Universitetet i Oslo
Institutt for informatikk

Flerkonfigurasjons-ruting (MRC) som utgangspunkt for resursallokering i datanettverk

Multiple Routing Configurations (MRC) As A Basis For Resource Allocation In Computer Networks

Johan H. W. Basberg

Cand. Scient. Hovedfag

Mai MMVII

Innhold

Figurer

Tabeller

Sammendrag

I denne oppgaven presenterer vi våre funn relatetert til bruk av Flerkonfigurasjonsruting (MRC, eng.: Multiple Routing Configurations) i datanettverk som benytter en sentral nettverksadministrator, slik som en båndbreddemegler (eng.: Bandwidth Broker) til håndtering av nettverksresurser og tjenester (IntServ, eng.: Integrated Services). Til dette har vi utviklet simulatoren SAK (Simulering av kapasitet).

For å bedre formidle hvordan MRC fungerer, og hva som skjer under allokering av nettverksresurser har vi vektlagt visualisering. SAK viser derfor både isolerings- og allokeringsprosessen i sanntid, hvis brukeren ønsker kan disse prosessene ytterligere forsinkes (eller akselereres), slik at det blir lettere (vanskeligere) å følge med på hva som skjer.

Vi har også sett nærmere på i hvilken grad allokeringskapasiteten påvirkes av antall konfigurasjoner som produseres av MRC, hvorvidt mange eller få konfigurasjoner gir den beste allokeringskapasiteten.

Våre eksperimenter gir en sterk indikasjon på at MRC yter bedre enn ruting etter en enkelt rutetabell og topologiens korteste sti fra inngangs- til utgangsnode. Vi fant også en klar indikasjon på at antall konfigurasjoner øker allokeringskapasiteten i nettverket.

Forord

Jeg vil gjerne takke min veileder Olav Lysne for god hjelp, tålmodighet og støtte underveis i denne altfor lange prosessen. Takk også til Audun Fosselie Hansen for topologier, innspill og god hjelp, Amund Kvalheim for kommentarer underveis og bidrag i begynnelsen av oppgaven. Takk til Joakim Jacobsen for helt nødvendig hjelp med statistikken. Jeg vil også rette en kjempetakk til kjæresten min Linlin Liu, som har vært en fantastisk hjelp hele veien. Takk også til min venn Jon Mugaas, min far Jon Erik Basberg og min mor Line Kloed Tandberg for både bistand og god støtte. Takk til hele familien Austvik, som var en helt nødvendig distraksjon fra tid til annen. Takk til Terje Sanner for korrekturlesing og gode tilbakemeldinger, familien Wenstøp for nødvendig bistand, og Rune Spaans for grafisk assistanse.

Jeg vil også få takke øvrige venner og familie for at dere var der når jeg trengte dere, og takk til Nico som ikke kan være her, men som ville vært det hadde han kunnet. Arbeidet med denne oppgaven var til tider ikke bare en gledelig opplevelse, og da var det ekstra godt å ha dere rundt meg.

Hele denne oppgaven har jeg skrevet mens jeg har bodd på Blindern Studenterhjem, som er et fantastisk sted å bo for oss studenter. Jeg vil avslutningsvis derfor gjerne takke Hans Majestet den Blinde Bukk, Blindernånden og mine brødre i Blindern Bad- og Badstuforening.

1 Introduksjon

Datanettverk er i utgangspunktet en resurs lik alle andre, og er i motsetning til menneskers tilsynelatende umettelige ønske om stadig mer, underlagt visse begrensninger. Vi jobber derfor tilnærmet kontinuerlig med å finne nye måter å øke utnyttelsen av våre resurser for å få mer ut av det vi har. Denne oppgaven har til hensikt å bidra på dette området ved å se nærmere på hvorvidt en spesifikk nettverksteknologi kan føre til økt utnyttelse av tilgjengelige datanettverksresurser.

Teknologien vi skal ta for oss er flerkonfigurasjonsruting, eller på engelsk Multiple Routing Configurations (MRC), som i utgangspunktet ble utviklet til å håndtere feil i nettverk. Teknologien har vist resultater som indikerer at protokollen kan bli et solid alternativ for feilbeskyttelse i nettverk. Metoden som implementeres av MRC har foreløpig ikke blitt anvendt til reservering av resurser i et *tilgangsstyrt nettverk*, men vi mener denne teknologien i en slik sammenheng vil kunne bidra til å forenkle rutekonfigurering, redusere vedlikeholdsarbeidet og bidra til å øke reservasjonskapasiteten i et nettverk. Ved å tilpasse eksisterende ruteprotokoller og resursallokeringsmetoder slik at de kan anvende MRC, vil nye resursanmodninger lettere kunne penses over til mindre belastede linjer, og anvende noder med mer ledig kapasitet enn hvis ruting foregår utelukkende etter en enkel rutetabell.

I denne oppgaven anvender vi derfor teknologien MRC til å unngå mettede komponenter under reservering av nettverksresurser. Begrepet *mettet* blir i denne sammenhengen *ikke* direkte tilknyttet antall tapte pakker ved køen, eller andre måltall direkte tilknyttet de faktiske forhold i et nettverk. Vi lar begrepet i stedet beskrive tilstanden i en node eller link, når tilgjengelige resurser kryper under enn gitt terskelverdi. I utgangspunktet sier vi at en node eller link er mettet når 100% av de reserverbare resursene er reserverert, eller når en ankommet reserveringsanmodning vil føre til at mer enn 100% av disse resursene blir reservert *etter* fullført reservering.

Vi har valgt å begrense oss til allokeringsprosessen i denne oppgaven fordi vi allerede vet at MRC kan anvendes til ruting av pakker etter bare små tilpasninger av eksisterende systemer [27]. Det vi mener nå er av interesse er å finne svar på *hvor godt MRC egner seg til allokering av nettverksresurser i et tilgangskontrollert autonomt datanettverk*. Dette er problemstillingen vi vil behandle i denne oppgaven.

Gjennomgående i denne oppgaven refererer vi derfor til begrepet *resursallokeringskapasitet*. Begrepet beskriver evnen til å *allokere* resurser, som med andre ord betyr at dette ytelsesmålet ser bort fra relaterte operasjoner, som vedlikehold av polisedatabasen, kommunikasjon med relevante noder i forbindelse med en reservering, og sletting av utdaterte reservasjoner. Resursallokeringskapasiteten til en topologi eller nettverk oppgis i prosent, og er i prinsippet direkte overførbart til metningsprosenten; vi foretrekker derimot det førstnevnte begrepet for det bedre beskriver hva vi faktisk måler. Vi finner også begrepet metningsprosent noe misvisende, fordi ytelsen vi måler kun inkluderer de reserverbare resursene i nettverket, metningsprosenten derimot, skiller som regel ikke mellom reserverbare og ikke-reserverbare resurser.

Av årsaker vi presenterer nærmere i kapittel 2.1, er MRC godt egnet for sentral administrasjon, initialisering og vedlikehold. Derfor legger vi opp denne oppgaven slik at MRC håndteres av en sentral enhet, som også kontrollerer nettverksresursene og -reservasjonene. Til å utføre en slik jobb, mener vi det er naturlig å benytte en sentral båndbreddemegler (BB). BB er en administrativ komponent i et tilgangskontrollert nettverk som tar seg av rutinemessige oppgaver tilknyttet, men ikke begrenset til, administrering av nettverksresursene.

Datastrømmer som rutes av BB følger signaliserte eller på annen måte forhåndsbestemte stier i nettverket. Dette er essensielt fordi det reelle reserverte resursnivået for hver enkelt node i nettverket må være kjent. Dette betyr også at flyktige måltall som dynamisk skifter tilstand i henhold til et eller flere forhold i nettverket, slik som transmisjonsfeil, forsinkelse og nettbelastning, ikke med letthet kan benyttes. I stedet anvendes ett eller en kombinasjon av flere

statiske måltall, slik som båndbredde, linklengde[1] og antall hopp. Sistnevnte er et naturlig og populært valg når et måltall skal velges i forbindelse med ruting i nettverk.

Ruteprotokoller i dag ruter pakker etter den korteste[2] stien i nettverket, hvilket er både effektivt og fornuftig. Pakker som rutes opptar på denne måten minimalt med resurser i nettverket, hvilket også gjerne betyr at et maksimalt antall pakker kan håndteres. Men gitt at den korteste stien passerer gjennom en mettet komponent vil dette som i utgangspunktet fremstår som en styrke, blokkere trafikken og føre til at reservasjonsanmodningen blir avslått.

I denne oppgaven skal vi finne ut om MRC er egnet til å påvirke reservasjonskapasiteten i et nettverk, og sammenligne en implementasjon av MRC med ruting etter en enkel rutetabell og korteste sti. Oppgaven ser nærmere på hvordan kapasiteten påvirkes av henholdsvis få reservasjoner som ber om mye båndbredde (elefanter) og mange reservasjoner som ber om lite båndbredde (mus). I tillegg ser vi hvordan reservasjonskapasiteten påvirkes av forholdsantallet mellom noder og linker (koblingsgrad) og antall konfigurasjoner brukt av MRC (isoleringstetthet).

Til dette arbeidet benyttes en abstrakt resursallokeringssimulator kalt SAK, som er spesielt utviklet til dette formålet. Vi tenker oss at BB implementerer en tilsvarende mekanisme som den vi har brukt i SAK, slik at MRC kan benyttes til både allokering av resurser og ruting av reserverte pakkestrømmer i et tilgangsstyrt nettverk.

SAK står for "Simulering Av Kapasitet", hvilket vi mener er et navn som enkelt betegner hensikten med programmet. Simulatoren sammenligner reservasjonskapasiteten til MRC med ruting etter en enkel rutetabell (SRT, eng.: Single Routing Table). Vi ser nærmere på begge disse teknologiene i kapittel 2.1. Målet med simuleringen er å finne svar på hvilken teknologi som gir best utnyttelse av tilgjengelige nettverksresurser, og hvilke faktorer som påvirker

[1] Gjerne gitt av forsinkelsen til linken når denne står ubrukt.

[2] Pakker rutes også etter den billigste stien, hvis noe annet enn hopp benyttes til å identifisere den minst resurskrevende stien i nettverket.

ytelsen til MRC.

MRC er underlagt kun fire krav[3], hvilket i utgangspunktet innebærer at teknologien ikke er en utpreget kompleks mekanisme. Likevel er vi klar over at MRC introduserer en rekke nye begreper, og behandler nettverkstopologien på en utradisjonell måte, hvilket for mange kan gjøre det unødvendig forvirrende å forstå hvordan teknologien fungerer. Men i motsetning til andre teknologier, som fremstår som komplekse i både implementering og bruk, er MRC tilnærmet selvkonfigurerende, selvom teknologien også lett kan tilpasses spesielle behov. Sluttproduktet gir nettverksadministratoren en abstrakt og nyttig arbeidsmodell som gjør det lettere å forstå konsekvensen av enkeltstående feil i nettverket. Du kan lese mer om MRC og slik den originalt ble utformet i kapittel 2 fra side 13.

Selvom MRC ikke nødvendigvis oppfattes som særlig kompleks, er det visse aspekter ved MRC som likevel lettere lar seg forklare ved hjelp av illustrasjoner. På generelt grunnlag tilsier også våre erfaringer at en form for visualisering for mange gir et bedre overblikk, og fører til en grundigere konseptuell forståelse. Under utviklingen av SAK satset vi derfor på sanntidsvisualisering[4] både av hvordan konfigurasjonene genereres, og hvordan disse påvirker traverseringen av nettverket under den etterfulgte allokeringsprosessen. Visualiseringen bistod også utviklingen, og ble ved flere anledninger brukt til å korrigere og optimalisere kode.

Under kjøring vil SAK i første omgang lagre punktgrafikk[5] av både topologien som er lastet og konfigurasjonene som er klare for reservering[6]. Det er den samme grafen på skjermen som lagres til fil, og fordi denne benytter farger gir den en svært god og faktisk unik oversikt over den endelige strukturen i hver

[3]Du kan lese mer om MRC og slik den originalt ble utformet i kapittel 2 fra side 13.

[4]Animasjonen kan forøvrig forsinkes, slik at man tydeligere kan følge med på hvert enkelt steg foretatt av MRC. Utgaven av SAK som leveres med denne oppgaven er kun tidsforsinket, men det er allerede lagt opp til at brukeren lett kan endre koden slik at animasjonen venter på tastetrykk før den fortsetter.

[5]Punktgrafikk er i motsetning til vektorgrafikk basert på et rutenett av bildepunkter, og blir derfor kornete etter tilstrekkelig med forstørring.

[6]Filene navngis utifra topologiens navn og antall endelige konfigurasjoner. Med andre ord vil samme topologi isolert to ganger, overskrive bildefilene hvis begge kjøringene ender med det samme antall konfigurasjoner.

enkelt konfigurasjon. Tidligere arbeid har ikke vektlagt visualisering av MRC på noen måte, SAK bidrar derfor med unike muligheter for visuell analyse. Dette mener vi burde kunne bidra til å gjøre det lettere for andre forskere, nettverksadministratorer og generelt nysgjerrige å ta en nærmere titt på MRC. Hvilket igjen kan bidra til økt interesse og kanskje også mer forskning på MRC. Les mer om hvordan SAK og visualiseringsprosessen fungerer i kapittel 4.2.2.

Vi har tro på at i nevnte scenario vil bruk av flerkonfigurasjonsruting bidra til å øke antall oppnåelige forbindelser i nettverket, som også betyr økt reservasjonskapasitet. MRC gjør dette mulig ved å tilføye effektiv omkjøring av nettverkstrafikk slik at nye datastrømmer kan unngå allerede mettede noder og linker.

Du kan lese mer om SAK på side 33 og MRC på side 13. SAK er forøvrig tilgjengelig i form av en CD-ROM bakerst i dette heftet. Hvis du derimot leser denne oppgaven digitalt, kan både kildekode og nødvendige programmeringsbibliotek lastes ned fra `http://heim.ifi.uio.no/johanba/`.

1.1 Tilgangskontroll

Tilgangskontroll er noe vi alle har et forhold til, tross alt er formålet med en høyst ordinær lås å kontrollere tilgang til en resurs eller verdi. Innen informasjonsteknologi er tilgangskontroll tilknyttet datasikkerhet, som kan inkludere både maskin- og mykvare. Mykvare blir i denne oppgaven brukt som et samlebegrep for både programvare og tilknyttet data, ettersom autentiserings- og autoriseringssystemer avhenger av begge deler.

Bruk av store nettverk (WAN, eng.: Wide Area Network) og Internett, gjør tilgangskontroll betydelig viktigere og vanskeligere. Informasjonssystemer kan bli aksessert fra en vilkårlig ubemannet dataterminal, eller fra tilnærmet hvor som helst via Internett. For å løse dette sikkerhetsproblemet innføres ett eller flere sikkerhetstiltak, som gjerne tilhører en av følgende tre kategorier:

- Kunnskap; “noe vi vet”, slik som koder og passord

- Objekter; "noe vi har", som for eksempel nøkkelkort og en kodebrikke.
- Biometri; "noe vi er", slik som fingeravtrykk (mest utbredt), DNA, ansiktsform, netthinne- og irisavlesning.

Til autentisering er samtlige av disse tiltakene avhengig av referansedata; oppgitte data som på et eller annet en-entydig vis passer sammen med en oppgitt eller medbrakt motpart. Hvorvidt de passer sammen avgjør om personen har tilgang eller ikke. Det samme gjelder kommunikasjon mellom maskiner, selvom maskiner gjerne er begrenset til kunnskaprelaterte løsninger som benytter mykvare ved autentisering. I denne oppgaven tildeler vi BB ansvaret for å autentisere og autorisere reservering og bruk av nettverksresurser.

I et åpent brev, skrevet av Steve Jobs og publisert på nettsidene til Apple Inc. i februar 2007, kan vi lese om problemene forbundet med oppdatering av systemet som forhindrer fri kopiering av kjøpt musikk [21]. Problemene er i hovedsak tilknyttet hemmeligholdelse av autentiseringsprosessen (FairPlay) og oppdateringen av nevnte prosess: hver gang sikkerheten i FairPlay brytes blir Apple Inc. tvunget til å oppdatere både sin nettbutikk iTunes, og samtlige av sine spillere[7] via oppdateringer lastet ned fra Internett. En slik oppdatering er både resurs- og tidkrevende, hvilket også gjerne gjør den kostbar. Fordelene forbundet med en sentralisert autentiseringsenhet, slik som en båndbreddemegler, blir åpenbare; faren for synkroniseringsproblemer og oppdateringsanomalier reduseres betraktelig, og i tillegg kan den distribuerte mekanismen tilknyttet autentiseringsprosessen forenkles, hvilket gjerne fører til lavere implementeringsutgifter, og reduserte vedlikeholds- og oppgraderingskostnader [8].

På en annen side, fører sentralisering også gjerne til flaskehalser og økt følsomhet for feil tilknyttet den administrative enheten i nettverket. Når det oppstår feil i en slik implementasjon, kan selv en kortvarig feil få kostbare driftsmessige konsekvenser. En sentral autentiseringsprosess fører også med seg et behov

[7] Dette inkluderer fastvaren også i solgte spillere.

[8] Digresjon; en oppdatering i forbindelse med Fairplay: den 2. april offentliggjorde EMI Music at de vil legge ut hele sin musikkutvalg for salg på iTunes uten digital kopibeskyttelse [32].

for nye sikkerhetstiltak; sensitiv informasjon som sendes over store avstander må beskyttes mot både innsyn og manipulering. I den forbindelse er kryptering en utbredt løsning, og har særlig fått betydning for elektronisk handel, der det brukes for å sikre sensitive transaksjonsdetaljer. I tillegg vil opphopning av autentiseringsdata isolert sett føre til økt verdi og fare for misbruk, slik at ytterligere datasikringstiltak må implementeres sentralt.

For å løse disse og andre svakheter forbundet med det initielle konseptet om en båndbreddemegler [36], drives det kontinuerlig forskning på alternative implementasjoner av både en enkel og flere båndbreddemeglere i ett og flere AS [46, 5].

1.2 Forutsetninger, abstraksjoner og avgrensninger

Arbeidet med oppgaven startet med en grundig gjennomgang av eksisterende nettverkteknologi. Målet var å etablere et solid utgangspunkt for bearbeiding, slik at eventuelle forutsetninger, abstraksjoner og avgrensninger ikke ville redusere verdien av det som presenteres. Denne tilnærmelsen har den fordelen at vi i utgangspunktet er åpne for at en vilkårlig del av dette råmaterialet, i prinsippet kan bli en komponent i det endelige systemet.

Fordi vi ikke ønsker å binde våre simuleringsresultater til en bestemt implementasjon av en eller flere båndbreddemeglere, har vi valgt å holde funksjoner ikke direkte relevant for *resursallokeringsprosessen* utenfor denne oppgaven. Vi forutsetter derfor at båndbreddemegleren selv er i stand til å ivareta sikkerheten tilknyttet kontrollmeldingene som benyttes til å administrere nettverksresursene, og at dette gjøres med resurser reservert utelukkende til dette formålet. Videre forutsetter vi at feilhåndteringsmekanismene i BB sørger for at kontrollaget opererer tilsynelatende uten feil, slik at reservasjonshåndteringen til en hver tid kjører som normalt. For å unngå å introdusere nye forhold som må tas med i denne betraktningen, vil en enkel BB ta hånd om reserveringsprosessen og vedlikeholde polisedatabasen (PDB). Polisedatabasen inneholder informasjon om hver enkelt tjenesteavtale og samtlige fremtidige og aktive reservasjoner, og

er med andre ord en helt fundamental komponent for BB. Ettersom denne oppgaven studerer nærmer en alternativ metode for hvordan stier tilknyttet resursreservasjoner identifiseres, er det ikke nødvendig å ta stilling til hvordan PDB lagres, brukes og vedlikeholdes;vi trenger heller ikke ta stilling til hvorvidt den er distribuert eller ligger sentralt lagret i topologien, fordi dette ikke vil påvirke våre resultater tilknyttet resursallokeringsprosessen i et gitt topologi.

Med disse forutsetningene kan vi se bort fra kontrollaget, og i stedet fokusere utelukkende på *i hvilken grad* resursreserveringskapasiteten i nettverket blir påvirket av MRC, sammenlignet med SRT.

1.3 Et historisk overblikk

4. oktober 1957 ble historiens første menneskeskapte satellitt skutt opp i bane rundt jorden. Sputnik var Sovjetisk og hendelsen sjokkerte amerikanerne nok til at Presidenten utnevnte en vitenskaplig rådgiver som i løpet av et år opprettet både den sivile romforskningsorganisasjonen NASA, og det militære forsvarsforskningsbyrået ARPA (eng.: Advanced Research Projects Agency, heter nå DARPA[9]). Det var ARPAs tredje direktør Jack Ruina som innså at datateknologi ville kunne bidra til å redusere problemene forbundet med militær kommunikasjon, kontroll og kommando [6] og som derfor etablerte forskningsorganet IPTO[10]. Psykologen Joseph Licklider ved Massachusets Instiute, som i mars året før publiserte artikkelen Man-Computer Symbiosis[11], ble av Ruina satt til å lede IPTO.

I mai 1961 beskrev MIT studenten Kleinrock i sin avhandlingsplan [24] til sin doktorgrad for første gang konseptet om pakkesvitsjing. Kleinrock fullførte doktoravhandlingen i løpet av det påfølgende året, og parallelt publiserte Licklider en rekke artikler der prinsippene for et globalt nettverk diskuteres ytterligere.

[9] Defense Advanced Research Agancy, het ARPA i 1958-72 og 1993-96.

[10] IPTO; Information Processing Techniques Office, forskningsorgan underlagt ARPA, etablert 1961. ARPA er et administrativt organ som tildeler fremtredende forskere stilling som prosjektleder. Disse prosjektlederne blir gitt stor frihet til å forfølge og støtte forskning de mener kan være til nytte for militæret.

[11] Artikkelen presenterte visjonen om et symbiotisk samarbeid mellom mennesker og beregningsmaskiner (eng.: computing machines), som ville kunne effektivisere intellektuelle prosesser og bidra til ny innsikt og nye slutninger innen teknisk og vitenskaplig tenkning.

Licklider mente en forutsetning for å oppnå et slikt effektivt samarbeid mellom datamaskiner ville være en videreutvikling innen beregningstidsdeling (eng.: computer time sharing), av minnekomponenter, minneorganisering, og programmeringsspråk, samt innen utstyr for inn- og utputt (eng.: input/output). Og det var nettopp tidsdeling av dataresurser over nettverk som var utgangspunktet for ARPAnet, som ble etablert syv år senere. Arbeidet til Kleinrock og Licklider var således fundamentalt i utviklingen av nettverkskommuniserende maskiner og det som etterhvert skulle bli et globalt nettverk bedre kjent som Internett.

Helt fra begynnelsen[12] har internettet vært i stadig vekst og utvikling. Mer enn noe annet medium, har Internett åpnet for nye muligheter innen datateknologi og kommunikasjon. Helt siden de første eksperimentene med pakkebasert stemmekommunikasjon ble utført på 70-tallet har stadig flere tjenester valgt å benytte seg av dette generelle og (etterhvert) globale nettverket fremfor proprietære løsninger. For mange var det særlig introduksjonen av hyperteksting og nettleseren Mosaic [1] i 1993 som gjorde internettet synlig på en tilstrekkelig brukervennlig og tilgjengelig måte, slik at langt flere så verdien av dette globale internettverket. Og det tok bare et par år før den vanlige bruker også hadde tilgang til lyd og bilde [42] via Internett. Året etter, i 1996, åpnet USA for fri konkurranse av levering av internett, hvilket førte med seg en betraktelig økning i antall husstander med fiberoptisk datakommunikasjon. Etterhvert som enda flere får bredbånd, blir det også mer interessant for større bedrifter å satse tungt på IP-tjenester, for eksempel beskriver The New AT&T dette som den kommende IP-revolusjonen [49].

I dag kan vi nyte godt av både enveis og toveis lyd og video, og kvaliteten på videoen som sendes har økt sakte men sikkert hele veien. Tilbudet av video via Internett har siden 2005 opplevd en dramatisk vekst, i 2006 alene dukket det opp en rekke nye publiseringsløsninger for nettvideo (Brightcove, Clipstream, m.fl), sammen med videotilbydere (click.tv, veoh, YouTube, vimeo, m.fl.), i tillegg

[12]Internettet kan sies å være født enten 2. sept. 1969 da den første datamaskinen kommuniserte med en ruter, men det var ikke før 29. okt. 1969 at de to første datamaskinene kommuniserte med hverandre via en ruter i et nettverk.

finnes det nå også muligheter for videosøk hos blant annet Google og Yahoo. I dag kan det kjøpes episoder av TV-serier dagen etter at de er sendt på TV [7], også CD-er og filmer [10] kan kjøpes og sendes over nettet. I 2006 var det også klare signaler fra markedsanalytikere, nettbrukere og innholdsleverandører som indikerte at internettvideo og -TV var i ferd med å bli allemannseie [13]. Mange av dagens mest populære tv-serier kan i USA finnes *gratis* på Internett, publisert av tv-selskapene selv, og vår egen riksdekkende tv-kanal NRK legger ut tilnærmet alt sitt innhold både innen radio og tv som lyd- og videostrømmer på Internett. Markedsundersøkelser utført av flere store analysebyråer[13] i 2006 er enige i at markedet er modent for IP-TV. Dog skal det nevnes at tanken om lyd og bilde over nett har eksistert helt fra første stund, og har vært med på å formgi protokollene som har ledet oss til der vi står i dag.

Til tross for denne utviklingen opererer dagens pakkesvitsjete Internett fortsatt i all hovedsak etter prinsippet om ytelse etter beste evne, som betyr at all datatrafikk konkurrerer om de samme resursene. Behovet for båndbredde kan mettes ved overdimensjonering (eng.: overbuild) slik at det alltid er nok båndbredde til alle brukere, også i rushtider (eng.: peak hours), men dette er både dyrt og tidkrevende. Et alternativ til fysisk overdimensjonering er å introdusere diskriminering og forskjellsbehandling av datapakker og -strømmer ved hjelp av programvare. I dag finnes det er rekke slike mekanismer med mål om å sikre nødvendig tjenestekvalitet for prioritert data, gjerne på bekostning av øvrig trafikk i nettverket. Sistnevnte moment er gjerne også årsaken til den regjerende skepsis forbundet med denne type teknologi, da forskjellsbehandling generelt betraktes som noe negativt og gjerne strider mot det grunnleggende prinsippet for Internett; der vi alle bidrar til fellesskapets beste.

