blive invalideret, og der skal gennemsnitligt benyttes flere pakketransmissioner til at forsøge at få en bekræftelse for et link, der ikke længere virker, og til at sende "Route Error"-pakker tilbage til de oprindeligt afsendende knuder. Denne tydelige effekt kan hænge sammen med, at der i konfigurationerne kun er 5 knuder, og den maksimale rutelængde (målt i pakketransmissioner) vil derfor være på 5 (inkl. den sidste bekræftelse jvf. afsnit 3.8.2). Når der dertil skal lægges et antal gensendelser som følge af RM, vil det hurtigt være nok til at påvirke det gennemsnitlige antal pakketransmissioner.

Konfidens	Udvidelser	Konfidensinterval for $\mu_f - \mu_s$
95%	Ingen:	[-0,40;4,47]
	CRR:	**[0,07;4,10]**
	SO:	[-0,17;7,61]
	CRR og SO:	**[0,39;7,11]**
90%	Ingen:	**[0,08;4,00]**
	CRR:	**[0,46;3,71]**
	SO:	**[0,62;6,83]**
	CRR og SO:	**[1,06;6,44]**

Tabel 5.9: 95% og 90% konfidensintervaller for forskellen på "slow"- og "fast"-konfigurationernes middelværdi for antal pakketransmissioner pr. afsendt pakke

Konfidens	Udvidelser	Konfidensinterval for $\mu_f - \mu_s$
95%	Ingen:	[-1,19;3,47]
	CRR:	[-0,95;2,61]
	SO:	[-0,91;5,26]
	CRR og SO:	[-0,44;2,45]
90%	Ingen:	[-0,74;3,01]
	CRR:	[-0,61;2,27]
	SO:	[-0,30;4,65]
	CRR og SO:	[-0,18;2,18]
80%	Ingen:	[-0,27;2,55]
	CRR:	[-0,26;1,92]
	SO:	**[0,32;4,02]**
	CRR og SO:	**[0,11;1,89]**

Tabel 5.10: 95% og 90% konfidensintervaller for forskellen på "slow"- og "fast"-konfigurationernes middelværdi for antal pakketransmissioner ved modtagelsen af pakker

Påvirkes pakkefremkomsthastigheden af knudernes hastighed?

Tabel 5.10 viser konfidensintervaller for forskellen i middelværdien på det antal pakketransmissioner, der skal benyttes, før en pakke når frem – baseret på tallene fundet i figur 5.4 på side 118.

Her er man kun i stand til at få et resultat, når man sætter konfidensen ned til 80% – og kun for de tilfælde, hvor udvidelsen SO er slået til i DSR-modellen. Her vil antallet af pakketransmissioner, før pakken når frem til modtageren, stige, når hastigheden sættes op. Dette hænger godt sammen med diskussionen i foregående afsnit, da fejlende ruter (der vil resultere i flere pakketransmissioner p.g.a. RM) netop ikke tæller med i disse tal.

Det betyder i denne forbindelse intet, at tallene allerede i 95% konfidensintervaltilfældet ligger meget tæt på 0 fra den ene side, da man som tidligere beskrevet intet kan sige om hvor i konfidensintervallet, middelværdien kan ligge. Konfidensintervaller er f.eks. ikke normalfordelte. Man kan dog alligevel udtrække *nogen* information af dette: Når hastigheden sættes op, vil middelværdien for antallet af pakketransmissioner, før pakken når frem til modtageren, maksimalt falde med 1,19 (og under 1, hvis en eller flere udvidelser er slået til), hvorimod den kan stige med op til 5,26 (i tilfældet, hvor kun SO-udvidelsen er slået til). D.v.s. når man går fra en "slow"- til en "fast"-konfiguration er det sandt, at middelværdien kan falde – men maksimalt med 1,19 pakketransmission (med en konfidens på 95%).

Konklusioner for knudernes hastighed

Man kan ud fra dette konkludere, at i DSR-modellen har en forhøjet hastighed for knuderne generelt en negativ effekt på både pakkefremkomst-procentdelen, antal pakketransmissioner i alt og antal pakketransmissioner, før pakker når frem, som forventeligt.

Med størst konfidens kan det ses, at det totale pakketransmissionsantal stiger, når knudernes hastighed sættes i vejret, mens effekten er noget mindre på antallet af fremkomne pakker og er relativt svært at måle på antal pakketransmissioner, før pakkerne når frem.

Hvis ruter oftere invalideres som følge af knuders forhøjede hastighed, vil RM sørge for, at der kommer flere gentransmissioner i netværket. Da der er brugt relativt korte ruter i de benyttede testkonfigurationer, vil disse gentransmissioner påvirke pakketransmissionsantallet i relativt høj grad, og dette kan være grunden til, at man med så høj en konfidens kan se et resultat på trods af det relativt lave antal observationer.

For at man med højere konfidens skal kunne konkludere noget om de øvrige egenskaber, bør der dog laves flere testkørsler.

5.3.3 Påvirkes DSR-modellens effektivitet af "Cached Route Reply"?

I dette afsnit vil jeg undersøge om de i figur 5.2-5.4 på side 116–118 fundne måledata påvirkes af hvorvidt, udvidelsen "Cached Route Reply" (CRR) er slået fra eller til i DSR-modellen. Dette sker v.h.a. et antal statistiske formler, der præsenteres i appendix G.5.

Påvirkes pakkefremkomsten af CRR?

Ud fra tabel 5.5 på side 119 kunne det godt se ud, som om gennemsnits-pakkefremkomst-procentdelen falder, hvis man slår udvidelsen CRR til. Dette vil jeg undersøge nærmere i det nedenstående.

I modsætning til analysen i forrige afsnit er de datasæt, der er tale om her, baserede på, at det er de samme kommandolister, der er kørt for hver kombination af udvidelser. Det betyder, at i modsætning til når man sammenligner testobservationerne i f.eks. "slow"- og "fast"-grupperingerne, er testobservationerne på tværs af de benyttede udvidelser i DSR-modellen *afhængige* af hinanden. I appendix G.1 beskrives det, hvordan dette gør, at konfidensintervallet bliver smallere.

Konfidensintervallerne i tabel 5.11 på næste side er fundet ud fra et antal statistiske formler, der præsenteres i appendix G.5.

I "slow"-konfigurationerne er konfidensintervallet altid angivet som "—". I virkeligheden er det i disse tilfælde udregnet til [0;0], men angivelsen "—" skyldes, at antallet af observationer er for lille til, at en forskel kunne ses i de udtrukne måledata fra testkørslerne. Man kan overbevise sig om dette ved at sammenligne kolonne 1 og 2 samt 3 og 4 i figur 5.2 i "slow"-konfigurationerne eller ved at kigge på de tilsvarende indgange i tabel F.1 på side 212 i appendix F.1. Man kan derfor ikke slutte noget om CRR's virkemåde på pakkefremkomst-procentdelen i "slow"-konfigurationer.

I "medium"- og "fast"-konfigurationerne kan det ses, at hvis man kigger på 95%-konfidensintervallet, indeholder de alle 0, d.v.s. at man ikke kan inferere noget om hvorvidt, CRR har en positiv eller negativ effekt på den gennemsnitlige pakkefremkomst-procentdel.

Sætter man konfidensen for, at middelværdien er indeholdt i konfidensintervallet, ned til 90%, får man dog et enkelt tilfælde, hvor man kan se en effekt: Når SO er slået til, indeholder konfidensintervallet i "fast"-konfigurationen ikke længere 0, og man har altså 90% konfidens for, at den rigtige middelværdi for pakkefremkomst-procentdelen altid vil være mindre, når CRR er slået til, end når den er slået fra.

Sættes konfidensen yderligere ned til 80%, får man, at "fast"-konfigurationen – både med og uden SO slået til – ikke længere indeholder 0, d.v.s. at CRR har en negativ effekt på gennemsnits-pakkefremkomst-procentdelen.

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	—
		SO	—
	"Medium"	Ingen	[-4,92;2,07]
		SO	[-4,92;2,07]
	"Fast"	Ingen	[-8,34;1,91]
		SO	[-11,43;0,91]
90%	"Slow"	Ingen	—
		SO	—
	"Medium"	Ingen	[-4,20;1,35]
		SO	[-4,20;1,35]
	"Fast"	Ingen	[-7,28;0,85]
		SO	**[-10,15;-0,36]**
80%	"Slow"	Ingen	—
		SO	—
	"Medium"	Ingen	[-3,49;0,63]
		SO	[-3,49;0,63]
	"Fast"	Ingen	**[-6,23;-0,20]**
		SO	**[-8,89;-1,62]**

Tabel 5.11: 95%, 90% og 80% konfidensintervaller for forskellen på pakkefremkomstmiddelværdien i konfigurationerne med og uden udvidelsen CRR. Bemærk at tallene er i procentpoint.

Dette betyder altså, at for testkonfigurationer, hvor knudernes hastighed er helt i top, har CRR en negativ effekt på pakkefremkomst-procentdelen med 80% konfidens. Dette giver fint mening, for CRR fungerer på den måde, at en knude kan svare på en "Route Request"-pakke med information fra dens "Route Cache" (jvf. afsnit 2.3.6). Når knudernes hastighed er helt i top, er sandsynligheden for, at den svarende knudes "Route Cache" indeholder ukurant information forhøjet (da ruter hurtigere invalideres), og sandsynligheden for, at afsenderen af "Route Request" får forkert information tilbage, forhøjes derfor tilsvarende – hvilket betyder, at pakken kan blive sendt via en invalideret rute og vil gå tabt.

Påvirkes pakketransmissionsantallet af CRR?

I tabel 5.12 på næste side er i stedet måledataene figur 5.3 på side 117 – antal pakketransmissioner pr. afsendt pakke – sammenlignet.

Her har man allerede i 95% konfidenstilfældet et resultat: I "slow"-konfigurationer vil middelværdien med 95% konfidens falde, når man går fra en konfiguration med CRR slået fra, til en konfiguration med CRR slået til.

Men selv hvis man sætter konfidensen ned til 90%, får man ikke et entydigt resultat i samme stil for "medium"- eller "fast"-konfigurationerne, og for 80% får man kun et lignende result i "medium"-konfigurationen hvor SO ikke er slået til.

CRR har altså kun en negativ effekt på pakketransmissionsantallet i "slow"-netværk. Dette kan som før beskrevet skyldes, at CRR i hurtigere netværk vil medføre, at flere invaliderede ruter vil blive benyttet, og RM som beskrevet i afsnit 5.3.2 ud fra dette skal foretage flere gentransmissioner, hvorved en eventuel positiv effekt af CRR ikke længere kan måles ud fra observationerne.

Påvirkes pakkefremkomsthastigheden af CRR?

I tabel 5.13 på næste side gentages de tidligere analyser, denne gang for datasættet fra figur 5.4 på side 118.

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	**[-1,01;-0,29]**
		SO	**[-0,88;-0,19]**
	"Medium"	Ingen	[-1,65;0,77]
		SO	[-1,05;0,31]
	"Fast"	Ingen	[-1,61;0,41]
		SO	[-2,02;1,01]
90%	"Slow"	Ingen	**[-0,94;-0,36]**
		SO	**[-0,81;-0,26]**
	"Medium"	Ingen	[-1,40;0,52]
		SO	[-0,91;0,17]
	"Fast"	Ingen	[-1,40;0,20]
		SO	[-1,71;0,69]
80%	"Slow"	Ingen	**[-0,86;-0,44]**
		SO	**[-0,74;-0,33]**
	"Medium"	Ingen	[-1,15;0,27]
		SO	[-0,77;0,03]
	"Fast"	Ingen	**[-1,20;-0,01]**
		SO	[-1,40;0,38]

Tabel 5.12: 95%, 90% og 80% konfidensintervaller for forskellen på middelværdien for antal pakketransmissioner pr. afsendt pakke i konfigurationerne med og uden udvidelsen CRR

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	**[-0,87;-0,24]**
		SO	**[-0,76;-0,46]**
	"Medium"	Ingen	[-1,50;0,37]
		SO	**[-0,68;-0,10]**
	"Fast"	Ingen	[-2,13;0,40]
		SO	[-4,78;1,23]
90%	"Slow"	Ingen	**[-0,80;-0,30]**
		SO	**[-0,73;-0,49]**
	"Medium"	Ingen	[-1,31;0,18]
		SO	**[-0,62;-0,16]**
	"Fast"	Ingen	[-1,87;0,14]
		SO	[-4,16;0,61]
80%	"Slow"	Ingen	**[-0,74;-0,37]**
		SO	**[-0,70;-0,52]**
	"Medium"	Ingen	**[-1,11;-0,01]**
		SO	**[-0,56;-0,22]**
	"Fast"	Ingen	**[-1,61;-0,12]**
		SO	**[-3,55;-0,01]**

Tabel 5.13: 95%, 90% og 80% konfidensintervaller for forskellen på middelværdien for antal pakketransmissioner ved modtagelsen af pakker i konfigurationerne med og uden udvidelsen CRR

Her kan man se, at allerede i 95% konfidenstilfældet for "slow"-konfigurationen er middelværdiforskellen for hvor mange pakketransmissioner, der skal benyttes, før pakkerne når frem til den endelige modtagerknude, under 0, når man sammenligner konfigurationer med CRR slået fra og CRR slået til – bemærk, at der her altså kun kigges på pakketransmissionsantallet for de pakker, der når frem. Sættes konfidensen ned til 80%, får man, at middelværdiforskellen er under 0 for alle konfigurationskombinationer.

CRR vil dermed med 80% konfidens have en positiv effekt på hastigheden (målt i antal pakketransmissioner jvf. afsnit 5.2.2) for middelværdien for hvor hurtigt, pakker når frem til modtageren.

Konklusioner for CRR

Hvis man sammenholder resultaterne fra pakkefremkomst-procentdelen, det totale pakketransmissionsantal og pakketransmissionantallet, inden pakkerne når frem, bliver brugen af CRR en afvejning af, om man gerne vil have et gennemsnitligt mindre pakketransmissionsforbrug i "slow"-type netværk og generelt færre pakketransmissioner, før pakkerne når frem (i alle typer netværk), eller man gerne vil undgå, at gennemsnitligt færre pakker når frem i "fast"-type netværk.

Bemærk i denne forbindelse, at jvf. afsnit 5.1.3 må "slow"-konfigurationen vurderes til at være tættest på "virkelige" trådløse netværk af de tre mulige hastighedsindstillinger, så ud fra dette vil jeg konkludere, at det vil være gavnligt at inkludere CRR i en generel implementation af DSR.

5.3.4 Påvirkes DSR-modellens effektivitet af "Salvage Operations"?

I dette afsnit vil jeg gerne undersøge, om de måledata, der blev fundet i figur 5.2-5.4 på side 116–118 påvirkes af, hvorvidt udvidelsen "Salvage Operations" (SO) er slået fra eller til i DSR-modellen. Dette sker v.h.a. et antal statistiske formler, der præsenteres i appendix G.5.

Påvirkes pakkefremkomsten af SO?

Hvis man kigger på tabel 5.5 på side 119, kunne det godt se ud, som om SO-udvidelsen ikke har nogen nævneværdig effekt på gennemsnits-pakkefremkomst-procentdelen. I histogrammerne i figur 5.2 på side 116 kan man kun se en forskel på figur 5.2(i) og 5.2(k) – de øvrige histogrammer er ens, når man går fra en konfiguration uden SO til en tilsvarende konfiguration med SO. Som i tabel 5.11 på side 125 i afsnit 5.3.3 betyder dette, at resten af konfidensintervallerne i tabel 5.14 på næste side er angivet til "—" – i virkeligheden er de udregnet til [0;0].

Jeg vil her undersøge, om man ud fra de fundne observationsværdier alligevel kan drage en konklusion om, at pakkefremkomst-procentdelen påvirkes den ene eller den anden vej. Dette gør jeg v.h.a. samme metode som i forrige afsnit, hvilket giver konfidensintervallerne vist i tabel 5.14 på næste side.

Her kan man se, at hverken med 95%, 90% eller 80% konfidens kan man inferere, at SO har en effekt på pakkefremkomst-procentdelens middelværdi.

At man ikke kan inferere noget om pakkefremkomst-procentdelen her, kan skyldes, at SO kun fungerer i relativt specifikke tilfælde: Som beskrevet i afsnit 2.4.6 vil udvidelsen kun blive brugt, hvis en knude, der skal til at videresende en pakke på dennes vej til den endelige modtagerknude, opdager, at linket til naboknuden (angivet i source-ruten i DSR-optionsheaderen) ikke fungerer, men knuden samtidigt kender en anden rute til den endelige modtagerknude. Da der kun blev udsendt gennemsnitligt 10 pakker i hver testkørsel i afsnit 5.2.2, er det sandsynligt, at denne situation ikke når at opstå – og tabel 5.14 på næste side siger da netop også, at hvis situationen opstår, opstår den ikke tit nok til, at man kan uddrage nogle konklusioner om den.

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	—
		CRR	—
	"Medium"	Ingen	—
		CRR	—
	"Fast"	Ingen	[-11,17;15,25]
		CRR	—
90%	"Slow"	Ingen	—
		CRR	—
	"Medium"	Ingen	—
		CRR	—
	"Fast"	Ingen	[-8,45;12,53]
		CRR	—
80%	"Slow"	Ingen	—
		CRR	—
	"Medium"	Ingen	—
		CRR	—
	"Fast"	Ingen	[-5,73;9,82]
		CRR	—

Tabel 5.14: 95%, 90% og 80% konfidensintervaller for forskellen på pakkefremkomstenmiddelværdien i konfigurationerne med og uden udvidelsen SO. Bemærk at tallene er i procentpoint.

Påvirkes pakketransmissionsantallet af SO?

Heller ikke hvad angår en forskel i middelværdierne på pakketransmissionsantallet (som vist i tabel 5.15 på næste side) kan man udtale sig om en positiv eller negativ effekt på pakketransmissionsantallet – og dette gælder uanset, om man benytter 95%, 90% eller 80% konfidens.

Påvirkes pakkefremkomsthastigheden af SO?

Hvis man kigger på forskellen i middelværdierne på antal pakketransmissioner før pakkerne når frem (som vist i tabel 5.16 på næste side), får man samme resultat som før: Man kan hverken udtale sig om en positiv eller negativ effekt på pakketransmissionsantallet – og dette gælder både, hvis man kigger på intervallerne, man får, når man benytter 95%, 90% og 80% konfidens.

Konklusioner for SO

Når man kigger på, hvad der sker med middelværdierne, når man går fra testkonfigurationer, hvor SO er slået fra, til testkonfigurationer, hvor den er slået til, er jeg ikke i stand til at måle en hverken tydeligt positiv eller negativ effekt i nogen af måledataene.

Dette kan skyldes flere forskellige ting: Det kan skyldes, at antallet af afsendte pakker i hver testkørsel, der af tidsmæssige årsager blev sat lavt, er sat så lavt, at SO ikke når at blive brugt ofte nok til at kunne give et målbart resultat for de datasæt, jeg har valgt at kigge på. Det kan også skyldes, at antallet af knuder i hver simulering er for lavt – d.v.s., at en mere målbar effekt først nås med flere knuder. En tredie årsag kan være, at indførelsen af "Mini-Salvage Operations" (Mini-SO, afsnit 3.5.5) fjerner for meget af SO's effekt. Endeligt kan det også skyldes, at SO-udvidelsen vitterligt ikke *har* en effekt. Om SO har en effekt, kan derfor kun med tilstrækkelig konfidens konkluderes, hvis man indsamler nye observationer – denne gang flere og for længere kommandolister, med et varierende antal knuder, og hvor Mini-SO bliver slået til og fra sammen med SO selv (hvilket vil kræve en udvidelse af modellen). Dette vil

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	[-0,24;0,29]
		CRR	[-0,36;0,64]
	"Medium"	Ingen	[-2,26;2,87]
		CRR	[-1,26;2,00]
	"Fast"	Ingen	[-6,07;9,50]
		CRR	[-5,22;8,83]
90%	"Slow"	Ingen	[-0,18;0,23]
		CRR	[-0,26;0,54]
	"Medium"	Ingen	[-1,73;2,34]
		CRR	[-0,92;1,67]
	"Fast"	Ingen	[-4,47;7,89]
		CRR	[-3,77;7,38]
80%	"Slow"	Ingen	[-0,13;0,18]
		CRR	[-0,15;0,43]
	"Medium"	Ingen	[-1,20;1,82]
		CRR	[-0,59;1,33]
	"Fast"	Ingen	[-2,87;6,29]
		CRR	[-2,33;5,94]

Tabel 5.15: 95%, 90% og 80% konfidensintervaller for forskellen på middelværdien for antal pakketransmissioner pr. afsendt pakke i konfigurationerne med og uden udvidelsen SO

Konfidens	Hastighed	Øvrige udv.	Konfidensinterval
95%	"Slow"	Ingen	[-0,45;0,55]
		CRR	[-0,32;0,31]
	"Medium"	Ingen	[-1,58;1,60]
		CRR	[-1,06;1,43]
	"Fast"	Ingen	[-6,23;8,40]
		CRR	[-1,40;1,74]
90%	"Slow"	Ingen	[-0,35;0,45]
		CRR	[-0,26;0,25]
	"Medium"	Ingen	[-1,25;1,28]
		CRR	[-0,80;1,18]
	"Fast"	Ingen	[-4,72;6,89]
		CRR	[-1,08;1,42]
80%	"Slow"	Ingen	[-0,24;0,35]
		CRR	[-0,19;0,18]
	"Medium"	Ingen	[-0,92;0,95]
		CRR	[-0,55;0,92]
	"Fast"	Ingen	[-3,22;5,39]
		CRR	[-0,76;1,09]

Tabel 5.16: 95%, 90% og 80% konfidensintervaller for forskellen på middelværdien for antal pakketransmissioner ved modtagelsen af pakker i konfigurationerne med og uden udvidelsen SO

muligvis kunne give et mere klart resultat for SO, men jeg har af tidsmæssige årsager ikke kunnet gøre dette i dette speciale.

Selvom jeg ikke finder et klart resultat for SO (som jeg gjorde for CRR), er det værd at fremhæve, at jeg i forbindelse med undersøgelsen af de to udvidelser har vist, hvordan man kan benytte statistiske metoder til at undersøge udvidelser generelt. Dette kan benyttes til at undersøge også de øvrige DSR-udvidelser under forskellige konfigurationer (ikke kun variation af knudernes hastighed jvf. afsnit 5.1.1). På samme måde kan man også undersøge, om indførelsen af Mini-SO (se afsnit 3.5.5) vil have en gavnlig effekt på DSR-protokollens virkemåde.

5.4 Relateret arbejde

I forbindelse med andres arbejde med modellering af MANET-protokoller er der lavet adskillige andre simuleringer. I de foregående afsnit er der allerede refereret adskillige gange til [KCC05], der giver et overblik over en række af disse. Jeg har derfor i de foregående afsnit sammenlignet mine simuleringer af DSR-CPN-modellen med disse simuleringer.

Her blev det konstateret, at antallet af knuder i simuleringerne i dette kapitel er lavt (5 mod mellem 10 og 30000, se afsnit 5.1.2), mens DSR-CPN-modellen ligger cirka it midten i forhold til de andre simuleringer, hvis man sammenligner på forholdet mellem antennerækkevidde og områdets størrelse samt m.h.t. knudernes tæthed (se afsnit 5.1.3).

I retrospekt var det for lidt at køre simuleringerne med kun 5 knuder. Bedre resultater (specielt med hensyn til undersøgelserne af SO) ville muligvis have kunnet nås, hvis dette var blevet sat op til 6 eller 7. Da simuleringstiden er proportional med antallet af knuder jvf. afsnit 5.1.2, ville dette have været praktiskt muligt. Et bedre resultat ville sandsynligvis også have kunnet nås med *længere* testkørsler (altså flere pakketransmissioner pr. simulering), men dette ville have givet en eksponentiel tidsforøgelse og ville ikke i praksis kunnet have lade sig gøre. Endelig er det muligt, at *flere* simuleringer ville have indskrænket konfidensintervallerne så meget, at flere konklusioner kunne drages med en høj nok konfidens. Simuleringer, hvor der samtidigt med SO også varieres på, hvorvidt Mini-SO (der kan ses som en delmængde af SO, men ikke fungerer sådan i modellen) er slået til eller fra, vil muligvis også give et bedre resultat for SO.

I [Dem01] er der også foretaget en undersøgelse af DSR baseret på simuleringer. Her undersøges forskellen på den benyttede rutelængde og den optimale rutelængde, antallet af RD-pakker og antallet af tabte pakker under forskellige modelkonfigurationer. Forskellen på den benyttede rutelængde og den optimale rutelængde kan ses som en variation over pakkefremkomsthastigheden, og antallet af tabte pakker kan ses som en variation over pakkefremkomst-procentdelen – begge parametre blev undersøgt i dette kapitel.

Simuleringerne udførtes i [Dem01] med variation af både antallet af knuder i det simulerede netværk, antennerækkevidden og knudernes hastighed. Til sammenligning sammenlignede jeg kun på knudernes hastighed i dette kapitel (jvf. afsnit 5.1.3). I [Dem01] udførtes der dog kun en enkelt simulering pr. testkonfiguration, hvor jeg udførte et antal simuleringer for at få et passende statistisk materiale. Til gengæld afsendtes der i løbet af en enkelt simulering i [Dem01] 10000 pakker, hvor der typisk kun blev sendt 10 i hver simulering i dette kapitel. Det vil sige, at et udsving i, hvordan en enkelt pakke opfører sig i en simulering, ikke vil påvirke resultatet af den simulering i nær så høj grad, som det gør i dette kapitel. Alle konklusionerne i artiklen er draget ud fra plottede måledata i stedet for ud fra statistiske metoder.

De simulerede knuder i [Dem01] bevæger sig efter en model, der svarer meget præcist til den, der er beskrevet i afsnit 3.6.1 (altså en modificeret "Boundless Simulation Area Mobility Model"). Knuderne kan dog kun bevæge i en af de fire retninger: Stik nord, stik syd, stik øst og stik vest, og de bevæger sig altid med præcist samme hastighed. Her understøtter modellen i dette speciale en noget mere nuanceret knudemobilitet m.h.t. både retning og hastighed. Det

benyttede modelsprog i [Dem01] angives ikke.

Artiklen finder, at når man sætter knudernes hastighed i modellen op, fås resultater, der er i tråd med dem, jeg finder i afsnit 5.3.2 – protokollen klarer sig minimalt dårligere hvad angår forskellen på benyttede rutelængder og optimale rutelængder. Dette forværres også (som man kunne forvente), når antallet af knuder eller knudernes antennerækkevidde sættes ned (begge uden at områdets størrelse samtidigt påvirkes).

Kigger man i stedet på antallet af benyttede RD-pakker, finder artiklen også egenskaber, man kunne forvente: Antallet falder, jo flere knuder, der er i netværket og jo større, antennerækkevidderne er. Også antallet af tabte pakker opfører sig som forventet: Antallet falder, når man sætter antallet af knuder op eller sætter antennerækkevidden op.

Hvad angår antallet af benyttede RD-pakker og af tabte pakker, kan en entydig effekt ikke på samme måde ses af en ændring i knudernes hastighed, hvilket artiklen benytter til at konkludere, at DSR-protokollen er god til at justere sig til knudernes hastighed.

Arbejdet i dette speciale og [Dem01] komplementerer hinanden. I begge er der lavet simuleringer ud fra hver sin model (bygget uafhængigt af hinanden), og i begge er der undersøgt beslægtede, men ikke helt de samme egenskaber for DSR-modellernes funktionalitet. Der, hvor der har været overlap mellem undersøgelserne, viser specialet og [Dem01] resultater, der er i tråd med hinanden. Dette styrker den tiltro, man kan have til, at de byggede modeller opfører sig ens og dermed med øget sandsynlighed korrekt, hvilket styrker den tiltro, man kan have til de fundne resultater – både hvor de overlapper, og hvor de ikke gør. Men det er naturligvis samtidigt klart, at dette ikke kan tages som et bevis for, at modellerne opfører sig korrekt.

Effekten af valgfrie DSR-udvidelser på [Dem01]'s model undersøges ikke. Jeg har ikke været i stand til at finde en artikel, der undersøger effekten af de i [JMH05] foreslåede DSR-udvidelser.

M.h.t. brugen af statistiske metoder siger [KCC05], at det er farligt at tage et enkelt sæt simuleringsresultater og præsentere disse som "sandheden". For at højne troværdigheden bør man udføre et antal simuleringer, udtrække måledata fra disse og finde konfidensintervaller for hvor man kan regne med, at de "sande" resultater befinder sig. Denne metode er netop benyttet i dette kapitel, men det er relativt unormalt at gøre – i de undersøgte artikler i [KCC05] blev det kun gjort i 14 ud af 112 tilfælde.

6 Konklusion

I dette speciale har jeg undersøgt de såkaldt MANET'er – mobile ad hoc-netværk. Jeg har præsenteret området generelt og har brugt dette som baggrund for at fokusere på en enkelt protokol: Dynamic Source Routing [JM96, JMB01, JMH05]. Denne protokol er undersøgt inden for to områder: Specifikationskvaliteten (mere præcist identifikation af eventuelle problemområder i protokollen) og effektiviteten af foreslåede, valgfrie optimeringer til protokollen.

6.1 Specifikationskvalitet

Jeg har identificeret en række problemområder i protokollen, hvilket gør, at protokollen endnu ikke kan anses som produktionsklar jvf. indledningen. Identifikationen er sket v.h.a. en generel gennemgang af protokollen og en modellering af protokollen i Coloured Petri Nets [Jen92, Jen94]. Gennemgangen af problemområderne svinger fra præcisering af en detalje i den officielle specifikation [JMH05] over påvisning af specialtilfælde, der ikke bliver dækket i specifikationen, til identifikation af områder, hvor datastrukturerne beskrevet i protokollen skal udvides.

Præcisering af den officielle specifikation

I afsnit 3.4.4 præciserede jeg, hvordan piggybacking til netværkspakker skal foregå, hvis DSR-protokollen skal fungere korrekt. Når DSR-optionsheaders skal piggybackes til en netværkspakke, er det nødvendigt, at de bliver indsat *før* de øvrige optionsheaders i netværkspakken. Dette skyldes, at optionsheaders skal behandles i den rækkefølge, de står i netværkspakken, og de piggybackede optionsheaders bliver netop ofte piggybacket, fordi de skal modificere, hvordan en modtagende netværksdeltager skal behandle de øvrige optionsheaders i netværkspakken.

Fejlretning af den officielle specifikation

I afsnit 3.4.5 beskriver jeg behandlingen af videresendelse af "Route Request"-pakker i et "Bi-dir-only"-netværk (defineret i afsnit 2.2.5) for "Route Request"-pakker, der er modtaget fra en nabo, der i knudens "Blacklist"-tabel er markeret som "unidirectionality questionable". Før en sådan videresendelse skal foregå, skal knuden undersøge, om linket mellem naboknuden og

knuden selv er bidirektionelt. Denne undersøgelse tager ikke hensyn til specialtilfældet, hvor en naboknude har implementeret "Cached Route Reply" (CRR) og dermed kan svare på en "Route Request"-pakke, selvom den ikke er målet for pakken. Dette kan gøre, at den undersøgende knude uforvarende kan komme til at benytte unidirektionelle eller ikke-eksisterende links i ruter, protokollen senere går ud fra kun indeholder bidirektionelle links. Jeg præsenterer et forslag til udbedring af problemet.

Andre problemområder i den officielle specifikation

I afsnit 3.5.4 beskrives det, hvordan det i specifikationen kræves, at modtagne "Route Error"-pakker piggybackes til fremtidige "Route Request"-pakker. Da modtagelsen af "Route Error"-pakker og udsendelsen af "Route Request"-pakker kan foregå på forskellige tidspunkter, og der ikke er defineret en datastruktur til at gemme "Route Error"-pakker i specifikationen [JMH05], vil det derfor ikke altid være muligt at foretage denne piggybacking. Fejlen vil kunne udbedres ved at specificere denne datastruktur.

I afsnit 3.4.6 beskriver jeg, hvordan man i specifikationen [JMH05] ikke tager hensyn til et specialtilfælde, hvor to knuder samtidigt udsender "Route Discovery"-pakker gående på hinanden i et netværk af typen "Frequently-unidir" (defineret i afsnit 2.2.5). Ud fra de datastrukturer, der præsenteres i specifikationen, vil man være nødsaget til at vælge enten en løsning, der strider imod de generelle retningslinier, der i øvrigt gives i specifikationen, eller en løsning, som jeg demonstrerer kan resultere i en deadlock: De to knuder kan broadcaste et højt antal "Route Request"-pakker ud i netværket med hinanden som destination uden nogensinde at få et svar tilbage, selvom der måtte findes en eller flere ruter mellem de to knuder i netværket. Jeg skitserer desuden en mulig løsning på problemet, der netop inkluderer, at de benyttede datastrukturer i protokollen skal udvides.

Forslag til optimering af den officielle specifikation

Når en knude opdager, at en pakke ikke kan videresendes til en naboknude, den ifølge rutelisten skulle have videresendt den til, skal en "Route Error"-pakke genereres og tilbagesendes til pakkens oprindeligt afsendende knude. Hvis udvidelsen "Salvage Operations" (SO) er inkluderet i implementationen af protokollen, og knuden har en alternativ rute til den endelige modtagerknude, kan knuden vælge at sende pakken videre ad denne rute i stedet – med "salvage_counter"-feltet talt op.