IntServ er et eksempel på en teknologi som forskjellsbehandler brukerne, og fungerer ved å prioritere de mer eller mindre sårbare trafikkstrømmene i henhold til en forhåndsutstedt tjenestekontrakt eller -garanti som sikrer dataflyten mot et eller flere av problemene som kan oppstå i et nettverk. Hvis ikke nok resurser

[13]IDC, GfK NOP og Frost & Sullivan

kan tilbys til å hedre kontrakten kan datastrømmen og den tilhørende tjenesten eventuelt bli blokkert, fremfor at brukeren av tjenesten opplever uakseptable resultater. En tjeneste av denne typen, som avhenger av en viss båndbredde kalles *uelastisk*, i motsetning til *elastiske* tjenester som fungerer uavhengig av hvor mye båndbredde som faktisk tildeles. Men selv elastiske tjenester vil under visse forhold være tjent med en tjenestenivåavtale (SLA, eng.: Service Level Agreement). Frogner Kino [8] kan være et eksempel på dette. Åpningen av kinosalen i 2006 åpnet for digital filmdistribusjon i Oslo. Levering av nye filmer via Internett er en typisk kostnadsbesparende og elastisk dataleveranse, som under visse forhold kan være avhengig av en tjenesteavtale som sikrer en viss overføringshastighet. Og det er all grunn til å tro at antall brukere av tilsvarende elastiske tjenester vil fortsette å øke i tiden fremover, og dermed også antallet aktive tjenesteavtaler.

Til å administrere dette mangfoldet av tjenesteavtaler kan en båndbreddemegler benyttes. Målet med en BB er i henhold til RFC2638 [36] å holde orden på "eksisterende allokering av markedstrafikk, og å vurdere merking av ny trafikk i lys av retningslinjer og eksisterende allokeringer". Bruk av en båndbreddemegler muliggjør automatisering og sentralisering av prosessene forbundet med sikkerhet, differensiering og administrering av trafikk i henhold til krav om tjenestekvalitet. En båndbreddemegler er med andre ord en dedikert nettverksagent eller komponent som tar seg av trafikkontroll, prioriteringer, resursforhandling og -fordeling[14]. Disse oppgavene forutsetter oppdatert kjennskap til gjeldende prioriteringer, ytelseskrav og resursforbruk, og innebærer en svært rutinemessig behandling av store datamengder, hvilket gjør oppgaven spesielt godt egnet for automatisering. Sentralisering betyr også gjerne lettere integrering av nødvendige systemer til fakturering og kundebehandling.

Fordelene og behovet for en slik administrativ enhet er også grunnen til at det fortsatt forskers relativt aktivt på båndbreddemeglere [5, 30, 25, 22, 9]. Det har

[14]Studier [23] indikerer at en BB gjør det lettere å tilby tjenestekvalitet til sanntidsapplikasjoner, i tillegg økes nettverksutnyttelsen og dermed også fortjenesten til internettjenestetilbydere (eng.: ISP, Internet Service Providers).

derimot vært liten aktivitet forbundet med forskning og utvikling av IntServ, noe av grunnen til dette kan være det at behovet for en slik nettverksmodell uteblir. Dagens kommersielle nettverk har tilsynelatende tilstrekkelig nettverkskapasitet til å betjene sine (større) kunder, og ser ingen reell fortjeneste tilknyttet slik forskning slik situasjonen er i dag. Slik kan det også fortsette å være frem til en ny banebrytende teknologi ser dagnes lys og permanent endrer hvordan nettverket brukes, og hvilke krav som stilles til både båndbredde og tjenestegarantier.

Våre resultater er direkte knyttet til sentral administrering av nettverksresurser, og vil avgjøre hvorvidt MRC er en mekanisme som eksempelvis en båndbreddemegler (BB, eng.: Bandwidth Broker) vil være tjent med å implementere.

2 Bakgrunn og relatert arbeid

Internett er i stadig vekst og forandring; eksisterende tjenester forbedres, nye digitale tjenester utvikles, og stadig flere brukere anvender internettet som sin primære kommunikasjonskanal. Denne avhengigheten øker behovet for et stabilt og tilgjengelig nettverk med tilstrekkelig kapasitet og ytelse. En kombinasjonen av trådløs og landfast nettverksteknologi benyttes gjerne for å dekke dette behovet, og forventingene fra brukerne fører til hyppigere oppdatering og utskiftning av både maskin- og programvare. I tillegg er eksisterende løsninger ofte komplekse både i implementasjon og anvendelse, hvilket uansett *aldri* er formålstjenlig. Disse faktorene er opplagt medvirkende til at de fleste nettverksfeil oppstår som et resultat av feilkonfigurering [33].

Når det oppstår feil i et nettverk, utløses en IP-konvergering. Denne prosessen er ofte tidkrevende, og fører med seg en periode med ustabilitet i nettverket. Hvis denne prosessen kunne utsettes, forutsatt at den oppdagete feilen kan håndteres på et annet vis, ville nettrafikken opplagt være tjent med dette.

Mye arbeid har blitt nedlagt i forskjellige løsninger som tar sikte på å redusere tiden det tar for nettverket å stabilisere seg etter en feilkorrigering, likevel har sanntidsapplikasjoner problemer med den resulterende forsinkelsen.

2.1 Flerkonfigurasjonsruting (MRC)

Flerkonfigurasjonsruting eller MRC, er i utgangspunktet en gjenopprettelsesmetode (eng.: recovery) som introduserer global og lokal feiloppretting i IP-baserte nettverk ved å benytte et sett pregenererte redundante rutetabeller til å omgå feil som har oppstått i nettverket [26]. At en mekanisme er gjenopprettende betyr at både feilbeskyttelse (eng.: error protection) og ombygging (eng.: restoration) benyttes til feilhåndtering. Dette innebærer både at alternative nettverksstier forberedes før en feil inntreffer, slik at feilen kan korrigeres svært raskt, og at permanente feil fører til en global rekonfigurering og oppdatering av rutetabeller. Ved initialisering av MRC vil ett sett med et nødvendig antall rutetabeller,

kalt konfigurasjoner, bli generert og deretter distribuert til samtlige noder i nettverket. Samtlige konfigurasjoner i et slikt sett er unike, og lagd på en slik måte at datatrafikken tvinges utenom visse komponenter i hver konfigurasjon. Når en feil oppdages vil *feilens plassering i nettverket diktere hvilken av disse konfigurasjonene som benyttes* videre til ruting av pakker, slik at komponenten som har feilet blir unngått.

De alternative rutekonfigurasjonene etableres ved å manipulere kostnaden forbundet med ruting over utvalgte linker, slik at bruken av disse begrenses og noder i praksis fremstår som endenoder - og derfor ikke blir brukt til datatransitt. En node eller link kalles isolert når den ikke benyttes til transitt eller videresending av data. Alle noder og linker isoleres i eksakt en konfigurasjon, og så sant nettverket er flerkoblet (eng.: biconnected) er det strukturen i topologien som avgjør hvor mange konfigurasjoner som kreves for å feilbeskytte nettverket. Kombinasjonen av en isolert node og en begrenset link kan betraktes som en tradisjonell blindvei, trafikken kan både nå og stamme fra den isolerte noden, men ikke passere noden som et hopp i en lengre sti på vei mot en annen node. At et nettverk er flerkoblet innebærer at en vilkårlig node kan fjernes fra topologien uten at nettverket segmenteres, dette er et krav for at MRC skal kunne fungere. Når en feil oppstår i en komponent er det dermed *kun* nødvendig å identifisere hvilken konfigurasjon som isolerer feilkomponenten, for så å velge nettopp den konfigurasjonen til å rute etterfølgende trafikk. På denne måten vil trafikken garantert aldri traversere komponenten som har feilet.

Et problem gjerne forbundet med IP-baserte nettverk, er at det opererer med kun en feilklasse. Rutere har ingen mulighet til å skille mellom midlertidige og permanente feil, og kan dermed opplagt heller ikke håndtere dem forskjellig. Konsekvensen av dette er at den samme tidkrevende ruteoppdateringsmekanismen benyttes uavhengig av feilens natur og varighet. Global feilhåndtering er tidkrevende, og fører underveis gjerne til både forsinkelser og pakketap. Studier viser at hendelser som utløser en slik feilhåndtering forekommer hyppig, at de

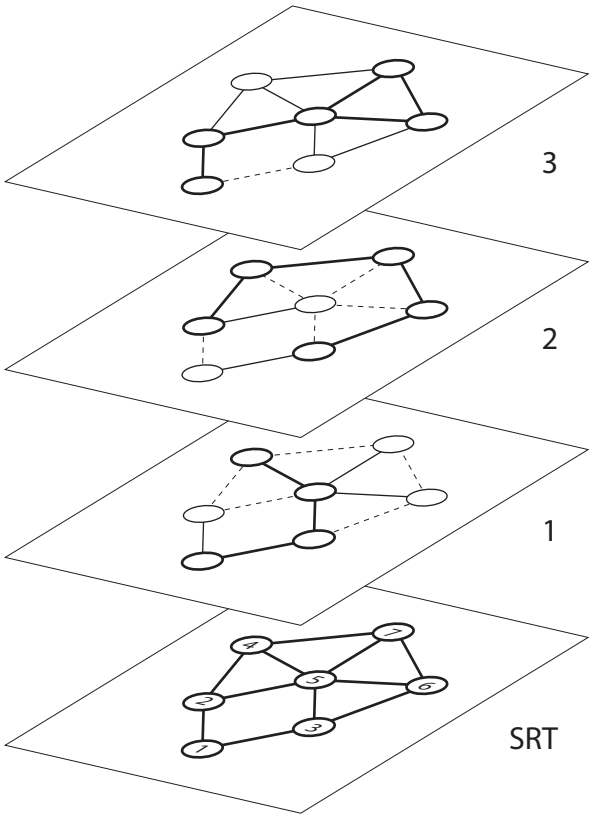
er kortvarige[15] og er gjerne utløst av eksterne ruteprotokoller.

Ved innføring av feilbeskyttelsesmetoden MRC vil midlertidige feil kunne håndteres lokalt. De forhåndsgenererte konfigurasjonene benyttes kun i det en node oppdager at sending til en nedstrømsnode eller -link [27] feiler. Lokal feiloppretting blir på ingen måte annonsert til øvrige noder i nettverket hvilket muliggjør en tilnærmet øyeblikkelig feilkorrigering og kontinuerlig ruting av pakker. Global feiloppretting utløses derimot sentralt og innebærer utsending av oppdaterte konfigurasjoner til samtlige noder i nettverket, som opplagt er vesentlig tregere enn lokal feiloppretting. Hver gang en node eller link permanent blir tilføyd eller fjernet fra nettverket blir en global feiloppretting utløst.

MRC har en rekke fordelaktige egenskaper, slik som lokal og derfor tilnærmet øyeblikkelig feiloppretting, teknologien bidrar til å utsette nettverkskonvergeringen hvilket er en fordel fordi de fleste feil er kortvarige, MRC krever ingen kjennskap til feilkilden, og kan implementeres uten særlige endringer i rutemekanismen til eksisterende teknologier [27].

Flerkonfigurasjonsruting er en videreføring av et konsept som stammer fra en teknologi kalt Robuste Rutelag (RRL, eng.: Resilient Routing Layers) [26, 17, 18, 27]. RRL er selv avledet fra et tidligere arbeid rettet mot ruting og det å hindre deadlocks i irregulære klasenettverk (eng.: cluster network) kalt Up*/Down*. Begge benytter virtuelle rutelag (les: sjikt), men kun førstnevnte kan benyttes til ruting i generelle pakkenettverk. RRL som ikke trenger å ta hensyn til deadlocks[16], fokuserer i hovedsak på å sikre feiltollerant ruting. RRL gjør dette ved å *fjerne komponenter* i topologien, som deretter grupperes på en bestemt måte i virtuelle lag. Når en feil oppstår kan RRL-laget som ikke inneholder komponenten som har feilet, brukes til ruting av pakker; slik også MRC fungerer. Men MRC skiller seg fra RRL ved at komponenter ikke fjernes fra topologien, men at pakkene og valget av en sti gjennom nettverket styres ved å *manipulere kostnaden* forbundet med bruk av de forskjellige nettverkskomponentene.

[15] Vanligvis er feilkilden korrigert i løpet av ett minutt [33].
[16] Deadlocks er ikke et problem i pakkesvistjete nettverk.



Figur 1: Lagdelt visning av hver enkelt konfigurasjon generert i forbindelse med isolering av eksempeltopologien (E3). Denne løsningen består av tre konfigurasjoner. Hver linje som er stiplet indikerer en isolert link, en link tegnet med tynn strek er begrenset, en node med tynn strek er isolert. Legg merke til at de øvrige komponenten danner et sti gjennom topologien som samtlige isolerte noder er tilkoblet. Denne stien er hovedåren (eng.: backbone) i konfigurasjonenen, og det kreves av MRC at denne er uavbrutt i hver konfigurasjon.

I denne oppgaven vil MRC som nevnt *ikke* bli benyttet til feilkorrigering, men til å unngå komponenter som ikke har tilstrekkelig med kapasitet til å akseptere gjeldende resursanmodning. BB benytter konfigurasjonene som generes av MRC til å identifisere alternative stier (eng.: trails) gjennom nettverket, som deretter undersøkes for tilstrekkelige ressurser. En sti består av sammenkoblede noder der hver enkel node kun forekommer en enkel gang. Hvis en konfigurasjon ikke har kapasitet langs den korteste stien mellom inngangs- og utgangsnoden, blir neste konfigurasjon undersøkt. Søket starter i den eller de konfigurasjonene som

tilbyr den korteste stien før øvrige konfigurasjoner med lengre stier undersøkes etter tur. Hvis ingen konfigurasjon lykkes i å tilby en sti med tilstrekkelige resurser, blir resursanmodningen avslått.

Sett i sammenheng med resursreservering og tilgangskontroll, er det blant annet denne egenskapen som gjør MRC spennende. Tenk deg en situasjon der korteste sti inneholder en eller flere fullreserverte komponenter (les: noder og linker), hvilket uten flerkonfigurasjonsruting fører til at resursanmodningen blir avslått. MRC derimot, prøver flere forskjellige stier og gir ikke opp før *samtlige konfigurasjoner* har avslått reserveringen, hvilket intuitivt burde føre til et økt antall godkjente reserveringer. Særlig i de tilfeller der flere konfigurasjoner tilbyr en løsning med like mange hopp, men som likevel ikke følger nøyaktig samme sti i nettverket. Slike stier kaller vi *likeverdige*, og forekommer oftere i et løst enn i et tett isolert nettverk. En løs isolering innebærer at den ferdigisolerte topologien benytter flere konfigurasjoner enn det som strengt tatt er nødvendig[17]. Dette er mulig ettersom MRC unnlater å spesifisere en overordnet rekkefølge nettverket skal traverseres under isoleringsprosessen. Dette påvirker ikke bare hvilke noder og linker som isoleres i hver enkel konfigurasjon, men også antall konfigurasjoner som benyttes av det ferdigisolerte nettverket. Likeverdige stier oppstår oftere hvis det er flere konfigurasjoner fordi det også betyr at færre komponenter er isolert i hver enkel konfigurasjon, og dermed også at flere komponenter er felles for hver enkel sti.

Figur 2 (s. 18) viser en topologi isolert ved hjelp av fire konfigurasjoner (E4), hvilket fremstår som en løsere isolering sammenlignet med figur 1 som bruker tre (E3). Ved nærmer undersøkelse ser vi at E3 etablerer 2 likeverdige stier[18], og at E4 tilbyr det dobbelte[19]. Ved ruting mellom node 1 og 7, gitt at node 3 og 4 er mettet, vil E3 fortsatt kunne godta en resursallokeringsanmodning (RAR, eng: Resource Allocation Request). Det samme kan derimot ikke sikkert sies om E4, ettersom MRC ikke nødvendigvis sørger for at node 5 inngår i den likeverdige

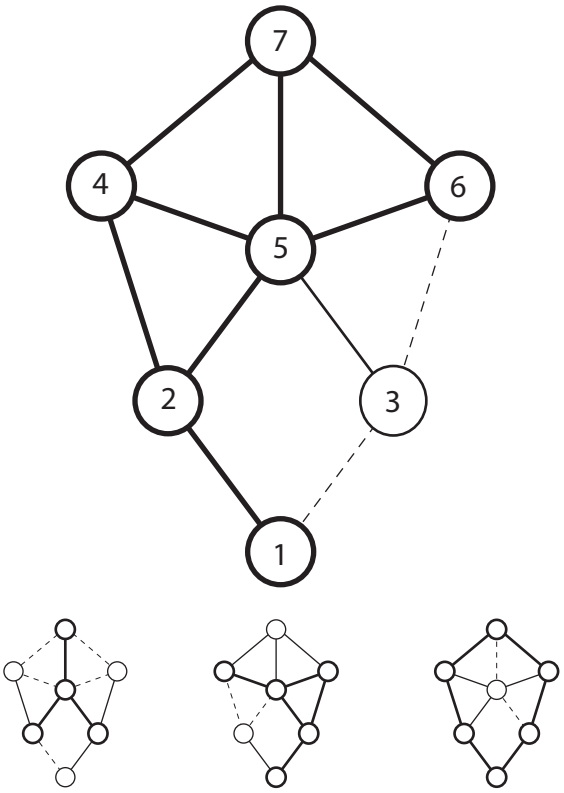
[17] Det finnes per i dag ingen metode som kan bestemme nedre grense for antall konfigurasjoner nødvendig for å isolere en nettverksgraf.

[18] Begge de likeverdige stiene i figur 1 er samlet i konfigurasjon 3: 6-4 og 3-7.

[19] Isolasjonsløsningen vist i figur 2 tilbyr 4 likeverdige stier: 1-7, 2-7, 3-7 og 4-6.

stien som anvendes mellom 1 og 7 i den øverste konfigurasjonen (fig. 2). Det foreligger i dag ingen mekanisme innen MRC som utnytter dette fenomenet til å øke reservasjonskapasiteten i nettverket. Vi drøfter en mulig implementasjon i kapittel 6.2.

Hvis en konfigurasjon aksepterer en reservering som allerede har blitt avslått av en annen konfigurasjon med en likeverdig sti, er det opplagt komponentene som ikke er felles som gjør det mulig. Hvor hyppig likeverdige stier forekommer i forhold til isoleringstettheten, er i den forbindelse et interessant spørsmål. Vi holder denne tanken friskt i minnet når vi senere drøfter mekanismen nevnt over.



Figur 2: Dette er eksempeltopologien isolert ved hjelp av 4 konfigurasjoner, som sammenlignet med figur 6 fremstår som en løsere isolering som derfor inneholder flere likeverdige stier. Legg merke til at det mellom node 1 og 7 finnes 3 likeverdige stier, hvorav en av disse inngår i *to forskjellige* konfigurasjoner.

Et spennende forhold tilknyttet dette og bruk av MRC er stienes lengdeøkning sammenlignet med SRT. Tidligere simulerte forsøk viser at metoden skallerer godt [26] i nettverk med opptil flere hundre noder, men også at omveiene i gjennomsnitt blir 15% [18] lenger enn de optimale stiene gjennom nettverket. Ettersom denne oppgaven ser nærmere på allokeringskapasiteten i nettverk, er opplagt metningsgraden av stor betydning. En generell økning av lengden på stien som reserveres i nettverket betyr jo kort og godt at det blir færre ressurser tilgjengelig for nye reservasjoner. Sånn sett fremstår MRC som en metode som *aggressivt* løser ressursproblemene for gjeldende reservering på bekostning av etterfølgende anmodninger.

Hvis vi tenker oss at trafikken i et nettverk oftere krysser kjernenodene enn kantnodene, fremstår MRC som et fornuftig alternativ: nodene som er mest utsatt for trafikk kan enkelt unngås ved å rute trafikken rundt kantene i nettverket. Men dette reiser en annen problemstilling tilknyttet isoleringsprosessen; i hvilken grad blir stiene langs kanten holdt inntakt under isolering, og hva kan gjøres for å påvirke dette. Denne problemstillingen er et egnet tema for videre arbeid og blir derfor ikke dekket ytterligere i denne oppgaven.

Ettersom MRC tvinger datatrafikk til å følge definerte omveier i nettverket, kan det bli oppfattet som galt å kalle nettopp en slik omvei for korteste sti, men sett i forhold til ruting med flere konfigurasjoner er dette begrepet fortsatt dekkende. En interressant observasjon verdt å bemerke i denne sammenheng er at det lett kan argumenteres for flere forskjellige tilnærmelser forbundet med valg av alternativ konfigurasjon når en mettet komponent blir identifisert. Med mindre den mettede komponenten er inngangs- eller utgangsnoden, er kanskje den mest innlysende fremgangsmåten å velge nettopp den konfigurasjonen der den mettede komponenten er isolert, slik at datastrømmen garantert omgår denne komponenten. Men fordi dette fører til et oppslag i en konfigurasjon som i utgangspunktet unngår komponenten fordi den er isolert, kan vi slutte at ruting etter korteste sti i samtlige konfigurasjoner allerede inkluderer nevnte alternativ og dermed også fremgangsmåte. For å skille mellom den korteste

stien i en topologi, og den ofte lengre korteste stien i en konfigurasjon, blir førstnevnte referert til som topologiens korteste sti, eller bare TKS, og sistnevnte som konfigurasjonens korteste sti (KKS).

Utifra målet om å maksimere reservasjonskapasiteten, er det ønskelig at hver enkel reservasjon alltid benytter korteste ledige sti. Denne forutsetningen krever at BB for hver innkommende RAR ordner samtlige konfigurasjoner i økende rekkefølge etter stiens lengde, og deretter undersøker samtlige konfigurasjoner etter tilstrekkelige resurser. Det er ikke mulig å sikre at korteste ledige sti brukes hvis MRC først sjekker konfigurasjonen som isolerer den første mette komponenten langs stien, fordi det selvsagt ikke nødvendigvis er denne konfigurasjonen som tilbyr den kortere stien med tilstrekkelige resurser. Det er derfor nødvendig å alltid starte med konfigurasjonen som stiller med den korteste stien, før en likeverdig eller lengre sti kontrolleres.

Det kan argumenteres for at bruk av korteste ledige sti i prinsippet vil føre til at MRC svært ofte vil følge stiene reservert ved bruk av SRT, og at poenget med MRC dermed forsvinner, eller i det minste blir redusert. Dette stemmer så lenge reservasjonene følger ubrukte stier, men etterhvert som nettverket gradvis mettes vil SRT feile stadig oftere fordi TKS er fullreservert, hvilket gjør at vi får bruk for øvrige konfigurasjoner tilbudt av MRC, i det trafikken presses til å følge alternative stier.

Fra tidspunktet der trafikken ikke alltid benytter TKS, vil nettverksutnyttelsen begynne å falle. Hvis vi forutsetter at anmodninger kan komme fra en hvilken som helst node i nettverket, mener vi det er rimelig å anta at SRT underveis i denne sluttfasen innimellom vil klare å reservere nye resurser. For oss fremstår det derfor som meningsfylt å se om dette gjør MRC totalt sett mindre effektiv enn SRT, om vi kan observere karakteristiske forhold som avgjør hvorvidt MRC eller SRT er best egnet til resursallokering, og til slutt om vi kan identifisere eventuelle egenskaper som når kombinert med metningsnivået gjør at MRC og SRT egner seg best til allokering av resurser.

Vi vil også se nærmer på hvordan MRC dynamisk kan omdirigere reservert

trafikk for å frigjøre resurser i en eller flere definerte nettverkskomponenter. Selvom dette ikke implementeres av SAK, vil funksjonaliteten drøftes og vurderes i kapittel 6.2.

2.1.1 Algoritmen til MRC

Protokoller spesifiseres gjerne kun tilstrekkelig til at kompatibilitet og rimelig ytelse kan forsikres. Dette kalles gjerne underspesifisering, og åpner for ulike implementasjoner og dermed gjerne også varierende ytelse. Noen implementasjoner håndterer visse trafikktyper bedre enn andre og det kan derfor være nødvendig å foreta et representativt og velbegrunnet utvalg blant alternative implementasjoner, slik at simuleringsresultatet blir representativt for faktisk ytelse i et *reelt* nettverk. Men det er også slik at kjente svakheter forbundet med en spesifikk implementasjon utbedres og at trafikkbildet i nettet endres over tid. Det er derfor hensiktsmessig å velge en simuleringsløsning som unngår å benytte en implementering med kjente svakheter kjørt i dagens trafikkmiks ettersom begge deler kan bli utdatert innen kort tid.

SAK implementerer MRC slik algoritmen beskrives i "Fast IP Network Recovery using Multiple Routing Configurations" [18], for et overblikk kan pseudokoden vist i figur 3 være til hjelp. For å gjøre det lettere å følge teksten i avsnittene under, henviser vi til pseudokoden med et linjenummer skrevet i parentes.

Som tidligere nevnt, er utgangspunktet for MRC en flerkoblet nettverkstopologi. Algoritmen til MRC traverserer hver enkel node inneholdt i grafen (1), og fordi algoritmen er rekursiv (45) er det nødvendig å starte med å undersøke hvorvidt gjeldende node allerede er isolert (3). Hvis noden allerede er isolert, går algoritmen direkte til neste node i topologien. Hvis noden derimot ikke er isolert, og det forsatt foreligger uprøvde konfigurasjoner, velges en uprøvd konfigurasjon fra settet med etablerte konfigurasjoner hvor gjeldende node forsøkers isolert (3).

Allerede nå ser vi hvordan MRC har muligheten til å styre rekkefølgen for både hvordan konfigurasjonene traverseres (51) og hvordan neste node velges

(1). Også traverseringen av linkene i topologien (7) forblir uspesifisert, selvom det stilles krav til isoleringen.

Så lenge noden forblir uisolert og ikke alle konfigurasjonenen er prøvd, vil MRC ikke gi opp isoleringsprosessen. Hvis derimot alle konfigurasjoner er prøvd og noden forblir uisolert, utstedes en unntakstilstand som avslutter algoritmen (56). Så lenge topologien er flerkoblet, innebærer dette at antall konfigurasjoner må økes med +1.

Hvis isolering av noden ikke segmenterer hovedåren gjennom gjeldende konfigurasjonen (5), vil denne noden bli forsøkt isolert i konfigurasjonen. Isoleringen skjer ved å traversere samtlige tilkoblede linker. Hvis isoleringen av gjeldende node er utløst av et rekursivt kall, vil linken som er oppgitt bli isolert først. Dette er for å sikre at rotnoden som utløste det rekursive kallet ved å velge gjeldende node, faktisk kan isoleres i den konfigurasjonen rotnoden foreløpig er merket som isolert. Hvis gjeldende node dermed ikke kan isoleres uten å segmentere hovedåren i samtlige konfigurasjoner (5) så feiler isoleringen ved at en unntakstilstand kastes (56).

En node kan isoleres så lenge den innehar minst en ikke-isolert link, som i denne sammenhengen betyr enten en link som er vektet normalt eller en som er begrenset (eng.: restricted). Det rekursive kallet går fra node til node helt til isoleringskjeden fullføres, deretter lokaliseres en uisolert node som isoleres ved neste gjennomkjøring. Dette kan best forklares nærmere med et konkret eksempel. Vi tar utgangspunkt i figur 1, hvilket fører til følgende isoleringsgang:

Vi starter med node 6, som i utgangspunktet er tilfeldig valgt av MRC. Noden isoleres, som betyr å sette vekten til samtlige linker (7), men unntak av én (39) til uendelig (36). Den gjenværende linken begrenses, som betyr å sette vekten til summen av *samlige vekter* i hele topologien (40). Dette garanterer at node 6 i konfigurasjon 1, kun kan nås via linken fra node 5. For å sikre at node 6 virkelig kan isoleres, uten å segmentere hovedåren i topologien, isoleres node 5 rekursivt (45). Feiler isoleringskjeden (21), fordi noden som forsøkes isolert ikke likevel kan isoleres i gjeldende konfigurasjon vil isoleringsprosessen

tilbakestille vektendringene som er foretatt (gjøres i forbindelse med 21) før en ny konfigurasjon forsøkes brukt.

Node 5 forsøkes isolert, men kan ikke isoleres i konfigurasjon 1 fordi det hadde ført til at node 6 selv hadde mistet sin eneste operative link til hovedåren i konfigurasjonen. Derfor forsøkes konfigurasjon 2. Node 5 isoleres, som betyr å sette vekten til samtlige linker (7), men unntak av én (40) til uendelig (36). I dette tilfelle er det linken som er koblet til node 2, som *ikke* isoleres, men settes til summen av *samlige vekter* i hele topologien (40). Dette fortsetter det rekursive kallet som, som nå forsøker å isolere node 2 i eksempeltopologien (45).

Node 2 kan isoleres i konfigurasjon 1, som innebærer nøyaktig det samme fremgangsmåten som med node 5, bortsett fra at MRC denne gangen er tvunget til å isolere node 1 neste gang.

Også node 1 isoleres rekursivt, som betyr at den er tvunget til å starte med å isolere linken som er gitt som parameter (koblet til node 2 i dette eksempelet), før noen andre linker isoleres. Men før vi går så langt, må konfigurasjonen som skal brukes velges. Fordi node 1 ikke kan isoleres i samme konfigurasjon som sin rotnode (hvilket hadde endret vektendringen i linken mellom node 1 og 2), velges konfigurasjon 2. Linken koblet til node 2 isoleres, deretter isoleres samtlige andre linker. Men fordi det her bare finnes en annen link, og den er ikke-isolert (11, 32) tvinges denne (34) til å bli begrenset (40). Hvilket utløser nok et rekursivt kall. Node 3 er dermed neste node som rekursivt isoleres.

Hvis node 3 isoleres i konfigurasjon 1 brytes kravet til MRC om at hovedåren i konfigurasjonen skal forbli inntakt (5), derfor forsøkes konfigurasjon 2, men det samme gjelder her. Noden isoleres derfor til slutt i konfigurasjon 3. Og som påkrevd isoleres først linken tilkoblet rotnoden (node 1). Men fordi node 5 og 6 begge to har sørget for at de resterende linkene tilkoblet node 3 allerede er isolert i en annen konfigurasjon, blir *begge* disse linkene begrenset.

Dette fullfører isoleringskjeden startet av node 6. Algoritmen til MRC kan derfor nå velge fritt hvilken node som skal isoleres neste gang, eventuelle noder (node 1, 2, 3, 5 og 6) som allerede er isolert hoppes over (3).

I figuren velges deretter node 6 tilfeldig av MRC, som tvangsbegrenser linken mellom node 5 og 6, og isolerer øvrige linker. Dermed avslutter isoleringskjeden, denne gangen uten å utløse et rekursivt kall. Node 5 isoleres til slutt, som tvinges til å bruke konfigurasjon 3 av de tilstøtende og isolerte linkene i konfigurasjon 1, og av regelen om å ikke segmentere hovedåren i konfigurasjon 2.

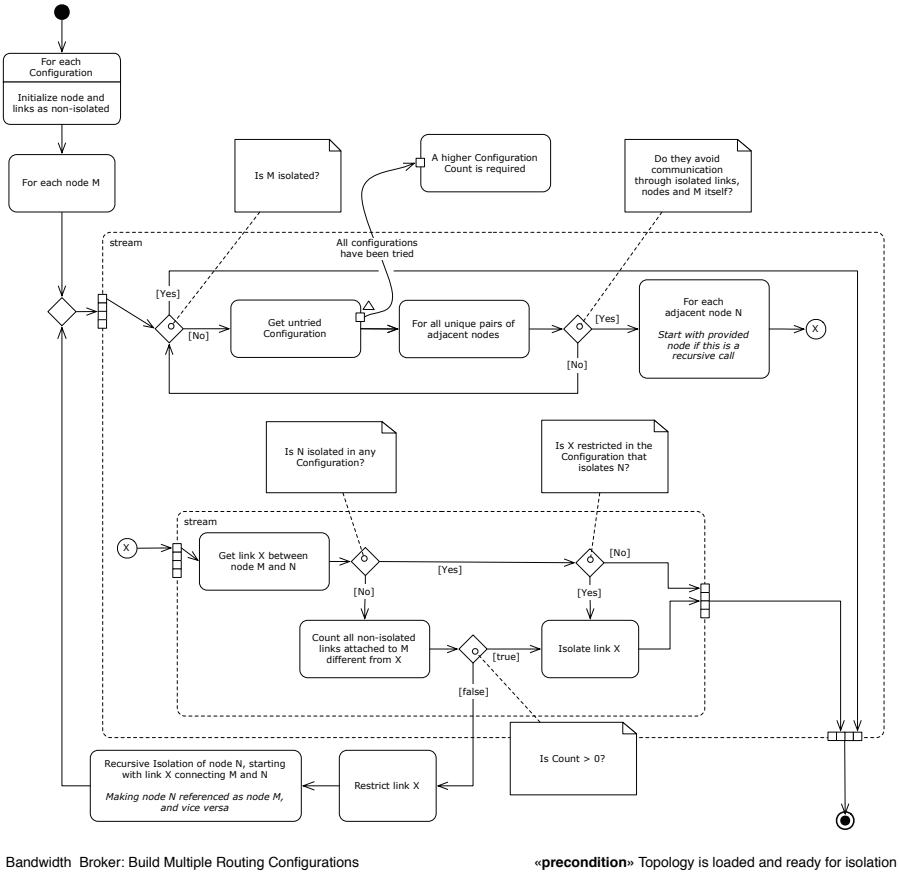
Implementeringen av denne algoritmen tar utgangspunkt i kildefilen `no.gatada.sak.simulator/`*`Simulation.java`*, fra linje 265 til 428.

Pseudokode som illustrerer hvordan MRC isolerer topologien

```
1  foreach vertex_V in Graph
2  {
3    while vertex_V != isolated in configuration_P and not all configurations  are tried
4    {
5      if isolating vertex_V does not divide configuration_P
6      {
7        foreach incident edge_J of vertex_V
8        {
9          Let vertex_W be incident to vertex_V and connected by edge_J
10
11         if vertex_W is isolated in some configuration_Q
12         {
13           if weight of edge_J == RESTRICTED
14           {
15             if there exists some non-isolated edge incident to vertex_V different from edge_J
16             {
17               edge_J = ISOLATED
18             }
19             else
20             {
21               break;
22             }
23
24           }
25           elseif weight of edge_J == ISOLATED but not in configuration_P
26           {
27             edge_J = RESTRICTED
28           }
29         }
30         else
31         {
32           // vertex_W er ikke isolert
33
34           if there exists some non-isolated edge incident to vertex_V different from edge_J
35           {
36             edge_J = ISOLATED
37           }
38           else
39           {
40             edge_J = RESTRICTED
41
42             // For å sikre at edge_J virkelig kan sioleres i konfigurasjon Q
43             // blir vertex_W forsøkt isolert som neste node, ved å starte med edge_J
44
45             isolateInConfigurationQ(vertex_W, starting with edje_J);
46           }
47         }
48       }
49     }
50
51     Select next Configuration as configuration_P
52   }
53
54   if vertex_V != isolated && tried all Configurations
55   {
56     throw exception!
57   }
58 }
```

Illustrasjon til hovedfagsoppgave MMVII

Figur 3: Pseudokode som illustrerer hvordan MRC traverserer topologien når nettverkskomponentene isoleres. Linje 47 er et rekursivt kall som iverksetter isolering av nødvendige nabokomponenter, slik at vi er sikre på at alle komponenter i topologien kan isoleres. Forutsatt at topologien er flerkoblet, vil linje 58 bety at antall konfigurasjoner må økes fordi isolering av samtlige komponenter feilet med gjeldende antall. At et link (edge) blir isolert, settes tilknyttet vekt/kost til uendelig, hvis den begrenses (restricted) settes vekten til summen av alle vekter i topologien. Dermed er vi garantert at en begrenset link ikke blir brukt med mindre det ikke finnes en alternativ sti til eller fra gjeldende node.



Figur 4: Viser aktivitetsdiagram av pseudokoden som vises i figur 3. Det rekursive kallet i pseudokoden, linje 47, kan sees nederst til venstre i dette diagrammet. Dette aktivitetsdiagrammet leses fra øverste ventre hjørne, til nederst i høyre hjørne og følger pilene fra en aktivitet til neste. Sirkelen som inneholder X markerer kun at aktiviteten er delt for å få plass på arket, og betyr ingenting utover dette.

2.2 Relatert arbeid

Ettersom dette arbeidet anvender MRC på en måte som hittil er helt uprøvd, er det lite tilgjengelig arbeid som direkte er relatert til MRC og resursallokering. Vi vil derfor her gjennomgå arbeid som er relevant for MRC og resursallokering, men ikke kombinasjonen av disse.

Når det dreier seg om resursreservering i nettverk, tenker antakelig de aller fleste på bruk av RSVP (eng.: Resource Reservation Protocol) [43]. Som navnet tilsier er RSVP i seg selv ingen ruteprotokoll, i stedet dreier det seg om en kontrollprotokoll som håndterer reservasjoner av resurser. RSVP er således lagd for å kjøre parallelt med eksisterende og fremtidige ruteprotokoller, og er ansvarlig for meldingsutvekslingen i forbindelse med oppkobling av resursreserveringer langs en sti i nettverket. I denne oppgaven bruker MRC til nettopp å etablere et utgangspunkt for resursreservering i et nettverk, vi anser det derfor ikke som et utenkelig konseptuelt sprang å benytte RSVP sammen med MRC i et gitt nettverk.

Grunnideen til MRC er å benytte nettverkstopologien med tilhørende kostnadene tilknyttet hver enkelt link, til å generere et sett deltopologier, hvorav hver deltopologi inneholder et unikt utvalg med kostnadsjusterte linker. Ved å øke kostnaden forbundet med bruk av bestemte linker, vil trafikken i praksis bli manipulert til å unngå utvalgte stier i nettverket. Vårt mål med denne oppgaven har vært å maksimere antall resursallokeringer som nettverket kan håndtere, ved å benyttet MRC til å påvirke hvilke stier som benyttes ved reservering av resurser i et nettverk; uten å ta stilling til feil i nettverket og hvordan feil eventuelt kunne ha påvirket allokeringskapasiteten.

Gopalan et al. jobber med LCBR (eng.: Link Critical Based Routing), drevet av et ønske om å maksimere antall reserveringer som kan bæres av nettverket, og samtidig tilby spesifisert tjenestekvalitet [12]. LCBR oppnår høy nettverksresursutnyttelse, og tilbyr reserveringer i form av QoS-garanterte VPN-kanaler (QVPN) med ende-til-ende forsinkelse- og båndbreddegarantier. I arbeidet deres introduserer ytelsesmål tiknyttet flere LCBR-algoritmer, og tar utgangspunkt i

et enkelt AS der samtlige resurser er under full kontroll av en enkelt operatør.

For MPLS-baserte IP-nettverk har Lee et al. sett på belastningballansering ved å benytte tre hybrid flerstialgoritmer (eng: multipath algorithms) til å fordele belastningen forbundet med inngangs- og utgangsnoden over flere stier i nettverket [31]. Dette blir dermed tilsvarende utnyttelsen av likeverdige stier med MRC.

Et sentralt mål lagt til grunn ved utformingen av MRC er blant annet robust håndtering av fysiske feil i IP-baserte nettverk. Det var også et sentralt mål at MRC skulle lette arbeidet forbundet med både implementering og administrering, i tillegg til håndtering av enkeltstående feil.

De fleste alternative løsninger til MRC som tilbyr feilhåndtering, benytter forhåndsgenererte frikoblede stier mellom inngang- og utgangsnoden [50]. Metodene krever at det genereres en løsning for hver mulig node og link som kan feile. Utfordringen forbundet med en slik løsning er at antallet stier gjerne blir uhåndterlig mange, selv i relativt små nettverk. Grover og Stamatelakis har introdusert en feilbeskyttelsesmetoder som benytter strukturerte støttestier (eng: backup trails), hvilket tilsvarende MRC, resulterer i nettverksabstraksjoner som er lettere å håndtere [14] enn frikoblede stier. Det finnes også arbeid nedlagt i tilsvarende mekanisme for MPLS, hovedsaklig anvendt i optiske nettverk, men de resulterende støttestiene er uakseptabelt lange [48].

Redundante trær er et alternativ som tilsvarende MRC, kan benyttes til å beskytte vilkårlig flerkoblede nettverk [35]. Denne metoden benytter datastrukturer til å generere både link og node-redundante trær for vilkårlig tokoblete nettverk [35]. Disse trærene, kalt røde og blå trær, kobler hver til en felles rot, slik at når det oppstår en feil i et tre kan et annet benyttes. Xue et al. har videreutviklet algoritmen til å generere tjenestekvalitetsorienterte trær som tar hensyn til kostnad og forsinkelse [51]. Selvom denne teknologien i utgangspunktet er utarbedet for optiske nettverk, har denne teknikken også blitt foreslått brukt til MPLS-feilhåndtering.

Barthos og Ramon demonstrerer utnyttelsen av Medards metode to-trær

metode for MPLS gjenoppretting. Deres metode skiller seg fra [35] ved at egressnoden brukes som rotnode. I tillegg, kalkuleres optimal sti ved hjelp av blå og røde trær kun for gjenoppretting. Med andre ord, i tillegg til å kalkulere primærestien for hver par med inngang- og utgangsnoder, kalkuleres rød-blå trær også for hver utgangsnode. Forfatterne demonstrerer at denne tilnærmelsen krever få merkelapper (eng.: labels) og at støttestienes lengde ikke blir betraktelig lengre enn for MPLS fast-reroute.

Grover og Stamatelakis introduserer et annet konsept som gjør nettverket tollerant for feil med noe de kaller beskyttelseskretser [15, 16]. Målet deres er å etablere raske kretser som tilbyr feilkorrigering like raskt som vi vanligvis bare finner i ringtopologier. Deres metode kalkulerer en eller flere kretser som når alle noder i nettverket. Metoden er optimalisert til å håndtere linkfeil, men også for å minimalisere behovet for overdimensjonering av linkene i nettverket. Når en link feiler, blir trafikken lokalt flyttet over og rutet etter denne kretsen i stedet for TKS, noe som bidrar til en vesentlig økning av stiens lengde. Grover og Stamatelakis har også tilpasset dette konseptet til IP/MPLS-nettverk [47, 48].

3 Metode

Problemstillingen som søkte å finne ytelsesrelaterte svar relatert til bruk av en ikke-eksisterende nettverksplattform, gjorde det klart allerede fra begynnelsen at oppgaven ville ta i bruk simulering. Det var derimot langt i fra gitt hvordan simulatoren skulle realiseres. En rekke alternative løsninger var oppe til vurdering, hvorav noen inkluderte bruk av maskinvare. Men etterhvert som arbeidet skred frem med oppgaven, ble det stadig klarere at befatning med annet enn mykvare ville etter all sannsynlighet innebære en arbeidsmengde uforenelig med denne oppgavens varighet, og det uten nødvendigvis å tilføye konklusjonen noe av nevneverdig verdi.

På dette tidspunktet hadde Amund Kvalheim ved Simula Senteret fullført utviklingen av et sett komponenter til bruk i simuleringsmiljøet J-Sim [38].

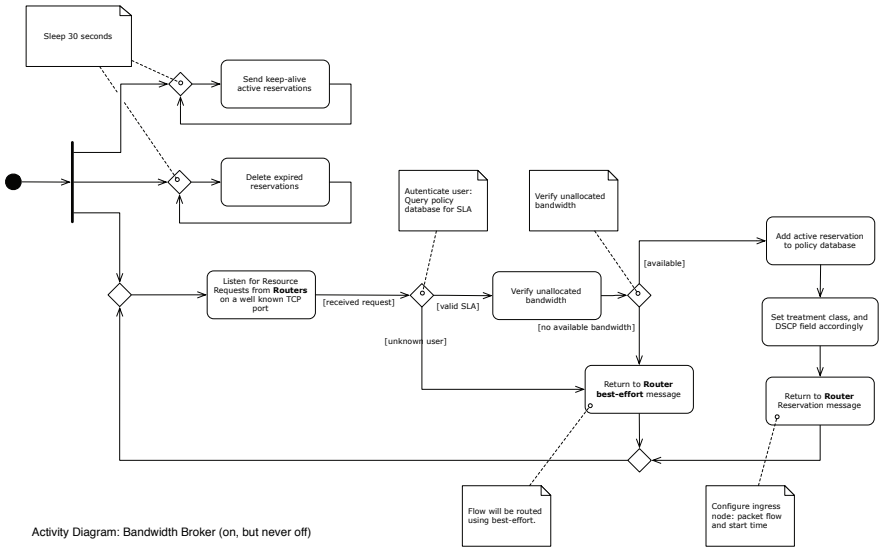
Komponentene ble lagd for å teste reaksjonstid og feilopprettingsevnen til Resilient Routing Layers, en feilbeskyttelsesteknologi nært beslektet med teknologien planlagt brukt i denne oppgaven. J-Sim var derfor et naturlig startsted i søken etter et egnet simuleringsmiljø for denne oppgaven. Men ved nærmere undersøkelse ble visualiseringsmulighetene til J-Sim funnet utilstrekkelige, derimot gjorde erfaringene knyttet til utprøvningen av J-Sim det naturlig å fortsette implementeringsarbeidet i Java.

Nå som Java hadde blitt valgt, ble formgivningsarbeidet av selve simulatoren påbegynt, og parallelt med dette startet et rimelig bredt søk etter mulige programmeringsbibliotek som kunne bistå visualiseringen av MRC.

Utviklingsprosessen av SAK gikk gjennom fire faser, hvorav den initielle systemutformingsfasen fulgte gangen for objektorientert analyse, hvilket innebærer å avdekke, analysere og spesifisere systemkrav som objekter med identitet, tildelte egenskaper og operasjoner [11]. Under denne prosessen utviklet simulatoren seg fra å være en altomfattende modell av faktiske nettverksforhold (ref. fig. 5), til en abstrakt og svært fokusert simulering av resurshåndteringen som utføres av en sentral enhet, slik som en båndbreddemegler. Denne abstraksjonsprosessen var helt nødvendig for å få redusert antall variabler til noe som kunne håndteres. Den resulterende strukturen ble dokumentert med det forente modelleringsspråket UML (eng.: Unified Modeling Language) 2.0 [20] og implementert i Java i de tre etterfølgende fasene.

Innen systemet var ferdig spesifisert hadde søket etter et visualiseringsbibliotek endt med kandidaten[20] Cytoscape [45], et bibliotek primært utviklet for analysering og visualisering av biologisk nettverksdata. Men hverken verktøy eller tilgjengelige resurser var til særlig hjelp ved utviklingen av simulatoren. Så da SAK var ferdigspesifisert og første implementasjonsfase tok til, ble det i lengre tid utviklet kode som ikke støttet noen form for visualisering. Dette skulle brått bli forandert da en artikkel [40] ble funnet som anvendte JUNG [37] biblioteket.

[20]Les mer om programmeringsbibliotekene som ble vurdert i kapittel 4.1.2.



Figur 5: Arbeidet med utformingen av SAK tok utgangspunkt i at alle aspekter som lot seg beskrive i et UML-diagram, også skulle implementeres. Etterhvert ble det tydeligere hvilke funksjoner som kunne abstraheres uten at det ville ugyldiggjøre simuleringsresultatene, og SAK ble til slutt opp en abstrakt nettverksimulator, som utelukkende fokuserte på resursallokeringskapasiteten i et tilgangsstyrt nettverk.

Introduksjonen av JUNG førte til den første store endringen av kodebasen. Med ustrakt støtte for analyse, modellering og visualisering av nettverksdata, overflødiggjorde JUNG mye av arbeidet allerede nedlagt i SAK. For å ta skikkelig i bruk de resursene biblioteket stilte til rådighet var det derfor ønskelig fra forfatterens side å skrape all kode, men fremdriftsplanen gjorde det nødvendig å beholde flere av de allerede ferdigstilte komponentene, slik som det tekstbaserte kommandogrensesnittet og isoleringsalgoritmen (som på dette tidspunktet var delvis ferdig).

Utviklingen av SAK gikk dermed inn i sin tredje fase, som introduserte visualisering og førte med seg en dypere forståelse av isoleringsalgoritmen som til frem til nå kun hadde kommunisert med tekst. Visualiseringen av MRC virket svært oppklarende og forenklet feilsøk drastisk. Mye tid ble i denne fasen lagt ned i å finne en god visalisering av algoritmene som traverserte nettverkstopologien,

og det var på dette tidspunktet en fundamental feil ble avslørt; et resultat av en overoptimalisert programflyt i et av de mer sentrale akvitietsdiagrammene brukt til implementering av SAK. Feilen førte til at isoleringsalgoritmen som var nøye spesifisert og implementert i henhold til gjeldende skjema, traverserte og feilregistrerte både linjer og noder som isolerte. Etter en rekke forsøk på å korrigere feilen ble det klart at koden ikke stod til å redde, og med det gikk prosjektet inn i sin neste og siste fase.

Etter mye arbeid med koden, gjennomlesing av artikler [27] [17] [18] og korrigering av UML-diagrammer fremstod koden som tilsynelatende feilfri, i tillegg var JUNG bedre integrert nå enn hva som var mulig å få til med den gamle koden. Etter dette fulgte det en lang periode med generell kodekorrigering og lettere optimalisering, før koden var klar til bruk.

SAK ble deretter brukt til å produsere et stort antall data, som vi i utgangspunktet analyserte ved hjelp av deskriptiv statistikk. Den foreløpige analysen avdekket et forholdsvis høyt standardavvik, hvilket utelukket direkte sammenligning av gjennomsnittet (s. 69). Vi bekreftet deretter normalfordelingen ved bruk av histogrammer (s. 71 og 70), og fordi tallmaterialet var såpass stort valgte vi å benytte U-testen, se under, til hypotesetesten på side 71.

$$U = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{1}$$

4 Simulering av kapasitet (SAK)

Innføring av nye tjenester får gjerne uforutsigbare konsekvenser når de implementeres i et allerede presset nettverk. Nye protokoller avhenger derfor av skikkelig testing før de implementeres i eksisterende nettverk, og denne testing blir gjerne utført ved hjelp av simuleringer. Simuleringer brukes ved studier av dynamisk adferd av objekter eller systemer kjørt under forhold som ikke lett eller sikkert kan tilføyes reelt [6]. Sett i sammenheng med nettverkssimuleringer, brukes disse til å anslå kapasitet og ytelse av eksisterende nettverk, til å bistå under utvikling av nye protokoller, og til å forutsi ytelse og forventet adferd for nye komponenter implementert i operative nettverk [19].

Bruk av simulering til testing av nettverksprotokoller er i dag utbredt praksis. Dette skyldes både økt kapasitet og tilgjengelighet av billige maskiner til edb., og at nettverk i dag har blitt så komplekse at de *vanskelig* lar seg beskrive med matematiske modeller. Kompleksiteten er et resultat av at det implementeres en blanding av faste og trådløse nettverksteknologier, i tillegg til nye digitale tjenester som bruker nettverket på måter det i utgangspunktet ikke ble lagd for å støtte. Under slike omstendigheter blir det vanskelig å undersøke konsekvensene av ny teknologi på andre måter enn ved simulering. En simulering gjør det mulig å foreta presise og fullstendige målinger av et spesifisert scenario ved å observere hvordan netttrafikk og kapasitet blir påvirket ved endring av individuelle variable og komponenter. Et anvendt scenario defineres ved hjelp av regler og operative prosedyrer, og resulterer gjerne i nye situasjoner, og ofte betingelser, som igjen kan benyttes til å videreutvikle og forfine nye kjøringer av simulatoren. Ved hjelp av tidskomprimering kan interessante situasjoner gjerne også raskere produseres under simulering, enn ved observasjoner av den virkelige verden.

SAK er en abstrakt nettverkssimulator, utviklet for å undersøke nærmere hvor egnet MRC er til allokering av resurser i et tilgangskontrollert nettverk. I utgangspunktet ble det forsøkt anvendt andre teknologier og tilgjengelige

rammeverk, fremfor et egenutviklet system, men vi fant ingen løsning som lot oss fokusere på visualisering og resursallokering i tilgangskontrollstyrte nettverk. Det vi derimot fant var løsninger som til syvende og sist ville kreve mer arbeid og gi oss mindre frihet, enn hvis simulatoren ble lagd av oss fra bunn av. SAK fremstår i dag som en tilstrekkelig brukervennlig og robust nettverkssimulator, som foretar målinger av resursallokeringskapasiteten til både SRT og MRC i et vilkårlig flerkoblet nettverk, og gir brukeren en oversiktlig og ryddig visuell introduksjon til MRC ved hjelp av visuell sanntidsanimasjon.