Specialtilfældet, hvor knuden, der opdager et fejlende link til en anden knude, er den oprindeligt afsendende knude selv, er dog ikke behandlet af specifikationen [JMH05]. I afsnit 3.5.5 beskriver jeg indførelsen af "Mini-Salvage Operations" (Mini-SO), der dækker netop dette tilfælde.

Mini-SO minimerer antallet af gange, protokollen unødvendigt smider pakker væk, når der har været mobilitet i netværket. Den afviger fra SO på to områder: Den afsendende knude må foretage en "Route Discovery" hvis nødvendigt, og "salvage_counter" tælles ikke op (så nye rutelister kan blive høstet af knuder, der videresender pakken).

Fremtidigt arbejde

Fokus for dette speciale har bl.a. været at undersøge kvaliteten af DSR-protokollen, f.eks. ved at påvise eksistensen af problemområder deri. Udover at gøre dette, har jeg til en vis grad også argumenteret for konsekvenserne af disse fejl og mangler, og er som beskrevet ovenfor enten kommet med forslag til løsninger på dem eller har skitseret sådanne. En del af konsekvensundersøgelserne og løsningerne mangler dog stadig, og et logisk sted at starte det fortsatte arbejde vil derfor være at analysere fejlene og manglerne nærmere og finde sådanne løsninger på dem.

Det skal her bemærkes, at selvom modellen af protokollen afspejler en fuldt fungerende basis-protokol med to udvidelser (som beskrevet i næste afsnit), så er der stadig visse begrænsninger i hvad, der er medtaget i modelleringen af protokollen. Specielt kan modelleringen af de omkringliggende OSI-lags funktionalitet (som beskrevet i afsnit 3.6) ikke give en MAC-bekræftelse tilbage til DSR-protokollen i "Bidir-only"-netværk. I stedet valgte jeg at modellere brugen af "eksplicitte bekræftelser", da dette afspejler den største kompleksitet i DSR-laget af de to muligheder. Enkelte detaljer i modelleringen af de omkringliggende OSI-lags funktionalitet gør, at modellen heller ikke i stand til at simulere "Frequently-unidir"-netværk, og det er derfor kun funktionaliteten i "Mostly-bidir"-netværk, der er fuldt modelleret.

Modellen bør laves helt færdig, så den også fungerer helt, når den er konfigureret som en af de manglende netværkstyper. Dette vil gøre, at der under den resterende modellering kan blive afsløret yderligere problemområder i protokollen.

6.2 Valgfrie optimeringer

I den officielle DSR-specifikation [JMH05] er en række af protokollens elementer valgfrie. I min gennemgang har jeg separeret disse fra den normale gennemgang og har præsenteret dem samlet i afsnit 2.3.6 og 2.4.6.

Jeg har herefter valgt at fokusere på to udvidelser: "Route Discovery"-udvidelsen "Cached Route Reply" (CRR) og "Route Maintenance"-udvidelsen "Salvage Operations" (SO). Disse er modelleret i CPN-modellen og er benyttet i en række simuleringer. Simuleringerne er udført, hvor en enkelt parameter varieres: De modellerede netværksdeltagernes hastighed. Det var ikke muligt at gennemføre simuleringer, hvor flere parametre varieres, men jeg argumenterer for, at det er fornuftigt at udvælge netop netværksdeltagernes hastighed som parameter, jeg kan variere på, omend det er klart, at dette nedsætter pålideligheden af resultaterne.

Ud fra simuleringerne er diverse måledata udtrukket, og v.h.a. statistiske analyser bliver disse benyttet til at drage konklusioner om modellens egenskaber. For at få måledata nok, bliver i alt 84 simuleringer udført – 7 i hver kombination af netværksdeltagernes hastighed og benyttede optimeringer.

Analyser på de udtrukkede måledata gør mig i stand til at vise en positiv effekt af CRR, når de modellerede netværksdeltagere bevæger sig med en hastighed, der ikke er *meget* høj. Til gengæld er jeg ikke i stand til at vise en effekt af SO. Dette kan enten skyldes, at udvidelsen ikke vil have en effekt på protokollens effektivitet, eller det kan skyldes, at mine simuleringer har været for korte eller for få, til at effekten vil skinne igennem i de statistiske analyser. Indførelsen af Mini-SO kan desuden fjerne en del af effekten af SO, og endelig kan det naturligvis skyldes, at der kan være fejl i modellen i forhold til protokollen. Tests og demonstrationer (f.eks. i afsnit 3.8.4) har dog ikke været i stand til at afsløre sådanne eventuelle fejl.

Fremtidigt arbejde

Som beskrevet ovenfor var jeg ikke i stand til at afgøre, om der er en effekt ved SO eller ej. Det er derfor et åbenlyst sted at starte et videre arbejde med at gennemføre flere og længere simuleringer som beskrevet i afsnit 5.3.4.

For at styrke pålideligheden af resultaterne, bør CRR og SO også begge analyseres under andre konfigurationer af netværket end blot "Mostly-bidir", hvilket kræver, at modellen bliver modelleret færdig som beskrevet i forrige afsnit. Samtidigt bør modellen analyseres, hvor andre parametre end blot netværksdeltagernes hastighed varieres som beskrevet i afsnit 5.1 – f.eks. med variation i antallet af netværksdeltagere, hvilket i høj grad vil kunne styrke den tiltro, man kan have til de fundne resultater.

I afsnit 2.3.6 og 2.4.6 er en række andre valgfrie udvidelser nævnt ud over dem, jeg har valgt at fokusere på i dette speciale. Også disse bør analyseres på samme måde, som jeg har

gjort det for CRR og SO for at undersøge, om udbyttet af disse udvidelser er positivt, og man bør inkludere dem i en fremtidig implementation. Dette inkluderer også andet end valgfrie udvidelser, f.eks. er det i DSR tilladt selv at bestemme hvilken strategi, man vil benytte for at udvælge en "bedste" rute, når man har flere ruter at vælge imellem til en destination (som beskrevet i afsnit 2.2.4). En gennemgang af de mulige strategier, der findes, og en sammenlignende analyse af disse bør derfor foretages.

I afsnit 3.5.5 præsenterede jeg Mini-SO. Mini-SO kan benyttes i et specialtilfælde i protokollen til at sørge for, at knuder ikke unødvendigt smider pakker, de skal route til en anden knude, væk. En måling af den præcise effekt af denne dækning af specialtilfældet har dog ikke været med i fokus for dette speciale, og bør derfor undersøges nærmere.

Arbejdet i kapitel 5 kan ses som en form for "proof-of-concept" for undersøgelser af protokoludvidelser. De samme metoder kan benyttes til lave de ovenfor nævnte undersøgelser af de øvrige DSR-udvidelser og af Mini-SO, så en del af forarbejdet for disse er hermed allerede gjort.

A

Præsentationer af MANET-protokoller af forskellige typer

A.1 Optimized Link State Routing Protocol (en "link state", "proaktiv" protokol – afsnit 1.3.3)

Et eksempel på en "Link state", "Proaktiv" protokol er "Optimized Link State Routing Protocol" (OLSR), som beskrives i bl.a. [JMC+01, CJ03].

I sin rene form skal alle knuder i et "link state", "proaktivt" MANET på en eller anden måde udveksle information om alle de naboer, de har, til alle øvrige knuder i netværket med jævne mellemrum, så alle knuderne kan danne sig en komplet oversigt over hele netværkstopologien. Når en knude skal sende en pakke til en anden knude, kan pakken herefter altid følge den optimale rute (f.eks. målt i antal videresendelser) i netværket til destinationsknuden.

At propagere denne information om hver knudes naboer ud til alle andre knuder i netværket vil imidlertid give meget kontroltrafik i netværket. En grundlæggende idé i OLSR er, at meget af den kontroltrafik, der bliver sendt rundt i et "link state"-netværk, er redundant, og man kan derfor spare på denne trafik. Dette gøres ved, at det kun udvalgte knuder, der genererer og videresender information om link status ("link state") om de naboer, den har. Disse udvalgte knuder kaldes MPR'er ("Multipoint Relays"). Resten af knuderne skal hverken generere eller videresende kontrolpakker med "link state"-pakker, men de vil stadig modtage al information, der bliver broadcastet rundt i netværket.

En knude kan blive valgt som MPR af et antal af sine naboknuder via en distribueret udvælgelsesproces. Udvælgelsesprocessen sørger bl.a. for, at hver knude er nabo til et sæt MPR'er, der tilsammen kan nå alle de knuder, der er op til 2 hop (d.v.s. 2 netværksvideresendelser) væk fra knuden selv.

En MPR holder styr på hvilke knuder, der har den som MPR, og distribuerer denne "link state"-information til de øvrige MPR'er i netværket. Den ovenstående udvælgelse af MPR'er betyder, at selvom kun MPR'er broadcaster information videre i netværket, overhøres disse broadcastede pakker også af alle andre knuder i netværket.

Når en pakke skal sendes fra en knude til en anden knude i netværket, har afsenderknuden på grund af dette information nok til at beregne hele ruten til destinationsknuden. På trods af dette har man valgt at denne rute ikke gemmes i pakken – kun destinationsknudens adresse

gemmes, før pakken sendes til den første videresendende knude. Denne (og de efterfølgende videresendende knuder) slår op i deres interne tabeller for at finde ud af hvilken knude, de skal sende pakken videre til. Dette sker for at sikre, at pakker når frem, selvom netværkstopologien ændrer sig, mens pakken er i transit.

Man har altså fravalgt den normale praksis med, at løkker automatisk bliver modvirket i "link state"-protokoller (som beskrevet i afsnit 1.3.1). Den kontinuerlige udsendelse af "link state"-information gør dog, at knudernes interne tabeller gøres konsistente med hinanden, og dette modvirker, at eventuelle løkker bliver uendelige.

A.2 Destination-Sequenced Distance Vector (en "distance vector", "proaktiv" protokol – afsnit 1.3.3)

Et eksempel på en "Distance vector", "Proaktiv" protokol er "Destination-Sequenced Distance Vector" (DSDV), som beskrives i bl.a. [PB94, PB01].

I DSDV gemmer hver knude information om tilgængelige destinationsknuder i netværket. Knyttet til informationen om hver destination er information om hvilke naboknuder, der kan bruges som videresendende knude til destinationen, og hvor mange hop, der er fra denne naboknude til destinationsknuden – det vil sige hvor mange knuder, der skal foretage videresendelser mellem naboknuden og destinationsknuden, for at pakken kan nå frem.

Når en knude vil sende en pakke til en destinationsknude, kan den blot slå op i sine interne tabeller og finde ud af, hvilken naboknude, der har det mindste antal hop til destinationsknuden, og sende pakken til denne. Knuden skal ikke bekymre sig om resten af ruten. Når naboknuden modtager pakken og skal finde ud af, hvad den skal gøre med den, gør den blot det samme: Den slår op i sine interne tabeller og videresender pakken til den naboknude, der har det mindste antal hop til destinationsknuden. Dette gentager sig, indtil pakken når frem til destinationsknuden.

For at undgå, at informationen i to knuder i netværket bliver inkonsistent med hinanden (så de f.eks. begynder at pege på hinanden som den naboknude, der har det mindste antal hop til en destinationsknude, og der dermed kan opstå en løkke i routingen af en pakke), udveksler knuderne i et netværk periodisk information om indholdet af deres interne tabeller.

Dette foregår ved, at hver knude broadcaster en speciel kontrolpakke med jævne mellemrum. Pakken indeholder et sekvensnummer, der er unikt for knuden. Når en knude modtager pakken (enten direkte fra afsenderknuden eller videresendt fra en anden knude), kigger den på, om sekvensnummeret er større end det største sekvensnummer, den indtil videre har modtaget for den afsendende knude. Hvis det er det, opdateres knudens interne tabeller med informationer, der trækkes ud af pakken; mere præcist sættes den naboknude, knuden modtog pakken fra, som en "next hop"-knude for den afsendende knude. Denne information tilknyttes antallet af gange, pakken blev videresendt, inden den blev modtaget af knuden (d.v.s. hvor mange hop, der nu vil være fra naboknuden til den afsendende knude).

For at spare på kontroltrafikken, broadcastes hver enkelt kontrolpakke i praksis ikke ud enkeltvist fra knude til knude. I stedet sender hver knude med jævne mellemrum enten forskellen på dens interne tabeller fra sin sidste udsendelse eller et "fuldt dump" af alle sine interne tabeller ud til sine naboer. Det sidste kan f.eks. være nødvendigt, hvis knuden har fået en ny nabo, der skal opdateres m.h.t. hvilke øvrige naboknuder, knuden har.

Hvis en knude opdager, at et link til en naboknude ikke længere fungerer (enten fordi en OSI-lag 2-procedure giver en fejl tilbage, når knuden forsøger at sende en pakke til naboknuden, se afsnit 1.5.3, eller fordi den ikke har hørt et broadcast fra naboknuden i et stykke tid), sender knuden en kontrolpakke ud for de destinationer, der benytter denne knude som "next hop"-knude med et sekvensnummer sat til det sidst modtagne sekvensnummer plus en, og hvor antallet af hop sættes til "uendeligt". Knuder, der modtager sådanne kontrolpakker, kan

herefter ud fra sekvensnummeret og det til "uendeligt" satte antal af hop deducere sig frem til, hvordan deres interne tabeller skal opdateres.

For at undgå problemer med links, der ikke er symmetriske, gemmes kun links til en naboknude i en knudes interne tabeller, hvis knuden ved, at linket er symmetrisk.

A.3 Ad Hoc On-Demand Distance Vector Routing (en "distance vector", "on demand" protokol – afsnit 1.3.3)

Et eksempel på en "Distance vector", "On demand"-protokol er "Ad Hoc On-Demand Distance Vector Routing" (AODV), som beskrives i bl.a. [PR01, PBRD03].

AODV baserer sig på DSDV, men forsøger at minimere antallet af broadcasts i netværket ved kun at kræve, at ruter i netværket findes, når der er brug for dem, d.v.s. ved at lave protokollen om til en "on demand"-protokol. Når en knude skal sende en pakke til en knude, den ikke kender en rute til, igangsætter den en procedure kaldet "Path Discovery", som broadcaster en speciel "Route Request"-pakke rundt i netværket. Når denne specielle pakke modtages af en knude, videresender denne pakken.

Samtidigt gemmer knuden information om hvilken naboknude, de fik pakken fra, som et "next hop"-link i deres interne tabel, så de nu kender et link på vejen fra dem selv til den oprindelige afsenderknude. Bemærk, at dette betyder, at AODV går ud fra, at alle links er symmetriske.

For at undgå, at en knude videresender den samme "Route Request"-pakke adskillige gange, har hver pakke et sekvensnummer, der er unikt pr. oprindelig afsenderknude. Dette nummer gemmes af de videresendende knuder, og knuderne vil ikke videresende pakker, de ud fra dette nummer kan deducere sig frem til, at de allerede har videresendt én gang.

Når pakkens destinationsknude modtager en "Route Request"-pakke, kan denne nu blot sende en "Route Reply"-pakke til den nabo, den netop modtog "Route Request"-pakken fra. Naboen vil allerede kende en "next hop"-knude til den oprindelige afsenderknude (da den har videresendt "Route Request"-pakken), og kan videresende "Route Reply"-pakken til denne. Den sørger samtidigt for at gemme information om hvilken naboknude, den fik "Route Reply"-pakken fra som en "next hop"-knude for afsenderen af denne (d.v.s. for målet for "Path Discovery"-algoritmen). Dette vil gentage sig, ind til pakken når den oprindelige afsenderknude – som herefter kender "next hop" til destinationsknuden, og kan sende den oprindelige pakke til denne.

Hvis en knude forsøger at sende eller videresende en pakke via et link til en "next hop"-nabo og opdager, at linket ikke længere fungerer (f.eks. fordi naboknuden eller knuden selv har flyttet sig), sendes en "Link Failure Notification"-pakke tilbage til den oprindelige afsenderknude. Når en videresendende knude modtager (og skal videresende) en sådan pakke, sletter den de relevante indgang i sine tabeller, så ruten ikke længere benyttes. Når den oprindelige afsenderknude modtager pakken, kan den herefter igangsætte en ny "Path Discovery" til destinationsknuden for at finde en ny rute dertil.

AODV forbyder ikke brugen af beskeder, der sendes ud, selvom det ikke sker som følge af datatrafik. Sådanne beskeder må bruges, hvis en knude vil informere sine naboer om hvilke andre naboer, den har. Brugen af disse er dog ikke påkrævet, og benyttes de i en implementation af AODV, vil denne implementation i princippet ikke længere give en "on demand"-protokol.

A.4 Dynamic Source Routing (en "link state", "on demand" protokol – afsnit 1.3.3)

Et eksempel på en "Link state", "On demand" protokol er "Dynamic Source Routing" (DSR), som beskrives i bl.a. [JM96, JMB01, JMH05] samt i kapitel 2.

DSR er en “on demand”-protokol, der til forskel fra AODV er en “source routing”-protokol. Dette betyder, at når en knude vil sende en pakke til en destinationsknude, skal den inkludere hele ruten til destinationsknuden i pakken. De mellemliggende knuder behøver ikke kende noget til netværkstopologien – de kan læse i pakken selv hvilken knude, de har som nabo, og som de skal sende pakken videre til.

Når en knude vil sende en pakke til en destinationsknude, og den ikke kender en rute til knuden, igangsætter den en “Route Discovery”. Dette svarer til en “Path Discovery” i AODV med nogle få forskelle: Når en “Route Request”-pakke broadcastes rundt i netværket og en knude vil videresende pakken, tilføjer den sin adresse til en i pakken opsamlet ruteliste. Dette betyder, at når den endelige destinationsknude modtager pakken, kan den i pakken se den komplette rute, der blev brugt til at sende “Route Request”-pakken fra den oprindelige afsenderknude til destinationsknuden. Hvis det underliggende OSI-lag kun kan sende pakker via links, der er symmetriske f.eks. p.g.a. brugen af RTS-CTS-pakker (se afsnit 1.5.3), reverseres den opsamlede ruteliste blot, og pakken sendes via denne. Hvis ikke-symmetriske links understøttes i det underliggende netværk, er det muligt, at nogle links i den opsamlede rute ikke fungerer bidirektionelt. Derfor kan rutelisten ikke nødvendigvist bare reverseres og bruges. Hvis destinationsknuden ikke i forvejen kender en rute til den oprindelige afsenderknude, igangsætter den derfor en ny “Route Discovery” gående på den oprindelige afsenderknude. “Route Reply”-pakken inkluderes dog i “Route Request”-pakken for at undgå, at en “Route Discovery”-løkke opstår.

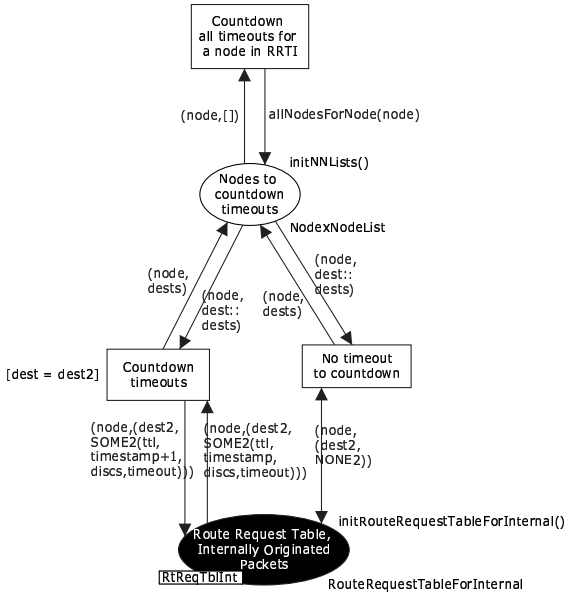
Når den oprindelige afsenderknude modtager “Route Reply”-pakken (enten direkte eller inkluderet i en “Route Request”-pakke), vil den få den komplette rute til destinationsknuden og kan benytte denne til at sende pakken dertil. Som før nævnt videresender hver af de mellemliggende knuder blot pakken til den adresse, som der står i pakken selv, at den skal sende pakken til. For at minimere antallet af fremtidige “Route Discoveries” i netværket, sørger de mellemliggende knuder dog også for at udtrække information om ruter fra disse pakker og gemme dem i deres interne tabeller.

Som i AODV sendes også en kontrolbesked tilbage til den oprindelige afsenderknude, hvis en mellemliggende knude opdager, at et link ikke længere fungerer. For at vedligeholde opsamlet information, sørger de knuder, der skal videresende pakken, ligeledes for at slette disse ruter fra deres tabeller.

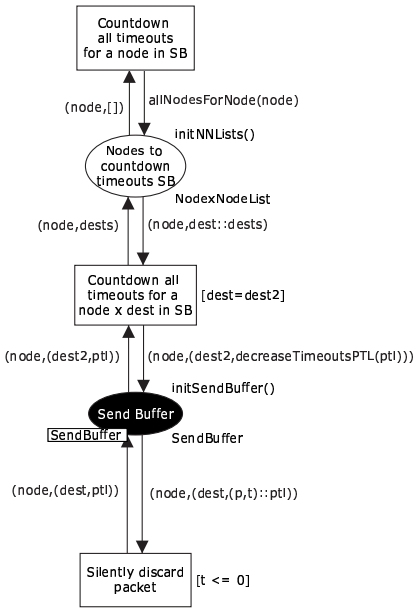
B
DSR-modellen

B.1 Resterende CPN-sider fra DSR-modellen i kapitel 3

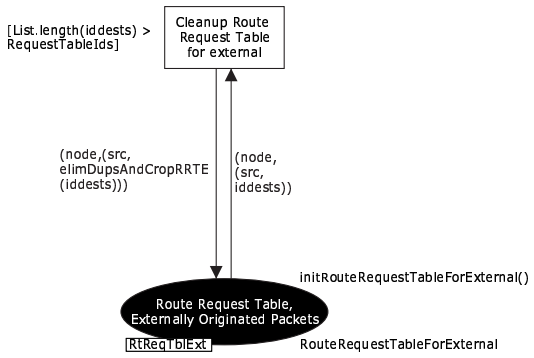
I dette afsnit vises de CPN-sider, der indgår i DSR-modellen, men som ikke blev vist i gennemgangen af denne i kapitel 3. Se afsnit 3.3-3.6 for en beskrivelse af indholdet af siderne.



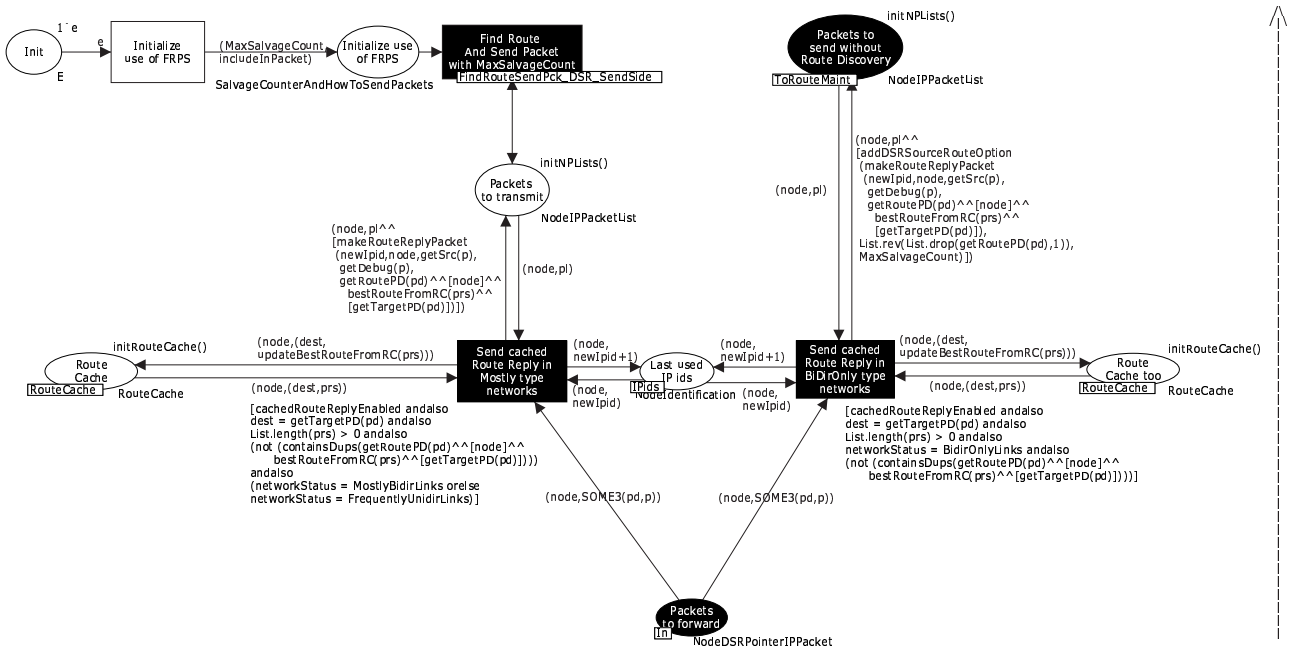
Figur B.1: DSR-modellens CPN-side “MaintainRouteReqTableInt_RD_SendSide”



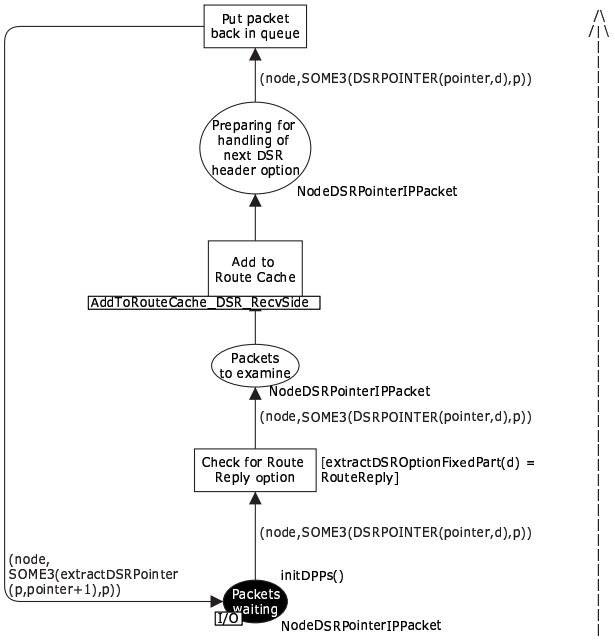
Figur B.2: DSR-modellens CPN-side “MaintainSendBuffer_RD_SendSide”



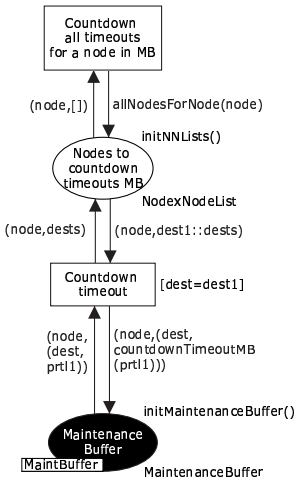
Figur B.3: DSR-modellens CPN-side “MaintainRouteReqTableExt_RD_RecvSide”



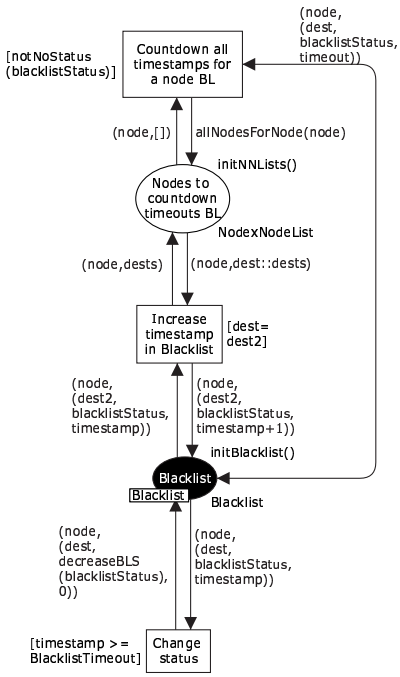
Figur B.4: DSR-modellens CPN-side “SendCachedRouteReply_RD_RecvSide”



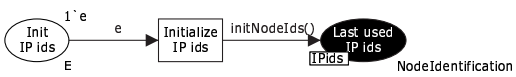
Figur B.5: DSR-modellens CPN-side “ProcRouteReply_RD_RecvSide”



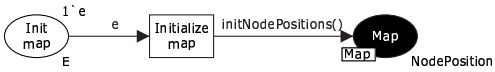
Figur B.6: DSR-modellens CPN-side "MaintainMaintBuffer_RM_SendSide"



Figur B.7: DSR-modellens CPN-side "MaintainBlacklist_RM_SendSide"



Figur B.8: DSR-modellens CPN-side "InitIPids_OSILayer3AndUp"



Figur B.9: DSR-modellens CPN-side "InitMap_OSILayer3AndUp"

B.2 ML-delen af DSR-modellen i kapitel 3

Dette er ML-delen af CPN-modellen af DSR-protokollen beskrevet i kapitel 3.

```
(* Standard declarations *)
     colset E = with e;
     colset INT = int;
     colset BOOL = bool;
     colset STRING = string;
     val noOfNodes = 4;
     val salvageEnabled = false;
     val cachedRouteReplyEnabled = false;
(* DSR globals *)
     val randomEvents = false;
     (* When this is set to true,
     eventChainToRun should be
     set to [] (unless you want a
     specified set of event to happen
     at the same time as random events *)
     val mapSize = 2000;
     val maxSpeed = 10;
     val maxSpeedChange = 2;
     val antennaDistance = real 300;
     val maxRandomPackets = 50;
     val packetIsMadePercentage = 20;
     (* Hvis en pakke kan laves, er der
     20% sandsynlighed for at den bliver
     det (for ikke at oversvoemme nettet). *)
     val collissionDetectionPercentage = 0;
     colset NetworkStatus =
     with FrequentlyUnidirLinks
          | MostlyBidirLinks
          | BidirOnlyLinks;
     val promiscModePossible = true; (* false; *)
     val timeoutFactor = 0.5;
     val stdTTL = 255;
     val DiscoveryHopLimit = 255; (* hops *)
     val BroadcastJitter = round(timeoutFactor * 10.0); (* ms - not used *)
     val RouteCacheTimeout = round(timeoutFactor * 300000.0); (* 300 seconds *)
     val SendBufferTimeout = round(timeoutFactor * 30000.0); (* 30 seconds *)
     val RequestTableSize = 64; (* nodes - not used *)
     val RequestTableIds = 16; (* identifiers *)
     val MaxRequestRexmt = 16; (* retransmissions *)
     val MaxRequestPeriod = round(timeoutFactor * 10000.0); (* 10 seconds *)
     val RequestPeriod = round(timeoutFactor * 500.0); (* 500 ms *)
     val NonpropRequestTimeout = round(timeoutFactor * 30.0); (* 30 ms *)
     val RexmtBufferSize = 50; (* packets - not used *)
     val MaintHoldoffTime = round(timeoutFactor * 250.0) (* 250 ms - not used *)
     val MaxMaintRexmt = 2; (* retransmissions *)
     val TryPassiveAcks = 1; (* attempt *)
     val PassiveAckTimeout = round(timeoutFactor * 100.0) (* 100 ms *)
     val GratReplyHoldoff = round(timeoutFactor * 1000.0) (* 1 second - not used *)
     val MaxSalvageCount = 15; (* salvages *)
     val MaintBufferTimeout = round(timeoutFactor * 500.0) (* 500 ms - not a std var! *)
     val BlacklistTimeout = round(timeoutFactor * 10000.0) (* 10000 ms - not a std var! *)
     val networkStatus = MostlyBidirLinks;
(* DSR colors - generel *)
     colset Timeout = int;
     colset Coor = int with 1..mapSize;
     colset Speed = int with ~maxSpeed..maxSpeed;
     colset SpeedChange = int with ~maxSpeedChange..maxSpeedChange;
     colset Node_ = string; (* "Node_" instead of "Node" to avoid CPN-bug 2078 *)
     colset NodeNode = product Node_ * Node_;
     colset NodeNodeNode =
     product Node_ * Node_ * Node_;
     colset NodeNodeNodeNode =
     product Node_ * Node_ * Node_ * Node_;
     colset Route = list Node_;
     colset NodeNodeList = list NodeNode;
     colset NodeList  = list Node_;
     colset NodexNodeList = product Node_ * Route;
     colset NodexNodeNodeList = product Node_  * NodeNodeList;
     colset NodeNodeNodeList = list NodeNodeNode;
     colset NodexNodeNodeNodeList = product Node_ * NodeNodeNodeList;
     colset RouteList = list Route;
     colset UserData = union SOMEDATA : STRING + NODATA;
     colset Result = bool;
```

```
      colset MapPosition = product Coor * Coor;
      colset Trajectory = product Speed * Speed;
      colset NodePosition = product Node_ * MapPosition;
      colset NodeTrajectory = product Node_ * Trajectory;
      colset MapRequest = product Node_ * Node_;
      colset MapRequestResult = product Node_ * Node_ * Result;
      colset MapRequestList = list NodexNodeList;
      colset MapRequestResultHuh
      = union REQ:MapRequest
      + REQRES:MapRequestResult;
      colset MapRequestResultHuhList = list MapRequestResultHuh;
      colset MapRequestResultHuhListList = list MapRequestResultHuhList;
      colset NodeResult = product Node_ * Result;
      colset NodeResultList = list NodeResult;
      colset NodexNodeResultList = product Node_ * NodeResultList;
      colset MapRequestResultList = list NodexNodeResultList;
      colset TTL = int;
      colset Counter = int;
      colset Identification = int with 1..100000;
      colset NodeIdentification = product Node_ * Identification;
      colset NoOfIdentification = product Identification * Counter;
      colset Timestamp = int;
      colset SalvageCounter = Counter;
      colset SegmentsLeft = Counter;
      colset DiscsCounter = Counter;
      colset RetransmissionsCounter = Counter;
      colset RandomEvent = bool;
      colset HowToSendPackets = with includeInPacket | saveInSendBuffer;
      colset SalvageCounterAndHowToSendPackets = product SalvageCounter * HowToSendPackets;
(* DSR colors - event chain (DSR globals no 2) *)
      val broadcastAddress = "Z" : Node_;
      colset MoveNode =
      record toPos: MapPosition;
      colset SendPacket =
      record toNode: Node_ *
      id: Identification * data: UserData;
      colset WaitForPacket =
      record id: Identification;
      colset WaitForTimeout =
      record timeout: Counter;
      colset Event =
      union MoveNode:MoveNode +
      SendPacket:SendPacket +
      WaitForPacket:WaitForPacket +
      WaitForTimeout:WaitForTimeout;
      colset EventChainForNode = list Event;
      colset EventChain = product Node_ * EventChainForNode;
      var event,event1,event2: Event;
      var events,events1,events2: EventChainForNode;
val demoChain1 : EventChain ms =
      1`("A",[MoveNode{toPos=(200,200)},
              SendPacket{toNode="B",id=1,data=SOMEDATA
                         ("pakke 1 fra A til B")}]) ++
      1`("C",[MoveNode{toPos=(200,400)}]) ++
      1`("D",[MoveNode{toPos=(200,600)}]) ++
      1`("E",[MoveNode{toPos=(200,800)}]) ++
      1`("B",[MoveNode{toPos=(200,1000)},
              WaitForPacket{id=1}]);
      val demoChain2 : EventChain ms =
      1`("A",[MoveNode{toPos=(200,200)},
              SendPacket{toNode="B",id=1,data=SOMEDATA
                         ("pakke 1")},
              WaitForTimeout{timeout=4000},
              SendPacket{toNode="B",id=2,data=SOMEDATA
                         ("pakke 2")},
              WaitForTimeout{timeout=50000},
              SendPacket{toNode="B",id=3,data=SOMEDATA
                         ("pakke 3 (genudsendelse af pakke 2)")}]) ++
      1`("C",[MoveNode{toPos=(200,400)}]) ++
      1`("D",[MoveNode{toPos=(200,600)}]) ++
      1`("E",[MoveNode{toPos=(200,800)},
              WaitForTimeout{timeout=2000},
              MoveNode{toPos=(2000,2000)}]) ++
      1`("B",[MoveNode{toPos=(200,1000)},
              WaitForPacket{id=1},
              WaitForPacket{id=3}]) ++
      1`("F",[MoveNode{toPos=(2000,2000)},
              WaitForTimeout{timeout=2000},
```

```
            MoveNode{toPos=(350,700)}]) ++
        1`("G",[MoveNode{toPos=(2000,2000)},
            WaitForTimeout{timeout=2000},
            MoveNode{toPos=(350,900)}]);
    val demoChain3 : EventChain ms =
        1`("A",[MoveNode{toPos=(200,200)},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="F",id=2,data=SOMEDATA
                ("pakke 2")},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="F",id=3,data=SOMEDATA
                ("pakke 3 (kan salvages)")}]) ++
        1`("B",[MoveNode{toPos=(200,400)}]) ++
        1`("C",[MoveNode{toPos=(200,600)},
            SendPacket{toNode="F",id=1,data=SOMEDATA
                ("pakke 1")}]) ++
        1`("F",[MoveNode{toPos=(200,1000)},
            WaitForPacket{id=1},
            WaitForPacket{id=2},
            WaitForPacket{id=3}]) ++
             (* WaitForPacket{id=3}: Vil kun blive opfyldt
                nr salvageEnabled = true *)
        1`("D",[MoveNode{toPos=(350,700)}]) ++
        1`("E",[MoveNode{toPos=(350,900)}]) ++
        1`("G",[MoveNode{toPos=(2000,2000)},
            WaitForTimeout{timeout=2000},
            MoveNode{toPos=(200,800)},
            WaitForTimeout{timeout=4000},
            MoveNode{toPos=(2000,2000)}]);
    val demoChain3a : EventChain ms =
        1`("A",[MoveNode{toPos=(200,200)},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="E",id=2,data=SOMEDATA
                ("pakke 2")},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="E",id=3,data=SOMEDATA
                ("pakke 3 (kan salvages)")}]) ++
        1`("B",[MoveNode{toPos=(200,400)},
            SendPacket{toNode="E",id=1,data=SOMEDATA
                ("pakke 1")}]) ++
        1`("C",[MoveNode{toPos=(150,600)},
            WaitForTimeout{timeout=2000},
            MoveNode{toPos=(0,600)},
            WaitForTimeout{timeout=4000},
            MoveNode{toPos=(150,600)}]) ++
        1`("D",[MoveNode{toPos=(400,600)},
            WaitForTimeout{timeout=2000},
            MoveNode{toPos=(250,600)},
            WaitForTimeout{timeout=4000},
            MoveNode{toPos=(400,600)}]) ++
        1`("E",[MoveNode{toPos=(200,800)},
            WaitForPacket{id=1},
            WaitForPacket{id=2},
            WaitForPacket{id=3}])
             (* WaitForPacket{id=3}: Vil kun blive opfyldt
                nr salvageEnabled = true *)
    val demoChain3b : EventChain ms =
        1`("A",[MoveNode{toPos=(200,200)},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="D",id=2,data=SOMEDATA
                ("pakke 2")},
            WaitForTimeout{timeout=4000},
            SendPacket{toNode="D",id=3,data=SOMEDATA
                ("pakke 3 (kan salvages)")}]) ++
        1`("B",[MoveNode{toPos=(200,400)},
            SendPacket{toNode="D",id=1,data=SOMEDATA
                ("pakke 1")}]) ++
        1`("C",[MoveNode{toPos=(300,600)}]) ++
        1`("D",[MoveNode{toPos=(200,800)},
            WaitForPacket{id=1},
            MoveNode{toPos=(200,600)},
            WaitForPacket{id=2},
            MoveNode{toPos=(200,800)},
            WaitForPacket{id=3}]) ++
             (* WaitForPacket{id=3}: Vil kun blive opfyldt
                nr salvageEnabled = true *)
        1`("E",[MoveNode{toPos=(2000,2000)}])
    val demoChain4 : EventChain ms =
```

```
230       1`("A",[MoveNode{toPos=(200,200)},
                  WaitForTimeout{timeout=2000},
                  SendPacket{toNode="B",id=2,data=SOMEDATA
                             ("pakke 2")}]) ++
          1`("C",[MoveNode{toPos=(200,400)}]) ++
          1`("D",[MoveNode{toPos=(200,600)},
                  SendPacket{toNode="B",id=1,data=SOMEDATA
                             ("pakke 1")}]) ++
          1`("E",[MoveNode{toPos=(200,800)}]) ++
          1`("B",[MoveNode{toPos=(200,1000)},
240               WaitForPacket{id=1},
                  WaitForPacket{id=2}]);
          val demoChain5 : EventChain ms =
          1`("A",[MoveNode{toPos=(200,200)},
                  SendPacket{toNode="D",id=1,data=SOMEDATA("pakke 1")},
                  WaitForTimeout{timeout=4000},
                  SendPacket{toNode="D",id=2,data=SOMEDATA("pakke 2")}]) ++
          1`("B",[MoveNode{toPos=(150,400)},
                  WaitForTimeout{timeout=2000},
                  MoveNode{toPos=(2000,2000)}]) ++
250       1`("C",[MoveNode{toPos=(2000,2000)},
                  WaitForTimeout{timeout=2000},
                  MoveNode{toPos=(250,400)}]) ++
          1`("D",[MoveNode{toPos=(200,600)},
                  WaitForPacket{id=1},
                  WaitForPacket{id=2}]);
          (* REMEMBER to set noOfNodes above depending on test *)
          val eventChainToRun = demoChain5;
          (* use "../dsr-perl/inputs_and_results/medium02input.sml";
          use "../latex/arb4/deadlock-trigger-input.sml"; *)
260   (* DSR colors - options header *)
        (* Fixed part of DSR options header: *)
          colset DSROptionsHeaderFixed =
          union DSRSourceRoute +
          RouteRequest +
          RouteReply +
          AcknowledgementRequest +
          Acknowledgement +
          RouteError;
        (* Variable part of DSR options header: Route Requests *)
270       colset RouteRequestOption =
          record id: Identification
                    * target: Node_
                    * routeSoFar: Route;
        (* Variable part of DSR options header: Route Replies *)
          colset RouteReplyOption = record route: Route;
        (* Variable part of DSR options header: Route Errors *)
          colset ErrorType = with NODE_UNREACHABLE
                           (*   | FLOW_STATE_NOT_SUPPORTED
                             | OPTION_NOT_SUPPORTED *);
280       colset ErrorTypeSpecificInformation = union unreachableNodeAddress : Node_;
          colset RouteErrorOption =
          record errorType: ErrorType
                            * salvageCounter: SalvageCounter
                            * errorSrc: Node_
                            * errorDest: Node_
                            * typeSpec: ErrorTypeSpecificInformation;
        (* Variable part of DSR options header: Acknowledgement Requests *)
          colset AcknowledgementRequestOption = record id: Identification;
        (* Variable part of DSR options header: Acknowledgements *)
290       colset AcknowledgementOption = record id: Identification
                                                * ackSrc: Node_
                                                * ackDest: Node_;
        (* Variable part of DSR options header: DSR Source Route *)
          colset DSRSourceRouteOption =
          record salvageCounter: SalvageCounter
                       * segmentsLeft: SegmentsLeft
                       * route: Route;
        (* DSR options header: Combined *)
          colset DSROptionsHeaderVariable = union routeRequest : RouteRequestOption
300                                             + routeReply : RouteReplyOption
                                                + routeError : RouteErrorOption
                                                + acknowledgementRequest : AcknowledgementRequestOption
                                                + acknowledgement : AcknowledgementOption
                                                + dsrSourceRoute : DSRSourceRouteOption;
          colset DSROptionsHeader =
          product DSROptionsHeaderFixed *
                                           DSROptionsHeaderVariable;
```

```
        colset DSROptionsHeaderList = list DSROptionsHeader;
        colset DebugHeaderProduct =
        record debugHardwareSrc: Node_ (* _this_ source *)
        * debugHardwareDest: Node_  (* _this_ destination *)
        * debugIPId: Identification
        * debugIPTTL: TTL
        * debugIPSrc: Node_ (* IP-packets source *)
        * debugIPDest: Node_  (* IP-packets destination *)
        * debugDSR: DSROptionsHeaderList;
        colset DebugHeaderProductList = list DebugHeaderProduct;
        colset DebugHeader =
        record debugId: Identification
        * debugHeaderCopy: DebugHeaderProductList;
    (* DSR colors - IP *)
        colset IPPacket = record id: Identification
                                 * ttl: TTL
                                 * src: Node_ (* source address *)
                                 * dest: Node_ (* destination address *)
                                 * dsr: DSROptionsHeaderList
                                 * debug: DebugHeader
                                 * data: UserData;
        colset IPPacketIPPacket = product IPPacket * IPPacket;
        colset IPPacketList = list IPPacket;
        colset NodeIPPacket = product Node_ * IPPacket;
        colset NodeIPPacketList = product Node_ * IPPacketList;
        colset NodeNodeIPPacketList = product Node_ * Node_ * IPPacketList;
        colset DSRPointer = int;
        colset DSRPointerElement = product DSRPointer * DSROptionsHeader;
        colset DSRPointerElementOrNot =
        union DSRPOINTER : DSRPointerElement +
              NOMOREHEADERS;
        colset DSRPointerIPPacketElement = product DSRPointerElementOrNot * IPPacket;
        colset DSRPointerIPPacketElementOrNot =
        union SOME3 : DSRPointerIPPacketElement + NONE3;
        colset NodeDSRPointerIPPacket =
        product Node_ *DSRPointerIPPacketElementOrNot;
        colset NodeDSRPointerIPPacketIPPacket =
        product Node_ * DSRPointerIPPacketElementOrNot * IPPacket;
        colset NodeDSRPointerIPPacketIPPacketList =
        list NodeDSRPointerIPPacketIPPacket;
        colset NodeNodeDSRPointerIPPacketIPPacketList =
        product Node_ * NodeDSRPointerIPPacketIPPacketList;
        colset NodeNodeIPPacketProductList =
        product Node_ * Route * IPPacket;
        colset NodeNodeIPPacketProductListList =
        list NodeNodeIPPacketProductList;
        colset DSROptionsHeaderIPPacket =
        product DSROptionsHeader * IPPacket;
        colset DSROptionsHeaderIPPacketList =
        list DSROptionsHeaderIPPacket;
        colset NodeDSROptionsHeaderIPPacket =
        product Node_ * DSRPointer * DSROptionsHeaderIPPacket;
        colset NodeNodeResultPacket =
        product Node_ * Node_ * Result * IPPacket;
        colset NodeNodeResultPacketList =
        list NodeNodeResultPacket;
        colset NodeNodeResultPacketListList =
        list NodeNodeResultPacketList;
        colset NodeNodeListPacket =
        product Node_ * Route * IPPacket;
        colset TimestampNodeNodeListPacket =
        product Timestamp * Node_ * Route * IPPacket;
        colset NodeNodeListPacketList = list NodeNodeListPacket;
        colset NodeNodeListPacketListList  = list NodeNodeListPacketList;
        colset NodeNodePacketPointerTimeout
        = product Node_ * Node_ *
        IPPacket *
        DSRPointerElementOrNot *
        Timeout;
    (* DSR colors - caches and structures *)
        colset NodeRoutes = product Node_ * RouteList;
        colset RouteTime = product Route * Timestamp;
        colset PossibleRoutes = list RouteTime;
        colset PossibleRoutesToDest
              = product Node_
                        * PossibleRoutes;
        colset RouteCache =
        product Node_ * PossibleRoutesToDest;
```

```
(* Send Buffer: *)
   colset IPPacketTime =
   product IPPacket * Timestamp;
   colset IPPacketTimeList =
   list IPPacketTime;
   colset NodeIPPacketTimeList =
   product Node_ * IPPacketTimeList;
   colset SendBuffer =
   product Node_ *
   NodeIPPacketTimeList;
(* Route Request Table: *)
   (* The Id should be either on battery backup (preferred) or random at startup: *)
   colset RouteRequestIds = int with 1..1000000;
   val routeRequestId = RouteRequestIds.ran();
   colset IdNode = product Identification * Node_;
   colset IdNodeList = list IdNode;
   colset RouteRequestTableForInternalElement =
                                product TTL
                                        * Timestamp (* last sent time *)
                                        * DiscsCounter
                                        * Timeout;
   colset RouteRequestTableForInternalElementOrNot =
   union SOME2: RouteRequestTableForInternalElement + NONE2;
   colset RouteRequestTableForInternalForANode =
                              product Node_ (* destination address *) *
                                        RouteRequestTableForInternalElementOrNot;
   colset RouteRequestTableForInternal = product Node_ * RouteRequestTableForInternalForANode;
   colset RouteRequestTableForExternalForANode =
   product Node_ (* original sender *)
    * IdNodeList (* id number and final destination address *);
   colset RouteRequestTableForExternal =
   product Node_ * RouteRequestTableForExternalForANode;
(* Network Interface Queue *)
   colset NetworkInterfaceQueueForANode = list NodeIPPacket;
   colset NetworkInterfaceQueue =
   product Node_ *
   NetworkInterfaceQueueForANode;
   colset NIQNIQ =
   product NetworkInterfaceQueue * NetworkInterfaceQueue;
   colset NIQNIQList =
   list NIQNIQ;
(* Maintenance Buffer *)
   colset Waitingpacket = product IPPacket
                               * RetransmissionsCounter
                               * Timestamp; (* last sent time *)
   colset WaitingpacketList = list Waitingpacket;
   colset MaintenanceBufferForANode =
   product Node_ (* destination *)
           * WaitingpacketList;
   colset MaintenanceBuffer =
   product Node_ * MaintenanceBufferForANode;
(* Blacklist *)
   colset BlacklistStatus = with Probable | Questionable | NoStatus;
   colset BlacklistNode = product Node_ * BlacklistStatus * Timestamp;
   colset Blacklist = product Node_ * BlacklistNode;
(* DSR variables *)
   var pd: DSRPointerElementOrNot;
   var mr: MapRequest;
   var wp: Waitingpacket;
   var wpl: WaitingpacketList;
   var sc: SalvageCounter;
   var pointerelm: DSRPointerElementOrNot;
   var pointerelmpacket: DSRPointerIPPacketElementOrNot;
   var doSalvage: BOOL;
   var possibleRoutes: PossibleRoutes;
   var blacklistStatus: BlacklistStatus;
   var mrl,mrl1,mrl2: MapRequestList;
   var mrr: MapRequestResult;
   var mrrl: MapRequestResultList;
   var mrrh, mrrh1, mrrh2: MapRequestResultHuh;
   var mrrhl, mrrhl1, mrrhl2: MapRequestResultHuhList;
   var mrrhll, mrrhll1, mrrhll2: MapRequestResultHuhListList;
   var nnpl: NodeNodeIPPacketProductList;
   var nnpll: NodeNodeIPPacketProductListList;
   var nnrpl: NodeNodeResultPacketList;
   var nnrpll: NodeNodeResultPacketListList;
   var nnlpl,nnlpl1,nnlpl2: NodeNodeListPacketList;
   var nnlpll: NodeNodeListPacketListList;
```

```
    var nndpl, nndpl1, nndpl2: NodeNodeDSRPointerIPPacketIPPacketList;
    var ndpl, ndpl1, ndpl2: NodeDSRPointerIPPacketIPPacketList;
    var destresult: NodeResult;
    var destresults: NodeResultList;
    var pos,pos1,pos2: MapPosition;
    var prt, prt1, prt2: Waitingpacket;
    var prtl, prtl1, prtl2: WaitingpacketList;
    var rc: PossibleRoutes;
    var str: STRING;
    var coor1,coor2: Coor;
    var pointer,pointer1: DSRPointer;
    var routetimes, newRoutetimes, routetime1,routetime2,routetime3: PossibleRoutes;
    var routetime: RouteTime;
    var traj: Trajectory;
    var node,node1,node2,sender,sender1,sender2,receiver,thisNode,
    aDest,aDest1,aDest2,failLink,link1,link2,
    dest, src, dest1, src1, dest2, src2, dest3, dest4, src3, nextHop: Node_;
    var salvage,salvageCount: SalvageCounter;
    var failList: NodeNodeList;
    var failList2: NodeNodeNodeList;
    var segmentsLeft: SegmentsLeft;
    var iddests: IdNodeList;
    var nodeList, route, route1, route2, dests, nodes: Route;
    var routes, routes1, routes2, rl: RouteList;
    var fixed: DSROptionsHeaderFixed;
    var data,data1,data2,userData: UserData;
    var dsrs: DSROptionsHeaderList;
    var ttl, ttl2: TTL;
    var result: Result;
    var p,p1,p2,packet,packet1,packet2: IPPacket;
    var pl, pl1, pl2, pl3, pl4, pl5: IPPacketList;
    var niq, niq1, niq2: NetworkInterfaceQueueForANode;
    var niqniqs: NIQNIQList;
    var prs: PossibleRoutes;
    var d, d1, d2, d3: DSROptionsHeader;
    var dsrvar: DSROptionsHeaderVariable;
    var dl, dl1, dl2, dl3: DSROptionsHeaderList;
    var dp, dp1, dp2, dp3: DSROptionsHeaderIPPacket;
    var dpl, dpl1, dpl2, dpl3: DSROptionsHeaderIPPacketList;
    var dhpl, dhpl1, dhpl2:
    DebugHeaderProductList;
    var debug, debug1, debug2:
    DebugHeader;
    var ptl, ptl1, ptl2, pts: IPPacketTimeList;
    var pt: IPPacketTime;
    var id,id2,id3,id4,debugid,ipid,newIpid,rrid: Identification;
    var userdata: UserData;
    var t,timeout,timeout1, timeout2: Timeout;
    var timestamp,timestamp2: Timestamp;
    var r: RetransmissionsCounter;
    var discs: DiscsCounter;
    var traceDebugId,debugId,debugId1,debugId2: Identification;
    var collissionHappens, packetIsMade: RandomEvent;
    var c1, c2: Counter;
    var howToSendPackets: HowToSendPackets;
(* DSR exceptions *)
    exception PDDSRException of DSRPointerElementOrNot;
    exception FindNodeNoInListDSRException of Node_;
    exception AllButLastElmDSRException;
    exception AddNodeToRouteRequestAtPointerDSRException
      of { dsrheaders: DSROptionsHeaderList, node: Node_, pointer: DSRPointer };
    exception ExtractDSROptionNoDSRException
      of { packet: IPPacket, pointer: DSRPointer };
    exception DecrementSegmentsLeftInDSRSourceRouteOptionDSRException;
    exception FindThisNodeInChainDSRException of Node_;
    exception NodeListToLastSenderInclDSRException
      of { segmleft: SegmentsLeft, nodelist: Route };
    exception RemoveListSectionFrom2ndElmIncl2ndElmDSRException
       of { nodelist: Route, node: Node_, no: INT};
    exception RemoveListSectionUntil2ndElmExcl2ndElmDSRException
      of { nodelist: Route, node: Node_, no: INT};
    exception IntendedRecipientPDSRException of IPPacket;
    exception IntendedRecipientPSDSRException of { packet: IPPacket, offset: INT };
    exception GetSegmentsLeftDSRException of IPPacket;
    exception GetSalvageCounterDSRException of IPPacket;
    exception GetFirstLinkDSRException of IPPacket;
    exception ExistsLinkInPacketDSRException
      of {src: Node_, dest: Node_, packet: IPPacket};
```

```
      exception FindRouteWithTimeInRCDSRException of Timestamp;
      exception CombineResultListsAndCopyDebugDSRException
        of {NNRL: NodexNodeResultList, NNLP: NodeNodeListPacket};
      exception UnHuhMRRDSRException of MapRequestResultHuhList;
      exception GetSalvageCountDSRException of IPPacket;
      exception SetSalvageCountDSRException of IPPacket;
      exception SetRouteDSRException of IPPacket;
      exception SetSegmentsLeftDSRException of IPPacket;
(* DSR functions - IP *)
      fun setId (newid: Identification, {id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=newid,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=dth,data=data};
      fun getId ({id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): Identification
      = id;
      fun setTTL (newttl: TTL, {id=id,ttl=ttl,src=src,dest=dest,
                  dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=newttl,src=src,dest=dest,dsr=dsrs,debug=dth,data=data};
      fun getTTL ({id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): TTL
      = ttl;
      fun setSrc (newSrc: Node_, {id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=newSrc,dest=dest,dsr=dsrs,debug=dth,data=data};
      fun getSrc ({id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): Node_
      = src;
      fun setDest (newDest: Node_, {id=id,ttl=ttl,src=src,dest=dest,
                dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=src,dest=newDest,dsr=dsrs,debug=dth,data=data};
      fun getDest ({id=id,ttl=ttl,src=src,dest=dest,
                dsr=dsrs,debug=dth,data=data}: IPPacket): Node_
      = dest;
      fun setDsrs (newDsrs: DSROptionsHeaderList, {id=id,ttl=ttl,src=src,dest=dest,
               dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=src,dest=dest,dsr=newDsrs,debug=dth,data=data};
      fun getDsrs ({id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): DSROptionsHeaderList
      = dsrs;
      fun setDth (newDth: DebugHeader, {id=id,ttl=ttl,src=src,dest=dest,
                dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=newDth,data=data};
      fun getDebug ({id=id,ttl=ttl,src=src,dest=dest,
                dsr=dsrs,debug=dth,data=data}: IPPacket): DebugHeader
      = dth;
      fun getDebugId ({id=id,ttl=ttl,src=src,dest=dest,
                dsr=dsrs,debug={debugId=id2, debugHeaderCopy=dhc},
                data=data}: IPPacket): Identification
      = id2;
      fun copyNormalHeaderToDebugHeader
      (thisSource: Node_,thisDest: Node_,{id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,
                debug={debugId=debugId,debugHeaderCopy=traceList},
                data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=
            {debugId=debugId,debugHeaderCopy=
                {debugHardwareSrc=thisSource,debugHardwareDest=thisDest,
                 debugIPId=id,debugIPTTL=ttl,
                 debugIPSrc=src,debugIPDest=dest,
                 debugDSR=[] (* dsrs *)}::[] (* traceList *)},
              data=data}
      fun setData (newData: UserData, {id=id,ttl=ttl,src=src,dest=dest,
                   dsr=dsrs,debug=dth,data=data}: IPPacket): IPPacket
      = {id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=dth,data=newData};
      fun getData ({id=id,ttl=ttl,src=src,dest=dest,
                 dsr=dsrs,debug=dth,data=data}: IPPacket): UserData
      = data;
      fun isSomedata (SOMEDATA(str): UserData): bool
          = true
      | isSomedata (NODATA) = false;
      fun isNodata (a: UserData): bool = not (isSomedata a);
(* DSR functions - generel *)
      fun mergePR((r1,t1)::l1: PossibleRoutes, (r2,t2)::l2: PossibleRoutes): PossibleRoutes
      = if t1>t2 (* Biggest first *)
         then (r1,t1)::mergePR(l1,(r2,t2)::l2)
         else (r2,t2)::mergePR((r1,t1)::l1,l2)
      | mergePR([],l2) = l2
      | mergePR(l1,[]) = l1;
      fun sortPR(xs: PossibleRoutes): PossibleRoutes
```

```
620     = if xs=[]
           then []
         else if tl(xs)=[]
                 then xs
                 else mergePR(sortPR(List.take(xs,List.length(xs) div 2)),
                              sortPR(List.drop(xs,List.length(xs) div 2)));
        fun routeExists(route: Route, (route2,time2)::
                    routetimes: PossibleRoutes): Timestamp option
        =       if (route = route2)
                then (SOME time2)
                else NONE
        | routeExists(route,[]) = NONE;
        fun min(a: int, b: int): int =
        if a < b
        then a
        else b;
        fun findAndElimDupInRC((route2,time2)::routetimes: PossibleRoutes): PossibleRoutes =
            if isSome(routeExists(route2,routetimes))
            then findAndElimDupInRC(routetimes)
            else (route2,time2)::findAndElimDupInRC(routetimes)
        | findAndElimDupInRC([]) = [];
        (* Sortering efter timestamp foregaar eksternt for denne funktion *)
        fun dupsExistInRC((route2,time2)::routetimes: PossibleRoutes): bool =
            isSome(routeExists(route2,routetimes)) orelse
            dupsExistInRC(routetimes)
        | dupsExistInRC([]) = false;
        fun tooBigTimestampExistsInRC((route,timestamp)::routetimes: PossibleRoutes): bool
        = if timestamp >= RouteCacheTimeout
           then true
           else tooBigTimestampExistsInRC(routetimes)
        | tooBigTimestampExistsInRC([]) = false;
        fun elimTooBigTimestampsInRC((route,timestamp)::
                        routetimes: PossibleRoutes): PossibleRoutes
        = if timestamp >= RouteCacheTimeout
           then elimTooBigTimestampsInRC(routetimes)
           else (route,timestamp)::elimTooBigTimestampsInRC(routetimes)
        | elimTooBigTimestampsInRC([]) = [];
        fun removeFromListRRTE(id: Identification, dest: Node_,
                    (id2,dest2)::iddests: IdNodeList): IdNodeList
        = if id=id2 andalso dest=dest2
           then removeFromListRRTE(id,dest,iddests)
           else (id2,dest2)::removeFromListRRTE(id,dest,iddests)
        | removeFromListRRTE(id,dest,[]) = [];
        fun elimDupsRRTE((id,dest)::iddests: IdNodeList): IdNodeList
        = (id,dest)::elimDupsRRTE(removeFromListRRTE(id,dest,iddests))
        | elimDupsRRTE([]) = [];
        (* The first ones in the list are the newest --
             everything should be deleted from the end *)
        fun elimDupsAndCropRRTE (iddests: IdNodeList): IdNodeList
        = let val elimedList = elimDupsRRTE(iddests)
           in List.take(elimedList,(min(RequestTableIds,List.length(elimedList))))
                    (* avoiding exception *)
           end;
        (* The first ones in the list are the newest --
              everything should be deleted from the end *)
        fun distance ((x1,y1): MapPosition, (x2,y2): MapPosition): real
        = Math.sqrt(Math.pow((real x1)-(real x2),real 2)+
                    Math.pow((real y1)-(real y2),real 2));
        fun randomPosition(): MapPosition
        = (Coor.ran(),Coor.ran());
        fun randomTrajectory(): Trajectory
        = (Speed.ran(),Speed.ran());
        fun randomId(): Identification = Identification.ran();
        fun allNodes(): Node_ ms
        = let
            fun an 0 = empty
              | an n = 1`Char.toString(Char.chr(n+64)) ++ (an (n-1))
          in
            an noOfNodes
          end;
        fun allNodesNodes(): NodeNode ms
        = let
          fun an 0 = empty
             | an n = (ext_col (fn m => (Char.toString(Char.chr(n+64)),m)) (allNodes()))
                         ++ (an (n-1))
          in
            an noOfNodes
          end;
```

```
fun initNodes(): Node_ ms
= allNodes();
fun allNodesExcept (n: Node_): Node_ ms
= allNodes() -- 1`n;
fun toAllExcept (except: Node_, packet: IPPacket)
= ext_col (fn recv => (packet, recv)) (allNodesExcept(except));
fun toAllExcept2 (except: Node_, packet: Node_): MapRequest ms
= ext_col (fn recv => (packet, recv)) (allNodesExcept(except));
fun makePacketsToAllExcept (except: Node_, packet: IPPacket): NodeNodeIPPacketProductList
= (except,(List.map (fn (packet,recv) => recv) (toAllExcept (except,packet))),packet);
fun initNodePositions(): NodePosition ms
= ext_col (fn n => (n,randomPosition())) (allNodes());
fun initNodeTrajectories(): NodeTrajectory ms
= ext_col (fn n => (n,randomTrajectory())) (allNodes());
fun initNPLists(): NodeIPPacketList ms
= ext_col (fn n => (n,[])) (allNodes());
fun initNNPLists(): NodeNodeIPPacketList ms
= ext_col (fn (n,m) => (n,m,[])) (allNodesNodes());
fun initNNDPLists(): NodeNodeDSRPointerIPPacketIPPacketList ms
= ext_col (fn n => (n,[])) (allNodes());
fun initNIQ(): NetworkInterfaceQueue ms
= ext_col (fn n => (n,[])) (allNodes());
fun initRLists(): NodeRoutes ms
= ext_col (fn n => (n,[])) (allNodes());
fun initNNLists(): NodexNodeList ms
= ext_col (fn n => (n,[])) (allNodes());
fun initNNNLists(): NodexNodeNodeList ms
= ext_col (fn n => (n,[])) (allNodes());
fun initNNNNLists(): NodexNodeNodeNodeList ms
= ext_col (fn n => (n,[])) (allNodes());
fun initSendBuffer(): SendBuffer ms
= ext_col (fn (n,m) => (n,(m,[]))) (allNodesNodes());
(* Produces a fitting ms with (n,m,[]) for all n,m *)
fun initMaintenanceBuffer(): MaintenanceBuffer ms
= ext_col (fn (n,m) => (n,(m,[]))) (allNodesNodes());
fun initBlacklist(): Blacklist ms
= ext_col (fn (n,m) => (n,(m,NoStatus,0: Timestamp))) (allNodesNodes());
fun initRouteCache(): RouteCache ms
= ext_col (fn (n,m) => (n,(m,[]))) (allNodesNodes());
fun initRouteRequestTableForInternal(): RouteRequestTableForInternal ms
= ext_col (fn (n,m) => (n,(m,NONE2))) (allNodesNodes());
fun initRouteRequestTableForExternal(): RouteRequestTableForExternal ms
= ext_col (fn (n,m) => (n,(m,[]))) (allNodesNodes());
fun initNodeIds(): NodeIdentification ms
= ext_col (fn n => (n,randomId())) (allNodes());
fun allNodesForNode(node: Node_): NodexNodeList
= (node,ms_to_list (allNodes()));
fun initDPPs(): NodeDSRPointerIPPacket ms
= ext_col (fn n => (n,NONE3)) (allNodes());
fun randomNode(notNode: Node_): Node_
= random (allNodesExcept(notNode));
fun randomMovement((x,y): MapPosition, (cx,cy): Trajectory): MapPosition
= (if x+cx > mapSize
    then mapSize
    else if x+cx < 1