For å gi en rask introduksjon til SAK, forløper en typisk kjøring av simulatoren på følgende måte:

1. `load 1b.top`

   SAK leser inn en topologifil og klargjør den nødvendig objektstrukturen til videre bearbeiding. Topologien som er lastet vises i sin helhet i et eget vindu, som vi refereres til som turvinduet. Navnet er passende fordi vinduet viser progresjonen under isoleringsprosessen, hvilket inkluderer *turen* som de rekursivene kallene tilbakelegger rundt i topologien.

2. Bruker organiserer den viste topologien i henhold til sine kriterier. Det er opplagt nyttig å kunne flytte noder slik at det oppnås forbedret oversikt, eller økt lesbarhet. I de tilfeller der topologien stammer fra reelle nettverk, kan det også være nyttig å organisere nodene i forhold deres geografiske beliggenhet.

3. `config`

   Denne kommandoen starter isoleringsprosessen. Hvis prosessen fullfører uten en feilmelding, har topologien blitt isolert. Dette betyr at et nødvendig antall rutekonfigurasjoner har blitt generert, slik at resursallokeringsprosessen kan ta fatt. Samtlige av de grafiske vinduene lagres som bilder i katalogen `./SAKoutput/images/`.

4. `route`

   Reservasjonsanmodninger genereres fra et ubegrenset antall tilfeldig valg-

te nodepar. Kommandoen avslutter ikke før 13 anmodninger er avslått *sammenhengende* av både SRT og MRC. Den resulterende statistikken lagres i katalogen `./SAKoutput/data/`.

5. `exit`

   Denne kommandoen lukker samtlige vinduer og avslutter SAK.

Du kan lese mer om hver enkelt kommando i kapittel 4.2.1, fra side 49. I resten av dette kapittelet skal vi gjennomgå systemspesifikasjonene til SAK og gi en innføring i hvordan SAK anvendes. Vi inkluderer også programmeringsjournalen, om av ikke annet enn historiske årsaker.

4.1 Systemspesifikasjon

Simulatoren SAK er implementert i Java versjon 1.5.0_04, og benytter JUNG[21], og Apache Jakarta Commons Collections 3.1[22] i tillegg til Javas standardbibliotek[23]. JUNG er et bibliotek basert på åpen kildekode, utviklet for å støtte modellering, visualisering og analyse av grafbasert data [37]. Jakarta Commons Collections er også åpen kildekode, utviklet på konsensusbasis, som tar sikte på å utvide funksjonaliteten til *Java Collections* med egne gjenbrukbare javakomponenter. Simulatoren ble i all hovedsak utviklet på en Apple PowerBook G4, og koden ble skrevet i Xcode og kompilert fra kommandolinjen med kommandoen `javac -d bin -sourcepath src src/no/gatada/sak/Sak.java`

Simulatoren ble utviklet som en del av denne Cand. Scient oppgaven, underlagt Instituttet for Informatikk ved Universitetet i Oslo. Hovedmålet med SAK var å generere ytelsesdata knyttet til reservasjonsallokeringsprosessen foretatt av en sentral nettverkskomponent, slik som en båndbreddemegler eller tilsvarende. Sekundært var det et ønske å visualisere MRC på en måte som kunne øke brukerens forståelse av denne mekanismen, hvilket førte til at både isole-

[21] Klippet fra nettsiden til JUNG: "JUNG provides facilities to dynamically change graphs, to programatically call code, and to output the results as the program continues."

[22] Klippet fra nettsiden til The Jakarta Project: ".. [Jakarta] has become the recognised standard for collection handling in Java."

[23] SAK kompilerer uten Cern Colt Scientific Library 1.2.0 og Xerces.

ringsprosessen og allokeringsfasen, nå med den endelige versjonen av SAK, blir visualisert i sanntid. Vi er overbevist om at denne visualiseringen burde være av interesse for de fleste som har befatning med MRC, hvilket også er hovedårsaken til at SAK ble gitt et både ryddig, brukervennlig og et relativt robust kommandolinjebasert grensesnitt. Under følger en punktvis liste med systemkravene som la grunnlaget for utformingen av SAK, kommentarer tilføyd i under eller i etterkant av utviklingen er skrevet i kursiv.

- Problemdomenet består av rutere, tilknyttede linker, trafikkprodusenter og -konsumenter. Det legges opp til at nevnte komponenter administreres sentralt, og at funksjonalitet utover resursallokeringsprosessen ikke blir behandlet videre i denne oppgaven.

- SAK skal ikke støtte modifisering av hverken topologifilene eller selve topologiene under kjøring. *Vi tar likevel vare på filnavnet på topologifilen ettersom dette benyttes ved lagring av topologibilder og statistikk.*

- Visualiseringen av av topologien skal ta i bruk egne vinduer, og lagre resultatet både som bildefiler og tekstbaserte datafiler. *For at datafilene skulle bli meningsfulle også for andre brukere av SAK, ble det tilføyd en selvforklarende topptekst i begge filene. Bildefilene burde gi mening for de fleste, men gir nok mer mening hvis brukeren har grunnleggende kjennskap til MRC. De endelige destinasjonene brukt til lagring kan kun endres ved å modifisere kildekoden.*

- For at SAK skal kunne lese inn en topologifil, må programmet kunne håndtere innlesing av et format kompatibelt med tilgjengelige filer. *Topologiene vi etterhvert fikk tak i ble levert av Rocketfuel [39], som skiller hver node med et mellomrom, og hvert nodepar med linjeskift. Hver enkel node identifiseres med en unik identifikator. Topologifilene vi har generert med BRITE [28] benytter i all hovedsak samme filformat som Rocketfuel, men starter filen med et tall som oppgir antall nodepar og avslutter med* `EOF`*.*

Filinnlesingen håndterer dette ved å varsle bruker om alle linjer som ikke kun inneholder to elementer per linje, men uten å avbryte innlesningen.

- Topologifilene som benyttes av SAK inneholder nettverkstopologien uten vekt/kost, utgangspunktet er derfor at hver komponent i nettverket gis 1 som vekt.

- Topologien som lastes skal vises i et eget vindu som gis tittelen "Loaded Topology". Samtlige vinduer som opprettes skal være av typen JFrame, og støtte interkasjon med musen. Dette vil muliggjøre manipulering av nodenes plassering slik at brukeren får bedre oversikt over nettverket. *Vinduet som viser topologien som er lastet refereres til som turtabellen. Tilpasning av nodenes plassering er kun mulig ved hjelp av en mus; SAK støtter ikke bruk av annen maskinvare til dette. Tilpasningen må forøvrig finne sted før konfigurasjonsprosessen iverksettes, ettersom nodenes plassering i vinduet til hver enkel konfigurasjon initialiseres utifra plasseringen i turtabellen.*

- Konfigurasjonsprosessen forutsetter at en gyldig topologi er opprettet, og har til hensikt å generere de nødvendig konfigurasjonene som skal til for å isolere samtlige komponenter i topologien. Prosessen kan avslutte på to måter, men det er kun når den fullfører etter en *vellykket isolering* at brukeren kan iverksette resursallokeringsprosessen. Hvis isoleringen ikke er vellykket, betyr dette at topologien enten ikke er flerkoblet, eller at et for lavt antall konfigurasjoner ble forsøkt benyttet. Vi undersøker om topologien er flerkoblet før isolering iverksettes. *Denne sistnevnte flerkoblingstesten gikk vi etterhvert bort fra, selvom vi nå i etterkant dessverre ikke kan forklare hvorfor. Konsekvensen av dette er at en ikke-flerkoblet topologi blir forsøkt isolert helt til det høyeste antall konfigurasjoner blir nådd og feiler. I gjeldende kode er dette tallet satt til 10 konfigurasjoner. Hvis SAK kjøres med forsinkelse, startes hvert konfigureringsforsøk manuelt, hvilket gjør at bruker kan velge å laste inn en annen topologi fremfor å utføre meningsløse konfigureringsforsøk av en ikke-flerkoblet topologi.*

- Etter fullført konfigurering skal det være mulig å manipulere nodeplasseringen i hvert enkelt vindu, uten at dette påvirker plasseringen av de ekvivalente nodene i de øvrige vinduene.

- Resursallokeringsprosessen skal kunne benytte samme isolerte topologi gjentatte ganger, som betyr at resursreserveringen i den klargjorte topologien må kunne nullstilles.

- Trafikken genereres ved at to tilfeldig valgte noder velges som henholdsvis inngangs- og utgangsnoder for hver enkel trafikkreservasjon. Det skal kunne genereres et ubegrenset antall trafikkreservasjoner. *Vi gjorde det også mulig å begrense antall nodepar, men antall reservasjonsanmodninger som utstedes er likevel kun begrenset av metningsnivået i topologien.*

- For at SAK skal kunne skrive både bilder og data til platelageret er det nødvendig at SAK undersøker om katalogen allerede finnes, og oppretter den om nødvendig. Hvis opprettelsesforsøket feiler skal katalogen som SAK kjøres fra benyttes. *Vi har ikke tatt stilling til eller spesifisert hva som skjer hvis SAK ikke klarer å skrive til platelageret. Bruker oppfordres derfor til å gi SAK skriverettigheter til* `./sak/bin/` *forut for kjøring.*

- For at visualiseringen av topologien skal være meningsfull og tilstrekkelig brukervennlig, er det nødvendig at alle ekvivalente noder har samme relative posisjon i hvert sitt vindu. Dette forutsetter at ett vindu bestemmer nodeuttegning (eng.: layout) til øvrige vinduer, og til dette skal turtabellen brukes. Ved kjøring kopieres posisjonen til nodene i turtabellen over til samtlige ekvivalente noder i hvert enkelt konfigurasjonsvindu. Denne kopieringsprosessen gjøres kun ved opprettelsen av konfigurasjonsvinduene. *Fordi SAK benytter to forskjellige uttegningsmetoder for henholdsvis turtabellen og konfigurasjonene, var vi nødt til å gjøre dette på et per-node nivå i hver enkel konfigurasjon i det vinduet blir opprettet og vist til skjerm.*

- Ønsket om at SAK skal kunne brukes av andre enn utvikleren selv, stiller

krav til brukervennlighet, funksjonalitet og robusthet.

- Simulatoren skal generere sammenlignbare ytelsesmålinger som kan benyttes som statistisk grunnlag i oppgaven. For at disse målingene skal være relevante er det nødvendig å simulere både store og mindre trafikkstrømmer i reelle nettverktopologier. Simulatoren må derfor kunne gjenskape nettverk basert på virkelige topologier. *Dette ble som nevnt løst ved at innlesingen av topologifiler, ble gjort kompatibel med både filene levert av Rocketfuel og de generert av BRITE.*

- SAK kjøres fra kommandolinjen og krever topologidata ved oppstart, enten i form av en enkel fil eller en katalog som inneholder topologidata. *Det er kun* `run <dir> <int> ls [dir]` *som aksepterer en katalog, øvrige kommandoer baserer seg på enkeltstående topologifiler.*

- Nettverket består av urettede linker og vektede noder. Ettersom MRC kontrollerer dataflyten i nettverket ved å manipulere vekten til linkene, må vekten til linkene kunne endres under kjøring. I MRC er det er høy kost som fører til at pakker finner alternative stier gjennom nettverket, og vekten til isolerte komponenter må kunne settes uendelig høyt. Hvis en node langs korteste sti ikke har tilstrekkelig med resurser, vil MRC forsøke korteste sti i en hittil uprøvd konfigurasjon. Samtlige konfigurasjoner deler de samme globale resursene, men hver konfigurasjon skal likevel lagre resursforbruket tillagt hver enkelt.

- Det skilles ikke mellom kantnodene og kjernenoder. Samtlige noder kan utstede en SLA for reservering av resurser. *Fordi SAK abstraherer hele resursforhandlingsprosessen, behandles ikke SLA som sådan - i stedet genereres resursanmodninger direkte av BB uten at det foreligger noen kontraktforhandling. I realiteten følger en slik anmodning etter en vellykket forhandling av en mottatt SLA.*

- Både MRC og SRT ruter strømmer etter korteste sti, og benytter ingen

form for feilkorrigering.

- Konfigurasjonsalgoritmen behandler samtlige noder sekvensielt og kun en enkelt gang. Hvis isoleringsprosessen av et flerkoblet nettverk feiler, skal den øke antallet konfigurasjoner og forsøke å gjennomføre en vellykket isolering. Algoritmen avslutter etter et bestemt antall isoleringsforsøk. Antall ganger settes i koden og kan ikke endres under kjøring.

- Konfigurasjonsalgoritmen går i løkke gjennom samtlige noder i nettverket, og prøver å isolere en node av gangen. En linje isoleres i samme iterasjon som en av sine tilknyttede noder isoleres.

- Hvis en node ikke kan isoleres i minst en konfigurasjon, forutsatt at topologien er flerkoblet, vil algoritmen avslutte med en feil. Feilen skal varsles til bruker, som enkelt skal kunne øke antall konfigurasjoner og kjøre et nytt isoleringsforsøk. *SAK foretar denne konfigurasjonsøkningen og iverksetter et nytt isoleringsforsøk automatisk, med mindre forsinkelse av kjøringen er aktivisert.*

- Algoritmen til MRC må implementeres og fordeles i en dertil egnet objektstruktur. *Ved innføringen av JUNG ble algoritmen påvirket, les mer om dette under kapittel 4.2.3.*

- Isoleringsprosessen går i dybden rekursivt, før den går i bredden. *Dette er det rekursive kallet på linje 45 i figur 3, side 25.*

- Under ruting av data vil utvalgte noder fungere som trafikkavsendere og andre som trafikkmottakere. Trafikkmønsteret genereres tilfeldig, og antallet SLA skal kunne settes og endres av bruker under kjøring. *Til å endre antall nodepar som produserer resursanmodninger, brukes kommandoen* `set [count]`*, se kommandoen nærmer forklart i kapittel 4.2.1.*

- Simulatoren benytter relative måltall.

- Ettersom det rutes etter eksisterende topologier, som ikke er tilpasset bruken av en dedikert båndbreddemegler, skal SAK benytte en virtuell båndbreddemegler.
- SAK tar ikke hensyn til feil i nettverket.

4.1.1 Topologiens og konfigurasjonens korteste sti

SRT og MRC kjøres opp mot hverandre i en og samme topologi. Etter at nettverkstopologien er lest inn fra fil og etablert, må topologien isoleres før den kan benyttes til ruting. Konfigurasjonsprosessen, som benytter MRC, stiller naturligvis samme minimumskrav til topologien som MRC, hvilket betyr at grafen må være flerkoblet[24]. MRC vil alltid klare å isolere en flerkoblet graf [27].

Konfigurasjonsprosessen går gjennom to faser; den første initierer avstandstabellen nødvendig for ruting etter TKS. Den etterfølgende fasen, gitt at topologien er flerkoblet, isolerer topologien. Isoleringsprosessen produserer et nødvendig antall konfigurasjoner som senere benyttes av BB til resursallokering.

Vanligvis når pakker skal rutes i en topologi, så brukes *en* rutetabell, og pakkene blir forsøkt rutet langs den stien som er tilknyttet lavest kost, sett i forhold til det kriteriet som anvendes. SAK benytter antall hopp til å identifisere både TKS og KKS. For å identifisere TKS blir en avstandtabell benyttet. Denne inneholder avstanden i antall hopp fra en node, til samtlige andre noder i topologien. Tabellen genereres ved hjelp av Dijkstras korteste sti algoritme [4], og hver individuelle node lagrer sin egen versjon av denne tabellen. Nodene er på sin side lagret i objektstrukturen som er tilknyttet den initielle grafen som opprettes når topologien lastes.

Algoritmen som søker frem TKS (TKS-algoritmen) starter med å undersøke om inngangs- og utgangsnoden har tilstrekkelig med resurser til å akseptere resursreserveringen. Hvis *begge* nodene kan bære reservasjonen, føyes inngangsno-

[24]Gjeldende implementasjon av SAK foretar ingen testing på om grafen er flerkoblet eller ikke. Konsekvensen av dette er at SAK ikke vil gi opp isoleringsforsøket før høyeste antall tillatte konfigurasjoner (som er satt til 10) er nådd. Implementering av en flerkoblingstest er anbefalt forut for isolering av virkelig store topologier. Nødvendig kode er allerede tilgjengelig og ville eventuelt naturlig hørt hjemme i innlesningsfunksjonen til SAK.

den til et `HashSet` som inneholder samtlige noder som utgjør TKS (TKS-settet). Deretter undersøkes avstandstabellen til *nabonodene* til inngangsnoden, og fordi inngangsnoden selv ikke er utgangsnoden, vet vi at nabonoden med kortest avstand til utgangsnoden også er første hopp i TKS.

Vi minner om at samtlige komponenter i topologien selv holder rede på hvor mye av deres egne resursene som er brukt; og fordi vi i denne oppgaven begrenser oss til allokering av resurser, blir det ikke vedlikeholdt informasjon om hvem som oppretter en reservasjon i topologien. Forøvrig blir all reservering som foretas ved hjelp av SRT, lagret i det originale `Graph`-objektet som visualiseres i turtabellen.

TKS-algoritmen undersøker deretter om linken som forbinder inngangsnoden og nabonoden (n1) har tilstrekkelig med ledige resurser, og kun hvis dette er tilfelle, går algoritmen rekursivt videre til n1. Har linken *ikke* tilstrekkelig med resurser, avslutter algoritmen med å tømme TKS-settet, som fører til at SRT gir avslag på resursanmodningen. Hvis linken mellom inngangsnoden og n1 derimot har tilstrekkelig med resurser tilføyer n1 seg selv til TKS-settet. Foreløpig er det kun inngangsnoden og n1 som er medlemmer av dette settet. Noden n1 undersøker deretter avstanden til utgangsnoden i samtlige av sine nabonoder. Hvis ingen av nabonodene er utgangsnoden, vet vi nok en gang at nabonoden (n2) med lavest kost er en del av TKS. Igjen unsersøkes det om linken mellom n1 og n2 har tilstrekkelig med resurser, før TKS-algoritmen rekursivt går til n2. Slik fortsetter det til utgangsnoden nås.

Avslutningsvis, tilføyes utgangsnoden når denne endelig nås av algoritmen. Vi vet allerede at denne noden har tilstrekkelig med resurser, så denne føyes til uten ytterligere kontroll. På dette tidspunktet vil algoritmen fullføre resursallokeringen med å traversere det komplette settet i den rekkefølgen det ble etablert, og reservere resursene i samtlige av komponentene inneholdt i settet. Også linken som kobler hvert nodepar sammen blir identifisert, og de nødvendige resursene reservert før TKS-algoritmen avslutter og RAR godtas.

Algoritmen som søker frem KKS (KKS-algoritmen) i hver enkel konfigura-

sjon fungerer på tilsvarende måte som TKS-algoritmen. Den mest signifikante forskjellen, er at KKS-algoritmen først ordner samtlige av de tilgjengelige konfigurasjonene etter avstanden til utgangnoden oppgitt avstandstabellen til inngangsnoden.

I utgangspunktet ble det forsøkt å lage en algoritme som søkte gjennom samtlige konfigurasjoner i sanntid etterhvert som hver enkelt RAR ble mottatt, men dette førte til unødig prosessering tilknyttet identifiseringen av den konfigurasjonen som kunne tilby den korteste KKS. Algoritmen kombinerte avstandstabellen som ble generert for SRT og vektinformasjonen tilknyttet hver enkelt konfigurajon, til å identifisere konfigurasjonens korteste sti (KKS). Denne løsningen gjorde det derimot svært vanskelig å ordne konfigurasjonene i en fornuftig rekkefølge, med andre ord risikerte algoritmen å å finne den korteste KKS med tilstrekkelig med resurser under testing av den siste tilgjengelige konfigurasjonen. Dette var opplagt ikke optimalt, hvilket er grunnen til at vi innførte bruk av avstandstabeller også for konfigurasjonene.

Avstandstabellen gjør det mulig å foreta et svært raskt oppslag i samtlige konfigurasjoner, slik at de kan ordnes i henhold til lengden på KKS. Deretter kan hver enkel KKS gjennomgås i stigende rekkefølge, og undersøkes for tilstrekkelige resurser. I tillegg blir traverseringen av hver enkel konfigurasjon svært effektiv, fordi algoritmen kun følger den korteste sti og unngår eventuelle blindveier.

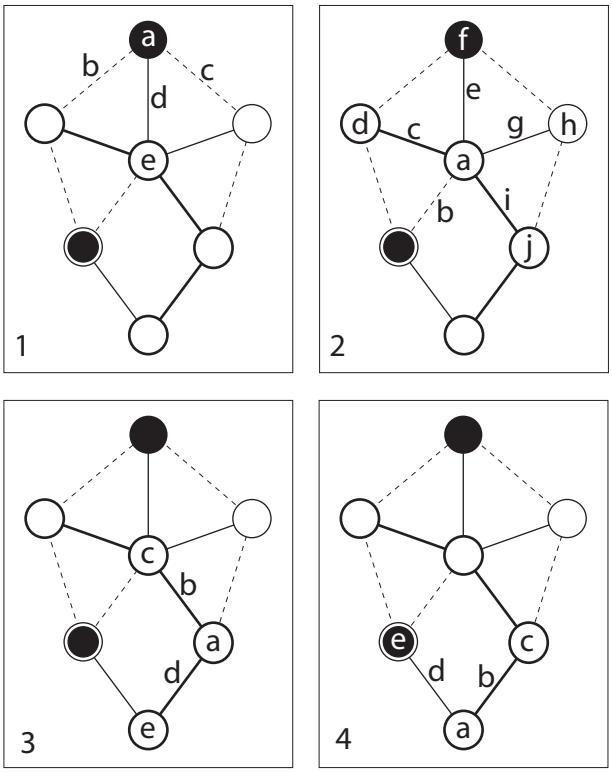
En topologi som er løst isolert, karakteriseres gjerne ved at minst en konfigurasjon kun inneholder en eller bare noen ytterst få isolerte noder. Dette innebærer at tilnærmet samtlige nodepar i en slik konfigurasjon kan kommunisere over en sti identisk med TKS, som med andre ord betyr at den korteste stien benyttes så lenge denne kan bære reservasjonen. Frem til den første reservasjonen blir avslått blir det på denne måten i praksis ingen forskjell mellom ruting etter MRC og SRT, som opplagt er positivt fordi resursutnyttelsen på denne måten maksimeres. Dette poenget nevnes kort i [18], men hvordan topologien kan isoleres for å oppnå en hensiktsmessig løst isolert topologi drøftes ikke videre.

SAK tar utgangspunkt i den første konfigurasjonen ved isolering av hver enkel node, og vil forsøke isolering i neste konfigurasjon kun hvis komponenten ikke lar seg isolere i gjeldende konfigurasjon. På den måten benyttes kun et minimum antall konfigurasjoner. I tillegg vil også traverseringen av topologien påvirke det endelige antallet konfigurasjoner. Vi presenterer ingen endelige konklusjoner her, men en interessant observasjon i denne sammenheng er at nodene i topologien `1b.top` alltid ble traversert på *nøyaktig den samme måten hver gang* den endelige løsningen begrenset seg til 3 konfigurasjoner.

Det er i prinsippet ingenting som hindrer SAK, eller en nettverksadministrator som har implementert MRC, i å tvinge MRC til å benytte unødvendig mange konfigurasjoner, slik at hver konfigurasjon i teorien kun har en enkel isolert node og en begrenset link (og et nødvendig antall isolerte linker). SAK legger derimot ikke føring på antall konfigurasjoner som benyttes, men benytter i stedet tilfeldig produserte løsninger til å sammenligne løst og tett isolerte topologier. *Tilfeldig* i denne sammenhengen innebærer at implementasjonen av MRC legger opp til å benytte så få konfigurasjoner som mulig, men uten direkte styring av neste *konfigurasjon*, *node* og *link*. Se forøvrig side 21 for flere detaljer om dette.

Figur 6 på side 45 viser trinnvis hvordan algoritmen traverserer konfigurasjonen for å klargjøre hvorvidt KKS har tilstrekkelig med kapasitet. Algoritmen tar utgangspunkt i en av nodene og jobber seg rekursivt fra denne *inngangsnoden* mot den andre *utgangsnoden*. Underveis legges samtlige noder med den korteste avstanden til utgangsnoden i et eget buffer, som i etterkant benyttes til selve reserveringen. Nodene mellomlagres for å lette en eventuell gjenoppretting hvis reservasjonen ikke kan bæres i gjeldende konfigurasjon. Kun hvis utgangsnoden nås, blir de faktiske resursersene i nettverket reservert i en etterfølgende og isolert prosess.

Algoritmen som søker seg gjennom hver enkel konfigurasjon, går først i bredden, og undersøker kun ikke-isolerte komponenter. Figur 6 tar utgangspunkt i en reservasjonsanmodning for nodeparet 7 og 2, med førstnevnte som inngangs-



Figur 6: Her vises hvordan KKS identifiserer og traverserer topologien for å finne konfigurasjonens korteste sti (KKS). I dette eksempelet starter vi i node 7 (øverst) og ender opp i node 1 (nederst). Algoritmen går gjennom fire reksursive kall vist i denne figuren. Hvert kall er nummerert nederst i venstre hjørne, og kun komponentene som er merket med en bokstav blir undersøkt eller traversert. Rekkefølgen linkene traverseres er i realiteten tilfeldig, selvom vi her har styrt traverseringen for å illustrere gangen i algoritmen så godt som mulig.

node. Figuren er delt i fire nummererte trinn, der hvert enkelt trinn illustrerer et enkelt steg i algoritmen. Kun komponentene som i figuren er merket med en bokstav blir traversert av algoritmen. At en node som ligger langs KKS sjekkes to ganger, f.eks. sjekkes node 5 av både 7 og 3 i figur 6, skyldes at grafen ikke er rettet. Det er to typer tester som kan utføres på en node under traverseringen av konfigurasjonen *a)* hvorvidt noden finnes i mellomlageret og dermed allerede er en del av KKS, eller *b)* om distansen til utgangsnoden er lenger enn øvrige nabonoder (hvilket den alltid vil være hvis den finnes i mellomlageret). Fordelen med å benytte *a)* er at neste nabonode kan undersøkes direkte, uten å sammen-

lignes med gjeldende korteste distanse. Men fordi det med tanke på ytelse, ikke var noen signifikant forskjell mellom disse to alternativene, foretar algoritmen for enkelhetens skyld kun oppslag i distansetabellen.