            then 1
            else x+cx,
    if y+cy > mapSize
    then mapSize
    else if y+cy < 1
           then 1
           else y+cy);
fun randomTrajectoryChange((cx,cy): Trajectory): Trajectory
= let val ccx = SpeedChange.ran();
        val ccy = SpeedChange.ran();
        val newccx = cx + ccx;
        val newccy = cy + ccy;
   in (if newccx > maxSpeed
        then maxSpeed
        else if newccx < ~maxSpeed
               then ~maxSpeed
               else newccx,
        if newccy > maxSpeed
        then maxSpeed
        else if newccy < ~maxSpeed
                then ~maxSpeed
                else newccy)
  end;
```

```
      fun randomData(sendNode: Node_, recvNode: Node_, debugId: Identification): UserData
      = SOMEDATA("Message from " ^ (sendNode) ^
        " to " ^ (recvNode) ^
        " (with debugId " ^ (Int.toString debugId) ^")");
      fun randomPacket(sendNode: Node_, id: Identification, debugId: Identification): IPPacket
      = let
          val recvNode = randomNode sendNode
        in
          {id=id, ttl=stdTTL, src=sendNode, dest=recvNode, dsr=[], debug={debugId=debugId,
           debugHeaderCopy=[]},
           data=randomData(sendNode,recvNode,debugId)}
        end;
      fun findNodeNoInList (thisNode: Node_, node::nodeList: Route): int
      = if thisNode = node
        then 0
        else 1 + (findNodeNoInList (thisNode,nodeList))
      | findNodeNoInList (thisNode, []) = raise FindNodeNoInListDSRException(thisNode);
      fun prevNode (thisNode: Node_, nodeList: Route): Node_
      = List.nth (nodeList,findNodeNoInList(thisNode,nodeList)-1);
      fun allbutlastelm (x::[]) = []
        | allbutlastelm [] = [] (* raise AllButLastElmDSRException *)
        | allbutlastelm (x::xs) = x::(allbutlastelm xs);
      fun lastelm (x::xs) = SOME (List.last (x::xs))
      | lastelm [] = NONE;
(* DSR functions - options header *)
      fun getPtrPD(DSRPOINTER(pointer,d)) = pointer
      | getPtrPD(x) = raise PDDSRException(x);
      fun getTypeSpecPD(DSRPOINTER(pointer,
      (RouteError,routeError({typeSpec=AS,...})))) = AS
      | getTypeSpecPD(x) = raise PDDSRException(x);
      fun getErrorDestPD(DSRPOINTER(pointer,
      (RouteError,routeError({errorDest=AS,...})))) = AS
      | getErrorDestPD(x) = raise PDDSRException(x);
      fun getErrorSrcPD(DSRPOINTER(pointer,
      (RouteError,routeError({errorSrc=AS,...})))) = AS
      | getErrorSrcPD(x) = raise PDDSRException(x);
      fun getAckDestPD(DSRPOINTER(pointer,
      (Acknowledgement,acknowledgement({ackDest=AS,...})))) = AS
      | getAckDestPD(x) = raise PDDSRException(x);
      fun getAckSrcPD(DSRPOINTER(pointer,
      (Acknowledgement,acknowledgement({ackSrc=AS,...})))) = AS
      | getAckSrcPD(x) = raise PDDSRException(x);
      fun getIdPD(DSRPOINTER(pointer,
      (Acknowledgement,acknowledgement({id=ID,...})))) = ID
      | getIdPD(DSRPOINTER(pointer,
      (AcknowledgementRequest,acknowledgementRequest({id=ID,...})))) = ID
      | getIdPD(DSRPOINTER(pointer,
      (RouteRequest,routeRequest({id=ID,...})))) = ID
      | getIdPD(x) = raise PDDSRException(x);
      fun getSalvCounPD(DSRPOINTER(pointer,
      (DSRSourceRoute,dsrSourceRoute({salvageCounter=SC,...})))) = SC
      | getSalvCounPD(DSRPOINTER(pointer,
      (RouteError,routeError({salvageCounter=SC,...})))) = SC
      | getSalvCounPD(x) = raise PDDSRException(x);
      fun getAckIdPD(x) = getIdPD(x);
      fun getIpId(x) = getId(x);
      fun getTargetPD(DSRPOINTER(pointer,
      (RouteRequest,routeRequest({target=SL,...}))))
      = SL
      | getTargetPD(x) = raise PDDSRException(x);
      fun getSegmLeftPD(DSRPOINTER(pointer,
      (DSRSourceRoute,dsrSourceRoute({segmentsLeft=SL,...}))))
      = SL
      | getSegmLeftPD(x) = raise PDDSRException(x);
      fun getRoutePD(DSRPOINTER(pointer,
      (DSRSourceRoute,dsrSourceRoute({route=nodeList,...})))) = nodeList
      | getRoutePD(DSRPOINTER(pointer,
      (RouteReply,routeReply({route=nodeList,...})))) = nodeList
      | getRoutePD(DSRPOINTER(pointer,
      (RouteRequest,routeRequest({routeSoFar=nodeList,...})))) = nodeList
      | getRoutePD(x) = raise PDDSRException(x);
      fun unHuhMRR(REQRES(n1,n2,res)::mrrhl: MapRequestResultHuhList): NodexNodeResultList
      = let val (src,destresults) = unHuhMRR(mrrhl)
          in (n1,(n2,res)::destresults)
          end
      | unHuhMRR(REQ(n1,n2)::mrrhl) = raise UnHuhMRRDSRException(REQ(n1,n2)::mrrhl)
      | unHuhMRR([]) = (broadcastAddress,[]);
      fun containsDups (node::nodes)
```

```
= (List.exists (fn x => (node = x)) nodes) orelse
    containsDups(nodes)
| containsDups ([]) = false;
fun getSalvageCount(p as {id=id, ttl=ttl, src=src, dest=dest,
      dsr=(DSRSourceRoute, dsrSourceRoute
           {salvageCounter=sc, segmentsLeft=sl, route=rt})::dsrs,
      debug=debug, data=data}: IPPacket): SalvageCounter
= sc
| getSalvageCount(p) = raise GetSalvageCountDSRException(p);
fun setSalvageCount(p as {id=id, ttl=ttl, src=src, dest=dest,
       dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter=sc,
       segmentsLeft=sl, route=rt})::dsrs, debug=debug, data=data}: IPPacket,
       newSc: SegmentsLeft): IPPacket
= {id=id, ttl=ttl, src=src, dest=dest, dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter
       =newSc, segmentsLeft=sl, route=rt})::dsrs, debug=debug, data=data}
| setSalvageCount(p,newSc) = raise SetSalvageCountDSRException(p);
fun setRoute(p as {id=id, ttl=ttl, src=src, dest=dest,
       dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter=sc,
       segmentsLeft=sl, route=rt})::dsrs, debug=debug, data=data}: IPPacket,
       newRoute: Route): IPPacket
= {id=id, ttl=ttl, src=src, dest=dest, dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter
       =sc, segmentsLeft=sl, route=newRoute})::dsrs, debug=debug, data=data}
| setRoute(p,newRoute) = raise SetRouteDSRException(p);
fun setSegmentsLeft(p as {id=id, ttl=ttl, src=src, dest=dest,
       dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter=sc,
       segmentsLeft=sl, route=rt})::dsrs, debug=debug, data=data}: IPPacket,
       newSL: SegmentsLeft): IPPacket
= {id=id, ttl=ttl, src=src, dest=dest, dsr=(DSRSourceRoute, dsrSourceRoute{salvageCounter
       =sc, segmentsLeft=newSL, route=rt})::dsrs, debug=debug, data=data}
| setSegmentsLeft(p,newSL) = raise SetSegmentsLeftDSRException(p);
fun removeRoutesOfLen (x: int, (r,t)::pr: PossibleRoutes): PossibleRoutes
= if List.length(r) = x
   then removeRoutesOfLen(x,pr)
   else (r,t)::removeRoutesOfLen(x,pr)
| removeRoutesOfLen(x,[]) = [];
fun routeOfLenExists (x:int, (r,t)::pr: PossibleRoutes): bool
= if List.length(r) = x
   then true
   else routeOfLenExists(x,pr)
| routeOfLenExists(x,[]) = false;
fun decreaseBLS(NoStatus: BlacklistStatus): BlacklistStatus
= NoStatus
| decreaseBLS(Probable) = Questionable
| decreaseBLS(Questionable) = NoStatus;
fun notNoStatus(NoStatus: BlacklistStatus): bool
= false
| notNoStatus(bls) = true;
fun addDSRSourceRouteOption
     (ippacket as {id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs,
        debug=dth, data=data}: IPPacket,
      theroute : Route, salvage: SalvageCounter) : IPPacket
= let
    val segmentsLeft = List.length theroute;
    val addressList = theroute;
  in
    setDsrs((DSRSourceRoute, dsrSourceRoute ({salvageCounter=salvage,
                  segmentsLeft=segmentsLeft, route=addressList}))
              ::(getDsrs ippacket),
              ippacket)
  end;
fun addRouteRequestOption
       (ippacket as {id=id2, ttl=ttl, src=src, dest=dest2, dsr=dsrs,
           debug=dth, data=data}: IPPacket,
        id: Identification, dest: Node_) : IPPacket
= setDsrs((getDsrs ippacket)^^[(RouteRequest, routeRequest
                          ({id=id, target=dest, routeSoFar=[src]}))], ippacket);
fun addNodeToRouteRequestAtPointer({id=id2, ttl=ttl, src=src, dest=dest,
       dsr=ds, debug=dh, data=userdata}:IPPacket,
       node: Node_, pointer: INT) : IPPacket
= let fun antrrat((RouteRequest, routeRequest({id=id, target=node2,
              routeSoFar=nodeList}))::ds, node, 0) =
       (RouteRequest, routeRequest({id=id, target=node2,
              routeSoFar=nodeList^^[node]}))::ds
    |  antrrat(d::ds, node, pointer) = d::(antrrat(ds, node, pointer-1))
    |  antrrat(x, node, pointer)
       = raise AddNodeToRouteRequestAtPointerDSRException
         ({dsrheaders=x, node=node, pointer=pointer})
              (* including the case where x = [] or
```

```
                        pointer=0 but x's first elm is not a routereq *)
    in
      {id=id2, ttl=ttl, src=src, dest=dest, dsr=antrrat(ds,node,pointer),
       debug=dh, data=userdata}
  end;
fun noOfDSROptions ({id=id, ttl=ttl, src=src, dest=dest,
                     dsr=dsr::dsrs, debug=dth, data=data}: IPPacket): INT
= 1 + (noOfDSROptions({id=id, ttl=ttl, src=src, dest=dest,
                       dsr=dsrs, debug=dth, data=data}))
| noOfDSROptions ({id=id, ttl=ttl, src=src, dest=dest, dsr=[],
                  debug=dth, data=data}) = 0;
fun extractDSROptionNo ({id=id, ttl=ttl, src=src, dest=dest,
        dsr=dsr::dsrs, debug=dh, data=ud}: IPPacket, 0: INT) : DSROptionsHeader
= dsr
| extractDSROptionNo ({id=id, ttl=ttl, src=src, dest=dest, dsr=[], debug=dh, data=ud},n)
= raise ExtractDSROptionNoDSRException({packet={id=id, ttl=ttl, src=src, dest=dest,
                  dsr=[], debug=dh, data=ud}, pointer=n})
| extractDSROptionNo ({id=id, ttl=ttl, src=src, dest=dest, dsr=dsr::dsrs, debug=dh, data=ud},n)
= extractDSROptionNo ({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=dh, data=ud},n-1);
fun extractDSRPointer (p: IPPacket, pointer: INT)
= if pointer >= noOfDSROptions(p)
then NOMOREHEADERS
else DSRPOINTER(pointer, extractDSROptionNo(p,pointer));
fun extractDSROptionFixedPart ((f,v): DSROptionsHeader): DSROptionsHeaderFixed = f;
fun decrementSegmentsLeftInDSRSourceRouteOption(((DSRSourceRouteOption,
    (dsrSourceRoute ({salvageCounter=salvageCounter, segmentsLeft=segmentsLeft,
    route=nodeList})))::dsrs): DSROptionsHeaderList): DSROptionsHeaderList
  = (DSRSourceRouteOption,(dsrSourceRoute ({salvageCounter=salvageCounter,
      segmentsLeft=segmentsLeft-1,route=nodeList})))::
      (decrementSegmentsLeftInDSRSourceRouteOption dsrs)
  | decrementSegmentsLeftInDSRSourceRouteOption (dsr::dsrs)
    = dsr::(decrementSegmentsLeftInDSRSourceRouteOption dsrs)
  | decrementSegmentsLeftInDSRSourceRouteOption [] = [];
fun decrementSegmentsLeftInDSRSourceRouteOptionP
       ({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=ud}: IPPacket): IPPacket
= {id=id, ttl=ttl, src=src, dest=dest, dsr=decrementSegmentsLeftInDSRSourceRouteOption(dsrs),
       debug=debug, data=ud};
fun decreaseTimeoutsPTL((p,t)::ptl: IPPacketTimeList): IPPacketTimeList
= (p,t-1)::(decreaseTimeoutsPTL(ptl))
| decreaseTimeoutsPTL([]) = [];
fun increaseTimestampsRT((r,t)::rts: PossibleRoutes): PossibleRoutes
= (r,t+1)::increaseTimestampsRT(rts)
| increaseTimestampsRT([]) = [];
fun lowestTimeInRC((r,t)::prs: PossibleRoutes): Timestamp
= min(t,lowestTimeInRC(prs))
| lowestTimeInRC([]) = RouteCacheTimeout * 10; (* Bigger than everything else *)
fun findRouteWithTimeInRC((r,t)::prs: PossibleRoutes, t2: Timestamp): Route
= if t = t2
   then r
   else findRouteWithTimeInRC(prs,t2)
| findRouteWithTimeInRC([],t2) = raise FindRouteWithTimeInRCDSRException(t2);
fun bestRouteFromRC(prs: PossibleRoutes): Route
= findRouteWithTimeInRC(prs,lowestTimeInRC(prs));
(* This implementation favours the newest discovered or used route *)
fun updateBestRouteFromRC((r,t)::prs: PossibleRoutes): PossibleRoutes
= if t = lowestTimeInRC((r,t)::prs)
   then (r,0)::prs (* No recursive call - only first best route is updated *)
   else (r,t)::updateBestRouteFromRC(prs)
| updateBestRouteFromRC([]) = [];
fun removeDSRHeaderOption({id=id, ttl=ttl, src=src, dest=dest,
          dsr=dsrs, debug=dth, data=userData}: IPPacket, d: DSROptionsHeader) : IPPacket
= let
    fun rdho [] = []
      | rdho (dsr::dsrs) = if d = dsr
                           then (rdho dsrs)
                           else dsr::(rdho dsrs)
  in
    {id=id, ttl=ttl, src=src, dest=dest, dsr=rdho dsrs, debug=dth, data=userData}
  end;
fun makeRouteRequestBroadcastPacket({id=id2, ttl=ttl2, src=source, dest=dest,
                dsr=dsr, debug=dh, data=userdata}: IPPacket,
                newIpid: Identification, ttl: TTL, id: Identification,
                debug: DebugHeader,
                howtosendpacket: HowToSendPackets,
                origdsrs: DSROptionsHeaderList, origuserdata: UserData): IPPacket
= addRouteRequestOption({id=newIpid, ttl=ttl, src=source, dest=broadcastAddress,
                dsr=(if howtosendpacket = includeInPacket
                     then origdsrs
```

```
                        else []),
                debug=debug,
                data=(if howtosendpacket = includeInPacket
                        then origuserdata
                        else NODATA)},
                id, dest);
fun findThisNodeInChain (thisNode: Node_, []: Route): INT
        = raise FindThisNodeInChainDSRException(thisNode)
    | findThisNodeInChain (thisNode, node::nodes)
        = if node = thisNode
          then 0
          else 1 + (findThisNodeInChain (thisNode, nodes));
fun getPreListsOfList ([]: Route): Route ms
        = []
  | getPreListsOfList (xs)
        = 1`xs ++ getPreListsOfList(allbutlastelm xs);
fun getRoutesAndPossiblyReversedRoutes (thisNode : Node_,
                    nodeLists : Route ms) : Route ms
        = case networkStatus
            of FrequentlyUnidirLinks => nodeLists
             | MostlyBidirLinks => nodeLists ++ (ext_col List.rev nodeLists)
             | BidirOnlyLinks => nodeLists ++ (ext_col List.rev nodeLists);
fun hasRouteReply ([]: DSROptionsHeaderList): bool = false
   | hasRouteReply ((RouteReply, routeReply y)::dsrs) = true
   | hasRouteReply (dsr::dsrs) = hasRouteReply dsrs;
fun nodeListToLastSenderIncl (segmentsLeft: SegmentsLeft, origNodeList: Route): Route
(* (nodeListToLastSenderIncl(2,[a,b,c,d,e]) => [a,b,c] *)
= let fun nltlsi (0, nodeList) = []
        | nltlsi (nodeNo, node::nodeList) = node::(nltlsi(nodeNo-1, nodeList))
        | nltlsi (nodeNo, []) = raise NodeListToLastSenderInclDSRException
                    ({segmleft=segmentsLeft,nodelist=origNodeList});
   in  nltlsi (List.length(origNodeList)-segmentsLeft (* +1 *), origNodeList)
end;
fun extractPossibleRoutes ((RouteRequest : DSROptionsHeaderFixed,
          routeRequest ({id=identification, target=destNode,
                  routeSoFar=nodeList})): DSROptionsHeader,
          p: IPPacket, thisNode: Node_) : Route ms
= if networkStatus = FrequentlyUnidirLinks orelse networkStatus = MostlyBidirLinks
   then 1`(nodeList^^[thisNode])
   else empty (* BidirOnly *)
| extractPossibleRoutes ((RouteReply, routeReply ({route=nodeList})), p, thisNode)
  = (if networkStatus = FrequentlyUnidirLinks orelse networkStatus = MostlyBidirLinks
      then 1`(nodeList (* ^^[thisNode] *))
      else empty (* BidirOnly, covered by the DSRSourceRoute-case below *))
| extractPossibleRoutes ((Acknowledgement, acknowledgement
          ({id=id,ackSrc=src,ackDest=dest})), p, thisNode)
  = 1`([src,dest])
| extractPossibleRoutes ((DSRSourceRoute, dsrSourceRoute
          ({salvageCounter=salvage,segmentsLeft=segmentsLeft,
            route=nodeList})), p, thisNode)
  = (if (networkStatus = FrequentlyUnidirLinks orelse
         networkStatus = MostlyBidirLinks orelse
       (networkStatus = BidirOnlyLinks andalso (not (hasRouteReply(getDsrs(p))))))
     then 1`((if salvage = 0 then [getSrc(p)] else [])^^nodeList^^[(getDest p)])
     else 1`((if salvage = 0 then [getSrc(p)] else [])^^
          (nodeListToLastSenderIncl(segmentsLeft,nodeList)^^[thisNode]))
            (* BidirOnly andalso IPPacket contains Route Reply *))
    ++
    (if (networkStatus = FrequentlyUnidirLinks orelse networkStatus = MostlyBidirLinks)
     then 1`((if salvage = 0 then [getSrc(p)] else [])^^
          (nodeListToLastSenderIncl(segmentsLeft,nodeList))^^[thisNode])
     else empty (* BidirOnly *))
| extractPossibleRoutes (d, p, thisNode) = empty
  (* Acknowledgement Request + Route Error (+ mismatchs) *);
fun removeListSectionUntilElmExclElm ([]: Route, node: Node_) : Route
    = []
  | removeListSectionUntilElmExclElm (x::xs, node)
    = if x = node
      then x::xs
      else (removeListSectionUntilElmExclElm(xs,node));
fun removeListSectionFrom2ndElmIncl2ndElm (nodeList: Route, node: Node_) : Route
(* ([a,b,c,d,e,c,f,g,c,h,i],c) => [a,b,c,d,e] *)
= let fun rlsf2ei2e (node2::nodeList, node, 0)
        = if node2 <> node
          then node2::(rlsf2ei2e (nodeList, node, 0))
          else node2::(rlsf2ei2e (nodeList, node, 1))
        | rlsf2ei2e ([], node, 0) = []
        | rlsf2ei2e (node2::nodeList, node, 1)
```

```
            = if node2 <> node
                then node2::(rlsf2ei2e (nodeList, node, 1))
                else []
            | rlsf2ei2e ([], node, 1) = []
            | rlsf2ei2e (nl, n, x) = raise RemoveListSectionFrom2ndElmIncl2ndElmDSRException
                                          ({nodelist=nl, node=n, no=x})
        in rlsf2ei2e (nodeList, node, 0)
        end;
    fun removeListSectionUntil2ndElmExcl2ndElm(nodeList: Route, node: Node_) : Route
    (* ([a,b,c,d,e,c,f,g,c,h,i],c] => [c,f,g,c,h,i] *)
    = let fun rlsu2ee2e (node2::nodeList, node, 0)
             = if node2 <> node
                then rlsu2ee2e (nodeList, node, 0)
                else rlsu2ee2e (nodeList, node, 1)
            | rlsu2ee2e ([], node, 0) = []
            | rlsu2ee2e (node2::nodeList, node, 1)
              = if node2 <> node
                 then rlsu2ee2e (nodeList, node, 1)
                 else node2::nodeList
            | rlsu2ee2e ([], node, 1) = []
            | rlsu2ee2e (nl, n, x) = raise RemoveListSectionUntil2ndElmExcl2ndElmDSRException
                                      ({nodelist=nl, node=n, no=x})
      in rlsu2ee2e (nodeList, node, 0)
      end;
    fun findSubsectionsOfNodeListStartingWithNode(nodeList: Route, node: Node_): Route ms
    = let val listFromHere = removeListSectionUntilElmExclElm(nodeList,node);
          fun subsections ([],node) : Route ms = empty
            | subsections (nodeList, node)
               = 1`(removeListSectionFrom2ndElmIncl2ndElm (nodeList,node)) ++
                 (subsections(removeListSectionUntil2ndElmExcl2ndElm(nodeList,node),node))
      in subsections (listFromHere, node)
      end;
    fun returnUntilExclSpecificNode ([]: Route, node: Node_): Route = []
      | returnUntilExclSpecificNode (node2::nodeList, node)
        = if node2 = node
          then []
          else node2::(returnUntilExclSpecificNode(nodeList,node));
    fun returnUntilDupNode ([]: Route): Route = []
      | returnUntilDupNode (node::nodeList)
        (* [a,b,c,d,e,c,f,g,c,h,i] => [a,b,c,d,e] *)
        = if List.exists (fn x => x = node) nodeList
          then node::(returnUntilExclSpecificNode(nodeList,node))
          else node::(returnUntilDupNode(nodeList));
    fun removeIfLengthLessThan(len, theList)
    = if List.length(theList) >= len
      then 1`theList
      else empty;
    fun returnRelevantRoutes (thisNode : Node_, dsrHeaderOption: DSROptionsHeader,
                              ipPacket: IPPacket) : RouteList
    = (ms_to_list
        (ext_ms (fn aList => (removeIfLengthLessThan(2,aList)))
          (ext_ms getPreListsOfList
                  (ext_col returnUntilDupNode
                           (ext_ms (fn x => findSubsectionsOfNodeListStartingWithNode
                                                (x,thisNode))
                                   (getRoutesAndPossiblyReversedRoutes
                                     (thisNode,
                                      extractPossibleRoutes
                                      (dsrHeaderOption,ipPacket,
                                       thisNode)))))))))
    fun intendedRecipient ({id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,
                    debug=debug,data=ud}: IPPacket, 0: INT, nodeList: Route): Node_ = dest
      | intendedRecipient (p,1,nodeList) =
        (case lastelm(nodeList) of SOME n => n
          | NONE => getSrc(p))
      | intendedRecipient (p,n,nodeList) = intendedRecipient(p,n-1,allbutlastelm(nodeList));
    fun intendedRecipientP(p as {id=id,ttl=ttl,src=src,dest=dest,dsr=(DSRSourceRoute,
                dsrSourceRoute({salvageCounter=sc,segmentsLeft=sl,route=rt}))::dsrs,
                debug=db,data=data}: IPPacket): Node_
    = intendedRecipient(p,sl,rt)
    | intendedRecipientP (p) = raise IntendedRecipientPDSRException(p);
    fun intendedRecipientPS(p as {id=id,ttl=ttl,src=src,dest=dest,dsr=(DSRSourceRoute,
                dsrSourceRoute({salvageCounter=sc,segmentsLeft=sl,route=rt}))::dsrs,
                debug=db,data=data}: IPPacket, offset: INT): Node_
    = intendedRecipient(p,sl+offset,rt)
    | intendedRecipientPS (p, offset2)
    = raise IntendedRecipientPSDSRException({packet=p,offset=offset2});
    fun DSROptionsAlikeExceptForSegmentsLeftAndStuff(
```

```
            (DSRSourceRoute, dsrSourceRoute({salvageCounter=salvage1,
               segmentsLeft=segmentsLeft1, route=nodeList1}))
             ::dsrs1: DSROptionsHeaderList,
            (DSRSourceRoute, dsrSourceRoute({salvageCounter=salvage2,
                segmentsLeft=segmentsLeft2, route=nodeList2}))
             ::dsrs2: DSROptionsHeaderList): bool
      = (salvage1 = salvage2) andalso (nodeList1 = nodeList2) andalso
          (segmentsLeft1+1 = segmentsLeft2) andalso
          DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1, dsrs2)
      | DSROptionsAlikeExceptForSegmentsLeftAndStuff(
            (AcknowledgementRequest, acknowledgementRequest{id=id})::dsrs1, dsrs2)
        = DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1, dsrs2)
      | DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1,
            (AcknowledgementRequest, acknowledgementRequest{id=id})::dsrs2)
        = DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1, dsrs2)
      | DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsr1::dsrs1, dsr2::dsrs2)
         = (dsr1 = dsr2) andalso
            DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1, dsrs2)
      | DSROptionsAlikeExceptForSegmentsLeftAndStuff([],[]) = true
      | DSROptionsAlikeExceptForSegmentsLeftAndStuff(x,y) = false;
      fun removePacketFromWaitingPackets
              ({id=id1, ttl=ttl1, src=src1, dest=dest1, dsr=dsrs1,
                 debug=debug1, data=ud1}: IPPacket,
            (((prt as ({id=id2, ttl=ttl2, src=src2, dest=dest2, dsr=dsrs2,
                 debug=debug2, data=ud2},
               rc, ts))::prtl)) : WaitingpacketList): WaitingpacketList
      = if (id1 = id2 andalso
            (* This is the reason, we included the id-field in our IP-packet *)
            ttl1+1 = ttl2 andalso
            src1 = src2 andalso
            dest1 = dest2 andalso
            DSROptionsAlikeExceptForSegmentsLeftAndStuff(dsrs1, dsrs2) andalso
            (* No, we mustn't check debug/debug2! *)
            ud1 = ud2)
      then prtl
      else prt::prtl
      | removePacketFromWaitingPackets(p,[]) = [];
      fun getSegmentsLeft ({id=id, ttl=ttl, src=src, dest=dest,
          dsr=(DSRSourceRoute, dsrSourceRoute({salvageCounter=sc,
          segmentsLeft=sl, route=rt}))::dsrs, debug=debug, data=data}: IPPacket): SegmentsLeft
      = sl
      | getSegmentsLeft p = raise GetSegmentsLeftDSRException(p);
      fun addAckReq({id=id2, ttl=ttl, src=src, dest=dest, dsr=dsrs,
                 debug=debug, data=data}: IPPacket, id: Identification): IPPacket
      = {id=id2, ttl=ttl, src=src, dest=dest, dsr=dsrs^^[(AcknowledgementRequest,
                 acknowledgementRequest({id=id}))], debug=debug, data=data};
      fun addAck(p as {id=id2, ttl=ttl, src=src, dest=dest, dsr=dsr, debug=debug,
                  data=data}: IPPacket, id: Identification, src2: Node_, dest2: Node_): IPPacket
      = {id=id2, ttl=ttl, src=src, dest=dest, dsr=dsr^^[(Acknowledgement,
                      acknowledgement({id=id, ackSrc=src2, ackDest=dest2}))], debug=debug, data=data};
      fun removeAckReq(p as {id=id2, ttl=ttl, src=src, dest=dest, dsr=(AcknowledgementRequest,
                acknowledgementRequest({id=id}))::dsrs, debug=debug, data=data}: IPPacket): IPPacket
      = removeAckReq({id=id2, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=data})
      | removeAckReq(p as {id=id, ttl=ttl, src=src, dest=dest, dsr=dsr::dsrs, debug=debug,
                data=data}: IPPacket): IPPacket
      = let val {id=id2, ttl=ttl2, src=src2, dest=dest2, dsr=dsr2, debug=debug2, data=data2} =
            removeAckReq({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=data})
        in {id=id2, ttl=ttl2, src=src2, dest=dest2, dsr=dsr::dsr2, debug=debug2, data=data2}
        end
      | removeAckReq(p as {id=id, ttl=ttl, src=src, dest=dest, dsr=[], debug=debug, data=data}) = p;
      fun removeDSRSourceRoute(p as {id=id, ttl=ttl, src=src, dest=dest, dsr=(DSRSourceRoute,
                dsrSourceRoute({salvageCounter=sc, segmentsLeft=sl, route=rt}))::dsrs,
                debug=debug, data=data}: IPPacket): IPPacket
      = removeDSRSourceRoute({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=data})
      | removeDSRSourceRoute(p as {id=id, ttl=ttl, src=src, dest=dest, dsr=dsr::dsrs,
                debug=debug, data=data}: IPPacket): IPPacket
      = let val {id=id, ttl=ttl2, src=src2, dest=dest2, dsr=dsr2, debug=debug2, data=data2} =
            removeDSRSourceRoute({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=data})
        in {id=id, ttl=ttl2, src=src2, dest=dest2, dsr=dsr::dsr2, debug=debug2, data=data2}
        end
      | removeDSRSourceRoute(p as {id=id, ttl=ttl, src=src, dest=dest, dsr=[], debug=debug, data=data}) = p;
      fun containsAck({id=id2, ttl=ttl, src=src, dest=dest, dsr=(Acknowledgement,
                    acknowledgement({id=id, ackSrc=ackSrc, ackDest=ackDest}))::dsrs,
                               debug=debug, data=data}: IPPacket) : bool = true
      | containsAck({id=id, ttl=ttl, src=src, dest=dest, dsr=dsr::dsrs, debug=debug, data=data})
          = containsAck({id=id, ttl=ttl, src=src, dest=dest, dsr=dsrs, debug=debug, data=data})
      | containsAck({id=id, ttl=ttl, src=src, dest=dest, dsr=[], debug=debug, data=data}) = false;
      fun containsAckReq({id=id3, ttl=ttl, src=src, dest=dest, dsr=(AcknowledgementRequest,
```

```
            acknowledgementRequest{id=id1}::dsrs,debug=debug,data=data} : IPPacket,
            id2: Identification) : bool
    = if id1=id2
       then true
       else containsAckReq({id=id3,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=debug,data=data},id2)
    | containsAckReq({id=id3,ttl=ttl,src=src,dest=dest,dsr=dsr::dsrs,debug=debug,data=data},id2)
      = containsAckReq({id=id3,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=debug,data=data},id2)
    | containsAckReq({id=id3,ttl=ttl,src=src,dest=dest,dsr=[],debug=debug,data=data},id2) = false;
    fun containsAnyAckReq({id=id,ttl=ttl,src=src,dest=dest,dsr=
        (AcknowledgementRequest,x)::dsrs,debug=debug,data=data}: IPPacket): bool = true
    | containsAnyAckReq({id=id,ttl=ttl,src=src,dest=dest,dsr=dsr::dsrs,
        debug=debug,data=data}) = containsAnyAckReq({id=id,ttl=ttl,src=src,
        dest=dest,dsr=dsrs,debug=debug,data=data})
    | containsAnyAckReq({id=id,ttl=ttl,src=src,dest=dest,dsr=[],debug=debug,
         data=data}) = false;
    fun elimAckReqs ((p,r,t)::ptl2: WaitingpacketList, id: Identification) : WaitingpacketList
    = if containsAckReq(p,id) then elimAckReqs(ptl2,id) else (p,r,t)::elimAckReqs(ptl2,id)
    | elimAckReqs ([], id) = [];
    fun countdownTimeoutMB((p,r,t)::prtl : WaitingpacketList) : WaitingpacketList
    = (p,r,t-1)::countdownTimeoutMB(prtl)
    | countdownTimeoutMB([]) = [];
    fun getSalvageCounter({id=id,ttl=ttl,src=src,dest=dest,dsr=(DSRSourceRoute,dsrSourceRoute{
            salvageCounter=sc,segmentsLeft=sl,route=rt})::dsrs,debug=debug,
            data=data}: IPPacket): SalvageCounter
    = sc
    | getSalvageCounter(p) = raise GetSalvageCounterDSRException(p);
    fun getFirstLink({id=id,ttl=ttl,src=src,dest=dest,dsr=(DSRSourceRoute,dsrSourceRoute{
            salvageCounter=sc,segmentsLeft=sl,route=[]})::dsrs,debug=debug,
            data=data}: IPPacket): Node_
    = dest
    | getFirstLink({id=id,ttl=ttl,src=src,dest=dest,dsr=(DSRSourceRoute,dsrSourceRoute{
            salvageCounter=sc,segmentsLeft=sl,route=node::nodes})::dsrs,debug=debug,
            data=data})
    = node
    | getFirstLink(p) = raise GetFirstLinkDSRException(p);
    fun getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,dsr=(Acknowledgement,
          varAck)::dsrs,debug=debug,data=data}: IPPacket): DSROptionsHeaderList
       = (Acknowledgement,varAck)::getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,
          dsr=dsrs,debug=debug,data=data})
    | getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,dsr=(RouteError,
         varRouteError)::dsrs,debug=debug,data=data})
       = (RouteError,varRouteError)::getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,
          dsr=dsrs,debug=debug,data=data})
    | getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,dsr=dsr::dsrs,debug=debug,
          data=data})
       = getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,dsr=dsrs,debug=debug,data=data})
    | getAcksAndRouteErrors({id=id,ttl=ttl,src=src,dest=dest,dsr=[],debug=debug,data=data}) = [];
    fun makeLinkForAllNodes(n1: Node_,n2: Node_): NodeNodeList
    = List.map (fn x => (x, n2)) (allNodes())
    fun makeLinkForAllNodes2(node: Node_, link1: Node_, link2: Node_): NodeNodeNodeNode ms
    = ext_col (fn x => (node,x,link1,link2)) (allNodes())
    fun makeLinkForAllNodes3(src: Node_, failLink: Node_): NodeNodeNodeList
    = List.map (fn x => (x,src,failLink)) (allNodes())
    fun existsLinkInRoute(fail1: Node_, fail2: Node_, node1::node2::route: Route): bool
    = if node1 = fail1 andalso node2 = fail2
       then true
       else existsLinkInRoute(fail1,fail2,node2::route)
    | existsLinkInRoute(x,fail1,fail2) = false; (* where x has 0 or 1 elms *)
    fun removeRouteWithLink (src:Node_, dest: Node_, route: Route,
                  fail1: Node_, fail2: Node_): RouteList
    = if existsLinkInRoute(fail1,fail2,src::(route^^[dest]))
       then []
       else [route];
    fun existsLinkInPacket(link1: Node_,link2: Node_,{id=id,ttl=ttl,src=src,dest=dest,
            dsr=(DSRSourceRoute,dsrSourceRoute({salvageCounter=sc,segmentsLeft=sl,
            route=rt}))::dsrs,debug=debug,data=data}: IPPacket): bool
    = existsLinkInRoute(link1,link2,src::(rt^^[dest]))
    | existsLinkInPacket(link1,link2,p) = false;
    fun extractPacketsWithLinkFromMB(link1: Node_, link2: Node_,
                (p,r,t)::prtl: WaitingpacketList): IPPacketList
    = if existsLinkInPacket(link1,link2,p)
       then p::(extractPacketsWithLinkFromMB(link1,link2,prtl))
       else extractPacketsWithLinkFromMB(link1,link2,prtl)
    | extractPacketsWithLinkFromMB(link1,link2,[]) = [];
    fun removePacketsWithLinkFromMB(link1: Node_, link2: Node_,
                (p,r,t)::prtl: WaitingpacketList): WaitingpacketList
    = if existsLinkInPacket(link1,link2,p)
       then removePacketsWithLinkFromMB(link1,link2,prtl)
```

```
    else (p,r,t)::(removePacketsWithLinkFromMB(link1,link2,prt1))
| removePacketsWithLinkFromMB(link1,link2,[]) = [];
fun extractPacketsWithLinkFromNIQ(link1: Node_, link2: Node_,
            (n,p)::rl: NetworkInterfaceQueueForANode, dest: Node_): IPPacketList
= if existsLinkInPacket(link1,link2,p) andalso intendedRecipientP(p) = dest
    then p::(extractPacketsWithLinkFromNIQ(link1,link2,rl,dest))
    else extractPacketsWithLinkFromNIQ(link1,link2,rl,dest)
| extractPacketsWithLinkFromNIQ(link1,link2,[],dest) = [];
fun removePacketsWithLinkFromNIQ(link1: Node_, link2: Node_,
          (n,p)::rl: NetworkInterfaceQueueForANode,
          dest: Node_): NetworkInterfaceQueueForANode
= if existsLinkInPacket(link1,link2,p) andalso intendedRecipientP(p) = dest
    then removePacketsWithLinkFromNIQ(link1,link2,rl,dest)
    else (n,p)::(removePacketsWithLinkFromNIQ(link1,link2,rl,dest))
| removePacketsWithLinkFromNIQ(link1,link2,[],dest) = [];
fun removeLinkFromRC(src: Node_, dest: Node_, (route, timeout)::rc: PossibleRoutes,
                     fail1: Node_, fail2: Node_): PossibleRoutes
= (List.map (fn x => (x,timeout)) (removeRouteWithLink(src,dest,route,fail1,fail2)))^^
                     removeLinkFromRC(src,dest,rc,fail1,fail2)
| removeLinkFromRC(src,dest,[],fail1,fail2) = [];
fun inIdNodeList(id: Identification, node: Node_, (id2,node2)::iddests: IdNodeList): bool
= if id = id2 andalso node = node2
    then true
    else inIdNodeList(id,node,iddests)
| inIdNodeList(id,node,[]) = false;
fun combineRequestPacketAndResultLists
      ((src1,destresults): NodexNodeResultList,
       (src2,dests,p): NodeNodeListPacket): NodeNodeResultPacketList
= ListPair.map (fn ((d1,r1),(d2)) =>
       if d1 <> d2
       then raise CombineResultListsAndCopyDebugDSRException
                  ({NNRL=(src1,destresults), NNLP=(src2,dests,p)})
       else (src1,d1,r1,copyNormalHeaderToDebugHeader(src1,d1,p))
    )
    (destresults,dests);
fun filterResultDests x =
(List.map (fn (n1,n2,result,p) => n2)
  (List.filter (fn (n1,n2,result,p) => result) x));
fun makeSalvagedPacket (node,p,sc,route): IPPacket
= removeAckReq
     (setSegmentsLeft
       (setRoute
         (setSalvageCount
           (p,sc),
          [node]^^route),
        List.length(route)))
fun makeRouteErrorPacket (p,newIpid,node,dest) : IPPacket
={id=newIpid,ttl=stdTTL,src=node,
   dest=(if getSalvageCounter(p)>0 then getFirstLink(p) else getSrc(p)),
   dsr=(RouteError,routeError({errorType=NODE_UNREACHABLE,
              salvageCounter=0,errorSrc=node,
              errorDest=(if getSalvageCounter(p)>0 then getFirstLink(p) else getSrc(p)),
              typeSpec=unreachableNodeAddress(dest)}))::
         getAcksAndRouteErrors(p),debug=getDebug(p),data=NODATA}
fun makeRouteErrorPacketForFirstLinkInP (p,newIpid,node) : IPPacket
= {id=newIpid,ttl=stdTTL,
     src=node,dest=getDest(p),
     dsr=(RouteError,routeError
       ({errorType=NODE_UNREACHABLE,
          salvageCounter=0,errorSrc=node,
          errorDest=getDest(p),
          typeSpec=unreachableNodeAddress
                     (getFirstLink(p))}))::
       getAcksAndRouteErrors(p),
     debug=getDebug(p),data=NODATA};
fun makeRouteReplyPacket (newIpid,src,dest,debug,route): IPPacket
={id=newIpid,ttl=stdTTL,src=src,dest=dest,
    dsr=[(RouteReply,routeReply({route=route}))],
    debug=debug,data=NODATA}
fun makeAckPacket(ipid,ackid,src,dest,debug): IPPacket
=addAck({id=ipid,ttl=stdTTL,src=src,dest=dest,
                 dsr=[(DSRSourceRoute,dsrSourceRoute
                      {salvageCounter=0, segmentsLeft=0,route=[]})],
                debug=debug,data=NODATA},
               ackid,src,dest)
```