Før algoritmen kjøres rekursivt gjennom topologien, blir det først bekreftet at inngangs- og utgangsnoden har tilstrekkelig med resurser, hvis ikke så avsluttes algoritmen ved at RAR-en avslås. Hvis det derimot er tilstrekkelig med resurser i stiens endenoder, så tilføyes inngangsnoden til settet for *involverte noder*. Dette er settet med alle noder som er en del av KKS. Deretter sjekkes det for om vi har nådd utgangsnoden. Hvis vi har nådd utgangsnoden, så undersøkes linken mellom inngangsnoden og utgangsnoden, og hvis også denne har tilstrekkelig med resurser så godtas RAR-en. Har vi derimot ikke nådd utgangsnoden, så blir nabonoden (til inngangsnoden) som ligger nærmest utgangsnoden identifisert. Deretter undersøkes linken mellom inngangsnoden og denne utvalgte nabonoden for tilstrekkelig med resurser. Hvis det ikke er tilstrekkelig med resurser, så avslås RAR-en; er derimot nødvendige resurser ledige, så går algoritmen rekursivt til nabonoden og foretar samme undersøkelse derfra. Hele tiden velges den nabonoden som har den korteste avstanden til utgangsnoden. Dette kan du enkelt se i figur 6, der hvert rekursive kall er vist i en egen rubrikk og noden som er utgangspunktet for det rekursive kallet er merket *a*.

Som du skjønner tar algoritmen ikke hensyn til hvorvidt det finnes flere likeverdige stier, hvilket opplagt er en svakhet. Vi ser som nevnt nærmere på mulige løsninger for dette i kapittel 6.

4.1.2 JUNG, graf- og universalnettverksrammeverk

Som tittelen tilsier er JUNG et rammeverk egnet for alle typer nettverk eller grafer. JUNG tilbyr et bredt visualiseringsbibliotek, med lett forståelig og tilgjengelig kode, og gode manualer med fine og saklige eksempler. De mest brukte algoritmene for håndtering, traversering og analysering av grafer følger med pakken, eller bibliotekene som er nødvendig for at JUNG skal kjøre. Nedlastning av samtlige bibliotek gikk smertefritt, og installasjon innebar kun å tilføye jar-

filene til Javas klassesti (eng.: class path). Diskusjonsgruppen var svært aktiv, og spørsmål av alle typer ble både stilt og besvart innen kort tid - gjerne innen et par timer. Hjelpen som ble gitt var helhjertet og detaljert, hvilket gjorde det enkelt å både spørre og å løse eventuelle problemer. Selvom biblioteket til en viss grad lider av inkonsekvent navngivning og innimellom noen rare utformingsvalg, var det lett å navigere de medfølgende Javadok-ene, selvom disse ikke alltid var like utfyllende i sine kommentarer[25].

JUNG tilbyr løsninger for blant annet visualisering, interaksjon med både tastatur og mus, samt lagring av nettverksrelatert metadata, og ble etter vurdering av den medfølgende kildekoden til de nettbaserte visualiseringsdemonstrasjonene oppfattet som oversiktlig og tilnærmet perfekt for SAK.

4.2 Hvordan simulatoren SAK fungerer

Simulatoren SAK er tekststyrt og gir ved oppstart en kort beskrivelse av alle tilgjengelige kommandoer, deretter er SAK avhengig av tekstbaserte kommandoer fra bruker. For å etablere et simuleringsgrunnlag må en tekstbasert topologifil bli lest inn i minnet. Topologifilen inneholder samtlige av nettverkets noder; to tilgrensende noder per link, og nodene i hvert enkelt par er adskilt med et mellomrom. Hver enkelt av disse nodene er identifisert med en unik identifikasjonsnøkkel, som ikke selv kan inneholde mellomrom. Du kan finne samtlige topologifiler som ble benyttet i denne oppgaven sammen med kildekoden[26]. Valg av topologier var i utgangspunktet styrt av ønsket om å benytte reelle nettverkstopologier, og ettersom ingen av de tilgjengelige nettverkene var strukturert for bruk av en båndbreddemegler, introduserer og benytter simulatoren en virtuell båndbreddemegler. En virtuell båndbreddemegler forenkler nettverkstopologien ved å fjerne linker og resurser utelukkende forbeholdt kontrolltrafikk, slik at gjenværende resurser uforkortet kan benyttes til ruting. Bruk av virtuell

[25] Mangelfulle kommentarer i Javadoc-ene kan ha blitt rettet opp i siste versjon av JUNG som er en ekstensiv oppgradering av biblioteket.

[26] En CD-ROM som inneholder kildekoden er vedlagt papirutgaven av denne oppgaven, forøvrig finnes også kildekoden tilgjengelig på nett, ref. 1.

båndbreddemegler innebærer en abstraksjon som ikke påvirker utfallet av eventuelle simuleringer, fordi kontrolltrafikken i utgangspunktet ikke benytter eller på noen måte påvirker resursene tilgjengelig for reservering. Fordelen med dette er at samtlige nettverkstopologier kan benyttes av SAK til testing, samtidig betyr det også at måltallene som benyttes og produseres av simulatoren er relative til resursene forbeholdt reservering.

Under simulering vil nye resursallokeringsanmodninger kontinuerlig ankomme BB, som så er ansvarlig for kontraktforhandlingene om en ny SLA ved å egenhendig foreta en helhetsvurdering av nettverksresursene, samt eksisterende avtaler før avtalen inngås. Ettersom SAK fokuserer på reserveringsprosessen, vil samtlige kontraktforhandlinger bli godkjent av båndbreddemegler så sant nettverket har tilstrekkelig med kapasitet. Nettverket har tilstrekkelig med kapasitet så lenge ingen komponenter i nettverket har reservert mer enn 100% av resursene forbeholdt resursreservering etter at den nye reserveringen er tildelt sine resurser.

Antall reserveringsanmodninger som genereres er satt till 200. Dette tallet er valgt etter en rekke prøvesimuleringer, og er stort sett det antall anmodninger som var nødvendig for å fullstendig mette forsøkstopologiene. I utgangspunktet ble det sendt nye anmodninger helt til både SRT og MRC hadde avslått totalt 10 reserveringer, men en stor variasjon i antall reservasjoner gjorde det vanskelig å sammenligne data fra forskjellige kjøringer. Hver enkel anmodning ber om reservering av resurser til ruting av data mellom to noder. Etter at en topologi er både lest inn og isolert, er det mulig å bestemme antall unike nodepar som kan utstede resursanmodninger. Hver anmodning som tildeles resurser følger den korteste stien gjennom nettverket. SAK er lagd slik at selvom det foreligger flere alternative stier med samme lengde, vil kun ett og alltid det samme alternativet bli valgt til ruting. Under visse forhold fører dette til reserveringsavslag selvom et likeverdig alternativ til den korteste sti fortsatt har tilstrekkelig med resurser. Vi drøfter i kapittel 6 hvordan MRC bedre kan håndtere en slik situasjon.

4.2.1 Simulatorskallet

Simulatoren benytter et kommandolinjebasert grensesnitt, som innebærer at simulatoren utelukkende vil kontrolleres ved hjelp av tekstbaserte kommandoer. Ettersom beskrivelsen av hver enkelt funksjon innad i programmet er forholdsvis kort, følger en mer detaljert beskrivelse under. *Påkrevde* parametere vises med *<spissparentes>* og *valgfrie* parametere vises med *[hakeparentes]*. Legg også merke til at vi i teksten under oppgir den alternativ versjonen for alle kommandoer som har en slik, og at primær versjonen er gitt i parentes. Den alternative formen blir ikke vist av SAK sin innebygde hjelpefunksjon.

1. `dir [katalog]` (ls)

   Hjelpefunksjon som viser alle filer i oppgitte katalog, og i gjeldende katalog hvis inger er oppgitt. Kataloger og filer vises adskilt, og gitt at SAK tolker filen som en topologifil, vises antall noder og antall linker i listen. Tekstbaserte filer med nøyaktig to tegn eller ord på en eller flere linjer blir tolket som en topologfil. Vi valgte denne løsningen fremfor å tvinge brukeren til å bruke en bestemt filforlengelse.

2. `l <topology>` (load)

   Forut for en simulering er det nødvendig å opprette en topologi. Dette gjøres med denne kommandoen. Filennavnet som oppgis kan inkludere en relativ eller absolutt mappesti. Den innleste topologien vises i et eget vindu, og det er anbefalt at nodene organiseres forut for en eventuell konfigurering.

3. `go` (config)

   Hvis en gyldig topologi har blitt lest inn kan brukeren starte konfigurasjonsprosessen med denne kommandoen. Hvis prosessen kjøres uforsinket vil den ikke avslutte før topologien er isolert, eller maksimalt antall konfigurasjoner er nådd. Hvis ønskelig, kan denne prosessen forsinkes ved hjelp av `delay`-kommandoen. I så fall vil bruker selv måtte iverksette hvert enkelt isoleringsforsøk manuelt.

4. `s [count]` (set)

   Bruk `set` til å endre antall nodepar som utsteder resursreserveringer. Hvis ingen verdi settes vil antall nodepar være ubegrenset.

5. `r [count]` (route)

   Etter at komponentene i topologien er isolert kan resursallokeringsfasen iverksettes med denne kommandoen. En kjøring avslutter ikke før 200 resursallokeringer er utstedt. Parameteren setter hvor mange kjøringer som skal startes før den avslutter, hvis det ikke oppgis et tall vil kun en kjøring bli gjennomført.

6. `run <folder> <int>`

   Dette er i prinsippet en oppsamlingsfunksjon som sekvensielt utfører følgende kommandoer for hver enkelt topologi inneholdt i `folder`: leser inn topologien, isolerer topologien og metter topologien `int` ganger. På denne måten kan brukeren spare mye tid når et stort antall topologier skal anvendes av SAK.

7. `slow` (delay)

   Brukeren kan ved hjelp av denne kommandoen forsinke SAK slik at det blir lettere å følge med på nøyaktig hvordan MRC traverserer topologien. Kommandoen kan brukes forut for konfigureringsfasen eller den etterfølgende resursallokeringsfasen. Ved bruk skrur kommandoen vekselvis av og på forsinkelsen av SAK.

8. `h` (history)

   Hjelpefunksjon som viser brukeren de ti siste gyldige kommandoene som er brukt. Den siste kommandoen gis alltid kommandonummer 0 (null), som betyr at kommandoene listes i omvendt kronologisk rekkefølge og at historiekommandoen selv alltid gis id 0.

   For å bruke en historiekommando brukes prefikset *!* (et utropstegn) direkte foran historieidentifikatoren uten noe mellomrom.

9. `?` (help)

   Hjelpefunksjon som gir en kort beskrivelse av hver enkelt kommando som støttes av simulatoren. Også denne hjelpefunksjonen viser påkrevde parametere med <spissparentes> og valgfrie parametere med [hakeparentes].

10. `x` (exit)

    Denne kommandoen lukker eventuelt åpne grafvinduer og avslutter simulatoren.

Organisering av noder gjøres ved å trekke og slippe nodene der de ønskes plassert, dette er mulig i samtlige vinduer. Hvis det gjøres i turvinduet før konfigurasjonsfasen iverksettes, vil samtlige genererte konfigurasjoner bli tildelt samme nodeplassering.

Det er mulig å forsinke isoleringsprosessen slik at det blir lettere å følge med på hva som skjer. Forsinket visning skrus av og på med `Delay` kommandoen, og må selvsagt brukes forut for den eventuelle fasen som ønsket forsinket. Forsinkelsen forblir aktiv inntil den deaktiviseres.

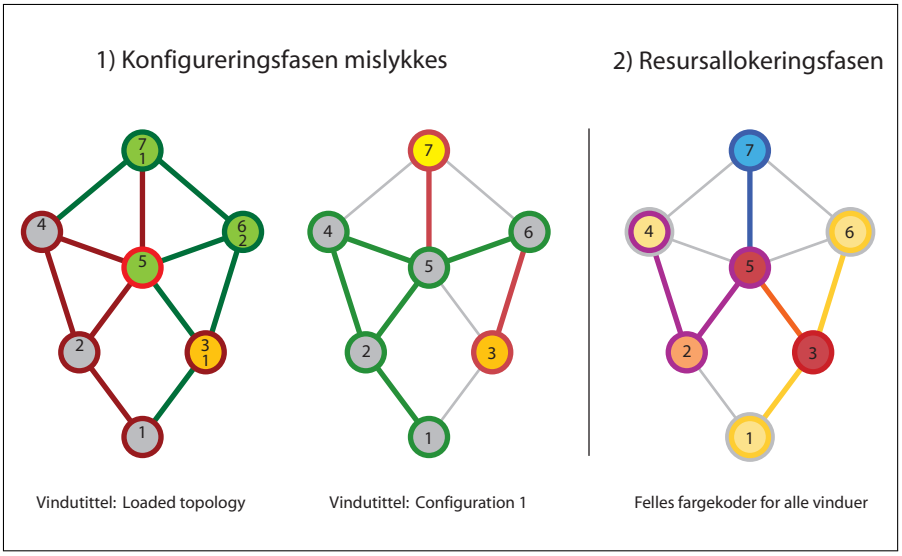
SAK skal kunne håndtere kommandoer fra bruker, initialisering av simuleringer basert på inndata fra enten en enkelt fil eller fra flere i form av en katalog, og gi tilbakemeldinger til bruker under kjøring.

Grafer, rådata og andre produkter generert under simulering, produseres av de respektive objektene, og blir ikke behandlet noe videre av simulatoren.

4.2.2 Visualisering av MRC

SAK har to forskjellige visualiseringsmodi, en for konfigurasjonsfasen og en for den etterfølgende resursallokeringsfasen. Begge fasene bruker farger, men ettersom hensikten med den første fasen er isolering og den neste er å mette nettverket, brukes fargene forskjellig. Dette kan i utgangspunktet virke forvirrende, men det fungerer godt i praksis. I utgangspunktet er det to typer vinduer som vises under bruk av SAK, til venstre i figur 7 vises begge. Hver enkel konfigurasjon vises i et eget vindu, og grafen i midten viser fargebruken til et vindu av

denne typen. Grafen til venstre viser fargene til den andre vindustypen; som i denne fasen refereres til som turtabellen. I utgangspunktet viser dette vinduet den innleste topologien, hvilket er årsaken til vindustittelen. Helt til høyre er et eksempel på fargebruken under reservasjonprosessen, som er felles for begge vindustypene. Vi forklarer hver enkel fase for seg i de etterfølgende avsnittene.



Figur 7: Farger benyttes av SAK til å visualisere progresjon og gjeldende tilstand i nettverket ved bruk av MRC. Grafene til venstre illustrerer et mislykket isoleringsforsøk med to konfigurasjoner, og viser turtabellen (t.v.) og den ene av de to brukte konfigurasjonene (midten).

Som vi vet fra 2.1 går MRC gjennom to faser, under den første[27] fasen brukes fargene til å vise hvilke komponenter som er isolert, hvilke linker som er begrenset og hvilke komponenter som ruter som normalt. Isolerte noder gis et rødt omriss i den konfigurasjonen der de er isolert, rødt ble valgt fordi de stopper, eller aldri videresender datatrafikk. Alle linkene som brukes til datatransport og er tilknyttet denne noden, vil i den samme konfigurasjonen være begrenset. Ettersom en isolert node forutsetter en eller flere begrensete linker, blir disse gitt den samme rødfargen som den isolerte noden selv. Slik kan man lett se hvilke linker som kommuniserer med en isolert node i en gitt konfigurasjon. Isolerte linker som aldri brukes til kommunikasjon gis derimot en gråfarge fordi dette

[27]Konfigurasjonsfasen er første fasen som MRC går gjennom, ref. kapittel 2.1 for detaljer.

gjerne assosieres med deaktiverte komponenter.

Turtabellen samler inn informasjonen fra samtlige konfigurasjonsvinduer, og viser hvordan isoleringsprosessen traverserer grafen og hvilken konfigurasjon som isolerer hver enkelt node. Dette tallet kan sees *under* nodeidentifikatoren, størrelsen på tallet er økt av hensyn til lesbarhet. Node 7 og 6 i figuren er begge isolert, sistnevnte er isolert i konfigurasjon 2, hvilket er grunnen til at noden vises som normal i konfigurasjon 1 ved siden av. I midten av turtabellen vises node 5 med et intenst rødt omriss, hvilket betyr at noden ikke lot seg isolere med gjeldende antall konfigurasjoner. Node 3 er oransje i fyllet, som betyr at dette er noden som for øyeblikket forsøkers isolert. Kombineres turtabellen med konfigurasjonen ved siden av, er det fullt mulig å analyser seg til nøyaktig hvordan grafen ble isolert før den feilet. Vi overlater dette til leseren som en øvelse.

Konfigurasjonen som forsøker å isolere en node vil fylle noden med den samme oransje fargen som brukes i turtabellen. Noder som derimot rekursivt isoleres som en følge av at startnoden (den oransje) forsøkes isolert, vises med et gult fyll. På den måten kan vi følge turen til det rekursive kallet gjennom topologien. Som nevnt i kapittel 4.2.1 er det mulig å forsinke isoleringsprosessen slik at denne traverseringen lettere kan følges.

Den neste fasen til MRC er resursallokeringsfasen av datatrafikk. Målet er å vise metningsgraden til komponentene i nettverket, alle vinduer viser og bruker derfor i denne fasen felles farger. Hver enkel konfigurasjon viser kun *sin andel* av den totale trafikken som er reservert i nettverket ved hjelp av MRC, og vil derfor hver for seg gi et mangelfullt bilde av tilstanden i nettet. Turtabellen derimot, viser den totale metningsgraden til SRT. Ubrukte noder og linker vises i blått, deretter går fargen fra gult til oransje, og til slutt til rødt etterhvert som komponentene mettes. Hvis lilla blir brukt i topologien, betyr dette at komponenten er overmettet. SAK vil derimot, slik koden foreligger, aldri tillate reservering av mer enn 100% av resursene i en vilkårlig komponent. Isolerte noder vises i konfigurasjonene med et grått omriss, og isolerte linker er gitt den

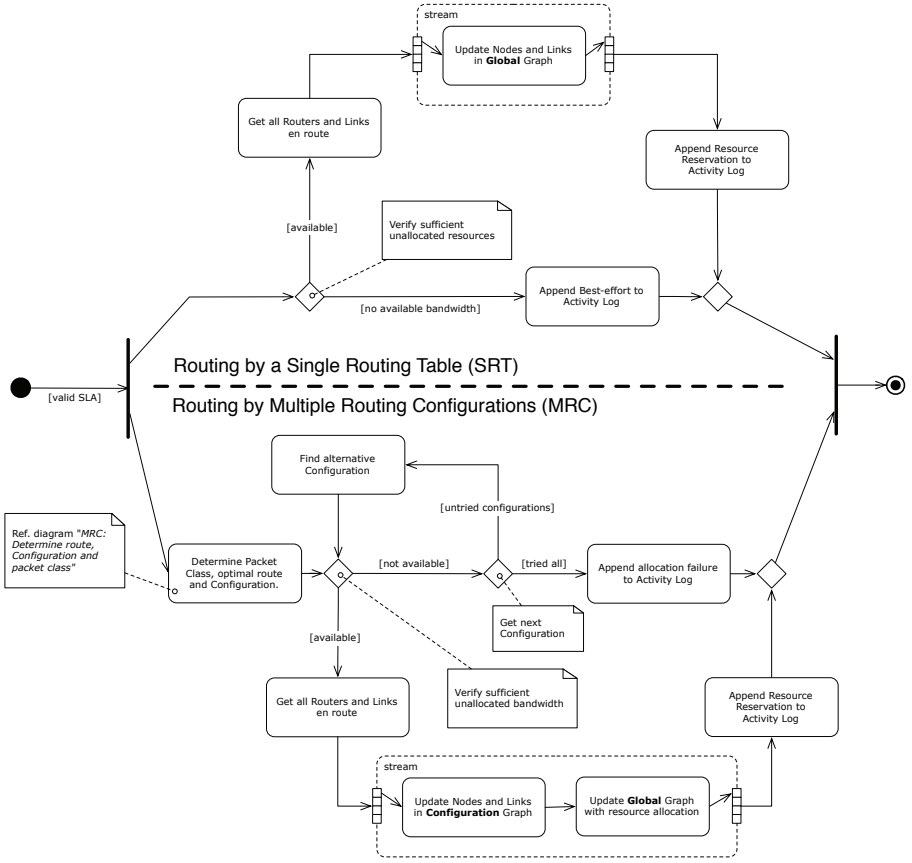
samme gråfargen. Det er med andre ord ikke lenger mulig å se hvilke linker som er begrenset i hver konfigurasjon, men fordi dette i denne fasen heller ikke er vårt fokus, blir denne tilleggsinformasjonen droppet av hensyn til oversiktlighet.

Nettsiden "Network Analysis And Visualization" [44] gir en god oversikt over tilgjengelige netverksanalyse- og -visualiseringsverktøy, også JUNG er oppført i denne listen.

4.2.3 Forretningslogikk

Topologien leses inn av SAK fra en tekstbasert fil, som samtidig oppretter den nødvendige objektstrukturen som benyttes av algoritmen. Fordi visualiseringen skjer ved hjelp av JUNG [37], blir datastrukturen som dette biblioteket benytter også brukt av SAK. Innleste noder og linker lagres som `Vertex` og `Edge` objekter i et globalt `Graph` objekt lagret i `Simulation`, som også kopieres og benyttes av `Configuration`-objektene som benyttes av algoritmen til MRC. Det globale grafobjektet opprettholder oversikten for alle reserverte resurser i topologien, og hver konfigurasjon lagrer i sine respektive grafer de resursene som håndteres av hver enkelt. Settet av vektjusterte komponenter i en gitt konfigurasjon, er med andre ord komponenter i den tilhørende grafkopien. Vekt og ledige resurser er inneholdt i hver node og link i grafobjektet, i form av et `UserDatum` objekt. Ved allokering av resurser, endres verdien til resursobjektet, slik at det til en hver tid indikerer korrekt andel reserverte resurser.

Samtlige noder i topologien er utstyrt med et sett av unike avstandstabeller. Tabellen er en hashbasert (eng.: hash map) slik at oppslag kan gjøres direkte, med en tabell for hver enkel konfigurasjon. I tabellen lagres en referanse til samtlige noder med en tilhørende avstand, som alltid er basert på KKS. Avstanden til noden selv er satt til null. Strukturelt er avstandstabellen med andre ord rent identisk med den ordinære avstandstabellen tilknyttet SRT, og den genereres ved hjelp av Dijkstras korteste sti algoritme. Her må det tilføyes at SAK implementerer en versjon av Dijkstras som er tilpasset både MRC og datastrukturen vi benytter i SAK.



Figur 8: Diagrammet viser forretningslogikken tilknyttet SRT og MRC; og er ment å gi en god oversikt over de forskjellige prosessene og testene som utføres i forbindelse med identifisering av TKS og KKS. Diagrammet er et UML2.0 aktivietsdiagram, og leses ved å følge pilene til hver enkelt aktivitet, fra venstre mot høyre.

Figur 8 viser aktivitetsdiagrammet for ruting etter både SRT og MRC. Hensikten med diagrammet er å gi en oversikt over programflyten, slik at følgende tekniske gjennomgang av forretningslogikken blir lettere å følge. Bruk også figur 9 til å identifisere de forskjellige klassene og hvilke pakker de tilhører.

Hver kommando som er tilgjengelig i SAK er basert på en `Command`, og tar i bruk ingen, en eller flere `Parameter`-objekter. Ved opprettelsen av en kommando spesifiseres det om hver enkelt parameter er påkrevd eller valgfri, hvilket vil påvirke hvordan kommandoen skrives når brukeren ber om `hjelp`.

Når vi under kjøring av SAK genererer nye resursanmodninger, kommer disse ikke "utenfra" slik de gjerne gjør i den virkelige verden. De produseres i stedet direkte av SAK og den aktiviserte utgaven av et ResourceReservation direkte, resursanmodningen er derfor alltid gyldig. Det utstedes med andre ord ikke noen SLA, i stedet er hver enkel resursanmodning i utgangspunktet gyldig. Det finnes flere forskjellige trafikkprodusenter i SAK, men kun en er aktivsert av gangen. Dette kan endres med kommandoen `set [count]`.

Hver resursanmodning blir ferdigbehandlet før en ny genereres. Det er kun visualiseringen som benytter en egen `Thread` i SAK. Dette betyr at vi unngår synkroniseringsproblemer, og potensielle oppdateringsanomalier, men det betyr også at visualiseringen til tider kan henge litt etter i forhold til faktisk progresjon. Dette er ikke et problem i praksis, visualiseringen innhenter simuleringen etter hvert, den ferdigisolerte eller fullmettede topologien blir alltid visuelt korrekt til slutt.

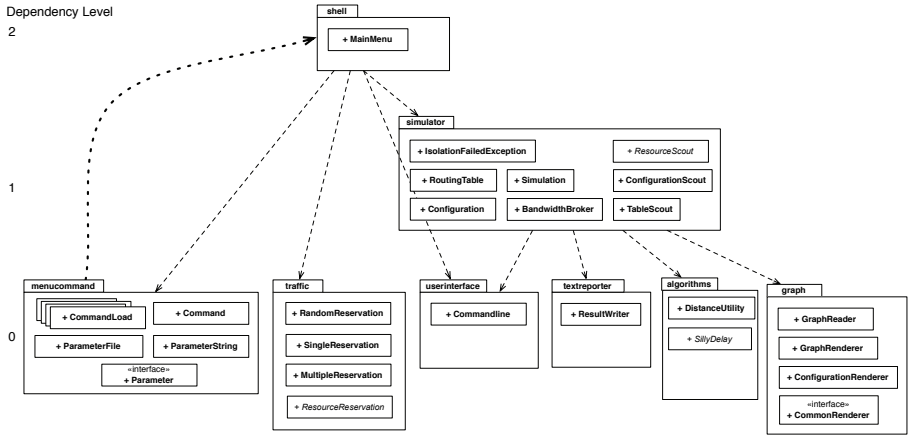
Aktivitetsloggen som både avslåtte og aksepterte resursallokeringer tilføyes, lagres i `./SAKoutput/data/`, og er en tab-formatert tekstfil. Etter at topologien er ferdig mettet, blir en rapport tilføyd topologiens ytelsesrapport i samme katalog. Disse filene slutter på `_summary.txt`. Begge disse rapportene etableres av `ResultWriter`.