C

Undersøgelser af samtidigt igangsatte "Route Discoveries"

C.1 Kommandolisten brugt til undersøgelserne i afsnit 3.4.6

```
val eventChainToRun =
  1`("A",[MoveNode{toPos=(200,200)},
          SendPacket{toNode="B",id=1,data=SOMEDATA
                     ("pakke 1 fra A til B")},
          WaitForPacket{id=2}]) ++

  1`("C",[MoveNode{toPos=(200,400)}]) ++

  1`("D",[MoveNode{toPos=(200,600)}]) ++

  1`("E",[MoveNode{toPos=(200,800)}]) ++

  1`("B",[MoveNode{toPos=(200,1000)},
          SendPacket{toNode="A",id=2,data=SOMEDATA
                     ("pakke 2 fra B til A")},
          WaitForPacket{id=1}])
```

C.2 Pakketransmissioner fra undersøgelserne i afsnit 3.4.6

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til "Frequently-unidir" uden nogen udvidelser medtaget:

Bemærk: Simulationen blev stoppet efter 2.000.000 steps.

```
(1,"A",["C"],{id=50037,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=48232,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})
```

```
(2,"B",["E"],{id=18455,ttl=255,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=66137,
                                               target="A",
                                               routeSoFar=["B"]}))],
              debug={debugId=2,...},
              data=NODATA})

(3,"C",["D","A"],{id=50037,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=48232,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"E",["D","B"],{id=18455,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=66137,
                                                   target="A",
                                                   routeSoFar=["B","E"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(5,"D",["E","C"],{id=50037,ttl=253,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=48232,
                                                   target="B",
                                                   routeSoFar=["A","C","D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(6,"D",["E","C"],{id=18455,ttl=253,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=66137,
                                                   target="A",
                                                   routeSoFar=["B","E","D"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(7,"E",["D","B"],{id=50037,ttl=252,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=48232,
                                                   target="B",
                                                   routeSoFar=["A","C","D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(8,"C",["D","A"],{id=18455,ttl=252,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=66137,
                                                   target="A",
                                                   routeSoFar=["B","E","D","C"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(9,"A",["C"],{id=50039,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=48233,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(10,"C",["D","A"],{id=50039,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=48233,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(11,"B",["E"],{id=18457,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=66138,
                                                target="A",
                                                routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(12,"D",["E","C"],{id=50039,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=48233,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(13,"E",["D","B"],{id=18457,ttl=254,src="B",dest="Z",
```

```
                    dsr=[(RouteRequest,routeRequest({id=66138,
                                                     target="A",
                                                     routeSoFar=["B","E"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(14,"E",["D","B"],{id=50039,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48233,
                                                     target="B",
                                                     routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(15,"D",["E","C"],{id=18457,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66138,
                                                     target="A",
                                                     routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(16,"C",["D","A"],{id=18457,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66138,
                                                     target="A",
                                                     routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(17,"A",["C"],{id=50041,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48234,
                                                 target="B",
                                                 routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(18,"C",["D","A"],{id=50041,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48234,
                                                     target="B",
                                                     routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(19,"D",["E","C"],{id=50041,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48234,
                                                     target="B",
                                                     routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(20,"E",["D","B"],{id=50041,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48234,
                                                     target="B",
                                                     routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(21,"B",["E"],{id=18460,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66139,
                                                 target="A",
                                                 routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(22,"E",["D","B"],{id=18460,ttl=254,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66139,
                                                     target="A",
                                                     routeSoFar=["B","E"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(23,"D",["E","C"],{id=18460,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66139,
                                                     target="A",
                                                     routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(24,"C",["D","A"],{id=18460,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66139,
```

```
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(25,"A",["C"],{id=50043,ttl=255,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48235,
                                                    target="B",
                                                    routeSoFar=["A"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(26,"C",["D","A"],{id=50043,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48235,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(27,"D",["E","C"],{id=50043,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48235,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(28,"E",["D","B"],{id=50043,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48235,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(29,"B",["E"],{id=18462,ttl=255,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66140,
                                                    target="A",
                                                    routeSoFar=["B"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(30,"E",["D","B"],{id=18462,ttl=254,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66140,
                                                    target="A",
                                                    routeSoFar=["B","E"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(31,"D",["E","C"],{id=18462,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66140,
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(32,"C",["D","A"],{id=18462,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66140,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(33,"A",["C"],{id=50045,ttl=255,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48236,
                                                    target="B",
                                                    routeSoFar=["A"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(34,"C",["D","A"],{id=50045,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48236,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(35,"D",["E","C"],{id=50045,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48236,
                                                    target="B",
```

```
                                                routeSoFar=["A","C","D"]}))],
                debug={debugId=1,...},
                data=NODATA})

(36,"E",["D","B"],{id=50045,ttl=252,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48236,
                                                target="B",
                                                routeSoFar=["A","C","D","E"]}))],
                debug={debugId=1,...},
                data=NODATA})

(37,"B",["E"],{id=18464,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=66141,
                                               target="A",
                                               routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(38,"E",["D","B"],{id=18464,ttl=254,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66141,
                                                target="A",
                                                routeSoFar=["B","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(39,"D",["E","C"],{id=18464,ttl=253,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66141,
                                                target="A",
                                                routeSoFar=["B","E","D"]}))],
                debug={debugId=2,...},
                data=NODATA})

(40,"C",["D","A"],{id=18464,ttl=252,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66141,
                                                target="A",
                                                routeSoFar=["B","E","D","C"]}))],
                debug={debugId=2,...},
                data=NODATA})

(41,"A",["C"],{id=50047,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=48237,
                                               target="B",
                                               routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(42,"C",["D","A"],{id=50047,ttl=254,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48237,
                                                target="B",
                                                routeSoFar=["A","C"]}))],
                debug={debugId=1,...},
                data=NODATA})

(43,"D",["E","C"],{id=50047,ttl=253,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48237,
                                                target="B",
                                                routeSoFar=["A","C","D"]}))],
                debug={debugId=1,...},
                data=NODATA})

(44,"E",["D","B"],{id=50047,ttl=252,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48237,
                                                target="B",
                                                routeSoFar=["A","C","D","E"]}))],
                debug={debugId=1,...},
                data=NODATA})

(45,"B",["E"],{id=18466,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=66142,
                                               target="A",
                                               routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(46,"E",["D","B"],{id=18466,ttl=254,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66142,
                                                target="A",
                                                routeSoFar=["B","E"]}))],
```

```
                    debug={debugId=2,...},
                    data=NODATA})

(47,"D",["E","C"],{id=18466,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66142,
                                                     target="A",
                                                     routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(48,"C",["D","A"],{id=18466,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66142,
                                                     target="A",
                                                     routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(49,"A",["C"],{id=50049,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=48238,
                                                 target="B",
                                                 routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(50,"C",["D","A"],{id=50049,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48238,
                                                     target="B",
                                                     routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(51,"D",["E","C"],{id=50049,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48238,
                                                     target="B",
                                                     routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(52,"E",["D","B"],{id=50049,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=48238,
                                                     target="B",
                                                     routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(53,"B",["E"],{id=18468,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=66143,
                                                 target="A",
                                                 routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(54,"E",["D","B"],{id=18468,ttl=254,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66143,
                                                     target="A",
                                                     routeSoFar=["B","E"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(55,"D",["E","C"],{id=18468,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66143,
                                                     target="A",
                                                     routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(56,"C",["D","A"],{id=18468,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=66143,
                                                     target="A",
                                                     routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til “Frequently-unidir” med både “Salvage Operations” og “Cached Route Reply” slået til:

Bemærk: Simulationen blev stoppet efter 2.000.000 steps.

```
(1,"B",["E"],{id=99459,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=37973,
                                                target="A",
                                                routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(2,"A",["C"],{id=86795,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=99346,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(3,"C",["D","A"],{id=86795,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99346,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(4,"E",["D","B"],{id=99459,ttl=254,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37973,
                                                    target="A",
                                                    routeSoFar=["B","E"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(5,"D",["E","C"],{id=86795,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99346,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(6,"D",["E","C"],{id=99459,ttl=253,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37973,
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(7,"E",["D","B"],{id=86795,ttl=252,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99346,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(8,"C",["D","A"],{id=99459,ttl=252,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37973,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(9,"A",["C"],{id=86797,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=99347,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(10,"C",["D","A"],{id=86797,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99347,
                                                     target="B",
                                                     routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(11,"D",["E","C"],{id=86797,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99347,
```

```
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(12,"E",["D","B"],{id=86797,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99347,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(13,"B",["E"],{id=99462,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37974,
                                                target="A",
                                                routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(14,"E",["D","B"],{id=99462,ttl=254,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=37974,
                                                    target="A",
                                                    routeSoFar=["B","E"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(15,"D",["E","C"],{id=99462,ttl=253,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=37974,
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(16,"C",["D","A"],{id=99462,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=37974,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(17,"A",["C"],{id=86799,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99348,
                                                target="B",
                                                routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(18,"C",["D","A"],{id=86799,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99348,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(19,"D",["E","C"],{id=86799,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99348,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(20,"E",["D","B"],{id=86799,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=99348,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(21,"B",["E"],{id=99464,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37975,
                                                target="A",
                                                routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(22,"E",["D","B"],{id=99464,ttl=254,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=37975,
                                                    target="A",
```

```
                                                routeSoFar=["B","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(23,"D",["E","C"],{id=99464,ttl=253,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37975,
                                                target="A",
                                                routeSoFar=["B","E","D"]}))],
                debug={debugId=2,...},
                data=NODATA})

(24,"C",["D","A"],{id=99464,ttl=252,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37975,
                                                target="A",
                                                routeSoFar=["B","E","D","C"]}))],
                debug={debugId=2,...},
                data=NODATA})

(25,"A",["C"],{id=86801,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99349,
                                                target="B",
                                                routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(26,"C",["D","A"],{id=86801,ttl=254,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99349,
                                                target="B",
                                                routeSoFar=["A","C"]}))],
                debug={debugId=1,...},
                data=NODATA})

(27,"D",["E","C"],{id=86801,ttl=253,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99349,
                                                target="B",
                                                routeSoFar=["A","C","D"]}))],
                debug={debugId=1,...},
                data=NODATA})

(28,"E",["D","B"],{id=86801,ttl=252,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99349,
                                                target="B",
                                                routeSoFar=["A","C","D","E"]}))],
                debug={debugId=1,...},
                data=NODATA})

(29,"B",["E"],{id=99466,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37976,
                                                target="A",
                                                routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(30,"E",["D","B"],{id=99466,ttl=254,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37976,
                                                target="A",
                                                routeSoFar=["B","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(31,"D",["E","C"],{id=99466,ttl=253,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37976,
                                                target="A",
                                                routeSoFar=["B","E","D"]}))],
                debug={debugId=2,...},
                data=NODATA})

(32,"C",["D","A"],{id=99466,ttl=252,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37976,
                                                target="A",
                                                routeSoFar=["B","E","D","C"]}))],
                debug={debugId=2,...},
                data=NODATA})

(33,"A",["C"],{id=86803,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99350,
                                                target="B",
                                                routeSoFar=["A"]}))],
```

```
                debug={debugId=1,...},
                data=NODATA})

(34,"C",["D","A"],{id=86803,ttl=254,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99350,
                                                 target="B",
                                                 routeSoFar=["A","C"]}))],
                debug={debugId=1,...},
                data=NODATA})

(35,"D",["E","C"],{id=86803,ttl=253,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99350,
                                                 target="B",
                                                 routeSoFar=["A","C","D"]}))],
                debug={debugId=1,...},
                data=NODATA})

(36,"E",["D","B"],{id=86803,ttl=252,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99350,
                                                 target="B",
                                                 routeSoFar=["A","C","D","E"]}))],
                debug={debugId=1,...},
                data=NODATA})

(37,"B",["E"],{id=99468,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37977,
                                                 target="A",
                                                 routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(38,"E",["D","B"],{id=99468,ttl=254,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37977,
                                                 target="A",
                                                 routeSoFar=["B","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(39,"D",["E","C"],{id=99468,ttl=253,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37977,
                                                 target="A",
                                                 routeSoFar=["B","E","D"]}))],
                debug={debugId=2,...},
                data=NODATA})

(40,"C",["D","A"],{id=99468,ttl=252,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=37977,
                                                 target="A",
                                                 routeSoFar=["B","E","D","C"]}))],
                debug={debugId=2,...},
                data=NODATA})

(41,"A",["C"],{id=86805,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99351,
                                                 target="B",
                                                 routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(42,"C",["D","A"],{id=86805,ttl=254,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99351,
                                                 target="B",
                                                 routeSoFar=["A","C"]}))],
                debug={debugId=1,...},
                data=NODATA})

(43,"D",["E","C"],{id=86805,ttl=253,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99351,
                                                 target="B",
                                                 routeSoFar=["A","C","D"]}))],
                debug={debugId=1,...},
                data=NODATA})

(44,"E",["D","B"],{id=86805,ttl=252,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=99351,
                                                 target="B",
                                                 routeSoFar=["A","C","D","E"]}))],
                debug={debugId=1,...},
```

```
                    data=NODATA})

(45,"B",["E"],{id=99470,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=37978,
                                                target="A",
                                                routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(46,"E",["D","B"],{id=99470,ttl=254,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37978,
                                                    target="A",
                                                    routeSoFar=["B","E"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(47,"D",["E","C"],{id=99470,ttl=253,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37978,
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(48,"C",["D","A"],{id=99470,ttl=252,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37978,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(49,"A",["C"],{id=86807,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=99352,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(50,"C",["D","A"],{id=86807,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99352,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(51,"D",["E","C"],{id=86807,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99352,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(52,"E",["D","B"],{id=86807,ttl=252,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=99352,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(53,"B",["E"],{id=99472,ttl=255,src="B",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=37979,
                                                target="A",
                                                routeSoFar=["B"]}))],
               debug={debugId=2,...},
               data=NODATA})

(54,"E",["D","B"],{id=99472,ttl=254,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37979,
                                                    target="A",
                                                    routeSoFar=["B","E"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(55,"D",["E","C"],{id=99472,ttl=253,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37979,
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                   debug={debugId=2,...},
                   data=NODATA})
```

```
(56,"C",["D","A"],{id=99472,ttl=252,src="B",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=37979,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                   debug={debugId=2,...},
                   data=NODATA})
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til "Mostly-bidir" uden nogen udvidelser medtaget:

```
(1,"A",["C"],{id=40996,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=44319,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(2,"B",["E"],{id=24143,ttl=255,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=20929,
                                               target="A",
                                               routeSoFar=["B"]}))],
              debug={debugId=2,...},
              data=NODATA})

(3,"C",["D","A"],{id=40996,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=44319,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"E",["D","B"],{id=24143,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=20929,
                                                   target="A",
                                                   routeSoFar=["B","E"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(5,"D",["E","C"],{id=40996,ttl=253,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=44319,
                                                   target="B",
                                                   routeSoFar=["A","C","D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(6,"D",["E","C"],{id=24143,ttl=253,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=20929,
                                                   target="A",
                                                   routeSoFar=["B","E","D"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(7,"E",["D","B"],{id=40996,ttl=252,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=44319,
                                                   target="B",
                                                   routeSoFar=["A","C","D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(8,"C",["D","A"],{id=24143,ttl=252,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=20929,
                                                   target="A",
                                                   routeSoFar=["B","E","D","C"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(9,"B",["E"],{id=24142,ttl=255,src="B",dest="A",
              dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                   segmentsLeft=3,
                                                   route=["E","D","C"]}))],
              debug={debugId=2,...},
              data=SOMEDATA("pakke 2 fra B til A")})

(10,"B",["E"],{id=24144,ttl=255,src="B",dest="A",
```

```
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=3,
                                                     route=["E","D","C"]})),
                     (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                debug={debugId=1,...},
                data=NODATA})

(11,"E",["D","B"],{id=24142,ttl=254,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["E","D","C"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2 fra B til A")})

(12,"A",["C"],{id=40995,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]}))],
               debug={debugId=1,...},
               data=SOMEDATA("pakke 1 fra A til B")})

(13,"A",["C"],{id=40997,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]})),
                    (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
               debug={debugId=2,...},
               data=NODATA})

(14,"E",["D","B"],{id=24144,ttl=254,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(15,"D",["E","C"],{id=24142,ttl=253,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["E","D","C"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2 fra B til A")})

(16,"C",["D","A"],{id=40995,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(17,"D",["E","C"],{id=24144,ttl=253,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(18,"C",["D","A"],{id=40997,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]})),
                        (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(19,"D",["E","C"],{id=40995,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C","D","E"]}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(20,"C",["D","A"],{id=24144,ttl=252,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["E","D","C"]})),
```

```
                    (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=1578}))],
                debug={debugId=1,...},
                data=NODATA})

(21,"D",["E","C"],{id=40997,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]})),
                    (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                debug={debugId=2,...},
                data=NODATA})

(22,"E",["D","B"],{id=40995,ttl=252,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["C","D","E"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=7277}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1 fra A til B")})

(23,"E",["D","B"],{id=40997,ttl=252,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["C","D","E"]})),
                    (RouteReply,routeReply({route=["B","E","D","C","A"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=7278}))],
                debug={debugId=2,...},
                data=NODATA})

(24,"B",["E"],{id=40995,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=7277,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=1,...},
               data=NODATA})

(25,"A",["C"],{id=24144,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=1578,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=1,...},
               data=NODATA})

(26,"C",["D","A"],{id=24142,ttl=252,src="B",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["E","D","C"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=1579}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2 fra B til A")})

(27,"B",["E"],{id=40997,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=7278,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=2,...},
               data=NODATA})

(28,"A",["C"],{id=24142,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=1579,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=2,...},
               data=NODATA})
```

```
(29,"D",["E","C"],{id=24144,ttl=253,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=94323}))],
                   debug={debugId=1,...},
                   data=NODATA})

(30,"C",["D","A"],{id=24144,ttl=252,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=1580}))],
                   debug={debugId=1,...},
                   data=NODATA})

(31,"C",["D","A"],{id=24144,ttl=255,src="C",dest="D",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=[]})),
                        (Acknowledgement,acknowledgement({id=94323,
                                                          ackSrc="C",
                                                          ackDest="D"}))],
                   debug={debugId=1,...},
                   data=NODATA})

(32,"A",["C"],{id=24144,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=1580,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=1,...},
               data=NODATA})
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til "Mostly-bidir" med både "Salvage Operations" og "Cached Route Reply" slået til:

```
(1,"A",["C"],{id=32818,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=88294,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(2,"B",["E"],{id=69457,ttl=255,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=31463,
                                               target="A",
                                               routeSoFar=["B"]}))],
              debug={debugId=2,...},
              data=NODATA})

(3,"C",["D","A"],{id=32818,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=88294,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"E",["D","B"],{id=69457,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=31463,
                                                   target="A",
                                                   routeSoFar=["B","E"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(5,"D",["E","C"],{id=69457,ttl=253,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=31463,
                                                   target="A",
                                                   routeSoFar=["B","E","D"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(6,"D",["E","C"],{id=13553,ttl=255,src="D",dest="A",
```

```
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                         segmentsLeft=1,
                                                         route=["C"]})),
                         (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(7,"C",["D","A"],{id=78317,ttl=255,src="C",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                         segmentsLeft=2,
                                                         route=["D","E"]})),
                         (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(8,"C",["D","A"],{id=13553,ttl=254,src="D",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                         segmentsLeft=0,
                                                         route=["C"]})),
                         (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=1578}))],
                    debug={debugId=1,...},
                    data=NODATA})

(9,"D",["E","C"],{id=78317,ttl=254,src="C",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                         segmentsLeft=1,
                                                         route=["D","E"]})),
                         (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(10,"A",["C"],{id=32817,ttl=255,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=3,
                                                         route=["C","D","E"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})

(11,"E",["D","B"],{id=78317,ttl=253,src="C",dest="B",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                          segmentsLeft=0,
                                                          route=["D","E"]})),
                          (RouteReply,routeReply({route=["B","E","D","C","A"]})),
                          (AcknowledgementRequest,acknowledgementRequest({id=7277}))],
                     debug={debugId=2,...},
                     data=NODATA})

(12,"A",["C"],{id=13553,ttl=255,src="A",dest="C",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=[]})),
                         (Acknowledgement,acknowledgement({id=1578,
                                                           ackSrc="A",
                                                           ackDest="C"}))],
                    debug={debugId=1,...},
                    data=NODATA})

(13,"C",["D","A"],{id=32817,ttl=254,src="A",dest="B",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                          segmentsLeft=2,
                                                          route=["C","D","E"]}))],
                     debug={debugId=1,...},
                     data=SOMEDATA("pakke 1 fra A til B")})

(14,"B",["E"],{id=69456,ttl=255,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=3,
                                                         route=["E","D","C"]}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(15,"B",["E"],{id=78317,ttl=255,src="B",dest="E",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=[]})),
                         (Acknowledgement,acknowledgement({id=7277,
                                                           ackSrc="B",
```

```
                                                    ackDest="E"}))],
                debug={debugId=2,...},
                data=NODATA})

(16,"D",["E","C"],{id=32817,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1 fra A til B")})

(17,"E",["D","B"],{id=69456,ttl=254,src="B",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=2,
                                                     route=["E","D","C"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2 fra B til A")})

(18,"D",["E","C"],{id=69456,ttl=253,src="B",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["E","D","C"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2 fra B til A")})

(19,"E",["D","B"],{id=32817,ttl=252,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["C","D","E"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=7278}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1 fra A til B")})

(20,"C",["D","A"],{id=69456,ttl=252,src="B",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["E","D","C"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=1579}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2 fra B til A")})

(21,"B",["E"],{id=32817,ttl=255,src="B",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                      (Acknowledgement,acknowledgement({id=7278,
                                                        ackSrc="B",
                                                        ackDest="E"}))],
                debug={debugId=1,...},
                data=NODATA})

(22,"A",["C"],{id=69456,ttl=255,src="A",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                      (Acknowledgement,acknowledgement({id=1579,
                                                        ackSrc="A",
                                                        ackDest="C"}))],
                debug={debugId=2,...},
                data=NODATA})
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til “Bidir-only” uden noden udvidelser medtaget:

```
(1,"B",["E"],{id=71555,ttl=255,src="B",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=74183,
                                                 target="A",
                                                 routeSoFar=["B"]}))],
                debug={debugId=2,...},
                data=NODATA})

(2,"A",["C"],{id=94250,ttl=255,src="A",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=82558,
                                                 target="B",
                                                 routeSoFar=["A"]}))],
                debug={debugId=1,...},
                data=NODATA})
```

```
(3,"E",["D","B"],{id=71555,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=74183,
                                                   target="A",
                                                   routeSoFar=["B","E"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(4,"C",["D","A"],{id=94250,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=82558,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(5,"D",["E","C"],{id=71555,ttl=253,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=74183,
                                                   target="A",
                                                   routeSoFar=["B","E","D"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(6,"D",["E","C"],{id=94250,ttl=253,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=82558,
                                                   target="B",
                                                   routeSoFar=["A","C","D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(7,"C",["D","A"],{id=71555,ttl=252,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=74183,
                                                   target="A",
                                                   routeSoFar=["B","E","D","C"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(8,"E",["D","B"],{id=94250,ttl=252,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=82558,
                                                   target="B",
                                                   routeSoFar=["A","C","D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(9,"A",["C"],{id=94251,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=3,
                                                     route=["C","D","E"]})),
                     (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                debug={debugId=2,...},
                data=NODATA})

(10,"B",["E"],{id=71556,ttl=255,src="B",dest="A",
                 dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                      segmentsLeft=3,
                                                      route=["E","D","C"]})),
                      (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                 debug={debugId=1,...},
                 data=NODATA})

(11,"C",["D","A"],{id=94251,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]})),
                        (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(12,"E",["D","B"],{id=71556,ttl=254,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(13,"D",["E","C"],{id=94251,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
```

```
                                                    route=["C","D","E"]})),
                        (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(14,"D",["E","C"],{id=71556,ttl=253,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(15,"E",["D","B"],{id=94251,ttl=252,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["C","D","E"]})),
                        (RouteReply,routeReply({route=["B","E","D","C","A"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=7277}))],
                    debug={debugId=2,...},
                    data=NODATA})

(16,"C",["D","A"],{id=71556,ttl=252,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=1578}))],
                    debug={debugId=1,...},
                    data=NODATA})

(17,"B",["E"],{id=71554,ttl=255,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["E","D","C"]}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(18,"B",["E"],{id=94251,ttl=255,src="B",dest="E",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                        (Acknowledgement,acknowledgement({id=7277,
                                                    ackSrc="B",
                                                    ackDest="E"}))],
                    debug={debugId=2,...},
                    data=NODATA})

(19,"A",["C"],{id=94249,ttl=255,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})

(20,"A",["C"],{id=71556,ttl=255,src="A",dest="C",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                        (Acknowledgement,acknowledgement({id=1578,
                                                    ackSrc="A",
                                                    ackDest="C"}))],
                    debug={debugId=1,...},
                    data=NODATA})

(21,"E",["D","B"],{id=71554,ttl=254,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["E","D","C"]}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(22,"C",["D","A"],{id=94249,ttl=254,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["C","D","E"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})
```

```
(23,"D",["E","C"],{id=71554,ttl=253,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["E","D","C"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2 fra B til A")})

(24,"D",["E","C"],{id=94249,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C","D","E"]}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(25,"C",["D","A"],{id=71554,ttl=252,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["E","D","C"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=1579}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2 fra B til A")})

(26,"E",["D","B"],{id=94249,ttl=252,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["C","D","E"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=7278}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(27,"A",["C"],{id=71554,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=1579,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=2,...},
               data=NODATA})

(28,"B",["E"],{id=94249,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=7278,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=1,...},
               data=NODATA})
```