Når SAK skal bekrefte at tilstrekkelig med resurser er tilgjengelig, som første test under resursallokeringsfasen, benyttes en `ResourceScout`. Både SRT og MRC har hver sin scout, `TableScout` og `ConfigurationScout`, med felles egenskaper arvet fra `ResourseScout`. Disse gjennomgår henholdsvis rutetabellen, som er det globale `Graph`-objektet, og hver enkel konfigurasjons kopi av dette objektet. Fordi resursene som allokeres tross alt er felles for alle konfigurasjoner, lagres allokerte resurser både i den globale, så vel som den lokale `Graph`-kopien.

Hvis en allokeringsanmodnig blir avslått, utstedes en `IsolationFailedException`. Slik SAK er implementert vil det være opp til bruker om en nytt isoleringsforsøk skal iverksettes kun hvis SAK kjøres i saktemodus. Under vanlig eller hurtigkjøring, vil SAK selv iverksette et nytt isolasjonsforsøk - helt til maksimumsgrensen

for antall tillatte konfigurasjoner nås. Etter dette stopper SAK med en feilmelding.

`SillyDelay` er en klasse som kun er inkludert for å gi brukeren muligheten til å forsinke isolerings- og resursallokeringsprosessen til SAK. Det er sikkert mulig å implementere dette på en smartere måte enn vi gjorde, derav navnet.



Figur 9: Her ser du klassene som utgjør SAK og deres avhengighetsnivå (t.v.). Målet her er å holde alle koblinger rettet mot et lavere avhengighetsnivå, det er derimot en stygg dobbelkobling som fører til en unødig tett binding mellom menykommandoene og hovedmenyen. Koblingen er et resultat av gjenbruk av kode, og kan i prinsippet lett fjernes.

5 Eksperimenter

Simulering av Internett byr på en rekke utfordringer [34], blant annet er nettrafikken ikke jevnt distribuert i et nettverk, noen linker brukes vesentlig mer enn andre [3]; et fenomen tilfeldig valgt trafikk ikke tar hensyn til. Hvis vi derimot tenker oss at nettverket har en basistrafikk, som tar en andel av de reserverbare resursene, så kan vi tenke oss at det på *toppen av dette* vil være ledige resurser som likevel kan allokeres uhindret av eksisterende trafikk i nettet. Til denne oppgaven har vi derfor tenkt oss av vi har abstrahert bort den allerede eksisterende trafikken, for så å fokusere utelukkende på resursene i nettverket som er forbeholdt reservering. Men fordi vi ikke kan utelukke at reservert trafikk også er ujevnt fordelt i nettverket, vil vi i tillegg til et tilfeldig valgt trafikkmønster, også utføre et eksperiment der vi begrenser antall nodepar som utsteder en reservasjonsanmodning. Resultatet herfra vil vi deretter sammenligne med resultatene fra øvrige eksperimenter med ubegrenset tilfeldig trafikk.

Som nevnt i kapittel 2.1 på side 17, vil MRC i prinsippet for hver enkel kjøring ende opp med en konfigurasjonløsning bestående av et varierende antall konfigurasjoner av varierende tetthet. Dette er en faktor som ikke får noen betydelige konsekvenser for nettverk som implementerer og bruker MRC til feilhåndtering, slik mekanismen i utgangspunktet var ment å bli brukt. Hensikten med MRC i en slik sammenheng er kun å unngå komponenten som har feilet, og det vil konfigurasjonen klare uavhengig av hvordan øvrige komponenter er isolert i den samme konfigurasjonen. Til *feilhåndtering* kan det faktisk være et mål i seg selv å isolere topologien så tett som mulig, ettersom det vil resultere i færre konfigurasjoner, og dermed et lavere minnebehov og mindre data å distribuere ved initialisering av nettverket. En tett isolert topologi er derimot ikke nødvendigvis hensiktsmessig hvis MRC implementeres og benyttes til *resursallokering*. For en tett isolert topologi burde vi intuitivt kunne anta at konsekvensen av MRC blir lettere å observere, ettersom det gjerne også medfører færre likeverdige stier og derfor større forskjeller fra ruting etter SRT som kun tilbyr TKS.

Vi mener MRC og en *løst isolert topologi* burde kunne erstatte SRT fullstendig når målet er å optimalisere resursallokeringskapasiteten i et nettverk. Hvis vi klarer å bruke MRC på en måte som ikke betyr lengre stier, burde det være mulig å øke antall resursallokeringer i et nettverk kun ved hjelp av denne mekanismen. Sammenlignet med SRT er dette mulig fordi MRC i en løst isolert topologi øker antallet likeverdige stier tilgjengelig for hver enkel resursallokering. Med en løst isolert topologi får vi dermed det beste fra begge verdener; trafikken følger TKS så lenge som mulig og kun når det oppstår et behov for å omgå en *mettet komponent*, benyttes den tidvis mer kostbare KKS. For å se hvilken innvirkning isoleringstettheten har på allokeringskapasiteten, har vi utført et eget eksperiment der vi sammenligner konfigurasjonsløsninger med henholdsvis 3, 4, 5 og 6 konfigurasjoner generert til to forskjellige topologier.

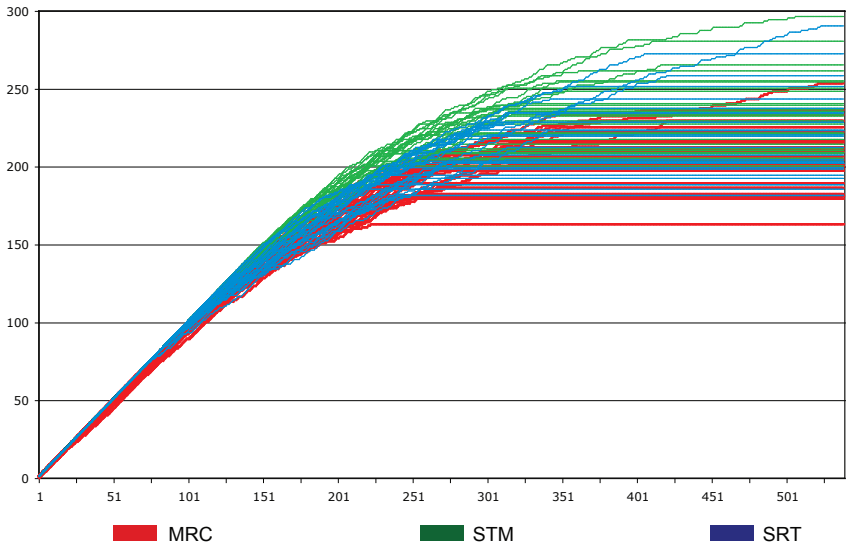
For å lettere kunne se hvor godt MRC yter, har vi i tillegg satt opp teknologikombinasjonen STM. STM står for "SRT Then MRC" og er simpelthen SRT kombinert med MRC: så lenge reservasjonen blir godkjent benyttes SRT, men i det reservasjonsanmodningen ikke blir innfridd med SRT skifter STM til MRC. Hvis bruk av MRC også fører til avslag, vil STM gi avslag på reservasjonsanmodningen.

5.1 Sammenligne SRT, MRC og STM

For å studere nærmere hvordan MRC påvirker allokeringskapasiteten utfører vi en rekke eksperimenter der vi sammenligner følgende simuleringsresultater:

1. SRT kjørt alene.
2. MRC kjørt alene.
3. SRT med støtte av MRC når resursallokeringen avslås. Resultater med STM vil i all hovedsak kun være veiledende, og benyttes som en referanse for resultatene fra både SRT og MRC.

I utgangspunktet er SAK satt opp til å velge et ubegrenset antall tilfeldig valgte nodepar fra gjeldende topologi, som deretter utsteder resursallokerings-



Figur 10: Slik ser det ut når nettverket mettes 30 ganger, hver farge er forbeholdt en teknologi. Både MRC og STM godkjente rundt 510 anmodninger på det meste (øverst til høyre). Figuren viser allokeringsevnen til SRT, MRC og STM for små reservasjoner (5% av resursene). Horisontalt ligger antall resursanmodninger og antall godkjente anmodninger kan leses av vertikalt. Grafen er et resultat av 96 forskjellige konfigurasjonsløsninger produsert under kjøring av SAK, 32 tilfeldig valgte syntetiske topologier ble mettet 30 ganger med hver teknologi.

anmodninger med en forhåndsbestemt størrelse. Hvert eksperiment blir kjørt to ganger, en gang for resursanmodninger som ber om 5% av de totale resursene (mus), deretter 30% av resursene (elefanter). Vi mener med andre ord at det er rimelig å anta at ingen SLA vil inngås for resurser under 5% av de resursene forbeholdt reservering, og samtidig at ingen avtale vil inngås som binder mer enn 30% av resursene til en enkel reservasjon.

SAK genererer nye reservasjoner helt til både SRT og MRC uavbrutt avslår 13 anmodninger. Tallet 13 ble valgt etter en rekke gjennomkjøringer med den største topologien vi skal benytte til simuleringen, og var det tallet som førte til en lettleselig graf og et *rimelig* antall resursallokeringsanmodninger. Rimelig i denne sammenheng innebærer at vi med dette tallet oppnådde en tilfredsstillende metningsgrad som tydelig kunne leses fra grafene vi produserte. Utflatningen av grafen kan sees i øverste høyre hjørne i figur 16. Det ble også forsøkt med et

fast antall reservasjonsanmodninger, men dette førte enten til at mindre nettverk mottok unødvendig mange RAR, eller at de større nettverkene ikke ble fullstendig mettet.

De 749 syntetiske topologiene vi benytter er generert ved hjelp av BRITE [28], som er et verktøy lagd for generering av representative Internett topologier. Programmet er ikke lenger i utvikling, men kan fortsatt lastes ned fra nettsiden. Topologiene ble generert med standardinnstillingene for Waxman modellen, med 2 og 3 ganger så mange linker i forhold til nodeantallet på 16, 32, 64 og 128. I tillegg til disse syntetiske topologiene, har vi også benyttet et sett bestående av 36 reelle topologier tilgjengeliggjort av Rocketfuel [39]. Gitt at størrelsen på en graf bestemmes av antall linker, består den minste topologien vi simulerer av 3 og den største av 476 linker (51 noder).

Resultatet fra en kjøring ender i to forskjellige filtyper; en fil inneholder informasjon om en enkelt kjøring. I denne kan vi se nøyaktig hvilke RAR som blir godkjent, og hvilke som blir avslått. Filen inneholder også identifikatoren til både ingress- og egressnoden, samt en anmodningsteller slik at filen blir lettere å både navigere og å lese. Sistnevnte teller opp for hver anmodning som mottas og behandles, og de tre etterfølgende kolonnene teller opp (les: økes) kun hvis henholdsvis SRT, MRC eller STM godkjenner RAR-en. Den andre filtypen er en oppsummeringsfil. En topologi som har blitt brukt kan ha flere slike, fordi det lagres en fil for alle løsninger med et bestemt antall konfigurasjoner. Slik samles alle isoleringsløsninger med samme isoleringstetthet i en og samme fil[28].

For statistikk der vi behandlet alle konfigurasjonsløsninger likt, var det nødvendig å samle data fra samtlige konfigurasjonsløsninger for å få et komplett bilde av ytelsen til MRC. For å samle statistikken lagde vi et PHP-skript som leste gjennom samtlige filer og lagret det endelige resultatet i en oversiktsfil som vi deretter behandlet videre. Vi har illustrert hvordan data fra 30 kjøringer av én enkel topologi ser ut i figur 16. Figuren viser SRT, MRC og STM benyt-

[28]For eksempel kan en topologi etter 10 simuleringer ende opp med 3 løsninger med 3 konfigurasjoner, 6 med 4 konfigurasjoner og 1 løsning med 5 konfigurasjoner. Dermed blir det lagd 3 oppsummeringsfiler, og f.eks. den førstnevnte oppsummeringsfilen vil da inneholde tre allokeringsresultater - et resultat for hver kjøring med dette antall konfigurasjoner.

tet til reservering av små resursanmodninger, og samtlige av de tilsammen 30 kjøringene av `T2-128-20.top`. Snittverdien for de forskjellige eksperimentene var henholdsvis 216,5 godkjente anmodninger for SRT, MRC godkjente 204,4 stykker og STM godkjente 232,0 anmodninger.

Totalt vil de 36 forskjellige reelle nettverkstopologier og 749 syntetisk genererte topologiene bli brukt i diverse kombinasjoner i følgende eksperimenter:

1. **Oversiktssimulering**

   Som et utgangspunkt skal vi foreta en metning av samtlige topologier. Målet med eksperimentet er å gi oss et utgangspunkt for videre arbeid. Vi simulerer både store og små reservasjonsanmodninger.

2. **Topologiens innvirkning på resultatet**

   Vi ser nærmere på hvordan strukturen i topologien påvirker resultatene, for å finne ut om MRC er mer eller mindre sensitiv til utformingen enn SRT.

3. **Simulering med et representativt utvalg**

   Basert på resultatene i det foregående eksperimentet, velger vi ut et representativt utvalg topologier og foretar et stort antall metninger.

4. **Løst og tett isolert topologi**

   Vårt siste eksperiment søker å finne svar på hvorvidt en topologi har best allokeringskapasitet når den er løst eller tett isolert.

Det første eksperimentet ble utført før de øvrige eksperimentene ble definert, og grafen i figur 11 viser resultatet; samtlige av de 785 topologiene er isolert og mettet 10 ganger. Forutsatt at du ser grafen i farger, kan en tydelig grønnfarge sees i toppen og minst like tydelig rødfarge i bunn. Grafen er ikke egentlig særlig leservennlig, fordi STM vist i grønt er gjort halvt gjennomsiktig og lagt over den røde grafen som viser ytelsen til MRC. Grafen kan derfor ved første øyekast gi en illusjon av at MRC alltid ligger under SRT, og motsatt for STM. Men dette er altså ikke tilfelle. Studeres grafen nærmere, er det to forskjellige grønnfarger og

den mørkeste av disse indikerer at den røde grafen ligger under, og motsatt for den lysegrønne delen av grafen. Figuren viser alle tre resultater i en og samme graf fordi vi mener at dette gir et nyttig overblikk der STM, MRC og SRT lett kan sammelignes og vurderes.

Med dette som utgangspunkt, ønsket vi kort og godt å se nærmere på hva som er årsaken til denne forskjellen. Dette innebar å prøve å identifisere karakteristiske trekk ved de forskjellige topologiene som er brukt. For å studere dette videre sorterte vi topologiene etter filnavn, for på den måten å gruppere de forskjellige topologiene sammen. De syntetiske topologiene ble dermed holdt adskilt fra de reelle nettverkstopologiene, og i tillegg ble hver individuelle gruppe med syntetiske topologier holdt samlet. Navnbasert sortering betyr også at de syntetiske topologiene med 128 noder ble ordnet foran topologiene med 16 noder fordi filnavnet inneholder nodeantallet, og uten et ledende nulltall. Kosekvensen av dette kan tydelig sees i figur 11. En utskrift fra terminalvinduet med filrekkefølgen kan leses i vedlegg C. De resulterende grafene kan sees som 12.

Som du ser er grafen (fig. 12) delt opp i segmenter merket med hvite bokstaver. Merkingen vil gjøre det lettere å referere til hver enkelt av disse topologigruppene når vi i kap. 5.2 skal analysere resultatet videre.

De øvrige eksperimentene følger naturlig fra resultatet til disse to eksperimentene, noe vi klargjør under våre observasjoner og den etterfølgende analysen.

5.2 Observasjon og analyse

Vi tar utgangspunkt i oversiktgrafen, vist i figur 11, som inkluderer samtlige observasjoner fra alle simuleringene utført på hele topologiutvalget. Resultatet er sortert etter prosentandelen resursallokeringer godkjent med SRT. Som eksperimentet tilsier er grafen i hovedsak inkludert for gi en god oversikt, og er ikke tilstrekkelig detaljert til at den kan fungere som et godt analytisk utgangspunkt. Oversiktsgrafen gir oss en åpenbar indikasjon på at STM produserer det jevnt over beste resultatet. Det røde feltet som markerer allokeringsevnen til MRC fluktuerer voldsomt, og indikerer en tidvis svært høy allokeringsevne, men fordi

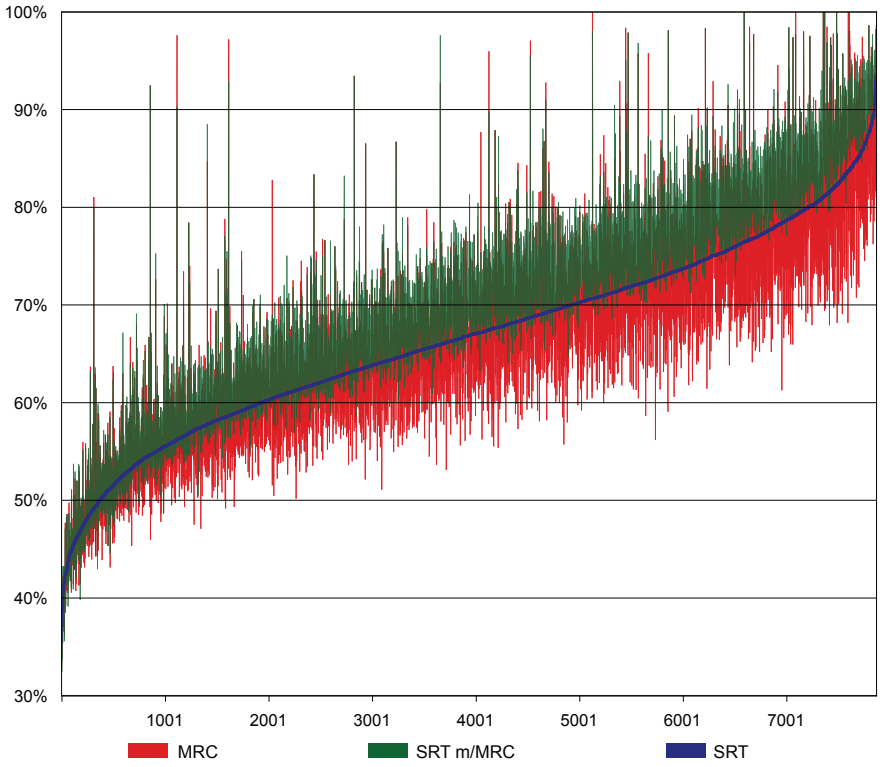
distribusjonen ligger tilnærmet symmetrisk om aksen til SRT, viser MRC også tilsvarende lav allokeringsevne. Grafen gjør oss klart oppmerksom på at MRC alene sannsynligvis ikke kan garantere beste resursallokeringsevne.

Grafen viser at kombinasjonen STM jevnt over bidrar til å øke reservasjonskapasiteten. Det er umulig å si hvorvidt MRC egentlig er bedre egnet for resursallokering enn SRT i disse grafene. Men det levnes ingen tvil om at MRC yter dårligere alene, enn sammen med SRT - en kombinasjon vi som nevnt har kalt STM.

Resultatet i figur 11 viser allokeringskapasiteten for små reservasjoner. Allokeringskapasiteten strekker seg fra tilnærmet 50% til 90%. Vi simulerte også store reservasjoner og resultatene fremstod nesten like entydig; grafen var flatere for de laveste allokeringsresultatene. STM gjorde det dårligere for små reservasjoner, men fremstår fortsatt jevnt over som bedre enn SRT (se figuren i vedlegg D). Ser vi nærmere på figur 11 så ser vi at STM løfter seg opp fra STR-grafen når nettverksutnyttelsen stiger over 60%. Dette er ikke like tydelig for de store reservasjonene i fig. 19, vi er ikke en gang sikre på om det kan sies at STM løfter seg i det hele tatt. Derimot ser vi at det er en stadig større spredning og fluktuering for STM etterhvert som utnyttelsen bedres.

For den samme topologien og store reservasjoner, 6 ganger så store som de små reservasjonene, var gjennomsnittet for antall godkjente anmodninger selvsagt mye lavere. SRT godkjente 33,87 anmodninger, MRC godkjente 32,37, og STM godkjente 36,1 anmodninger. Hvilket for denne kjøringen betyr at *STM* med *store reservasjoner* yter like bra som *SRT* for *små reservasjoner*. Dette er utifra en syntetisk generert topologi, hvilket vi snart vil vise har en signifikant påvirkning på allokeringskapasiteten.

Segmentet b venstre i den segmenterte oversiktsgrafen vist i figur 12 (s. 66) skiller seg tydelig ut fra segment *a* ved siden av, hvilket i seg selv er som forventet. Tross alt inneholder *a* segmentet en samling av alle typer topologier, og kan derfor betraktes som en svært komprimert versjon av resten av grafen. Det er også trivielt å se at grafen er langt fra kontinuerlig. Utifra dette mener vi det

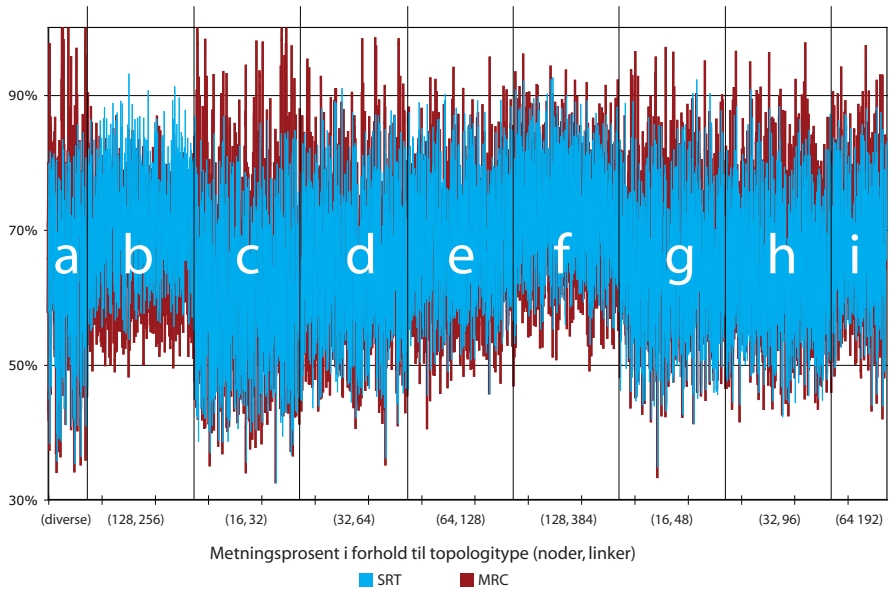


Figur 11: Grafen viser prosentandelen godkjente resursallokeringsanmodninger sortert etter SRT. MRC vises her i rødt, og SRT kombinert med MRC vises i gjennomsiktig grønt. Den mørke grønnfargen er resultatet av rødfargen under som skinner gjennom. Grafen viser 786 forskjellige topologier, 10 kjøringer av hver, antall observasjoner ligger horisontalt, allokeringskapasitet i prosent kan leses langs y-aksen.

er rimelig å fastslå at topologiens utforming får konsekvenser for allokeringskapasiteten. Det er derimot ikke så lett å fastslå hvordan MRC påvirkes i forhold til SRT, uten bruk av statistikk.

Det som nå gjenstår er å se nærmere på hvordan isoleringstetthet påvirker allokeringskapasiteten. Til dette valgte vi to topologier[29]; en syntetisk topologi og en reell nettverkstopologi med tilnærmet samme node og link antall, deretter kjørte vi isolering og metning av disse tilstrekkelig mange ganger til at det nødvendige tallmatrialet ble generert.

[29]Topologiene `7170.top` (reell) og `T3-16-43.top` (syntetisk) ble benyttet, med henholdsvis 18 noder og 60 links, og 16 noder og 48 links.



Figur 12: Grafen viser allokeringskapasiteten for topologiene når hver reservering ber om 5% av resursene (mus). Med unntak av gruppe `a` inneholder hvert segment merket med en bokstav resultatet fra allokeringsforsøk med syntetisk genererte topologier. Grafen er resultatet av 749 syntetiske og 36 reelle topologier kjørt 10 ganger hver. Node og link forholdet i hver enkelt gruppe kan leses i parentes horisontal under grafen, langs y-aksen vises allokeringskapasiteten.

Ved sammenligning av grafene som vises i figur 20 og 17, ser vi at mønsteret er det samme; dette gjelder også fig. 21 og 18. Utifra dette føler vi oss trygge på at disse to utvalgte topologiene er representative for ytelsen til både syntetiske og reelle topologier. Dermed er det interessant å se nærmere på variasjonen av allokeringskapasiteten, når vi sammeligner tette og løse isoleringsløsninger. Under dette eksperimentet produserte vi løsninger som krevde 3, 4, 5 og 6 konfigurasjoner for begge topologiene. Dog skal det nevnes at forholdet var tilnærmet 1:10000 for løsningene som ga 3 konfigurasjoner med den reelle nettverkstopologien. Dette skjer fordi MRC ikke direkte spesifiserer hvordan nettverket traverseres under isolering, hvilket betyr at SAK overlater dette til tilfeldigheter. Det endelige resultatet kan sees i tabell 3, med utvalgte representative grafer vist i figur 20 og 21.

5.3 Resultater

I forrige kapittel gikk vi gjennom en rekke observasjoner, og foretok en lett analyse av det inntrykket de forskjellige grafene ga oss. Nå skal vi foreta en statistisk gjennomgang av de samme simuleringsresultatene. Konklusjonen vi trekker i oppgavens neste og siste kapittel er i all hovedsak basert på de resultatene som presenteres her, det viser seg nemlig at vår statistiske analyse ikke helt sammenfaller med våre forventninger.

Vi tar til å begynne med tak i dataene brukt til å produsere fig. 11. Figur 15 og 14 viser begge et histogram for tettheten til allokeringskapasiteten til henholdsvis MRC og SRT, og som vi ser er de begge tilnærmet normalfordelte. Det kan også se ut som om MRC er noe høyreskjev, hvilket er positivt for MRC. I tabell 1 vises gjennomsnitt, standardavvik, og median for små reservasjoner. I tillegg har vi inkludert et avviksjustert gjennomsnitt, utregnet som i formel 2; for ordens skyld er målet et så høyt avviksjustert gjennomsnitt som mulig.

$$Avviksjustert\ gjennomsnitt = Gjennomsnitt/Standardavvik \tag{2}$$

Sammenligner vi gjennomsnittet i tabell 1, er snittet for MRC veldig likt for syntetiske og reelle topologier, derimot faller allokeringskapasiteten til både SRT og STM for reelle topologier sammenlignet med resultatene for de syntetiske. Standardavviket øker derimot for samtlige hvis vi sammenligner verdiene fra syntetiske og reelle nettverkstopologier.

Flytter vi blikket over til figur 12 og gjør samme utregninger for hvert enkelt segment med syntetiske topologier, får vi resultatene som vises i figur 13. I denne figuren skårer segment B[30] og F[31] høyest, hvilket også henger sammen med høyde (variasjon) og vertikal plassering (allokeringskapasitet) av segmentene.

Histogrammene til både SRT og MRC, figur 14 og 15, viser en tilnærmet normalfordelt allokeringskapasitet. selvom det kan se ut til at MRC er noe høyre-

[30] B: 9,53 (SRT), 8,65 (MRC) og 9,33 (STM)
[31] F: 9,92 (SRT), 8,33 (MRC) og 9,37 (STM)

	Syntetiske topologier			Reelle nettverkstopologier		
	SRT	**MRC**	*STM*	SRT	**MRC**	*STM*
Gj.snitt	66,70%	**66,52%**	*70,26%*	63,73%	**66,51%**	*66,46*
Std.avvik	9,48%	**10,28%**	*11,12%*	11,29%	**13,90%**	*13,51%*
Median	66,85%	**66,25%**	*70,35%*	64,28%	**67,10%**	*68,02%*
Av.Gj.snitt	7,038	**6,467**	*6,321*	5,645	**4,784**	*4,920*

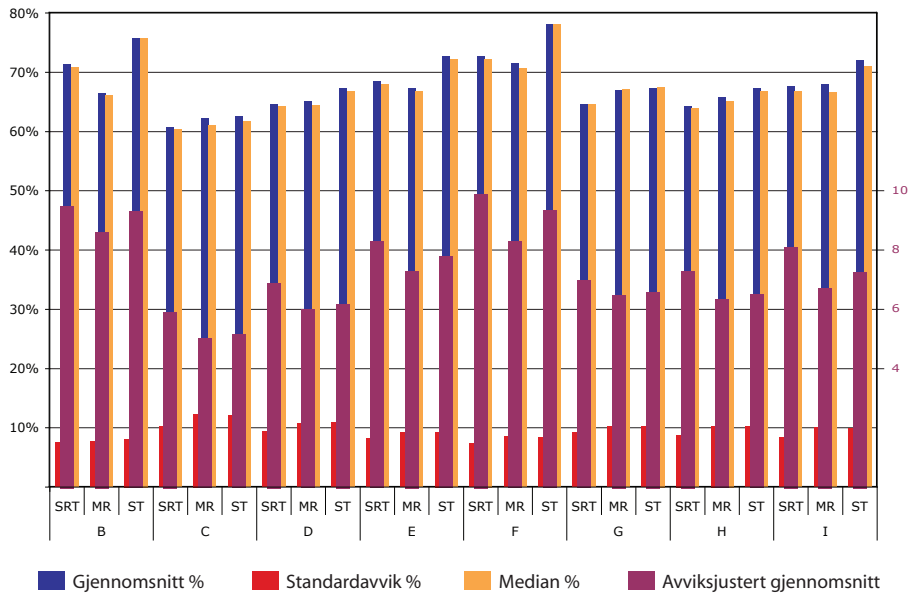
Tabell 1: Tabellen viser gjennomsnittlig allokeringskapasitet, standardavvik og median for mus fra 10 kjøringer av 749 syntetiske topologier (N=7490), og 36 reelle nettverkstopologier (N=360). Figurene 20 og 21, side 95, viser resultatene for *elefanter*.

skjev med en oftere forekomst av svært høy allokeringerkapasitet. Dette stemmer med våre forventninger fra figur 11.

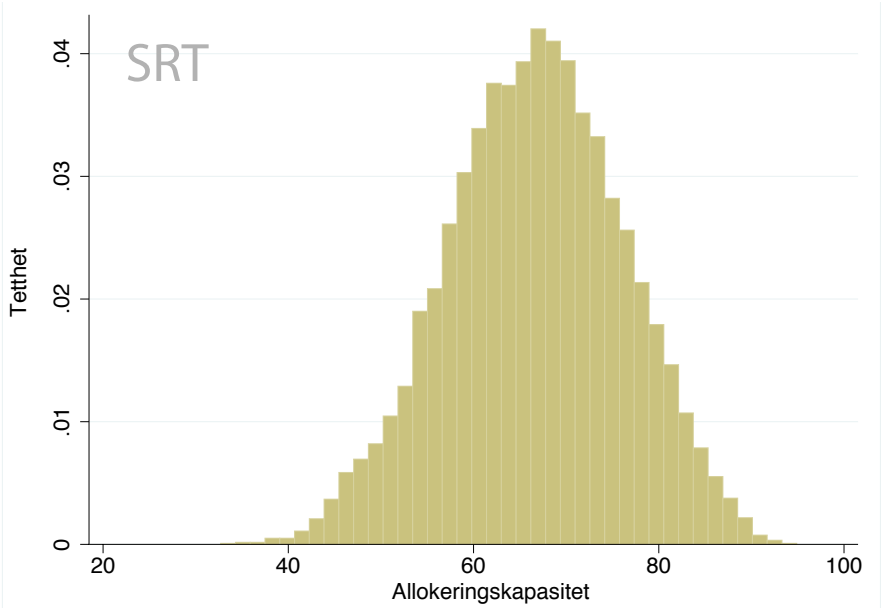
Boksplottet i figur 16 viser at STM har den høyeste median, men også at STM har et større standardavvik. Vi kan også se at MRC har flere høye uteliggere enn både SRT og STM, og i tillegg flere lave uteliggere. Med andre ord ser STM best ut hvis vi tar utgangspunkt i snittet alene, og SRT og MRC ser forholdsvis like ut, selvom avviket og uteliggerne til sistnevnte gjør teknologien tilsynelatende mindre attraktiv enn SRT.

	Syntetiske topologier		Reelle nettverkstopologier	
	Små reservasjoner	**Store**	Små reservasjoner	**Store**
SRT	68,59%	**42,50%**	64,13%	**33,50%**
MRC	67,70%	**42,07%**	66,73%	**34,16%**
STM	72,97%	**44,74%**	66,86%	**34,29%**

Tabell 2: Tabellen viser allokeringskapasiteten til syntetiske topologier og reelle nettverkstopologier; for både små og store reservasjonsanmodninger. Det ble tilfeldig valgt ut 32 syntetiske topologier (N=960), som deretter ble isolert og mettet sammen med de 36 reelle (N=1080) topologiene, begge gruppene ble mettet 30 ganger. Dette er således et oversiktsbilde av grafene som kan sees i figur 17 og 18.



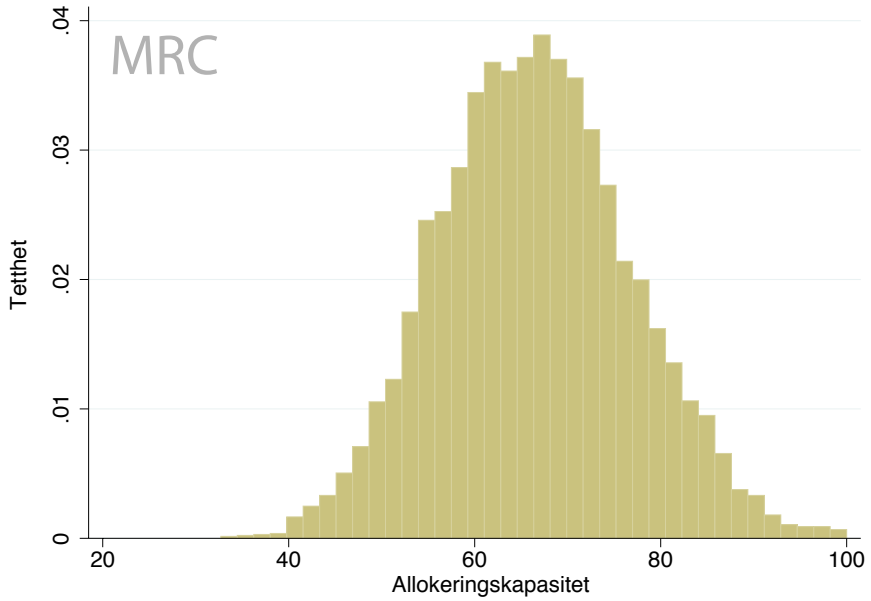
Figur 13: Figuren viser deskriptiv statistikk for hvert enkelt syntetiske segment vist i figur 12. Avviksjustert gjennomsnitt er er vist i lilla, og leses av på skalaen til *høyre side* av figuren. Samtlige topologier inneholdt i hvert segment er kjørt 10 ganger (antall topologier i hvert segment finner du som vedlegg C, side 91).



Figur 14: Histogram som viser distribusjonen til allokeringskapasiteten (horisontalt) i forhold til relativ tetthet (y-aksen). Dette er de samme data som er vist i tabell 1, selvom grafen kun dekker de syntetiske topologiene. Det var ingen betydelig forskjell mellom den syntetiske og reelle fordelingen, vi viser derfor kun dette histogrammet for SRT.

	Syntetiske topologier			Reelle nettverkstopologier		
	Gj.snitt	**Std.avvik**	*Av.gj.snitt*	Gj.snitt	**Std.avvik**	*Av.gj.snitt*
3	64,17%	**10,42%**	*6,16*	64,66%	**13,51%**	*4,79*
4	66,92%	**9,68%**	*6,91*	64,59%	**13,67%**	*4,73*
5	68,19%	**9,35%**	*7,30*	65,05%	**13,55%**	*4,80*
6	68,53%	**9,62%**	*7,12*	65,24%	**12,95%**	*5,04*

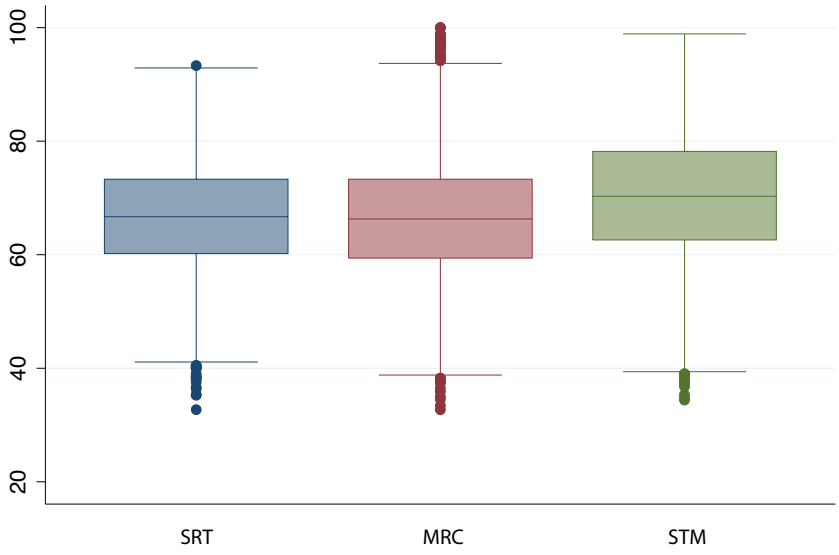
Tabell 3: Tabellen viser gjennomsnittlig allokeringskapasitet for mus i forhold til isoleringstetthet, gruppert etter topologitype og antall konfigurasjoner i hver enkelt isoleringsløsning produsert av SAK. Vi foretok et tilfeldig utvalg av 1100 resultater fra et stort antall kjøringer. Denne tabellen viser metningsprosenten for mus, vi har i tillegg vist to representative grafer for *elefanter* i figur 20 og 21, side 96.



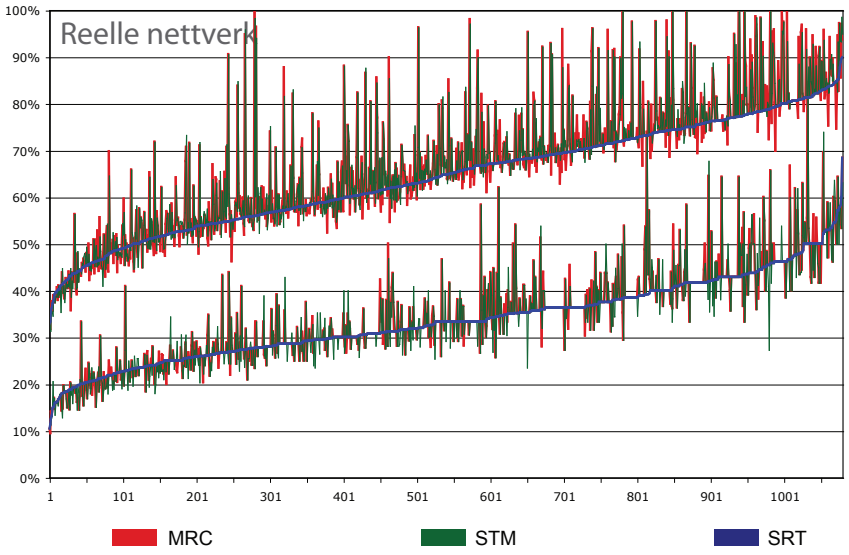
Figur 15: Histogram viser distribusjonen til allokeringskapasiteten (horitontalt) for MRC, basert på tetthet (y-aksen). Også denne grafen viser fordelingen kun for de 749 syntetiske topologiene, som hver er kjørt 10 ganger. Fordi det ikke var noen bemerkelsesverdig forskjell fra fordelingen til syntetiske og reelle, viser vi kun denne fordelingen av syntetiske topologier.

	Topologier		
	Alle topologier	*Syntetiske*	**Reelle**
N	7870	*7490*	**380**
U-statistikk	0,2625	*1,1462*	**-3,0256**
H_0: SRT = MRC	Ikke avvis	*Ikke avvis*	**Avvis**
H_0: SRT > MRC	-	-	**Avvis**
Konklusjon	SRT = MRC	*SRT = MRC*	**SRT < MRC**

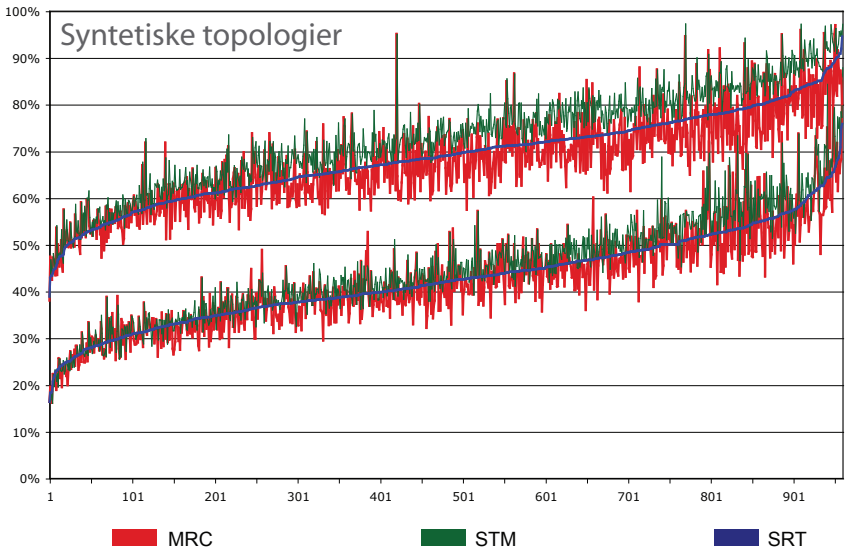
Tabell 4: Her vises resultatene av testene vi har utført for å finne ut hvorvidt ytelsesforskjellen til SRT og MRC er gyldig gitt en 5% signifikantsnivå. Basert på antall observasjoner, velger vi å benytte U-testen (ref. formel 1) til å teste hvorvidt $SRT = MRC$. Utifra testresultatet kan vi avvise hypotese H_0: $SRT = MRC$ for reelle topologier. Herfra tester vi hvorvidt SRT er bedre enn MRC i reelle nettverk, hvilket med 95% sikkerhet kan avvises, som betyr at MRC er bedre enn SRT i reelle topologier.



Figur 16: Figuren viser boksplottet for SRT og MRC, og små reservasjoner (5% av resursene). Grafen er resultatet av 749 syntetiske topologier kjørt 10 ganger hver. Boksen i midten er delt horisontalt ved medianen, med 25-prosentilet henholdsvis over og under medianen. Variasjonen vises med tynne linker utifra boksen, og uteliggere er merket som prikker utenfor variansen.



Figur 17: Figuren viser resultatene fra resursallokering av 36 reelle nettverkstopologier, isolert og mettet 30 ganger; først med mus deretter med elefanter. Den øverste grafen viser allokeringsevnen for små reservasjoner (5% av resursene) og den nederste viser store reservasjoner (30% av resursene). Denne grafen kan med fordel sammenlignes med fig. 18 som viser simulering av syntetiske topologier. Grafen er sortert etter allokeringskapasiteten til SRT, kjøringene er plassert horisontalt med allokeringskapasiteten langs y-aksen.



Figur 18: Her vises resultatene fra resursallokering av 32 syntetiske topologier, hver topologi isolert 30 ganger hver, først med elefanter, deretter med mus. Den øverste grafen viser allokeringskapasiteten ved små (5%) resursanmodninger (mus), den nederste viser reservasjoner med store resursallokeringsanmodninger (30% av resursene, elefanter). Figuren kan sammenlignes med fig. 17, som benytter reelle nettverkstopologier. Grafen er sortert etter allokeringskapasiteten til SRT, kjøringene er plassert horisontalt med allokeringskapasiteten langs y-aksen.

6 Konklusjon og refleksjon

Flerkonfigurasjonruting har vist gode resultater for feilhåndtering i en vilkårlig flerkoblet topologi, kombineres dette med fordelene tilknyttet MRC (s. 15), har vi årsaken til hvorfor vi ønsket å finne ut om teknologien er egnet til mer enn feilhåndtering.

Vi har i kapittel 2 redegjort for hvordan MRC fungerer, og i kapittel 1.3 gått gjennom en rekke båndbreddekrevende tjenester og nye bruksområder som gjerne kan være tjent med resursreservering, og på den måten klargjort behovet for teknologi som kan bidra til å maksimere reservasjonskapasiteten i nettverk. MRC presenteres som en mulig kandidat for nettopp dette. Resten av oppgaven har derfor vært dedikert til simuleringsprogrammet vi lagde for å studere nærmere resursallokeringskapasiteten til MRC, og eksperimentene vi utførte for å undersøke resursallokeringskapasiteten til MRC. Resultatene fra vår analyse av innsamlet data, samt en rekke observasjoner tilknyttet disse ble presentert i forrige kapittel; i teksten under vil vi trekke en konklusjon, reflektere over vår erfaring med SAK, og foreslå mulige temaer for videre arbeid.

Med utgangspunkt i resultatene vist i figur 11 og 19, side 65 og 94, ble vi presentert for et resultat som ikke virket udelt positivt for MRC. Teknologien demonstrerte god toppytelse, men viste seg også å være forholdsvis ustabil, med store variasjoner og uforutsigbar allokeringsevne. Vi ser i tabell 1 at MRC har et gjennomsnitt som ligger godt an både for syntetisk genererte topologier og særlig reelle nettverkstopologier, men standardavviket svekker verdien av dette resultatet. Ser vi på det avviksjusterte gjennomsnittet, ser det faktisk ut til å være SRT som leverer det beste resultatet; hvilket betyr at SRT jevnt over fremstår som det beste alternativet, til tross for at både MRC og STM har et høyere gjennomsnitt.

Ser vi på figur 13, som viser resultatene fra hvert enkelt segment med syntetiske topologier fra figur 12 er det ingen segmenter som har mer nytte av MRC, enn både SRT og STM. Det er derimot en rekke segmenter[32] der MRC gjør det

[32]MRC og STM yter tilnærmet like bra i segmentene C, D, G og H

tilnærmet like bra som STM, i disse segmentene ble en rekke RAR godkjent av både MRC (alene) og MRC-komponenten til STM, hvilket er årsaken til det sammenfallende resultatet.

Vi lurte også på hvordan isoleringstettheten påvirker resursallokeringsytelsen til MRC, og i tabell 3 ser vi svaret. Det er en tendens i tabellen som tilsier at en løsere isolert topologi får et lavere standardavvik og dermed også bedre resursallokeringsytelse. Dette var som forventet og betyr at MRC utnytter det faktum at et løst isolert konfigurasjonssett inneholder flere konfigurasjoner som kan tilby en sti like kostbar som TKS, hvilket er det optimale - så sant stien har kapasitet. Vi observerer også at standardavviket reduseres etterhvert som konfigurasjonsantallet stiger. Forskjellene vi har registrert i ytelsen til SRT, MRC og STM er ikke voldsomme. Det dreier seg sjelden om mer enn et par prosentpoeng, men innimellom kan dette være avgjørende for at nettverket nettopp skal klare å få akseptert nok en resursreservasjon. Hvis MRC-algoritmen aktivt hadde jobbet for bedre utnyttelse av likeverdige stier, mener vi resursallokeringskapasiteten hadde økt.

For å kunne konkludere hvorvidt MRC er bedre enn SRT, med utgangspunkt i et forholdsvis stort standardavvik, holder det ikke med en direkte sammenligning av gjennomsnittlig ytelse for SRT og MRC (fig. 2). Med utgangspunkt vårt store tallmateriale, benytter vi U-testen (s. 32) til å teste hypotesen: SRT = MRC. Gitt et signifikantsnivå på 5% kunne vi avvise hypotesen for reelle nettverkstopologier. Vi testet derfor hypotesen av SRT er bedre enn MRC, og gitt samme sikkerhet på 95%, kunne vi også avvise denne hypotesen for MRC.

Konklusjonen vår blir derfor at MRC med 95% sikkerhet vil yte bedre enn SRT i reelle nettverkstopologier. For at MRC (og dermed også STM) skal bli sterkere ledd i resursallokeringsprosessen, mener vi det vil være mest hensiktsmessig å jobbe for å *å redusere standardavviket* til MRC. Vi har tro på at dette kan gjøres ved å kontrollere hvordan konfigurasjonene, nodene og linkene traverseres under konfigurasjonsfasen til MRC, men vi overlater dette til eventuelt videre arbeid. Mye takket være de gode egenskapene til MRC, er vi

avslutningsvis av den oppfatning at MRC er et konkurransedyktig alternativ for resursallokering i tilgangsstyrte nettverk.

6.1 Visuelt assistert forståelse

Ved begynnelsen av dette prosjektet hadde vi en klar forståelse av MRC, vi kjente algoritmen og fremfor alt forstod den. Likevel opplevde vi at den unike og detaljerte visualiseringen av MRC implementert av SAK både skjerpet og fordypet vår forståelse ytterligere. En stor andel av forslagene til videre arbeid er et direkte resultat av tanker gjort under observasjoner av MRC i sanntid; animasjonen ble opplevd som både stimulerende og inspirerende. Utover det rent personlige forbundet med bruken av SAK, var animasjonen en opplagt fordel også for kommunikasjonen med andre. Det å kunne demonstrere MRC algoritmen visuelt gjorde det lettere å involvere, eller introdusere nye mennesker for prosjektet, og ikke minst teknologien.

At de resulterende konfigurasjonene blir lagret som bilder, gjør det også lett å sende konkrete konfigurasjonsløsninger til andre via f.eks. e-post. Bildefilene kan også muligens gjøre det lettere for eventuelle nettverksadministratorer å planlegge hvordan nettverket skal legges ut og implementeres. Visualiseringen kan også gjøre det enklere å følge opp krav til konfigurasjonsløsningen, som f.eks. plassering av isolerte komponenter, slik at to noder tilkoblet f.eks. samme strømkurs ikke isoleres i samme konfigurasjon. Her mener vi at det også er til hjelp at samtlige konfigurasjoner kan vises på skjermen samtidig, og at nodenes plassering kan ednres i henhold til brukerens egne behov.

Under utviklingen av SAK introduserte visualiseringen også en ny tilnærming til løsningen av feil og problemer, og var ved et par anledninger en direkte årsak til at feilen forholdsvis hurtig ble lokalisert og korrigert. Under utvikling brukte vi stort sett utelukkende topologien `1b.top`, som består av 17 noder og 27 linker, hvilket er en forholdsvis liten topologi. Dette har opplagt også påvirket visaliseringen, for eksempel er nodene gitt en størrelse som gjør topologien uoversiktlig hvis den overstiger tilnærmet 64 noder. Selvom dette enkelt kan

endres i koden, er det klart at visualiseringen ikke bidrar nevneverdig til oversikten når flere noder ligger skjult bak andre. Et annet problem for oversikten er at linkene tegnes som rette linjer. I visse situasjoner, er det derfor vanskelig å vite om linken stopper i den mellomliggende noden, eller om den går direkte til noden bortenfor. Både dette link- og nodeproblemet kan løses forholdsvis enkelt ved å endre visualiseringskoden til SAK. For eksempel kan det enkelt implementeres en dynamisk justering av nodestørrelsen i forhold til topologiens nodeantall[33], og linkene kan gjøres om til splines og gis en annen praktisk form. Det finnes flere demoer som følger med JUNG-biblioteket som viser hvordan dette eventuelt kan gjøres[34].

Det skal også nevnes at visualiseringen av topologien opplagt forsinker konfigurasjonsprosessen. Er det behov for produksjon av store mengder data ved help av SAK, er det derfor anbefalt at SAK settes i `fastest`-modus. Hvis vinduet som viser den innleste topologien lukkes *før* enten konfigurasjons- eller metningsprosessen iverksettes, vil mye tid bli spart.

6.2 Videre arbeid

Arbeidet med SAK tok til uten konkret tilretteleggelse for visualisering. Hadde listen [44] over programminergsbibliotek eller tilsvarende lister vært kjent tidligere i prosjektet, hadde vi mye tidligere satset på bruk av JUNG, eller et tilsvarende bibliotek, under utviklingen av SAK. Tidligere implementering av JUNG ville kunne ført til bredere interaktiv støtte, slik som støtte for kjøretidsredigering og generering av topologier, lagring av ferdigisolerte konfigurasjoner, og muligens implementering av en bedre utnyttelse av likeverdige stier i konfigurasjonssetet, og en rekke andre funksjoner. Hvis det arbeides videre med utviklingen av SAK, er det anbefalt å løse bindingen mellom kommandoskallet og kommandoene som i gjeldende implementering er toveis.

Spesifikasjonene av isoleringsalgoritmen til Multiple Routing Configurations

[33] Dette gjøres eventuelt i `no.gatada.sak.graph/ConfigurationRenderer.java`.
[34] Disse demoene kan forøvrig sees på nettet direkte fra JUNGs egen nettside.