Pakketransmissioner fra simulation af DSR-modellen konfigureret til "Bidir-only" med både "Salvage Operations" og "Cached Route Reply" slået til:

```
(1,"B",["E"],{id=14392,ttl=255,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=35992,
                                               target="A",
                                               routeSoFar=["B"]}))],
              debug={debugId=2,...},
              data=NODATA})

(2,"A",["C"],{id=54293,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=10407,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(3,"E",["D","B"],{id=14392,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=35992,
                                                   target="A",
                                                   routeSoFar=["B","E"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(4,"D",["E","C"],{id=14392,ttl=253,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=35992,
```

```
                                                    target="A",
                                                    routeSoFar=["B","E","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(5,"C",["D","A"],{id=54293,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=10407,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(6,"C",["D","A"],{id=14392,ttl=252,src="B",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=35992,
                                                    target="A",
                                                    routeSoFar=["B","E","D","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(7,"D",["E","C"],{id=54293,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=10407,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(8,"A",["C"],{id=54294,ttl=255,src="A",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                      segmentsLeft=3,
                                                      route=["C","D","E"]})),
                       (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                  debug={debugId=2,...},
                  data=NODATA})

(9,"E",["D","B"],{id=54293,ttl=252,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=10407,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                    debug={debugId=1,...},
                    data=NODATA})

(10,"C",["D","A"],{id=54294,ttl=254,src="A",dest="B",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=2,
                                                         route=["C","D","E"]})),
                          (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                     debug={debugId=2,...},
                     data=NODATA})

(11,"B",["E"],{id=14393,ttl=255,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=3,
                                                       route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(12,"D",["E","C"],{id=54294,ttl=253,src="A",dest="B",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=1,
                                                         route=["C","D","E"]})),
                          (RouteReply,routeReply({route=["B","E","D","C","A"]}))],
                     debug={debugId=2,...},
                     data=NODATA})

(13,"E",["D","B"],{id=14393,ttl=254,src="B",dest="A",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=2,
                                                         route=["E","D","C"]})),
                          (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                     debug={debugId=1,...},
                     data=NODATA})

(14,"D",["E","C"],{id=14393,ttl=253,src="B",dest="A",
                     dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=1,
                                                         route=["E","D","C"]})),
                          (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
```

```
                    debug={debugId=1,...},
                    data=NODATA})

(15,"E",["D","B"],{id=54294,ttl=252,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=["C","D","E"]})),
                         (RouteReply,routeReply({route=["B","E","D","C","A"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=7277}))],
                    debug={debugId=2,...},
                    data=NODATA})

(16,"C",["D","A"],{id=14393,ttl=252,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=["E","D","C"]})),
                         (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=1578}))],
                    debug={debugId=1,...},
                    data=NODATA})

(17,"B",["E"],{id=14391,ttl=255,src="B",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=3,
                                                     route=["E","D","C"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2 fra B til A")})

(18,"B",["E"],{id=54294,ttl=255,src="B",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=7277,
                                                       ackSrc="B",
                                                       ackDest="E"}))],
                debug={debugId=2,...},
                data=NODATA})

(19,"A",["C"],{id=54292,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=3,
                                                     route=["C","D","E"]}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1 fra A til B")})

(20,"E",["D","B"],{id=14391,ttl=254,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=2,
                                                         route=["E","D","C"]}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(21,"A",["C"],{id=14393,ttl=255,src="A",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=1578,
                                                       ackSrc="A",
                                                       ackDest="C"}))],
                debug={debugId=1,...},
                data=NODATA})

(22,"C",["D","A"],{id=54292,ttl=254,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=2,
                                                         route=["C","D","E"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})

(23,"D",["E","C"],{id=14391,ttl=253,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=1,
                                                         route=["E","D","C"]}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(24,"C",["D","A"],{id=14391,ttl=252,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
```

```
                                                    segmentsLeft=0,
                                                    route=["E","D","C"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=1579}))],
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2 fra B til A")})

(25,"D",["E","C"],{id=54292,ttl=253,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["C","D","E"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})

(26,"A",["C"],{id=14391,ttl=255,src="A",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                segmentsLeft=0,
                                                route=[]})),
                     (Acknowledgement,acknowledgement({id=1579,
                                                ackSrc="A",
                                                ackDest="C"}))],
                debug={debugId=2,...},
                data=NODATA})

(27,"E",["D","B"],{id=54292,ttl=252,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["C","D","E"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=7278}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1 fra A til B")})

(28,"B",["E"],{id=54292,ttl=255,src="B",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                segmentsLeft=0,
                                                route=[]})),
                     (Acknowledgement,acknowledgement({id=7278,
                                                ackSrc="B",
                                                ackDest="E"}))],
                debug={debugId=1,...},
                data=NODATA})
```

D

Pakketransmissioner fra demonstrationer af DSR-modellen

D.1 Demonstration af "DSR's hovedfunktionalitet" og "Route Discovery" (afsnit 3.8.1)

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"A",["C"],{id=14317,ttl=255,src="A",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=93500,
                                               target="B",
                                               routeSoFar=["A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(2,"C",["D","A"],{id=14317,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=93500,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"D",["E","C"],{id=14317,ttl=253,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=93500,
                                                   target="B",
                                                   routeSoFar=["A","C","D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"E",["D","B"],{id=14317,ttl=252,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=93500,
                                                   target="B",
                                                   routeSoFar=["A","C","D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(5,"B",["E"],{id=5844,ttl=255,src="B",dest="A",
              dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                   segmentsLeft=3,
                                                   route=["E","D","C"]})),
                   (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
```

```
                debug={debugId=1,...},
                data=NODATA})

(6,"E",["D","B"],{id=5844,ttl=254,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=2,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(7,"D",["E","C"],{id=5844,ttl=253,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=1,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(8,"C",["D","A"],{id=5844,ttl=252,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=62139}))],
                  debug={debugId=1,...},
                  data=NODATA})

(9,"A",["C"],{id=14316,ttl=255,src="A",dest="B",
              dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                   segmentsLeft=3,
                                                   route=["C","D","E"]}))],
              debug={debugId=1,...},
              data=SOMEDATA("pakke 1 fra A til B")})

(10,"C",["D","A"],{id=14316,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(11,"A",["C"],{id=14318,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=62139,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=1,...},
               data=NODATA})

(12,"D",["E","C"],{id=14316,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C","D","E"]}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(13,"E",["D","B"],{id=14316,ttl=252,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["C","D","E"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=61092}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1 fra A til B")})

(14,"B",["E"],{id=5845,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=61092,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=1,...},
               data=NODATA})
```

D.2 Demonstration af "Route Maintenance" (afsnit 3.8.2)

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"A",["C"],{id=15193,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=97597,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(2,"C",["D","A"],{id=15193,ttl=254,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=97597,
                                                   target="B",
                                                   routeSoFar=["A","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"D",["E","C"],{id=15193,ttl=253,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=97597,
                                                   target="B",
                                                   routeSoFar=["A","C","D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"E",["D","B"],{id=15193,ttl=252,src="A",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=97597,
                                                   target="B",
                                                   routeSoFar=["A","C","D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(5,"B",["E"],{id=25229,ttl=255,src="B",dest="A",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["E","D","C"]})),
                     (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
               debug={debugId=1,...},
               data=NODATA})

(6,"E",["D","B"],{id=25229,ttl=254,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=2,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(7,"D",["E","C"],{id=25229,ttl=253,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=1,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(8,"C",["D","A"],{id=25229,ttl=252,src="B",dest="A",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E","D","C"]})),
                       (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=16264}))],
                  debug={debugId=1,...},
                  data=NODATA})

(9,"A",["C"],{id=15192,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]}))],
               debug={debugId=1,...},
               data=SOMEDATA("pakke 1")})

(10,"A",["C"],{id=15194,ttl=255,src="A",dest="C",
```

```
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=16264,
                                                       ackSrc="A",
                                                       ackDest="C"}))],
                debug={debugId=1,...},
                data=NODATA})

(11,"C",["D","A"],{id=15192,ttl=254,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=2,
                                                     route=["C","D","E"]}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1")})

(12,"D",["E","C"],{id=15192,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1")})

(13,"E",["D","B"],{id=15192,ttl=252,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["C","D","E"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=69607}))],
                debug={debugId=1,...},
                data=SOMEDATA("pakke 1")})

(14,"B",["E"],{id=25230,ttl=255,src="B",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=69607,
                                                       ackSrc="B",
                                                       ackDest="E"}))],
                debug={debugId=1,...},
                data=NODATA})

(15,"A",["C"],{id=15195,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=3,
                                                     route=["C","D","E"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(16,"C",["D","A"],{id=15195,ttl=254,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=2,
                                                     route=["C","D","E"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(17,"D",["F","C"],{id=15195,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(18,"D",["F","C"],{id=15195,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=84698}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(19,"D",["F","C"],{id=15195,ttl=253,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C","D","E"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=84698}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})
```

```
(20,"D",["F","C"],{id=28635,ttl=255,src="D",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C"]})),
                        (RouteError,routeError({errorType=NODE_UNREACHABLE,
                                                salvageCounter=0,
                                                errorSrc="D",
                                                errorDest="A",
                                                typeSpec=unreachableNodeAddress("E")}))],
                   debug={debugId=2,...},
                   data=NODATA})

(21,"C",["D","A"],{id=28635,ttl=254,src="D",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,segmentsLeft=0,route=["C"]})),
                        (RouteError,routeError({errorType=NODE_UNREACHABLE,
                                                salvageCounter=0,
                                                errorSrc="D",
                                                errorDest="A",
                                                typeSpec=unreachableNodeAddress("E")})),
                        (AcknowledgementRequest,acknowledgementRequest({id=16265}))],
                   debug={debugId=2,...},
                   data=NODATA})

(22,"A",["C"],{id=15196,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=16265,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=2,...},
               data=NODATA})

(23,"A",["C"],{id=15198,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=97598,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=3,...},
               data=NODATA})

(24,"C",["D","A"],{id=15198,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=97598,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=3,...},
                   data=NODATA})

(25,"D",["F","C"],{id=15198,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=97598,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                   debug={debugId=3,...},
                   data=NODATA})

(26,"F",["G","D"],{id=15198,ttl=252,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=97598,
                                                    target="B",
                                                    routeSoFar=["A","C","D","F"]}))],
                   debug={debugId=3,...},
                   data=NODATA})

(27,"G",["F","B"],{id=15198,ttl=251,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=97598,
                                                    target="B",
                                                    routeSoFar=["A","C","D","F","G"]}))],
                   debug={debugId=3,...},
                   data=NODATA})

(28,"B",["G"],{id=25231,ttl=255,src="B",dest="A",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=4,
                                                    route=["G","F","D","C"]})),
                    (RouteReply,routeReply({route=["A","C","D","F","G","B"]}))],
               debug={debugId=3,...},
               data=NODATA})

(29,"G",["F","B"],{id=25231,ttl=254,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
```

```
                                                    segmentsLeft=3,
                                                    route=["G","F","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","F","G","B"]}))],
                    debug={debugId=3,...},
                    data=NODATA})

(30,"F",["G","D"],{id=25231,ttl=253,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["G","F","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","F","G","B"]}))],
                    debug={debugId=3,...},
                    data=NODATA})

(31,"D",["F","C"],{id=25231,ttl=252,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["G","F","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","F","G","B"]}))],
                    debug={debugId=3,...},
                    data=NODATA})

(32,"C",["D","A"],{id=25231,ttl=251,src="B",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["G","F","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","F","G","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=16266}))],
                    debug={debugId=3,...},
                    data=NODATA})

(33,"A",["C"],{id=15197,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=4,
                                                    route=["C","D","F","G"]}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (genudsendelse af pakke 2)")})

(34,"A",["C"],{id=15199,ttl=255,src="A",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=16266,
                                                    ackSrc="A",
                                                    ackDest="C"}))],
                debug={debugId=3,...},
                data=NODATA})

(35,"C",["D","A"],{id=15197,ttl=254,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","F","G"]}))],
                    debug={debugId=3,...},
                    data=SOMEDATA("pakke 3 (genudsendelse af pakke 2)")})

(36,"D",["F","C"],{id=15197,ttl=253,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["C","D","F","G"]}))],
                    debug={debugId=3,...},
                    data=SOMEDATA("pakke 3 (genudsendelse af pakke 2)")})

(37,"F",["G","D"],{id=15197,ttl=252,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["C","D","F","G"]}))],
                    debug={debugId=3,...},
                    data=SOMEDATA("pakke 3 (genudsendelse af pakke 2)")})

(38,"G",["F","B"],{id=15197,ttl=251,src="A",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["C","D","F","G"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=21782}))],
                    debug={debugId=3,...},
                    data=SOMEDATA("pakke 3 (genudsendelse af pakke 2)")})

(39,"B",["G"],{id=25232,ttl=255,src="B",dest="G",
```

```
dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                     segmentsLeft=0,
                                     route=[]})),
     (Acknowledgement,acknowledgement({id=21782,
                                       ackSrc="B",
                                       ackDest="G"}))],
debug={debugId=3,...},
data=NODATA})
```

D.3 Demonstration af “Cached Route Reply” (afsnit 3.8.3)

D.3.1 Kommandoliste

```
1`("A",[MoveNode{toPos=(200,200)},
        WaitForTimeout{timeout=2000},
        SendPacket{toNode="B",id=2,data=SOMEDATA("pakke 2")}]) ++

1`("C",[MoveNode{toPos=(200,400)}]) ++

1`("D",[MoveNode{toPos=(200,600)},
        SendPacket{toNode="B",id=1,data=SOMEDATA("pakke 1")}]) ++

1`("E",[MoveNode{toPos=(200,800)}]) ++

1`("B",[MoveNode{toPos=(200,1000)},
        WaitForPacket{id=1},
        WaitForPacket{id=2}])
```

D.3.2 Pakketransmissioner med optimeringen slået fra

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id’en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"D",["E","C"],{id=47891,ttl=255,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=65256,
                                                   target="B",
                                                   routeSoFar=["D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(2,"C",["D","A"],{id=47891,ttl=254,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=65256,
                                                   target="B",
                                                   routeSoFar=["D","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"E",["D","B"],{id=47891,ttl=254,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=65256,
                                                   target="B",
                                                   routeSoFar=["D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"B",["E"],{id=55809,ttl=255,src="B",dest="D",
              dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                   segmentsLeft=1,
                                                   route=["E"]})),
                   (RouteReply,routeReply({route=["D","E","B"]}))],
              debug={debugId=1,...},
              data=NODATA})

(5,"A",["C"],{id=47891,ttl=253,src="D",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=65256,
                                               target="B",
                                               routeSoFar=["D","C","A"]}))],
              debug={debugId=1,...},
              data=NODATA})
```

```
(6,"E",["D","B"],{id=55809,ttl=254,src="B",dest="D",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E"]})),
                       (RouteReply,routeReply({route=["D","E","B"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=15082}))],
                  debug={debugId=1,...},
                  data=NODATA})

(7,"D",["E","C"],{id=47890,ttl=255,src="D",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=1,
                                                       route=["E"]}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})

(8,"D",["E","C"],{id=47892,ttl=255,src="D",dest="E",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=[]})),
                       (Acknowledgement,acknowledgement({id=15082,
                                                         ackSrc="D",
                                                         ackDest="E"}))],
                  debug={debugId=1,...},
                  data=NODATA})

(9,"E",["D","B"],{id=47890,ttl=254,src="D",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=15083}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})

(10,"B",["E"],{id=55810,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=15083,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=1,...},
               data=NODATA})

(11,"A",["C"],{id=65979,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=46650,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=2,...},
               data=NODATA})

(12,"C",["D","A"],{id=65979,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=46650,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(13,"D",["E","C"],{id=65979,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=46650,
                                                    target="B",
                                                    routeSoFar=["A","C","D"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(14,"E",["D","B"],{id=65979,ttl=252,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=46650,
                                                    target="B",
                                                    routeSoFar=["A","C","D","E"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(15,"B",["E"],{id=55811,ttl=255,src="B",dest="A",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["E","D","C"]})),
                    (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
               debug={debugId=2,...},
```

```
                data=NODATA})

(16,"E",["D","B"],{id=55811,ttl=254,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(17,"D",["E","C"],{id=55811,ttl=253,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(18,"C",["D","A"],{id=55811,ttl=252,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["E","D","C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=47568}))],
                   debug={debugId=2,...},
                   data=NODATA})

(19,"A",["C"],{id=65978,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]}))],
               debug={debugId=2,...},
               data=SOMEDATA("pakke 2")})

(20,"A",["C"],{id=65980,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=47568,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=2,...},
               data=NODATA})

(21,"C",["D","A"],{id=65978,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2")})

(22,"D",["E","C"],{id=65978,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C","D","E"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2")})

(23,"E",["D","B"],{id=65978,ttl=252,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["C","D","E"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=15084}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2")})

(24,"B",["E"],{id=55812,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=15084,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=2,...},
               data=NODATA})
```

D.3.3 Pakketransmissioner med optimeringen slået til

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"D",["E","C"],{id=26758,ttl=255,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=58173,
                                                   target="B",
                                                   routeSoFar=["D"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(2,"C",["D","A"],{id=26758,ttl=254,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=58173,
                                                   target="B",
                                                   routeSoFar=["D","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"E",["D","B"],{id=26758,ttl=254,src="D",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=58173,
                                                   target="B",
                                                   routeSoFar=["D","E"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(4,"A",["C"],{id=26758,ttl=253,src="D",dest="Z",
                dsr=[(RouteRequest,routeRequest({id=58173,
                                                 target="B",
                                                 routeSoFar=["D","C","A"]}))],
                debug={debugId=1,...},
                data=NODATA})

(5,"B",["E"],{id=5393,ttl=255,src="B",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["E"]})),
                     (RouteReply,routeReply({route=["D","E","B"]}))],
                debug={debugId=1,...},
                data=NODATA})

(6,"E",["D","B"],{id=5393,ttl=254,src="B",dest="D",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E"]})),
                       (RouteReply,routeReply({route=["D","E","B"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=15082}))],
                  debug={debugId=1,...},
                  data=NODATA})

(7,"D",["E","C"],{id=26757,ttl=255,src="D",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=1,
                                                       route=["E"]}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})

(8,"D",["E","C"],{id=26759,ttl=255,src="D",dest="E",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=[]})),
                       (Acknowledgement,acknowledgement({id=15082,
                                                         ackSrc="D",
                                                         ackDest="E"}))],
                  debug={debugId=1,...},
                  data=NODATA})

(9,"E",["D","B"],{id=26757,ttl=254,src="D",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["E"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=15083}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})
```

```
(10,"B",["E"],{id=5394,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=15083,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=1,...},
               data=NODATA})

(11,"A",["C"],{id=24612,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=54843,
                                                target="B",
                                                routeSoFar=["A"]}))],
               debug={debugId=2,...},
               data=NODATA})

(12,"C",["D","A"],{id=24612,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=54843,
                                                    target="B",
                                                    routeSoFar=["A","C"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(13,"D",["E","C"],{id=26760,ttl=255,src="D",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                        segmentsLeft=1,
                                                        route=["C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(14,"C",["D","A"],{id=26760,ttl=254,src="D",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=15,
                                                        segmentsLeft=0,
                                                        route=["C"]})),
                        (RouteReply,routeReply({route=["A","C","D","E","B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=47568}))],
                   debug={debugId=2,...},
                   data=NODATA})

(15,"A",["C"],{id=24611,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=3,
                                                    route=["C","D","E"]}))],
               debug={debugId=2,...},
               data=SOMEDATA("pakke 2")})

(16,"A",["C"],{id=24613,ttl=255,src="A",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=47568,
                                                      ackSrc="A",
                                                      ackDest="C"}))],
               debug={debugId=2,...},
               data=NODATA})

(17,"C",["D","A"],{id=24611,ttl=254,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=2,
                                                        route=["C","D","E"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2")})

(18,"D",["E","C"],{id=24611,ttl=253,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["C","D","E"]}))],
                   debug={debugId=2,...},
                   data=SOMEDATA("pakke 2")})

(19,"E",["D","B"],{id=24611,ttl=252,src="A",dest="B",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["C","D","E"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=15084}))],
```

```
                    debug={debugId=2,...},
                    data=SOMEDATA("pakke 2")})

(20,"B",["E"],{id=5395,ttl=255,src="B",dest="E",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=15084,
                                                      ackSrc="B",
                                                      ackDest="E"}))],
               debug={debugId=2,...},
               data=NODATA})
```

D.4 Demonstration af "Salvage Operations" (afsnit 3.8.4)

D.4.1 Kommandoliste

```
1`("A",[MoveNode{toPos=(200,200)},
        WaitForTimeout{timeout=4000},
        SendPacket{toNode="E",id=2,data=SOMEDATA("pakke 2")},
        WaitForTimeout{timeout=4000},
        SendPacket{toNode="E",id=3,data=SOMEDATA
                                   ("pakke 3 (kan salvages)")}]) ++

1`("B",[MoveNode{toPos=(200,400)},
        SendPacket{toNode="E",id=1,data=SOMEDATA("pakke 1")}]) ++

1`("C",[MoveNode{toPos=(150,600)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(0,600)},
        WaitForTimeout{timeout=4000},
        MoveNode{toPos=(150,600)}]) ++

1`("D",[MoveNode{toPos=(400,600)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(250,600)},
        WaitForTimeout{timeout=4000},
        MoveNode{toPos=(400,600)}]) ++

1`("E",[MoveNode{toPos=(200,800)},
        WaitForPacket{id=1},
        WaitForPacket{id=2},
        WaitForPacket{id=3}])
         (* WaitForPacket{id=3}: Vil kun blive opfyldt
            når salvageEnabled = true *)
```

D.4.2 Pakketransmissioner med optimeringen slået fra

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"B",["C","A"],{id=93079,ttl=255,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=34970,
                                                   target="E",
                                                   routeSoFar=["B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(2,"C",["E","B"],{id=93079,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=34970,
                                                   target="E",
                                                   routeSoFar=["B","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"A",["B"],{id=93079,ttl=254,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=34970,
                                               target="E",
                                               routeSoFar=["B","A"]}))],
```

```
                debug={debugId=1,...},
                data=NODATA})

(4,"E",["C"],{id=33783,ttl=255,src="E",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C"]})),
                      (RouteReply,routeReply({route=["B","C","E"]}))],
                debug={debugId=1,...},
                data=NODATA})

(5,"C",["E","B"],{id=33783,ttl=254,src="E",dest="B",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=["C"]})),
                         (RouteReply,routeReply({route=["B","C","E"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=90854}))],
                    debug={debugId=1,...},
                    data=NODATA})

(6,"B",["C","A"],{id=93078,ttl=255,src="B",dest="E",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=1,
                                                         route=["C"]}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1")})

(7,"B",["C","A"],{id=33783,ttl=255,src="B",dest="C",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=[]})),
                         (Acknowledgement,acknowledgement({id=90854,
                                                           ackSrc="B",
                                                           ackDest="C"}))],
                    debug={debugId=1,...},
                    data=NODATA})

(8,"C",["E","B"],{id=93078,ttl=254,src="B",dest="E",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=["C"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=90855}))],
                    debug={debugId=1,...},
                    data=SOMEDATA("pakke 1")})

(9,"E",["C"],{id=93078,ttl=255,src="E",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                      (Acknowledgement,acknowledgement({id=90855,
                                                        ackSrc="E",
                                                        ackDest="C"}))],
                debug={debugId=1,...},
                data=NODATA})

(10,"A",["B"],{id=47682,ttl=255,src="A",dest="Z",
                 dsr=[(RouteRequest,routeRequest({id=28583,
                                                  target="E",
                                                  routeSoFar=["A"]}))],
                 debug={debugId=2,...},
                 data=NODATA})

(11,"B",["D","A"],{id=47682,ttl=254,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=28583,
                                                     target="E",
                                                     routeSoFar=["A","B"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(12,"D",["E","B"],{id=47682,ttl=253,src="A",dest="Z",
                    dsr=[(RouteRequest,routeRequest({id=28583,
                                                     target="E",
                                                     routeSoFar=["A","B","D"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(13,"E",["D"],{id=33785,ttl=255,src="E",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
```

```
                                                    segmentsLeft=2,
                                                    route=["D","B"]})),
                        (RouteReply,routeReply({route=["A","B","D","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(14,"D",["E","B"],{id=33785,ttl=254,src="E",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["D","B"]})),
                     (RouteReply,routeReply({route=["A","B","D","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(15,"B",["D","A"],{id=33785,ttl=253,src="E",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["D","B"]})),
                     (RouteReply,routeReply({route=["A","B","D","E"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=66895}))],
                debug={debugId=2,...},
                data=NODATA})

(16,"A",["B"],{id=47681,ttl=255,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["B","D"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(17,"A",["B"],{id=33785,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                     (Acknowledgement,acknowledgement({id=66895,
                                                       ackSrc="A",
                                                       ackDest="B"}))],
                debug={debugId=2,...},
                data=NODATA})

(18,"B",["D","A"],{id=47681,ttl=254,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["B","D"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(19,"D",["E","B"],{id=47681,ttl=253,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["B","D"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=46522}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(20,"E",["D"],{id=47681,ttl=255,src="E",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                     (Acknowledgement,acknowledgement({id=46522,
                                                       ackSrc="E",
                                                       ackDest="D"}))],
                debug={debugId=2,...},
                data=NODATA})

(21,"A",["B"],{id=47684,ttl=255,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["B","D"]}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (kan salvages)")})

(22,"B",["C","A"],{id=47684,ttl=254,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["B","D"]}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (kan salvages)")})
```

```
(23,"B",["C","A"],{id=47684,ttl=254,src="A",dest="E",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["B","D"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=47570}))],
                   debug={debugId=3,...},
                   data=SOMEDATA("pakke 3 (kan salvages)")})

(24,"B",["C","A"],{id=47684,ttl=254,src="A",dest="E",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["B","D"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=47570}))],
                   debug={debugId=3,...},
                   data=SOMEDATA("pakke 3 (kan salvages)")})

(25,"B",["C","A"],{id=93081,ttl=255,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=[]})),
                        (RouteError,routeError({errorType=NODE_UNREACHABLE,
                                                salvageCounter=0,
                                                errorSrc="B",
                                                errorDest="A",
                                                typeSpec=unreachableNodeAddress("D")})),
                        (AcknowledgementRequest,acknowledgementRequest({id=66896}))],
                   debug={debugId=3,...},
                   data=NODATA})

(26,"A",["B"],{id=93081,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=66896,
                                                      ackSrc="A",
                                                      ackDest="B"}))],
               debug={debugId=3,...},
               data=NODATA})
```

D.4.3 Pakketransmissioner med optimeringen slået til

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id’en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"B",["C","A"],{id=11346,ttl=255,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=44585,
                                                   target="E",
                                                   routeSoFar=["B"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(2,"C",["E","B"],{id=11346,ttl=254,src="B",dest="Z",
                  dsr=[(RouteRequest,routeRequest({id=44585,
                                                   target="E",
                                                   routeSoFar=["B","C"]}))],
                  debug={debugId=1,...},
                  data=NODATA})

(3,"A",["B"],{id=11346,ttl=254,src="B",dest="Z",
              dsr=[(RouteRequest,routeRequest({id=44585,
                                               target="E",
                                               routeSoFar=["B","A"]}))],
              debug={debugId=1,...},
              data=NODATA})

(4,"E",["C"],{id=50020,ttl=255,src="E",dest="B",
              dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                   segmentsLeft=1,
                                                   route=["C"]})),
                   (RouteReply,routeReply({route=["B","C","E"]}))],
              debug={debugId=1,...},
```

```
                data=NODATA})

(5,"C",["E","B"],{id=50020,ttl=254,src="E",dest="B",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["C"]})),
                       (RouteReply,routeReply({route=["B","C","E"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=90854}))],
                  debug={debugId=1,...},
                  data=NODATA})

(6,"B",["C","A"],{id=11345,ttl=255,src="B",dest="E",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=1,
                                                       route=["C"]}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})

(7,"B",["C","A"],{id=50020,ttl=255,src="B",dest="C",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=[]})),
                       (Acknowledgement,acknowledgement({id=90854,
                                                         ackSrc="B",
                                                         ackDest="C"}))],
                  debug={debugId=1,...},
                  data=NODATA})

(8,"C",["E","B"],{id=11345,ttl=254,src="B",dest="E",
                  dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                       segmentsLeft=0,
                                                       route=["C"]})),
                       (AcknowledgementRequest,acknowledgementRequest({id=90855}))],
                  debug={debugId=1,...},
                  data=SOMEDATA("pakke 1")})

(9,"E",["C"],{id=11345,ttl=255,src="E",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=90855,
                                                       ackSrc="E",
                                                       ackDest="C"}))],
                debug={debugId=1,...},
                data=NODATA})

(10,"A",["B"],{id=58924,ttl=255,src="A",dest="Z",
                 dsr=[(RouteRequest,routeRequest({id=36508,
                                                  target="E",
                                                  routeSoFar=["A"]}))],
                 debug={debugId=2,...},
                 data=NODATA})

(11,"B",["D","A"],{id=58924,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=36508,
                                                    target="E",
                                                    routeSoFar=["A","B"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(12,"D",["E","B"],{id=58924,ttl=253,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=36508,
                                                    target="E",
                                                    routeSoFar=["A","B","D"]}))],
                   debug={debugId=2,...},
                   data=NODATA})

(13,"E",["D"],{id=50022,ttl=255,src="E",dest="A",
                 dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                      segmentsLeft=2,
                                                      route=["D","B"]})),
                      (RouteReply,routeReply({route=["A","B","D","E"]}))],
                 debug={debugId=2,...},
                 data=NODATA})

(14,"D",["E","B"],{id=50022,ttl=254,src="E",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
```

```
                                                    route=["D","B"]})),
                    (RouteReply,routeReply({route=["A","B","D","E"]}))],
                debug={debugId=2,...},
                data=NODATA})

(15,"B",["D","A"],{id=50022,ttl=253,src="E",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["D","B"]})),
                    (RouteReply,routeReply({route=["A","B","D","E"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=66895}))],
                debug={debugId=2,...},
                data=NODATA})

(16,"A",["B"],{id=58923,ttl=255,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["B","D"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(17,"A",["B"],{id=50022,ttl=255,src="A",dest="B",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                     (Acknowledgement,acknowledgement({id=66895,
                                                       ackSrc="A",
                                                       ackDest="B"}))],
                debug={debugId=2,...},
                data=NODATA})

(18,"B",["D","A"],{id=58923,ttl=254,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["B","D"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(19,"D",["E","B"],{id=58923,ttl=253,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=["B","D"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=46522}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(20,"E",["D"],{id=58923,ttl=255,src="E",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                     (Acknowledgement,acknowledgement({id=46522,
                                                       ackSrc="E",
                                                       ackDest="D"}))],
                debug={debugId=2,...},
                data=NODATA})

(21,"A",["B"],{id=58926,ttl=255,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=2,
                                                    route=["B","D"]}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (kan salvages)")})

(22,"B",["C","A"],{id=58926,ttl=254,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["B","D"]}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (kan salvages)")})

(23,"B",["C","A"],{id=58926,ttl=254,src="A",dest="E",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["B","D"]})),
                    (AcknowledgementRequest,acknowledgementRequest({id=47570}))],
                debug={debugId=3,...},
                data=SOMEDATA("pakke 3 (kan salvages)")})
```

```
(24,"B",["C","A"],{id=58926,ttl=254,src="A",dest="E",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=1,
                                                        route=["B","D"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=47570}))],
                   debug={debugId=3,...},
                   data=SOMEDATA("pakke 3 (kan salvages)")})

(25,"B",["C","A"],{id=58926,ttl=254,src="A",dest="E",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=1,
                                                        segmentsLeft=1,
                                                        route=["B","C"]}))],
                   debug={debugId=3,...},
                   data=SOMEDATA("pakke 3 (kan salvages)")})

(26,"B",["C","A"],{id=11348,ttl=255,src="B",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=[]})),
                        (RouteError,routeError({errorType=NODE_UNREACHABLE,
                                                salvageCounter=0,
                                                errorSrc="B",
                                                errorDest="A",
                                                typeSpec=unreachableNodeAddress("D")})),
                        (AcknowledgementRequest,acknowledgementRequest({id=66896}))],
                   debug={debugId=3,...},
                   data=NODATA})

(27,"C",["E","B"],{id=58926,ttl=253,src="A",dest="E",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=1,
                                                        segmentsLeft=0,
                                                        route=["B","C"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=90856}))],
                   debug={debugId=3,...},
                   data=SOMEDATA("pakke 3 (kan salvages)")})

(28,"A",["B"],{id=11348,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=66896,
                                                      ackSrc="A",
                                                      ackDest="B"}))],
               debug={debugId=3,...},
               data=NODATA})

(29,"E",["C"],{id=58926,ttl=255,src="E",dest="C",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=90856,
                                                      ackSrc="E",
                                                      ackDest="C"}))],
               debug={debugId=3,...},
               data=NODATA})
```

D.5 Demonstration af "Mini-Salvage Operations" (afsnit 3.8.5)

D.5.1 Kommandoliste

```
1`("A",[MoveNode{toPos=(200,200)},
        SendPacket{toNode="D",id=1,data=SOMEDATA("pakke 1")},
        WaitForTimeout{timeout=4000},
        SendPacket{toNode="D",id=2,data=SOMEDATA("pakke 2")}]) ++

1`("B",[MoveNode{toPos=(150,400)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(2000,2000)}]) ++

1`("C",[MoveNode{toPos=(2000,2000)},
        WaitForTimeout{timeout=2000},
        MoveNode{toPos=(250,400)}]) ++
```

```
1 `("D",[MoveNode{toPos=(200,600)},
         WaitForPacket{id=1},
         WaitForPacket{id=2}])
```

D.5.2 Pakketransmissioner

Debug-headerne er fjernet fra pakketransmissionerne (udover debug-id'en, se afsnit 3.6.3). Pakketransmissionerne er på formen:

```
(pakketransmissionsnr, afsenderknude, [modtagende knuder], pakke)
```

```
(1,"A",["B"],{id=62704,ttl=255,src="A",dest="Z",
               dsr=[(RouteRequest,routeRequest({id=30888,
                                                target="D",
                                                routeSoFar=["A"]}))],
               debug={debugId=1,...},
               data=NODATA})

(2,"B",["D","A"],{id=62704,ttl=254,src="A",dest="Z",
                   dsr=[(RouteRequest,routeRequest({id=30888,
                                                    target="D",
                                                    routeSoFar=["A","B"]}))],
                   debug={debugId=1,...},
                   data=NODATA})

(3,"D",["B"],{id=26706,ttl=255,src="D",dest="A",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["B"]})),
                     (RouteReply,routeReply({route=["A","B","D"]}))],
               debug={debugId=1,...},
               data=NODATA})

(4,"B",["D","A"],{id=26706,ttl=254,src="D",dest="A",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["B"]})),
                        (RouteReply,routeReply({route=["A","B","D"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=22767}))],
                   debug={debugId=1,...},
                   data=NODATA})

(5,"A",["B"],{id=62703,ttl=255,src="A",dest="D",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=1,
                                                    route=["B"]}))],
               debug={debugId=1,...},
               data=SOMEDATA("pakke 1")})

(6,"A",["B"],{id=26706,ttl=255,src="A",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=22767,
                                                      ackSrc="A",
                                                      ackDest="B"}))],
               debug={debugId=1,...},
               data=NODATA})

(7,"B",["D","A"],{id=62703,ttl=254,src="A",dest="D",
                   dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                        segmentsLeft=0,
                                                        route=["B"]})),
                        (AcknowledgementRequest,acknowledgementRequest({id=22768}))],
                   debug={debugId=1,...},
                   data=SOMEDATA("pakke 1")})

(8,"D",["B"],{id=62703,ttl=255,src="D",dest="B",
               dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                    segmentsLeft=0,
                                                    route=[]})),
                    (Acknowledgement,acknowledgement({id=22768,
                                                      ackSrc="D",
                                                      ackDest="B"}))],
               debug={debugId=1,...},
```

```
                data=NODATA})

(9,"A",["C"],{id=62706,ttl=255,src="A",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["B"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(10,"A",["C"],{id=62706,ttl=255,src="A",dest="D",
                 dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                      segmentsLeft=1,
                                                      route=["B"]})),
                      (AcknowledgementRequest,acknowledgementRequest({id=76358}))],
                 debug={debugId=2,...},
                 data=SOMEDATA("pakke 2")})

(11,"A",["C"],{id=62706,ttl=255,src="A",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["B"]})),
                      (AcknowledgementRequest,acknowledgementRequest({id=76358}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(12,"A",["C"],{id=62708,ttl=255,src="A",dest="Z",
                dsr=[(RouteError,routeError({errorType=NODE_UNREACHABLE,
                                             salvageCounter=0,
                                             errorSrc="A",
                                             errorDest="D",
                                             typeSpec=unreachableNodeAddress("B")})),
                      (RouteRequest,routeRequest({id=30889,
                                                  target="D",
                                                  routeSoFar=["A"]}))],
                debug={debugId=2,...},
                data=NODATA})

(13,"C",["D","A"],{id=62708,ttl=254,src="A",dest="Z",
                    dsr=[(RouteError,routeError({errorType=NODE_UNREACHABLE,
                                                 salvageCounter=0,
                                                 errorSrc="A",
                                                 errorDest="D",
                                                 typeSpec=unreachableNodeAddress("B")})),
                         (RouteRequest,routeRequest({id=30889,
                                                     target="D",
                                                     routeSoFar=["A","C"]}))],
                    debug={debugId=2,...},
                    data=NODATA})

(14,"D",["C"],{id=26708,ttl=255,src="D",dest="A",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C"]})),
                      (RouteReply,routeReply({route=["A","C","D"]}))],
                debug={debugId=2,...},
                data=NODATA})

(15,"C",["D","A"],{id=26708,ttl=254,src="D",dest="A",
                    dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                         segmentsLeft=0,
                                                         route=["C"]})),
                         (RouteReply,routeReply({route=["A","C","D"]})),
                         (AcknowledgementRequest,acknowledgementRequest({id=40170}))],
                    debug={debugId=2,...},
                    data=NODATA})

(16,"A",["C"],{id=62706,ttl=255,src="A",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=1,
                                                     route=["C"]}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(17,"A",["C"],{id=26708,ttl=255,src="A",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                      (Acknowledgement,acknowledgement({id=40170,
```

```
                                                    ackSrc="A",
                                                    ackDest="C"}))],
                debug={debugId=2,...},
                data=NODATA})

(18,"C",["D","A"],{id=62706,ttl=254,src="A",dest="D",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=["C"]})),
                     (AcknowledgementRequest,acknowledgementRequest({id=40171}))],
                debug={debugId=2,...},
                data=SOMEDATA("pakke 2")})

(19,"D",["C"],{id=62706,ttl=255,src="D",dest="C",
                dsr=[(DSRSourceRoute,dsrSourceRoute({salvageCounter=0,
                                                     segmentsLeft=0,
                                                     route=[]})),
                     (Acknowledgement,acknowledgement({id=40171,
                                                    ackSrc="D",
                                                    ackDest="C"}))],
                debug={debugId=2,...},
                data=NODATA})
```

E

Analyseunderstøttende scripts

E.1 `generate_event_chain.pl` benyttet i afsnit 5.2.1

Dette er scriptet `generate_event_chain.pl` beskrevet i afsnit 5.2.1.

```
#!/usr/local/bin/perl -w

# File: generate_event_chain.pl, $Revision: 1.18 $

use strict;

my $spaces = " " x (length($0)+8);
my $usageText =
    "Usage: $0 <no_of_nodes>\n" .
    "$spaces<map_size_x_and_y>\n" .
    "$spaces<max_speed_x_and_y>\n" .
    "$spaces<max_speed_change_x_and_y>\n" .
    "$spaces<time_to_run>\n" .
    "$spaces<max_time_between_packets>\n" .
    "$spaces<max_time_between_movement>\n" .
    "$spaces<steps_with_movement_after_normal_run>\n";

my $noOfNodes = shift || die $usageText;
# Not yet able to handle > 25 nodes ("Z" is the broadcast-address)
if ($noOfNodes < 2 or $noOfNodes > 25) {
  die "<no_of_nodes> should be between 2 and 26\n";
}
my $mapSize = shift || die $usageText;
my $maxSpeed = shift || die $usageText;
my $maxSpeedChange = shift || die $usageText;
my $maxTime = shift || die $usageText;
my $maxPacketWait = shift || die $usageText;
my $maxMovementWait = shift || die $usageText;
my $stepsWithMovementAfterNormalRun = shift || die $usageText;

my @nodeNos = map { $_-1 } (1..$noOfNodes); # handy
my @nodeNames = map { chr(64+$_) } (1..$noOfNodes);
my @nodePosX = map { 1 + int(rand $mapSize) } (1..$noOfNodes); # 1..$mapSize
my @nodePosY = map { 1 + int(rand $mapSize) } (1..$noOfNodes); # 1..$mapSize
my @nodeSpeedX = map { int(rand(2*$maxSpeed+1)) - $maxSpeed }
                                    (1..$noOfNodes); # -$maxSpeed..$maxSpeed
my @nodeSpeedY = map { int(rand(2*$maxSpeed+1)) - $maxSpeed }
                                    (1..$noOfNodes); # -$maxSpeed..$maxSpeed

print "val eventChainToRun = \n" .
    "  (* generated with parameters:\n" .
    "     noOfNodes=$noOfNodes\n" .
    "     mapSize=$mapSize\n" .
```

```
      "       maxSpeed=$maxSpeed\n" .
      "       maxSpeedChange=$maxSpeedChange\n" .
      "       maxTime=$maxTime\n" .
      "       maxMovementWait=$maxMovementWait\n" .
      "       maxPacketWait=$maxPacketWait\n" .
      "       stepsWithMovementAfterNormalRun=$stepsWithMovementAfterNormalRun\n" .
      "   *)\n";
my %events = ();

foreach my $nodeNo (@nodeNos) {
  my @nodeEvents = (); # initialize
  push @nodeEvents, 'MoveNode{toPos=(' . $nodePosX[$nodeNo] . "," .
                                         $nodePosY[$nodeNo] . ")}";
  @{$events{$nodeNo}} = @nodeEvents;
}

my $debugId = 1; # The (external) packet debug id

foreach my $nodeNo (@nodeNos) {
  # Okay, vi har to tællere: $maxMovementWait og $maxPacketWait -
  # vi skal tage hensyn til begge.

  my $step = 0; # Not to be confused with a CPN-step - this is just
                # number of times, an event chain related transition
                # will fire - i.e. just a counter!

  my $movementWait = int(rand $maxMovementWait) + 1;
  my $packetWait = int(rand $maxPacketWait) + 1;

  my @nodeEvents = @{$events{$nodeNo}};

  while ($step < $maxTime) {

    if ($movementWait < $packetWait) {
      push @nodeEvents, "WaitForTimeout{timeout=$movementWait}";

      # Moving the nodes
      $nodePosX[$nodeNo] += $nodeSpeedX[$nodeNo];
      $nodePosY[$nodeNo] += $nodeSpeedY[$nodeNo];

      # Did a node move out of the map?
      if ($nodePosX[$nodeNo] < 1) { $nodePosX[$nodeNo] = 1;
                                    $nodeSpeedX[$nodeNo] = 0; }
      if ($nodePosY[$nodeNo] < 1) { $nodePosY[$nodeNo] = 1;
                                    $nodeSpeedY[$nodeNo] = 0; }
      if ($nodePosX[$nodeNo] > $mapSize) { $nodePosX[$nodeNo] = $mapSize;
                                           $nodeSpeedX[$nodeNo] = 0; }
      if ($nodePosY[$nodeNo] > $mapSize) { $nodePosY[$nodeNo] = $mapSize;
                                           $nodeSpeedY[$nodeNo] = 0; }

      # Adjusting trajectories for next time
      $nodeSpeedX[$nodeNo] += int(rand(2*$maxSpeedChange + 1)) - $maxSpeedChange;
      $nodeSpeedY[$nodeNo] += int(rand(2*$maxSpeedChange + 1)) - $maxSpeedChange;

      # ...but not above the "speed limit"
      if ($nodeSpeedX[$nodeNo] < -$maxSpeed) { $nodeSpeedX[$nodeNo] = -$maxSpeed; }
      if ($nodeSpeedY[$nodeNo] < -$maxSpeed) { $nodeSpeedY[$nodeNo] = -$maxSpeed; }
      if ($nodeSpeedX[$nodeNo] > $maxSpeed) { $nodeSpeedX[$nodeNo] = $maxSpeed; }
      if ($nodeSpeedY[$nodeNo] > $maxSpeed) { $nodeSpeedY[$nodeNo] = $maxSpeed; }

      push @nodeEvents, 'MoveNode{toPos=(' . $nodePosX[$nodeNo] . "," .
                                             $nodePosY[$nodeNo] . ")}";

      $step += $movementWait; # Elapsed time
      $packetWait -= $movementWait; # There's now a shorter time to the next packet
      $movementWait = int(rand $maxMovementWait) + 1; # Time until the next movement
    }
    else {
      # i.e. $packetWait <= $movementWait, d.v.s., hvis en pakke skal
      # sendes og noderne flyttes rundt samtidigt, så bliver pakken
      # sendt først

      push @nodeEvents, "WaitForTimeout{timeout=$movementWait}";

      my $fromNode = $nodeNames[$nodeNo];
      my $toNode =   $nodeNames[int(rand $noOfNodes)];
      while ($fromNode eq $toNode) {
        $toNode = $nodeNames[int(rand $noOfNodes)];
```

```
        }
        push @nodeEvents, 'SendPacket{toNode="' . $toNode . '",' . "id=$debugId," .
            "data=SOMEDATA(\"pakke $debugId fra $fromNode til $toNode\")}";
        $debugId++;

        $step += $packetWait; # Elapsed time
        $movementWait -= $packetWait; # There's now a shorter time to the next movement
        $packetWait = int(rand $maxPacketWait) + 1; # Time until the next packet

      }
    }
    while ($step < $maxTime + $stepsWithMovementAfterNormalRun) {
      push @nodeEvents, "WaitForTimeout{timeout=$movementWait}";

      # Moving the nodes
      $nodePosX[$nodeNo] += $nodeSpeedX[$nodeNo];
      $nodePosY[$nodeNo] += $nodeSpeedY[$nodeNo];

      # Did a node move out of the map?
      if ($nodePosX[$nodeNo] < 1) { $nodePosX[$nodeNo] = 1;
                                    $nodeSpeedX[$nodeNo] = 0; }
      if ($nodePosY[$nodeNo] < 1) { $nodePosY[$nodeNo] = 1;
                                    $nodeSpeedY[$nodeNo] = 0; }
      if ($nodePosX[$nodeNo] > $mapSize) { $nodePosX[$nodeNo] = $mapSize;
                                           $nodeSpeedX[$nodeNo] = 0; }
      if ($nodePosY[$nodeNo] > $mapSize) { $nodePosY[$nodeNo] = $mapSize;
                                           $nodeSpeedY[$nodeNo] = 0; }

      # Adjusting trajectories for next time
      $nodeSpeedX[$nodeNo] += int(rand(2*$maxSpeedChange + 1)) - $maxSpeedChange;
      $nodeSpeedY[$nodeNo] += int(rand(2*$maxSpeedChange + 1)) - $maxSpeedChange;

      # ...but not above the "speed limit"
      if ($nodeSpeedX[$nodeNo] < -$maxSpeed) { $nodeSpeedX[$nodeNo] = -$maxSpeed; }
      if ($nodeSpeedY[$nodeNo] < -$maxSpeed) { $nodeSpeedY[$nodeNo] = -$maxSpeed; }
      if ($nodeSpeedX[$nodeNo] > $maxSpeed) { $nodeSpeedX[$nodeNo] = $maxSpeed; }
      if ($nodeSpeedY[$nodeNo] > $maxSpeed) { $nodeSpeedY[$nodeNo] = $maxSpeed; }

      push @nodeEvents, 'MoveNode{toPos=(' . $nodePosX[$nodeNo] . "," .
                                             $nodePosY[$nodeNo] . ")}";

      $step += $movementWait; # Elapsed time
      $packetWait -= $movementWait; # There's now a shorter time to the next packet
      $movementWait = int(rand $maxMovementWait) + 1; # Time until the next movement
    }
    @{$events{$nodeNo}} = @nodeEvents;
  }

  my @eventStrings = ();
  foreach my $nodeNo (sort (keys %events)) {
    my $id = 1;
    my @nodeEvents = @{$events{$nodeNo}};

    # ms af lister-version:
    push @eventStrings,
        "  1 '(\"" . $nodeNames[$nodeNo] . "\",[" .
        (join ",\n          ", (map {"$_"} @nodeEvents)) .
        "])";

    # ren ms-version:
    #push @eventStrings, map {"  (\"" . $nodeNames[$nodeNo] . "\"," . $id++ . ",$_)"}
    #                         @nodeEvents;
  }

  print (join " ++\n", @eventStrings);
  print "\n";
```

E.2 examine_result.pl benyttet i afsnit 5.2.1

Dette er scriptet examine_result.pl beskrevet i afsnit 5.2.1.

```
#!/usr/local/bin/perl -w

# File: examine_result.pl, $Revision: 1.13 $

use strict;

my $prefix = shift || die "Usage: $0 <prefix> <result_postfix>\n";
my $resultpostfix = shift || die "Usage: $0 <prefix> <result_postfix>\n";
my $inputfile = $prefix . "input.sml";
my $resultfile = $prefix . "result-" . $resultpostfix . ".txt";

open INPUT, $inputfile or die "Could not open $inputfile\n";

my @packets = ();
my $trans = 0;
my $successfulTookNoOfTrans = 0;
my $routeLengths = 0;

my $currentNode = "";

while (<INPUT>) {
  if (m/\("(\w+)",\[/) { # "
    $currentNode = $1;
  }
  if (m/SendPacket\{toNode="(\w+)",id=(\d+),data=SOMEDATA/) {
    push @packets, {fromNode=>$currentNode,
                    toNode=>$1,
                    debugId=>$2,
                    noOfTransmissions=>0,
                    noOfReceivedTransmissions=>0,
                    successful=>0,
                    successfulTookNoOfTrans=>0};
    #print "$currentNode $1 $2\n";
  }
}

close INPUT;

open RESULT, $resultfile or die "Could not open $resultfile\n";

while (<RESULT>) {
  if (m/"(\w+)",\[([^\]]*)\],\{.*debugId=(\d+),.*\},data=(\w+)DATA/) {
    my $fromNode = $1;
    my $toNodes = $2;
    my $debugId = $3;
    my $data = $4;

    #print "$fromNode $toNodes $debugId $data\n";

    my %packet = %{$packets[$debugId-1]};

    $trans++;
    $packet{noOfTransmissions}++;

    if (($data eq "SOME") &&
        ((index $toNodes, $packet{toNode}) > -1)) {

      $packet{successful}++;

      if ($packet{successful} == 1) {
        $successfulTookNoOfTrans += $packet{noOfTransmissions};
        $packet{successfulTookNoOfTrans} = $packet{noOfTransmissions};

        # Udregn rutelængden i pakken, der nåede frem (fra
        # DSR-optionsheaderen, ikke den faktisk benyttede
        # rutelængde).

        if (m/dsrSourceRoute\({.*,route=\[([A-Z,""]*)\]}\)/) {
          my $route = $1;
          $route =~ s/"//g; # "
          $route =~ s/,//g;

          # Afsender og modtager ikke er i denne liste, 2 skal
```

```
            # lægges til, hvis begge ønskes talt med -- 1, hvis kun
            # en af dem skal tælles med:
            $routeLengths += (1 + length $route);
          }
          else {
            die "Parse error: $_\n";
          }

        }
      }

      $toNodes =~ s/"//g; # "
      my @toNodes = split /,/, $toNodes;

      $packet{noOfReceivedTransmissions} += scalar @toNodes;

      $packets[$debugId-1] = \%packet;
    }
  }

  close RESULT;

  my $minTrans = 1000000000; # FIXME
  my $minTransNo = 0;
  my $maxTrans = 0;
  my $maxTransNo = 0;
  my $minTransSucc = 1000000000; # FIXME
  my $minTransNoSucc = 0;
  my $maxTransSucc = 0;
  my $maxTransNoSucc = 0;
  my $minTransRecv = 1000000000; # FIXME
  my $minTransRecvNo = 0;
  my $maxTransRecv = 0;
  my $maxTransRecvNo = 0;
  my $minTransRecvSucc = 1000000000; # FIXME
  my $minTransRecvNoSucc = 0;
  my $maxTransRecvSucc = 0;
  my $maxTransRecvNoSucc = 0;
  my $noSuccessful = 0;
  my $transSuccessful = 0;

  foreach my $packet (@packets) {
    my %packet = %$packet;
    print "Pakke " . $packet{debugId} .
        " fra " . $packet{fromNode} .
        " til " . $packet{toNode} .
        ", #transm.sendt: " . $packet{noOfTransmissions} .
        ", #transm.modt.: " . $packet{noOfReceivedTransmissions} .
        ", succes: " . $packet{successful} . "\n";

    if ($packet{successful} > 0) {
      $transSuccessful += $packet{noOfTransmissions};
    }

    if ($packet{noOfTransmissions} < $minTrans) {
      $minTrans = $packet{noOfTransmissions};
      $minTransNo = $packet{debugId};
    }
    if ($packet{noOfTransmissions} > $maxTrans) {
      $maxTrans = $packet{noOfTransmissions};
      $maxTransNo = $packet{debugId};
    }
    if ($packet{noOfReceivedTransmissions} < $minTransRecv) {
      $minTransRecv = $packet{noOfReceivedTransmissions};
      $minTransRecvNo = $packet{debugId};
    }
    if ($packet{noOfReceivedTransmissions} > $maxTransRecv) {
      $maxTransRecv = $packet{noOfReceivedTransmissions};
      $maxTransRecvNo = $packet{debugId};
    }
    if ($packet{successful}) {
      $noSuccessful++;

      if ($packet{noOfTransmissions} < $minTransSucc) {
        $minTransSucc = $packet{noOfTransmissions};
        $minTransNoSucc = $packet{debugId};
      }
      if ($packet{noOfTransmissions} > $maxTransSucc) {
```

```
        $maxTransSucc = $packet{noOfTransmissions};
        $maxTransNoSucc = $packet{debugId};
      }
      if ($packet{noOfReceivedTransmissions} < $minTransRecvSucc) {
        $minTransRecvSucc = $packet{noOfReceivedTransmissions};
        $minTransRecvNoSucc = $packet{debugId};
      }
      if ($packet{noOfReceivedTransmissions} > $maxTransRecvSucc) {
        $maxTransRecvSucc = $packet{noOfReceivedTransmissions};
        $maxTransRecvNoSucc = $packet{debugId};
      }
    }
  }

  print "\n";
  print "Mindste antal sendte transmissioner: $minTrans (id: $minTransNo)\n";
  print "Største antal sendte transmissioner: $maxTrans (id: $maxTransNo)\n";
  print "Mindste antal sendte transmissioner (v/succ): $minTransSucc (id: $minTransNoSucc)\n";
  print "Største antal sendte transmissioner (v/succ): $maxTransSucc (id: $maxTransNoSucc)\n";
  print "Mindste antal modtagne transmissioner: $minTransRecv (id: $minTransRecvNo)\n";
  print "Største antal modtagne transmissioner: $maxTransRecv (id: $maxTransRecvNo)\n";
  print "Mindste antal modtagne transmissioner (v/succ): $minTransRecvSucc (id: $minTransRecvNoSucc)\n";
  print "Største antal modtagne transmissioner (v/succ): $maxTransRecvSucc (id: $maxTransRecvNoSucc)\n";
  print "\n";
  print "Antal successfulde routninger: $noSuccessful (" . ($noSuccessful*100/(scalar @packets)) . "%)\n";
  print "Totalt antal pakketransmissioner: $trans\n";
  print "Antal pakketrans. før successfulde routninger: $successfulTookNoOfTrans\n";
  print "Totalt antal pakketrans. for successfulde pakker: $transSuccessful\n";
  print "Samlet rutelængde i DSR-optionsheaders på fremkomne pakker: $routeLengths\n";
```

F Måledata

F.1 Måledata fra afsnit 5.2.2

De rå måledata i f.eks. figur F.1 viser nogle tal, der kan virke meget "præcise" (f.eks. 91,67%). Dette skal sammenholdes med, at der i hver testkørsel blev simuleret sendt mellem 7 og 13 pakker. 91,67% svarer blot til, at 11 ud af 12 pakker nåede frem i en testkørsel.

Testkonfi-guration	Test nr.	Antal pakker, der nåede den endelige modtager			
		Ingen opt.	"CRR"	"SO"	"CRR"+"SO"
"Slow"	1	100,00%	100,00%	100,00%	100,00%
	2	100,00%	100,00%	100,00%	100,00%
	3	91,67%	91,67%	91,67%	91,67%
	4	100,00%	100,00%	100,00%	100,00%
	5	100,00%	100,00%	100,00%	100,00%
	6	100,00%	100,00%	100,00%	100,00%
	7	100,00%	100,00%	100,00%	100,00%
"Medium"	1	77,78%	77,78%	77,78%	77,78%
	2	88,89%	88,89%	88,89%	88,89%
	3	100,00%	100,00%	100,00%	100,00%
	4	100,00%	90,00%	100,00%	90,00%
	5	91,67%	91,67%	91,67%	91,67%
	6	100,00%	100,00%	100,00%	100,00%
	7	87,50%	87,50%	87,50%	87,50%
"Fast"	1	100,00%	87,50%	100,00%	87,50%
	2	100,00%	90,00%	100,00%	90,00%
	3	100,00%	100,00%	100,00%	100,00%
	4	100,00%	100,00%	100,00%	100,00%
	5	54,55%	54,55%	54,55%	54,55%
	6	57,14%	57,14%	57,14%	57,14%
	7	85,71%	85,71%	100,00%	85,71%

Tabel F.1: Måledata: Procentdel af de afsendte pakker, der nåede frem til den endelige modtager – afbildet i histogrammer i figur 5.2 på side 116

Testkonfi-	Test	Antal pakketransmissioner pr. “SendPacket”			
guration	nr.	Ingen opt.	“CRR”	“SO”	“CRR”+“SO”
“Slow”	1	4,64	3,91	4,73	4,00
	2	4,67	4,22	4,78	4,78
	3	4,58	3,50	4,50	3,50
	4	4,50	4,50	4,63	4,50
	5	7,38	6,46	7,23	6,54
	6	4,75	3,75	4,83	4,00
	7	4,88	4,50	4,88	4,50
“Medium”	1	6,78	6,33	8,11	7,11
	2	4,22	4,22	4,67	4,67
	3	6,50	4,20	4,90	5,50
	4	4,80	5,20	4,90	5,10
	5	6,83	4,75	7,08	5,67
	6	3,78	3,89	3,78	3,67
	7	9,88	11,13	11,50	10,63
“Fast”	1	8,13	5,75	8,25	5,75
	2	10,80	9,30	10,90	8,80
	3	4,88	4,75	4,75	6,38
	4	3,25	3,25	3,25	3,25
	5	7,45	8,09	6,91	8,18
	6	9,14	8,00	14,00	12,29
	7	6,00	6,29	13,57	13,43

Tabel F.2: Måledata: Gennemsnitligt antal pakketransmissioner i netværket pr. afsendt pakke – afbildet i histogrammer i figur 5.3 på side 117

Testkonfi-	Test	Antal pakketransmissioner i snit før fremkomst			
guration	nr.	Ingen opt.	“CRR”	“SO”	“CRR”+“SO”
“Slow”	1	3,18	2,45	3,18	2,45
	2	3,11	2,78	3,11	2,78
	3	3,09	2,27	2,91	2,27
	4	3,00	3,00	3,38	2,88
	5	5,69	5,08	5,77	4,92
	6	3,00	2,00	2,83	2,25
	7	2,88	2,50	3,13	2,50
“Medium”	1	3,71	3,14	3,71	3,14
	2	2,38	2,38	2,88	2,88
	3	4,40	2,40	3,10	3,00
	4	2,80	1,78	2,70	1,89
	5	3,55	2,27	3,82	3,09
	6	1,78	1,56	1,78	1,56
	7	6,43	7,57	7,14	6,86
“Fast”	1	6,63	3,00	6,75	3,00
	2	8,50	6,89	8,20	6,11
	3	1,88	2,25	2,00	3,63
	4	2,25	2,25	2,25	2,25
	5	5,83	5,67	5,17	5,67
	6	3,50	3,00	4,00	3,25
	7	3,33	2,83	11,14	3,17

Tabel F.3: Måledata: Gennemsnitligt antal pakketransmissioner før en pakke når frem til modtager for succesfuldt modtagede pakker – afbildet i histogrammer i figur 5.4 på side 118

G
Statistiske metoder

Til brug for analyserne i afsnit 5.3 et antal statistiske "værktøjer" nødvendige. Dette appendix giver et overblik over disse metoder baseret på [AF97, Sør98]. Disse er benyttet til at danne tabellerne i afsnit 5.3 ud fra de måledata, der blev fundet i afsnit 5.2.2 (og vises i appendix F.1).

G.1 Om testkørslerne

Som beskrevet i afsnit 5.2 er et sæt testkørsler simuleret i CPN-modellen af DSR, og en række måledata er udtrukket herfra som vist i tabel F.1-F.3 på side 212–213. Jeg benytter betegnelsen "testkørsel" i det efterfølgende for den præcise kommandoliste (se afsnit 3.6.3), der er benyttet som grundlag for en simulering af DSR-modellen i CPN Tools, og hvorfra et sæt måledata derefter er udtrukket og vist i de førnævnte tabeller.

Jeg har 3 grupper af testkørsler: Som beskrevet i afsnit 5.1.3 opererer jeg med 3 forskellige konfigurationer ("slow", "medium" og "fast"), og det er i hver af disse, jeg har foretaget simuleringerne af 7 forskellige (og dermed uafhængige) testkørsler. For hver af tabellerne med måledata (tabel F.1-F.3 på side 212–213) giver dette i princippet 21 uafhængige observationer at arbejde med. Hver af disse observationer vil jeg betegne $x_{i,h},\ i = 1,\ldots,7,\ h \in \{\mathrm{s},\mathrm{m},\mathrm{f}\}$.

Hver af disse observationer er imidlertid igen underopdelt i fire observationer, der er *afhængige* af hinanden: Hver kommandoliste er simuleret fire gange, hvor DSR-modellen henholdsvist er konfigureret til ikke at slå nogle af de modellerede udvidelser af DSR til, til at slå "Cached Route Reply" (CRR) til, til at slå "Salvage Operations" (SO) til og endelig til at slå begge udvidelser til. Dette giver i alt 84 observationer betegnet $x_{i,h,u},\ i = 1,\ \ldots,7,\ h \in \{\mathrm{s},\mathrm{m},\mathrm{f}\},\ u \in \{\emptyset,\{\mathrm{c}\},\{\mathrm{s}\},\{\mathrm{c},\mathrm{s}\}\}$.

I mine analyser vil jeg gerne arbejde med at sammenligne, hvad der sker med observationerne, når jeg ændrer en konfiguration fra f.eks. "slow" til "fast" (ændring af h) eller når jeg slår en udvidelse til eller fra (ændring af u).

At de fire observationer på tværs af udvidelserne ($u \in \{\emptyset,\{\mathrm{c}\},\{\mathrm{s}\},\{\mathrm{c},\mathrm{s}\}\}$) er gjort afhængige af hinanden, d.v.s. at det er den samme testkørsel, der er brugt til hver af simuleringerne, kan virke som en hæmsko i det videre arbejde, idet forskellige beregningsmetoder skal benyttes, alt efter om man ønsker at sammenligne på tværs af hastighed eller udvidelser, men det giver som senere beskrevet en fordel, når man sammenligner observationer fra forskellige udvidelseskonfigurationer: Standardafvigelsen bliver mindre, og de fundne konfidensintervaller bliver derfor mere præcise.

Det ville derfor have givet mening også at lade observationerne på tværs af hastigheden ($h \in \{\mathrm{s},\mathrm{m},\mathrm{f}\}$) være afhængige af hinanden, men dette var umiddelbart sværere at gøre, idet det ville betyde, at kommandolisten (se afsnit 3.6.3) skulle gentænkes, så `MoveNode`-kommandoer og `SendPacket`-kommandoer ikke længere foregår i samme tempo hver gang – altså så `MoveNode`-kommandoerne for en knude ville kunne "afspilles" f.eks. fem gange så hurtigt som knudens `SendPacket`-kommandoer. Derfor er kun observationer på tværs af benyttede udvidelser gjort afhængige af hinanden.

G.2 Er måledataene normalfordelte?

For at man kan analysere på dataene, skal det verificeres, at de er normalfordelte, d.v.s. at $x_1, \ldots, x_7$ i hver gruppe af testkørsler er observationer af uafhængige, identisk fordelte stokastiske variable $X_1,\ldots,X_7$, så $X_i \sim N(\mu,\sigma^2),\ i = 1,\ldots,7$, hvor μ betegner middelværdien og σ betegner standardafvigelsen. Som beskrevet i [Sør98, kapitel 2] kan en grafisk metode (probitanalyser, der resulterer i probitdiagrammer) benyttes til at afgøre dette.

Her skal de 84 observationer ikke ses som en samlet observationsrække, da alle (ikke-sammenlignende) analyser kun foregår inden for en enkelt kombination af $h \in \{\mathrm{s},\mathrm{m},\mathrm{f}\}$ og $u \in \{\emptyset,\{\mathrm{c}\},\{\mathrm{s}\},\{\mathrm{c},\mathrm{s}\}\}$, hvorfor jeg kun viser, at observationerne inden for hver af disse 12 grupper er normalfordelte.

Probitdiagrammer for dataene i tabel F.1 på side 212 kan ses i appendix H.1, for dataene i tabel F.2 på side 213 kan de ses i appendix H.2, og for dataene i tabel F.3 på side 213 kan de ses i appendix H.3.

Probitdiagrammerne skal tilnærmelsesvist udgøre en lige linie (specielt omkring midten), før man kan konkludere, at dataene er normalfordelte. Denne linie vil dog kun meget tilnærmelsesvist ligne en lige linie ved meget lave observationsantal, som der er i dette tilfælde. Man kan dog se, at probitdiagrammerne i høj grad ligner probitdiagrammerne fra [Sør98, side 21], hvor eksempler for grupper med 5 observationer er vist.

Specielt skal man lægge mærke til, at hvis man sammenligner f.eks. probitdiagrammet for "slow"-, "medium"- og "fast"-konfigurationen for en fast udvidelseskonfiguration (f.eks. figur H.4(a), H.5(a) og H.6(a) på side 221–222) for et sæt måledata, så viser de kun meget tilnærmelsesvist lige linier, men måden, de grafisk er forskellige fra en lige linie, afviger fra figur til figur. Dette sker ikke, hvis man i stedet holder hastigheden fast og varierer hvilke udvidelser, der er slået til, da måledataene her netop er afhængige af hinanden (sammenlign f.eks. figur H.4(a)-H.4(d) på side 221).

På baggrund af dette vil jeg konkludere, at måledataene er normalfordelte.

G.3 Parameterinferens

Man kan ikke finde den sande værdi for μ (middelværdien) og σ (variansen), da jeg kun har måledata til rådighed. Man kan derimod estimere disse v.h.a. statistisk inferens – inferens betyder netop "forudsigelser om populationen på basis af observationer". Fra [AF97, Sør98] får man (for $h \in \{\mathrm{s},\mathrm{m},\mathrm{f}\},\ u \in \{\emptyset,\{\mathrm{c}\},\{\mathrm{s}\},\{\mathrm{c},\mathrm{s}\}\},\ n = n_{h,u} = 7$ observationer) følgende:

Observationsmiddelværdi:

$$\bar{x}_{h,u\cdot} = \frac{1}{n}\sum_{i=1}^{n} x_{i,h,u}$$

Varians:

$$s^2_{h,u} = \frac{1}{n-1}\sum_{i=1}^{n}(x_{i,h,u} - \bar{x}_{h,u\cdot})^2$$

Standardafvigelse:

$$s_{h,u} = \sqrt{s_{h,u}^2}$$

Standardfejl:

$$e_{h,u} = \frac{s_{h,u}}{\sqrt{n}}$$

Frihedsgrader:

$$f = n - 1$$

***t*-fraktil:**

$t_p(f)$ findes v.h.a. tabel i f.eks. [Blæ] eller [AF97]

95% konfidensinterval for middelværdien:

$$\bar{x}_{h,u}. - t_{.025}(f) * e_{h,u} \leq \mu_{h,u} \leq \bar{x}_{h,u}. + t_{.025}(f) * e_{h,u}$$

Bemærk, at jeg i de viste formler benytter en *t*-fordeling i stedet for en normalfordeling. Ifølge [AF97] bør man som hovedregel gøre dette, når man (som i dette tilfælde) har observationsgrupper på 30 observationer eller derunder.

Observationerne i "slow"-, "medium"- og "fast"-konfigurationerne er som tidligere nævnt uafhængige af hinanden. Jeg kunne i visse tilfælde vælge at slå observationerne fra simuleringerne af testkørslerne under disse tre konfigurationer sammen for at få flere observationer at se på – på tværs af hastighederne. Hastighederne i disse tilfælde er imidlertid ikke valgt tilfældigt, men er valgt til gennemsnitligt at være én af tre værdier, og resultaterne, der vil komme ud af denne analyse, vil lade til at være mere præcis (fordi de er baseret på tre gange på mange observationer), men den ikke-randomiserede generering af testene, der ligger til grundlag for testkørslerne, gør, at man ikke nødvendigvis kan benytte disse resultater til noget. Jeg har derfor valgt ikke at forsøge at gøre dette i analyserne i dette kapitel.

I afsnit 5.3.1 benyttes de ovenstående formler til at udregne de nævnte parametre for de i afsnit 5.2.2 fundne måledata.

G.4 Sammenligning af observationsgrupper på tværs af hastighed

Når jeg i de efterfølgende afsnit ønsker at foretage sammenligninger af observationsgrupper, ønsker jeg at gøre det på to forskellige måder. Dels sammenligner jeg på tværs af hastighed, dels på tværs af benyttede udvidelser i DSR-protokollen. I dette afsnit vil jeg finde formler, der kan benyttes til udregning af f.eks. 95% konfidensintervaller for det første af tilfældene – sammenligning på tværs af hastighed. I næste afsnit findes formlerne for sammenligning på tværs af benyttede udvidelser.

Som beskrevet i appendix G.1 er de enkelte observationer i observationsgrupperne uafhængige, når man sammenligner dem på tværs af den benyttede hastighed (d.v.s. på tværs af "slow"-, "medium"- og "fast"-konfigureringen af testkørslerne).

Jeg ønsker at sammenligne middelværdierne for to sådanne grupper for at se, om man f.eks. kan sige noget generelt om det gennemsnitlige antal transmissioner, når man sætter knudernes hastighed op. Fra [AF97] får man, at når man kun har et lavt antal observationer (under 20 i mindst en af grupperne), bør man som hovedregel benytte en *t*-fordeling i stedet for en

normal-fordeling. I praksis udregnes et 95% konfidensinterval for f.eks. $\mu_{f,u} - \mu_{s,u}$. Hvis hele dette konfidensinterval ligger under (eller over) 0, kan man udlede, at man med 95% konfidens har en middelværdi, der (som et snit over *alle* de mulige kørsler i disse konfigurationer) falder (eller stiger), når man går fra en "slow"-konfiguration til en "fast"-konfiguration.

Når jeg i det ovenstående siger, at man med 95% konfidens ved, at middelværdien ligger i et interval, så skal det ikke forstås på den måde, at der er 95% chance for, at det fundne konfidensinterval indeholder den rigtige middelværdi – enten indeholder konfidensintervallet middelværdien, eller også gør den ikke. Konfidensintervallet er ikke en sandsynlighed og bør ikke fortolkes som sådan. Den rigtige fortolkning er, at hvis man laver et antal nye sæt (af samme størrelse som det sæt, jeg kigger på) af simuleringer af nye testkørsler, og udregner konfidensintervallet for hvert af disse sæt, vil 95% af disse konfidensintervaller indeholde den rigtige middelværdi. Når jeg i dette appendix og i kapitel 5 skriver, at middelværdien med 95% konfidens ligger i et interval, skal det altså tolkes på denne måde.

Når 95% konfidensintervallet for forskellen på μ_1 og μ_2 skal udregnes, benyttes jvf. [AF97, Sør98] følgende formler:

Standardfejl:

$$e_{\text{diff}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Frihedsgrader:

$$c = \frac{s_1^2/n_1}{s_1^2/n_1 + s_2^2/n_2}$$

$$f_{\text{diff}} = \left\{ \frac{c^2}{n_1 - 1} + \frac{(1-c)^2}{n_2 - 1} \right\}^{-1}$$

***t*-fraktil:**

$t_p(f_{\text{diff}})$ findes v.h.a. tabel i f.eks. [Blæ] eller [AF97]

95% konfidensinterval for forskellen på middelværdien:

$$(\bar{x}_{2\cdot} - \bar{x}_{1\cdot}) - t_{.025}(f_{\text{diff}}) * e_{\text{diff}} \leq \mu_{\text{diff}} \leq (\bar{x}_{2\cdot} - \bar{x}_{1\cdot}) + t_{.025}(f_{\text{diff}}) * e_{\text{diff}}$$

I afsnit 5.3.2 benyttes disse formler til at undersøge, om ændringer i knudernes hastighed har en effekt på nogle af de i afsnit 5.2.2 fundne måledata.

G.5 Sammenligning af observationsgrupper på tværs af udvidelser

I modsætning til situationen i forrige afsnit har man, når man vil sammenligne observationsgrupper på tværs af de benyttede udvidelser (d.v.s. på tværs af, om "Cached Route Reply" (CRR) og/eller "Salvage Operations" (SO) er slået til eller fra), *afhængige* observationer. Dette skyldes som forklaret i appendix G.1, at det er de samme testkørsler, der benyttes i de fire forskellige konfigurationer af CPN-modellen af DSR.

Når man som i forrige afsnit ønsker at sammenligne middelværdierne for to sådanne grupper (bestående af observationerne x_{i1} og $x_{i2},\ i = 1, \ldots, n$), får man følgende formler fra [AF97, Sør98]:

Middelværdi af differencerne:

$$\bar{d}. = \frac{1}{n}\sum_{i=1}^{n}(x_{i2} - x_{i1})$$

Standardafvigelse af differencerne:

$$s_{\text{diff}} = \sqrt{\frac{\sum_{i=1}^{n}((x_{i2} - x_{i1}) - \bar{d}.)^2}{n-1}}$$

Standardfejl:

$$e_{\text{diff}} = \frac{s_{\text{diff}}}{\sqrt{n}}$$

Frihedsgrader:

$$f_{\text{diff}} = n - 1$$

***t*-fraktil:**

$t_p(f_{\text{diff}})$ findes v.h.a. tabel i f.eks. [Blæ] eller [AF97]

95% konfidensinterval for forskellen på middelværdien:

$$\bar{d}. - t_{.025}(f_{\text{diff}}) * e_{\text{diff}} \leq \mu_{\text{diff}} \leq \bar{d}. + t_{.025}(f_{\text{diff}}) * e_{\text{diff}}$$

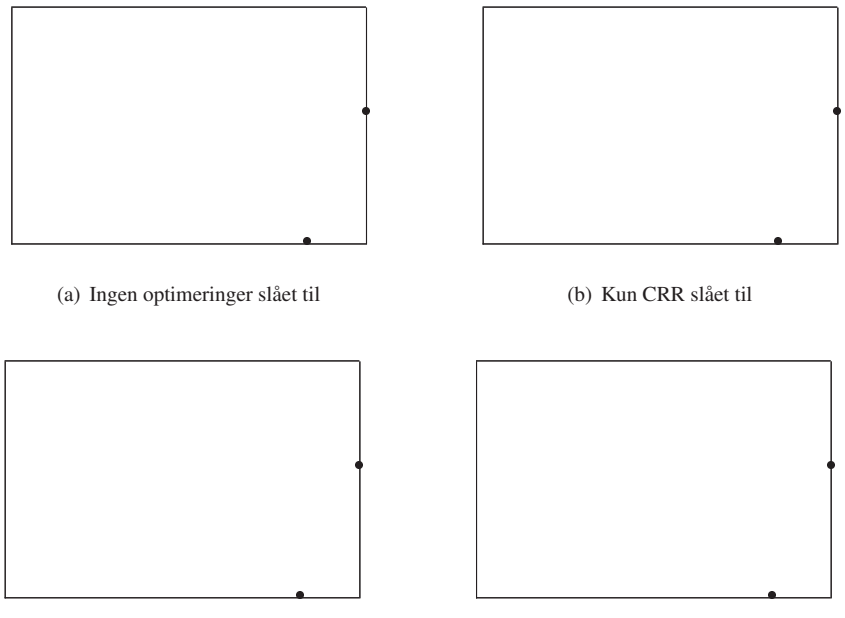
Bemærk, at konfidensintervallet vil være smallere her end i det uafhængige tilfælde (appendix G.4). Dette skyldes, at der i det uafhængige tilfælde både vil indgå måleusikkerhed og variation i de enkelte observationers niveau. Fra [Sør98] får man, at den sidste faktor bliver minimeret, når man kigger på det afhængige tilfælde.

I afsnit 5.3.3 og 5.3.4 benyttes disse formler til at undersøge, om ændringer af de benyttede udvidelser har en effekt på nogle af de måledata, der blev fundet i afsnit 5.2.2.

H

Analyseunderstøttende probitdiagrammer

H.1 Probitdiagrammer for pakkefremkomst-procentdele fra testkørslerne i tabel F.1 (afsnit 5.2.2)



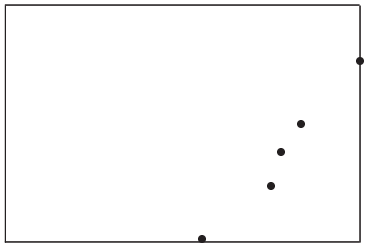
(a) Ingen optimeringer slået til

(b) Kun CRR slået til

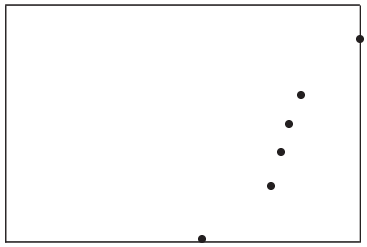
(c) Kun SO slået til

(d) Både CRR og SO slået til

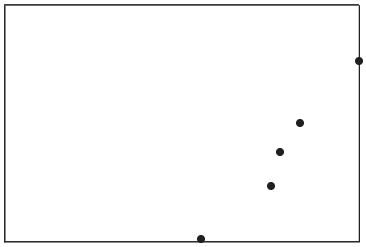
Figur H.1: Probitdiagrammer for tabel F.1 på side 212 (i “slow”-konfigurationen)



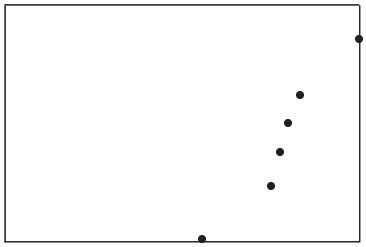
(a) Ingen optimeringer slået til



(b) Kun CRR slået til

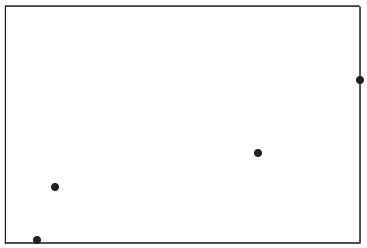


(c) Kun SO slået til

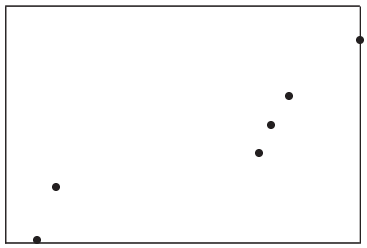


(d) Både CRR og SO slået til

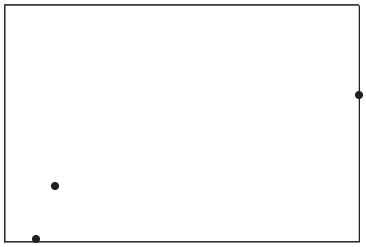
Figur H.2: Probitdiagrammer for tabel F.1 på side 212 (i "medium"-konfigurationen)



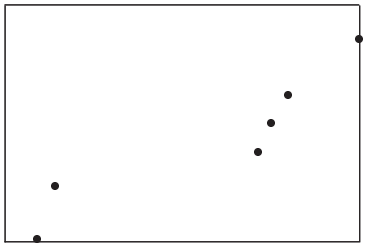
(a) Ingen optimeringer slået til



(b) Kun CRR slået til



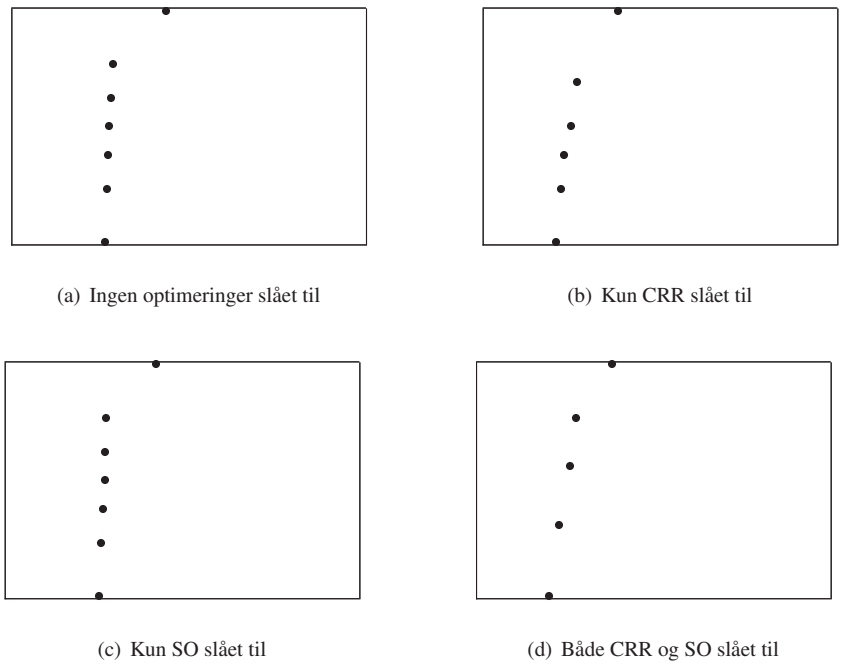
(c) Kun SO slået til



(d) Både CRR og SO slået til

Figur H.3: Probitdiagrammer for tabel F.1 på side 212 (i "fast"-konfigurationen)

H.2 Probitdiagrammer for antal pakketransmissioner fra testkørslerne i tabel F.2 (afsnit 5.2.2)



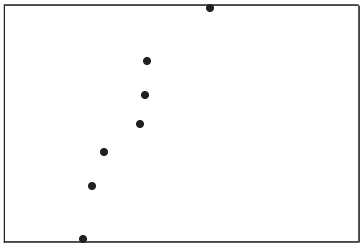
(a) Ingen optimeringer slået til

(b) Kun CRR slået til

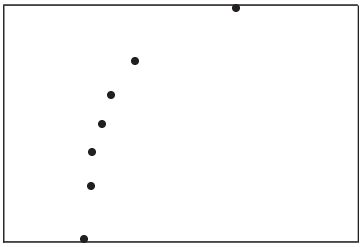
(c) Kun SO slået til

(d) Både CRR og SO slået til

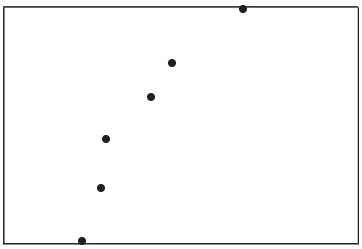
Figur H.4: Probitdiagrammer for tabel F.2 på side 213 (i “slow”-konfigurationen)



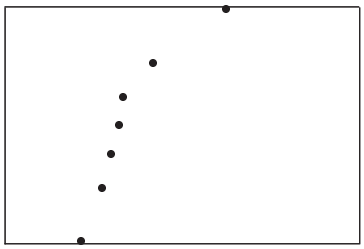
(a) Ingen optimeringer slået til



(b) Kun CRR slået til

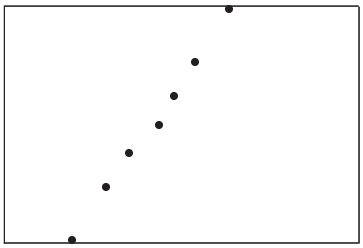


(c) Kun SO slået til

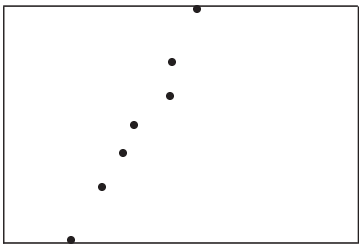


(d) Både CRR og SO slået til

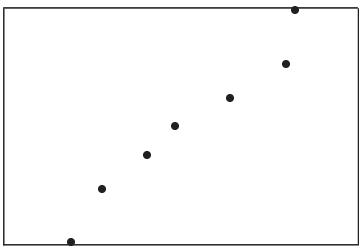
Figur H.5: Probitdiagrammer for tabel F.2 på side 213 (i "medium"-konfigurationen)



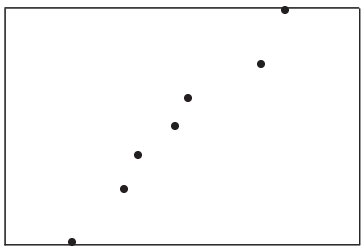
(a) Ingen optimeringer slået til



(b) Kun CRR slået til



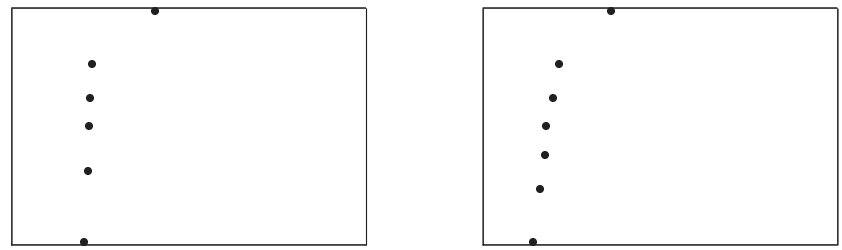
(c) Kun SO slået til



(d) Både CRR og SO slået til

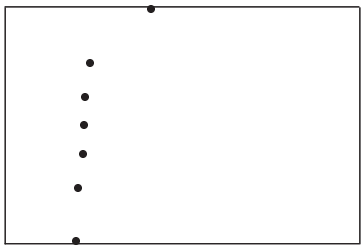
Figur H.6: Probitdiagrammer for tabel F.2 på side 213 (i "fast"-konfigurationen)

H.3 Probitdiagrammer for antal pakketransmissioner før pakkefremkomst fra testkørslerne i tabel F.3 (afsnit 5.2.2)

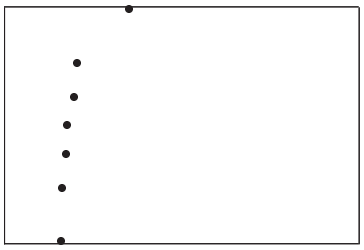


(a) Ingen optimeringer slået til

(b) Kun CRR slået til

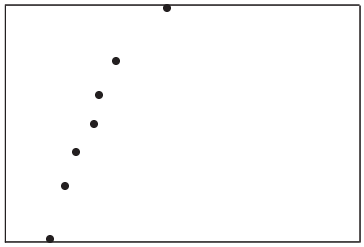


(c) Kun SO slået til

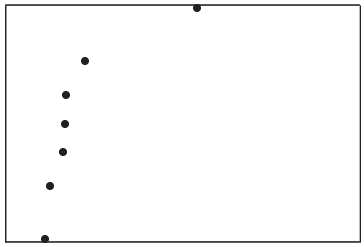


(d) Både CRR og SO slået til

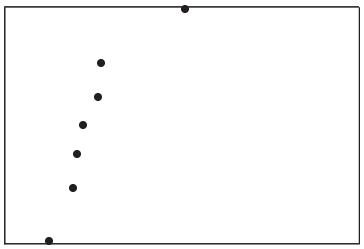
Figur H.7: Probitdiagrammer for tabel F.3 på side 213 (i “slow”-konfigurationen)



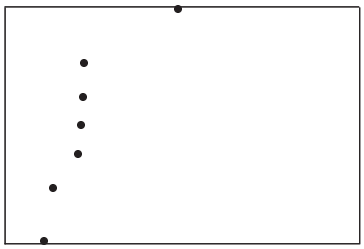
(a) Ingen optimeringer slået til



(b) Kun CRR slået til

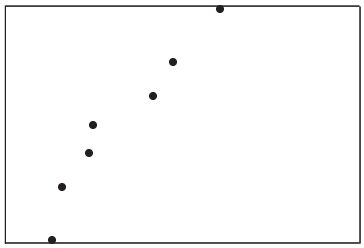


(c) Kun SO slået til

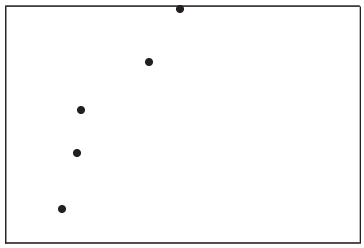


(d) Både CRR og SO slået til

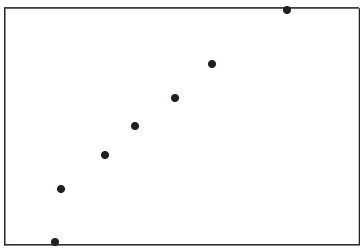
Figur H.8: Probitdiagrammer for tabel F.3 på side 213 (i "medium"-konfigurationen)



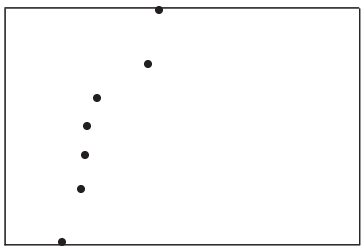
(a) Ingen optimeringer slået til



(b) Kun CRR slået til



(c) Kun SO slået til



(d) Både CRR og SO slået til

Figur H.9: Probitdiagrammer for tabel F.3 på side 213 (i "fast"-konfigurationen)

Litteratur

[AF97] Alan Agresti og Barbara Finlay. *Statistical Methods for the Social Sciences*. Prentice Hall, 3. udgave, 1997.

[AJ03] Eitan Altman og Tania Jiménez. NS Simulator for beginners. Lecture notes, 2003-2004, december 2003.

[BGH04] Jonathan Billington, Guy Edward Gullasch og Bing Han. A Coloured Petri Net Approach to Protocol Verification. I Jörg Desel, Wolfgang Reisig og Grzegorz Rozenberg, redaktører, *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, bind 3098 af *Lecture Notes in Computer Science*, side 210-290. Springer-Verlag, juni 2004.

[Blæ] Preben Blæsild. *Statistical Tables*.

[BMJ+98] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu og Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. I *Proceedings of 4th Annual International Conference on Mobile Computing and Networking (MobiCom '98)*, side 85-97, 1998.

[CBD02] Tracy Camp, Jeff Boleng og Vanessa Davies. A Survey of Mobility Models for Ad Hoc Network Research. *Wireless Communication & Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Application*, 2(5):483-502, 2002.

[CDK05] George Coulouris, Jean Dollimore og Tim Kindberg. *Distributed Systems – Concepts and Design*. Addison-Wesley, 4. udgave, 2005.

[CJ03] T. Clausen og P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, Network Working Group, oktober 2003. URL: `http://www.ietf.org/rfc/rfc3626.txt`.

[CK04] Jong-Mu Choi og Young-Bae Ko. A Performance Evaluation for Ad Hoc Routing Protocols in Realistic Military scenarios. I *Proc. of the 9th International Conference on Cellular and Intelligent Communications (CIC '04)*, oktober 2004.

[CM99] S. Corson og J. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, Network Working Group, januar 1999. URL: `http://www.ietf.org/rfc/rfc2501.txt`.

[CMB96] M. Scott Corson, Joseph Macker og Stephen G. Batsell. Architectural Considerations for Mobile Mesh Networking. I *Proceedings of Military Communications Conference (MILCOM '96)*, bind 1, side 225-229, oktober 1996.

[Com95] Douglas E. Comer. *Internetworking With TCP/IP*, bind I: Principles, Protocols, and Architecture. Prentice Hall, 3. udgave, 1995.

[CT] CPN Tools. Hjemmeside. URL: `http://www.daimi.au.dk/CPNtools/`.

[CTS] CPN Tools-support. Mailingliste. URL: http://mailman.daimi.au.dk/mailman/listinfo/cpntools-support.

[DC] Design/CPN. Hjemmeside. URL: http://www.daimi.au.dk/designCPN/.

[Dem01] Tamer Demir. Simulation of Ad Hoc Networks with DSR Protocol. I *Proceedings of the Sixteenth International Symposium on Computer and Information Sciences*, november 2001.

[Dij59] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269-271, oktober 1959.

[Dro97] R. Droms. Dynamic Host Configuration Protocol. RFC 2131, Network Working Group, marts 1997. URL: http://www.ietf.org/rfc/rfc2131.txt.

[FL01] James A. Freebersyser og Barry Leiner. A DoD Perspective on Mobile Ad Hoc Networks. I Charles E. Perkins, redaktør, *Ad Hoc Networking*, kapitel 2, side 29-51. Addison-Wesley, 2001.

[For03] Behrouz A. Forouzan. *Local Area Networks*. McGraw-Hill, 1. udgave, 2003.

[IBM96] IBM. Local Area Network Concepts and Products: Routers and Gateways. Redbook SG24-4755-00, IBM International Technical Support Organization, Raleigh Center, maj 1996. URL: http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg244755.html.

[IEE] IEEE 802.11 WLAN Working Group. IEEE 802.11 Wireless Local Area Networks – The Working Group for WLAN Standards. Hjemmeside. URL: http://grouper.ieee.org/groups/802/11/.

[IETa] IETF Secretariat. MANET Status Pages. Hjemmeside. URL: http://tools.ietf.org/wg/manet/.

[IETb] IETF Secretariat. Mobile Ad-hoc Networks (manet) Charter. Hjemmeside. URL: http://www.ietf.org/html.charters/manet-charter.html.

[JCK02] Kurt Jensen, Søren Christensen og Lars M. Kristensen. *CPN Tools Occurrence Graph Manual*. Computer Science Department, University of Aarhus, 2002. Version 0.1. URL: http://wiki.daimi.au.dk/cpntools-help/_files/manual.pdf.

[Jen92] Kurt Jensen. *Coloured Petri Nets*, bind 1: Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag, 1992.

[Jen94] Kurt Jensen. An Introduction to the Theoretical Aspects of Coloured Petri Nets. DAIMI PB 476, Computer Science Department, University of Aarhus, august 1994.

[JM96] David B. Johnson og David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. I Tomasz Imielinski og Hank Korth, redaktører, *Mobile Computing*, kapitel 5, side 153-181. Kluwer Academic Publishers, 1996.

[JMB01] David B. Johnson, David A. Maltz og Josh Broch. DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks. I Charles E. Perkins, redaktør, *Ad Hoc Networking*, kapitel 5, side 139-172. Addison-Wesley, 2001.

[JMC+01] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum og L. Viennot. Optimized Link State Routing Protocol for Ad Hoc Networks. I *IEEE INMIC Pakistan*, 2001.

[JMH05] David B. Johnson, David A. Maltz og Yih-Chun Hu. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR). Internet Draft Version 10, IETF MANET Working Group, januar 2005. Work in progress, opdaterede versioner tilgængelige fra `http://www.ietf.org/ids.by.wg/manet.html`. URL: `http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-10.txt`.

[JT87] John Jubin og Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21-32, januar 1987.

[KCC05] Stuart Kurkowski, Tracy Camp og Michael Colagrosso. MANET Simulation Studies: The Incredibles. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9(4):50-61, oktober 2005.

[KCJ98] Lars M. Kristensen, Søren Christensen og Kurt Jensen. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98-132, 1998.

[KJJ04] Lars Michael Kristensen, Jens Bæk Jørgensen og Kurt Jensen. Application of Coloured Petri Nets in System Development. I Jörg Desel, Wolfgang Reisig og Grzegorz Rozenberg, redaktører, *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, bind 3098 af *Lecture Notes in Computer Science*, side 626-685. Springer-Verlag, juni 2004.

[Koz05] Charles M. Kozierok. *The TCP/IP Guide – A Comprehensive, Illustrated Internet Protocols Reference*. No Starch Press, 2005.

[KV98] Young-Bae Ko og Nitin H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. I *Proceedings of 4th Annual International Conference on Mobile Computing and Networking (MobiCom '98)*, side 66-75, oktober 1998.

[MAN] MANET – Mobile Ad-hoc Networks. Mailingliste. URL: `https://www1.ietf.org/mailman/listinfo/manet`.

[MC98] Joseph P. Macker og M. Scott Corson. Mobile Ad Hoc Networking and the IETF. *ACM Mobile Computing and Communications Review*, 2(1):9-14, januar 1998.

[Mor00] Kjeld Høyer Mortensen. Automatic Code Generation Method Based on Coloured Petri Net Models Applied on an Access Control System. I Mogens Nielsen og Dan Simpson, redaktører, *Application and Theory of Petri Nets 2000, 21st International Conference*, bind 1825 af *Lecture Notes in Computer Science*, side 367-386. Springer-Verlag, juni 2000.

[MU90] Matematiklærerforeningen og Undervisningsministeriet. *Matematisk Formelsamling, Matematisk linje, Højt niveau*. Matematiklærerforeningen og Undervisningsministeriet, Direktoratet for Gymnasieskolerne og Højere Forberedelseseksamen, 1990.

[Nor05] Erik Nordstrom. AdHoc@UU: DSRUUImpl: DSR-UU – implementation. Hjemmeside, juni 2005. URL: `http://core.it.uu.se/AdHoc/DsrUUImpl`.

[NS2] The Network Simulator - ns-2. Hjemmeside. URL: `http://www.isi.edu/nsnam/ns/`.

[PB94] Charles E. Perkins og Pravin Bhagwat. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. I *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, side 234-244, 1994.

[PB01] Charles E. Perkins og Pravin Bhagwat. DSDV Routing over a Multihop Wireless Network of Mobile Computers. I Charles E. Perkins, redaktør, *Ad Hoc Networking*, kapitel 3, side 53-74. Addison-Wesley, 2001.

[PBRD03] C. Perkins, E. Belding-Royer og S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, Network Working Group, juli 2003. URL: `http://www.ietf.org/rfc/rfc3561.txt`.

[Per05] Charles E. Perkins. IP Flooding in Ad hoc Mobile Networks. Internet Draft Version 2, Mobile Ad Hoc Networking Working Group, juni 2005. Work in progress.

[PR01] Charles E. Perkins og Elizabeth M. Royer. The Ad Hoc On-Demand Distance-Vector Protocol. I Charles E. Perkins, redaktør, *Ad Hoc Networking*, kapitel 6, side 173-219. Addison-Wesley, 2001.

[PRG] Protean Research Group. MANET Implementation Survey. Hjemmeside. URL: `http://tang.itd.nrl.navy.mil/5522/manet/survey.html`.

[RMP00] Rice Monarch Project. Monarch Project: Implementation of Dynamic Source Routing. Hjemmeside, april 2000. URL: `http://www.monarch.cs.rice.edu/dsr-impl.html`.

[Sel03] Bran Selic. The Pragmatics of Model-Driven Development. *IEEE Software*, 20(5):19-25, september-oktober 2003.

[SNT] Scalable Network Technologies. Qualnet. Hjemmeside. URL: `http://www.scalable-networks.com/`.

[Son01] Alex Song. picoNet II – a wireless ad hoc network for mobile handheld devices. Specialeafhandling, Department of Information Technology and Electrical Engineering, University of Queensland, Australia, oktober 2001. URL: `http://piconet.sourceforge.net/thesis/`.

[Sta00] William Stallings. *Data & Computer Communications*. Prentice Hall, 6. udgave, 2000.

[SW99] Myra L. Samuels og Jeffrey A. Witmer. *Statistics for the Life Sciences*. Prentice Hall, 2. udgave, 1999.

[Sør98] Michael Sørensen. *Basal Statistik*. Department of Theoretical Statistics, University of Aarhus, juli 1998.

[Toh02] C.-K. Toh. *Ad Hoc Mobile Wireless Networks – Protocols and Systems – Age of Pervasive Mobile Networking and Computing*. Prentice Hall PTR, 2002.

[TV01] C.-K. Toh og Vasos Vassiliou. The Effects of Beaconing on the Battery Life of Ad Hoc Mobile Computers. I Charles E. Perkins, redaktør, *Ad Hoc Networking*, kapitel 9, side 299-321. Addison-Wesley, 2001.

[Ull98] Jeffrey D. Ullman. *Elements of ML Programming, ML97 Edition*. Prentice Hall, 2. udgave, 1998.

[WFA03] Wi-Fi Alliance. Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks. White Paper, april 2003. URL: `http://www.wi-fi.org/membersonly/getfile.asp?f=Whitepaper_Wi-Fi_Security4-29-03.pdf`.

[WZK05] Adam Wierzbicki, Aneta Zwierko og Zbigniew Kotulski. A New Authentication Protocol for Revocable Anonymity in Ad-Hoc Networks. I M.H. Hamza, redaktør, *Proceedings of the IASTED International Conference on Communication, Network, and Information Security, CNIS 2005*, side 30-35, november 2005.

[YGK03] Yunjung Yi, Mario Gerla og Taek Jin Kwon. Efficient Flooding in Ad hoc Networks: a Comparative Performance Study. I *Proc. of the IEEE International Conference on Communications*, maj 2003.