(MRC) definerer ingen bestemt isoleringsrekkefølge av nodene i nettverket, heller ikke den rekursive traverseringen av topologien, eller isoleringen av tilstøttende linker blir spesifisert. Vi mener det er selvsagt at disse prosessene påvirker hvordan det endelige settet med konfigurasjoner ser ut, og at de dermed også påvirker ytelsen til MRC. Vi nevnte i kapittel 5.2 (s. 66) hvor vanskelig det kan være å få ønsket resultat under isolering av en topologi, også da vi gjennomgikk algoritmen til MRC (s. 21) poengterte vi kodesegmentene som foreløpig vilkårlig velger en konfigurasjon, node og en link. En mer hensiktsmessig styring av disse prosessene mener vi er et svært godt egnet tema for videre forskning. I denne sammenheng kan vi også trekke frem følgende eksempel. Under utviklingen av SAK ble det til testing i all hovedsak benyttet en redusert utgave av `1.top` fra Rocketfuel [39]. Den reduserte testtopologien inneholdt samtlige noder som var del av det segmentet som var flerkoblet i den originale topologien. Testtopologien ble funnet som et godt egnet topologi til testing fordi isoleringsalgoritmen til MRC genererte alt fra 3 til 6 konfigurasjoner: 72% av løsningene benyttet 4 konfigurasjoner, 25% trengte 5 konfigurasjoner og de resterende 2% ga henholdsvis 3 og 6 konfigurasjoner. Hvis det var behov for å undersøke egenskaper ved MRC eller SAK som krevde enten 3 eller 6 konfigurasjoner, var vi nødt til å bruke uforholdsmessig mye tid på konfigurasjonsprosessen, kun på å finne konfigurasjonsløsningen som benyttet 3 eller 6 konfigurasjoner.

Et relatert tema i denne sammenheng er å studere nærmere konsekvensen av kontrollering av isoleringskjedens lengde. Vi beskriver hva en isoleringskjede er fra side 22. Det er mulig å tenke seg at MRC traverserer topologien på en slik måte at isoleringskjeden til en hver tid alltid holdes så kort som mulig, og opplagt er det mulig å også tenke seg det motsatte. Vi mener det hadde vært spennende å studere dette nærmere, å analyserer konsekvensene ved manipulering av isoleringskjedens lengde.

Det burde være mulig å finne et uttrykk som representerer det minimum antall konfigurasjoner som kreves for å isolere en topologi. En komponent i dette måltallet må være den lengste lenken i topologien. En lenke beskriver en

sammenkoblet rekke av minimum 3 noder som *ikke deltar i noen andre stier* i topologien, og som utgjør en (urettet) ring hvis endenodene i kjeden kobles sammen.

$$C = max(a, b) \tag{3}$$

Der `C` er antall konfiurasjoner, `a` er antall noder i den lengste lenken i topologien, og `b` er et tall som gis utifra egenskapene til topologien forøvrig. Vi mener et utgangspunkt for sistnevnte kan være den eller de lengste av topologiens isolasjonskjeder og størrelsen på tilstøtende isolasjonskjeder med en felles kant.

Hvis et nettverk implementerer MRC, kan vi også tenke oss at kontrollmeldinger rutes ved hjelp av MRC, slik at lokal feiloppretting er mulig også for kontrollmekanismen. Det er ingenting i veien for at feil i kontrollaget håndteres ved hjelp av de samme konfigurasjonene som allerede er distribuert i nettverket, hvilket betyr at ruting etter beste evne, resursallokering og kontrolllaget foregår ved hjelp av MRC.

I kapittel 4.1.1 forklares algoritmen som traverserer konfigurasjonen og identifiserer KSS. Hvordan valg av likeverdige stier kan foretas, ettersom de i gjeldende implementasjon faktisk ikke tar hensyn til dette - noe som resulterer i reservasjonsavslag til tross for at det finnes stier med tilstrekkelige resurser. En mekanisme som sørger for at den likeverdige stien mellom 1-7 inkluderer node 5 i E4, ettersom node 4 er inkludert i konfigurasjon 3 blant de nederst i figur 2. Dette kan opplagt *ikke* gjøres ved ytterligere manipulasjon av vektene i topologien. For å muliggjøre sortering av konfigurasjonene etter stienes lengde, introduserte vi avstandstabeller også for konfigurasjonene. Bruk av avstandstabellen presenteres på side 42. Stien som avstandstabellen identifiserer er basert på rekkefølgen i tabellen. Dette betyr at hvis node A og node B begge har avstand 3 til node D, så vil stien via node A bli valgt fordi denne er først i tabellen. Med andre ord, hvis vi kan manipulere rekkefølgen noden traverserer sine nabonoder for å finne

den korteste stien, slik at denne er forskjellig i hver konfigurasjon, så vil også den resulterende stien som velges i hver konfigurasjon bli forskjellig, selvom de er likeverdige. For oss fremstår det som kun en mulig løsning som sørger for at flere likeverdige stier benyttes, ved å manipulere *rekkefølgen nodene listes opp i avstandstabellene* tilknyttet de forskjellige konfigurasjonene. Ruteprotokollen som benyttes må da også oppfatte rekkefølgen i avstandstabellen som signifikant og rute deretter. SAK gjør dette ved at det alltid er den siste noden med en likeverdig sti som velges til ruting.

Reell nettrafikk vil ikke være uniformt distribuert i et nettverk [3]. Det er gjerne noen stier som er mer brukt enn andre, og det er rimelig å anta at noen av disse helt eller delvis krysser nettverket. Hvis en av disse stiene blir mettet, vil opplagt nettverket bli segmentert, hvilket kan gjøre det hensiktsmessig å benytte en grenseverdi som spesifiserer et metningspunkt for når MRC vil bli brukt til å omgå disse sentrale komponentene. Av denne grunnen ble det vurdert nok et eksperiment til denne oppgaven, der SRT ble benyttet frem til en spesifisert og muligens dynamisk satt terskelverdi utifra topologiens størrelse og kompleksitet; men fordi det foreligger så mange ubesvarte spørsmål tilknyttet et slikt eksperiment, mener vi denne problemstillingen er bedre egnet som tema for et selvstendig arbeid på et senere tidspunkt.

Forbundet med segmentering av nettverket, mener vi det vil være mulig å implementere en mekanisme som muliggjør dynamisk re-allokering av reserverte resurser. Dette vil bety at straks nodene i senter av topologien nærmer seg mettet, vil det være mulig å flytte trafikk bort fra sentrum og over til kantnodene i nettverket. Dynamisk reallokering av reserverte resurser vil kunne håndteres med en enkel bekreftelsesanmodning fra Båndbreddemegler til ingressnoden, som inneholder den nye konfigurasjonsnøkkelen som skal benyttes. Ved mottak av kontrollmeldingen merker ingressnoden pakkene med den oppdaterte konfigurasjonsnøkkelen før de sendes til egressnoden som vanlig. I det egressnoden mottar den første pakken rutet over denne nyvalgte konfigurasjonen, sender denne en bekreftelsesanmodning til BB som bekrefter reallokeringen. Egressnoden

legger deretter strømmen over i den nye konfigurasjonen, og når ingressnoden mottar pakkene over den nye konfigurasjonen, sendes en bekreftelsesmelding til BB som dermed vet at resursene langs den tidligere reserverte stien nå er frigjort.

Det kan forøvrig være hensiktsmessig å prioritere de anvendte konfigurasjonene som benyttes etter varighet og resursbehov. Dermed kan f.eks. langvarige elefantreservasjoner forsøke konfigurasjoner som leder trafikken utenom sentrale noder, før de mer sentraliserte konfigurasjonene undersøkes for tilgjengelige resurser. Dermed unngår man allokeringssegmentering av topologien og store omveier for mus og kortvarige elefanter.

Referanser

[1] Marc Andreessen. Mosaic web browser history - nsca, marc andreessen, eric bina. http://www.livinginternet.com/w/wi_mosaic.htm, 20. Feb. 1993.

[2] A. Baroni and F. Abreu. Formalizing object-oriented design metrics upon the uml meta-model, 2002.

[3] Supratik Bhattacharyya, Christophe Diot, and Jorjeta Jetcheva. Pop-level and access-link-level traffic dynamics in a tier-1 pop. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 39–53, New York, NY, USA, 2001. ACM Press.

[4] Paul E. Black. Dijkstra'a algorithm. *Dictionary of Algorithms and Data Structures [online], U.S. National Institute of Standards and Technology*, Sept. 2007.

[5] C. Bouras and K. Stamos. Examining the benefits of a hybrid distributed architecture for bandwidth brokers. In *Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International*, pages 491–498, 2005.

[6] Encyclopædia Britannica. computer simulation. Article@ http://www.britannica.com/eb/article-9001627, September 2006.

[7] Apple Corp. Apple - ipod + itunes. http://www.apple.no/, March 2006.

[8] Gilbert Cotton. Frogner går digitalt. Tidsskrift, Digital World, nr. 2, 2006.

[9] Zhenhai Duan, Zhi-Li Zhang, Yiwei Thomas Hou, and Lixin Gao. A core stateless bandwidth broker architecture for scalable support of guaranteed services. *IEEE Trans. Parallel Distrib. Syst.*, 15(2):167–182, 2004.

[10] Norsk Filminstitutt. Filmarkivet.no. http://www.filmarkivet.no/, March 2006.

[11] Donald G. Firesmith and Edward M. Eykholt. *Dictionary of Object Technology: The Definitive Desk Reference*. SIGS, 1995.

[12] K. Gopalan, Tzi cker Chiueh, and Yow-Jian Lin. Network-wide load balancing routing with performance guarantees. volume 2, pages 943–948, 2006.

[13] Peter Grant. Online video goes mainstream, sparking an industry scramble. *The Wall Street Journal*, XXIV(16):1, 36, February 21. 2006.

[14] W. Grover, J. Doucette, M. Clouqueur, D. Leung, and D. Stamatelakis. New options and insights for survivable transport networks. *Communications Magazine, IEEE*, 40(1):34–41, 2002.

[15] W. Grover and D. Stamatelakis. Cycle-oriented distributed preconfiguration: Ring-like speed with mesh-like capacity for self-planning network restoration, 1998.

[16] W. Grover and D. Stamatelakis. Self-organizing closed path configuration of restoration capacity in broadband mesh transport networks, 1998.

[17] A. F. Hansen, A. Kvalbein, T. Cicic, S. Gjessing, and O. Lysne. A comparison of different approaches for calculating resilient routing layers and multiple routing configurations. Technical Report 2005-15, Simula Research Laboratory, 2005.

[18] A. F. Hansen, A. Kvalbein, T. Cicic, S. Gjessing, and O. Lysne. Resilient routing layers for recovery in packet networks. In Bob Werner, editor, *International Conference on Dependable Systems and Networks (DSN 2005) Yokohama, Japan, June 28-July 1*. IEEE Computer Society, 2005. ISBN 0-7695-2282-3.

[19] John Heidemann, Kevin Mills, and Sri Kumar. Expanding confidence in network simulation. Research Report 00-522, USC/Information Sciences Institute, April 2000. submitted for publication, IEEE Computer.

[20] OMG Inc. Unified modeling language (uml), version 2.0. http://www.omg.org/technology/documents/formal/uml.htm, 1997-2006.

[21] Steve Jobs. Thoughts on music. http://www.apple.com/hotnews/thoughtsonmusic/, 6 Feb. 2007.

[22] Byeongsik Kim, Heesung Chae, Taeman Han, and Yoohyun Jeong. Bandwidth broker signaling for service level negotiation over heterogeneous ipv4/ipv6 diffserv networks. In *Advanced Communication Technology, 2004. The 6th International Conference on*, volume 2, pages 1097–1102, 2004.

[23] G. Kim, P. Mouchtaris, S. Samtani, R. Talpade, and L Wong. Qos provisioning for voip in bandwidth broker architecture: A simulation approach. In *Communication networks and distributed systems modeling and simulation conference*, 2001.

[24] Leonard Kleinrock. *Information Flow in Large Communication Nets*. PhD thesis, MIT, Research Laboratory of Electronics MA, July 1961.

[25] Chamil P. W. Kulatunga, Jesse Kielthy, Paul Malone, and Micheal O Foghlu ? Implementation of a simple bandwidth broker for diffserv networks. In *Proceedings 2nd International Workshop on Inter-Domain Performance and Simulation*, March 2004.

[26] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast recovery from link failures using resilient routing layers. In *10th IEEE Symposium on Computers and Communications (ISCC 2005)*, pages 554–560, Cartagena, Spain, June 27-30, 2005. IEEE Communications Society. ISSN 1530-1346, ISBN 0-7695-2373-0.

[27] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast ip network recovery using multiple routing configurations. In Zhili Zhang Arturo Azcorra, Joe Touch, editor, *INFOCOM 2006*, Barcelona, Spain April 23 – 29, 2006. IEEE.

[28] Anukool Lakhina, Alberto Medina, Ibrahim Matta, and John Byers. Brite: Boston university representative internet topology generator. `http://www.cs.bu.edu/brite/`, 2002.

[29] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development.* Prentice Hall PTR, 3 edition, October 20 2004.

[30] Bu-Sung Lee, Wing-Keong Woo, Chai-Kiat Yeo, Teck-Meng Lim, Bee-Hwa Lim, Yuxiong He, and Jie Song. Secure communications between bandwidth brokers. *SIGOPS Oper. Syst. Rev.*, 38(1):43–57, 2004.

[31] K. Lee, A. Toguyeni, and A. Rahmani. Hybrid multipath routing algorithms for load balancing in mpls based ip network. volume 1, pages 6 pp.–, 2006.

[32] Derick Mains and Tom Neumayr. Apple unveils higher quality drm-free music on the itunes store. `http://www.apple.com/pr/library/2007/04/02itunes.html`, 2. April 2007.

[33] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot. Characterization of failures in an ip backbone, 2004.

[34] Satoshi Matsuoka, Rodney R. Oldehoeft, and Marydell Tholburn, editors. *Why We Don't Know How To Simulate The Internet.* Springer, January 2000.

[35] M. Medard, S. G. Finn, R. A. Barry, and R. G. Gallager. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *Networking, IEEE/ACM Transactions on*, 7(5):641–652, 1999.

[36] K. Nichols, V. Jacobson, and L. Zhang. A two-bit differentiated services architecture for the internet. `http://www.ietf.org/rfc/rfc2638.txt`, July 1999.

[37] University of California. Jung java - universal network/graph framework. `http://jung.sourceforge.net/`, March 2006.

[38] University of Illinois. J-sim home page. http://www.j-sim.org/, 2007.

[39] University of Washington. Rocketfuel: An isp topology mapping engine. http://www.cs.washington.edu/research/networking/rocketfuel/, November 2006.

[40] Joshua O'Madadhain, Danyel Fisher, and Padhraic Smyth. Analysis and visualization of network data using jung. *Statistical Software*, 2005.

[41] Dan Pilone and Neil Pitman. *UML 2.0 in a Nutshell*. O'Reilly Media, 2 edition, June 2005.

[42] RealNetworks. Realnetworks granted fundamental streaming media patent. http://www.realnetworks.com/company/press/releases/2006/patent_cts.html, 2006.

[43] R. RFC. Resource reservation protocol, 1997.

[44] rowanxmas. Network and graph visualization. http://networkviz.sourceforge.net/, 2003.

[45] Chris Sander, Leroy Hood, Benno Schwikowski, Annette Adler, and Trey Ideker. Cytoscape. http://csbi.sourceforge.net/, Nov 2006.

[46] Shaleeza Sohail and Sanjay Jha. The survey of bandwidth broker, 2002.

[47] D. Stamatelakis and W. Grover. Rapid span or node restoration in ip networks using virtual protection cycles, 1999.

[48] D. Stamatelakis and W. D. Grover. Ip layer restoration and network planning based on virtual protection cycles. *Selected Areas in Communications, IEEE Journal on*, 18(10):1938–1949, 2000.

[49] The New AT & T. The new at & t: Driving the communications transformation. http://www.sbc.com/Common/files/pdf/AT\&TTransformation.pdf, 2006.

[50] D. Torrieri. Algorithms for finding an optimal set of short disjoint paths in a communication network. pages 11–15 vol.1, 1991.

[51] Guoliang Xue, Li Chen, and K. Thulasiraman. Delay reduction in redundant trees for preplanned protection against single link/node failure in 2-connected graphs. volume 3, pages 2691–2695 vol.3, 2002.

A Ordforklaring

API Application Programming Interface, programmeringsbibliotek, tilbyr et sett med ferdigimplementert funksjonalitet som bistår utviklingen av nye programmer.

Flerkoblet Betyr at en vilkårlig valgt node kan fjernes fra topologien uten at nettverket segmenteres.

KKS Konfigurasjonens korteste sti, som i motsetning til TKS ikke nødvendigvis er den korteste stien mellom inngangs- og utgangsnoden som topologien kan tilby, KKS er i gjennomsnitt 15% [18] lenger enn TKS.

Konfigurasjonsløsning En refererer til antallet konfigurasjoner produsert av MRC. Dette fører til at en isolert topologi som etter en kjøring benytter 3 konfigurasjoner og etter en annen benytter 6 konfigurasjoner, regnes som to forskjellige er. Det betyr også at resultatet fra to kjøringer, som begge to ender med 3 er, regnes som den samme en og derfor slås sammen i statistikken; selvom hver enkelt konfigurasjon i prinsippet kan inneholde forskjellige isolerte komponenter.

MRC Akronym, står for Multiple Routing Configurations.

PDB Polise database, database som vedlikeholdes av båndbreddemegler og brukes til å organisere informasjon tilknyttet tjenesteavtaler, i tillegg til fremtidige og aktive resursreserveringer.

SAK Står for "Simulering av kapasitet" og refererer til simulatoren som ble kodet i Java i forbindelsen med denne oppgaven.

SRT Single Routing Table, standard metode for ruting av data, refererer til alle rutingmetoder som benytter en enkel rutetabell til å identifisere korteste sti fra inngangs- til utgangsnode.

STM SRT Then MRC, henviser til kombinasjonen av SRT og MRC, hvilket betyr at hvis SRT ikke klarer å allokere tilstrekkelig med resurser, vil

MRC gjøre et forsøk, og hvis også MRC feiler blir anmodningen avslått.

TKS Topologiens korteste sti innebærer færrest mulig hopp fra inngangs- til utgangsnode, TKS er i motsetning til konfigurasjonens korteste sti alltid færrest mulige hopp i gjeldende topologi. Dette utelukker selvsagt ikke likeverdiger stier, dvs. stier med identisk kost, vekt, eller antall hopp.

WAN Wide Area Network, nettverkstopologi som strekker seg over et større geografisk område sammenlignet med Local Area Network (LAN) og Metropolitan Area Network (MAN).

B Hovedmenyen til SAK

```
Simulator v3.72

ls [dir]          Lists all files contained in the current folder
                  or specified by the optional [folder] parameter.

load <topology>   Loads a text file with a node pair on each line.

config            Starts the configuration of a loaded topology.
                  Requires a biconnected network graph.

set [count]       Sets the number of streams to be routed through the
                  network. No parameter results in randomly selected
                  limitless sources.

delay             Toggles between fast, slow and default execution to make
                  it easier to follow what is happening during simulation.

route [count]     Will start the process of generating resource requests
                  and routing traffic over the configured topology.
                  When the network is fully saturated it terminates.

run <dir> <int>   Runs the simulation int number of times for each
                  topology contained in the provided folder.

history           Shows a list of the 10 last entered commands. Type
                  !N to execute the command listed as number N.

help              Displays this message.

exit              Quits the simulator.
```

C Topologifiler

Til denne oppgaven ble det anvendt 786 forskjellige topologier, hvorav 36 stykker er basert på reelle nettverk og resten er syntetisk generert. Under følger en liste over samtlige av disse topologiene, med informasjon om antall noder (vertices) og linker (edges) for hver enkelt. Listen er sortert etter antall noder.

Lenger ned vil du også finne en oversikt over de 32 topologiene som ble tilfeldig valgt ut og brukt som grunnlag for oppgavens fokusstudium.

C.1 Reelle og syntetiske nettverkstopologier

```
 3 vertices    3 edges  All/1784.top
 3 vertices    3 edges  All/3701.top
 3 vertices    3 edges  All/10910.top
 4 vertices    5 edges  All/15290.top
 4 vertices    6 edges  All/7543.top
 5 vertices    5 edges  All/4513.top
 5 vertices    7 edges  All/4006.top
 5 vertices    7 edges  All/13129.top
 6 vertices    7 edges  All/15412.top
 7 vertices    9 edges  All/7176.top
 7 vertices   12 edges  All/3602.top
 7 vertices   13 edges  All/6539.top
 8 vertices   14 edges  All/6939.top
 8 vertices   18 edges  All/2497.top
10 vertices   16 edges  All/1785.top
10 vertices   17 edges  All/germantelxl.top
11 vertices   26 edges  All/cost239.top
12 vertices   17 edges  All/4565.top
13 vertices   30 edges  All/6467.top
13 vertices   37 edges  All/dfnxl.top
15 vertices   24 edges  All/11537.top
17 vertices   27 edges  All/1b.top
17 vertices   29 edges  All/network-USA-2.top
17 vertices   36 edges  All/7911.top
18 vertices   31 edges  All/3300.top
18 vertices   60 edges  All/7170.top
19 vertices   30 edges  All/geant.top
19 vertices   35 edges  All/7132.top
20 vertices   30 edges  All/16631.top
22 vertices   51 edges  All/6395.top
24 vertices   38 edges  All/5511.top
27 vertices   57 edges  All/4637.top
30 vertices   54 edges  All/cwxl.top
32 vertices   64 edges  All/1239.top
33 vertices   83 edges  All/209.top
46 vertices  268 edges  All/3356.top
68 vertices  353 edges  All/3320.top
```

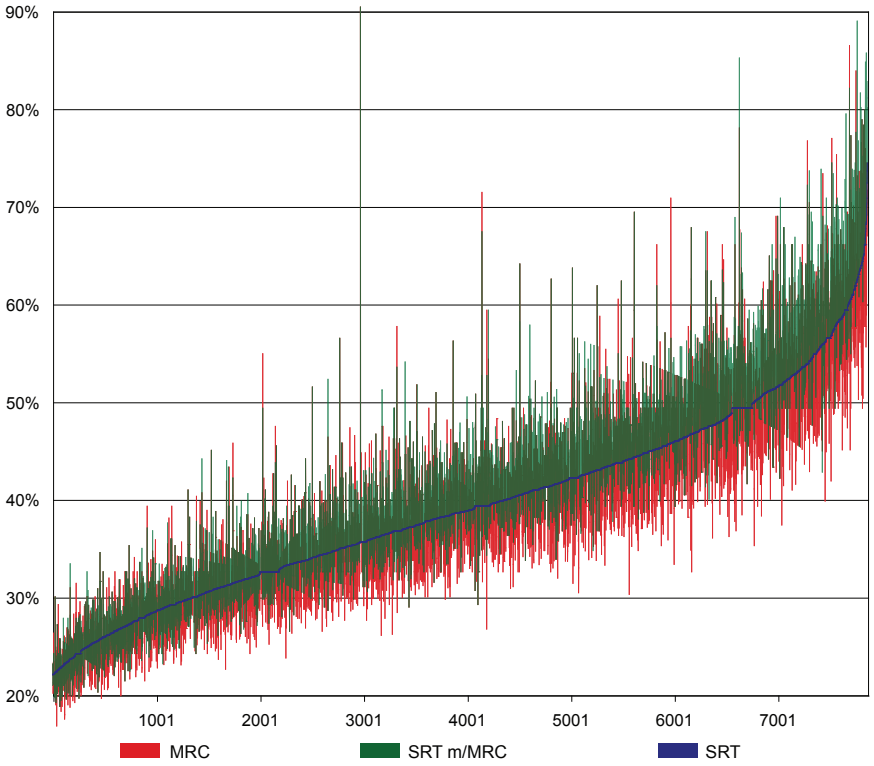
I tillegg til de reelle nettverkene, anvendte vi også 8 forskjellige typer topologikonfigurasjoner, og med mindre noe annet er oppgitt, brukte vi 100 varianter av hver konfigurasjon. I stedet for å liste opp samtlige topologier, viser vi under kun ett eksempel fra hver gruppe. Rekkefølgen på topologiene er slik de blir traversert av SAK.

```
128 vertices  256 edges  All/T2-128-266.top
 16 vertices   32 edges  All/T2-16-67.top
 32 vertices   64 edges  All/T2-32-66.top
 64 vertices  128 edges  All/T2-64-4.top
128 vertices  384 edges  All/T3-128-353.top (99 stk)
 16 vertices   48 edges  All/T3-16-11.top
 32 vertices   96 edges  All/T3-32-21.top   (99 stk)
 64 vertices  192 edges  All/T3-64-54.top   (51 stk)
```

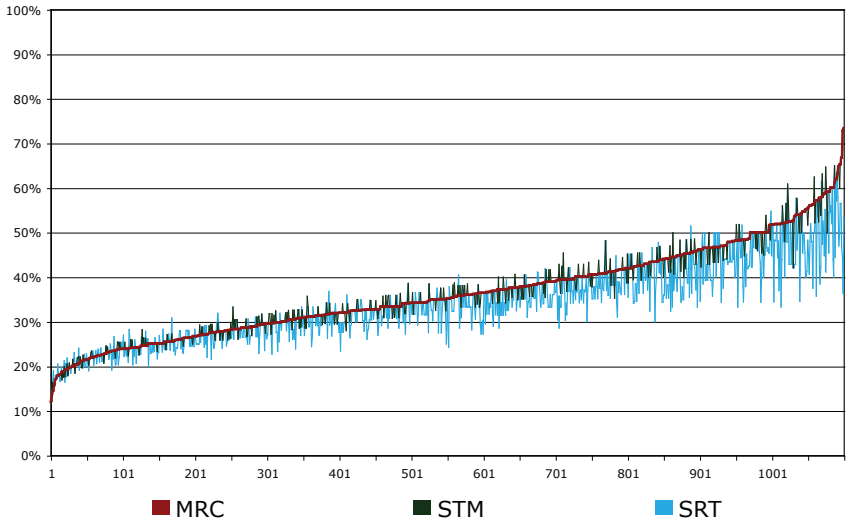
D Oversiktgraf for store reservasjoner

Denne grafen er utelukkende tatt med fordi vi fant det tydelige mønsteret rundt to områder langs grafen svært interessant. Legg merke til de to stjerneformete områdene rundt observasjon 2000 og 6700. Vi ikke gjort oss noen mening om hvorfor grafen flater ut og hvorfor disse tydelige mønstrene dannes. Uten mulighet til å følge opp denne observasjonen, overlater vi grafen til eventuelt fremtidig arbeid.

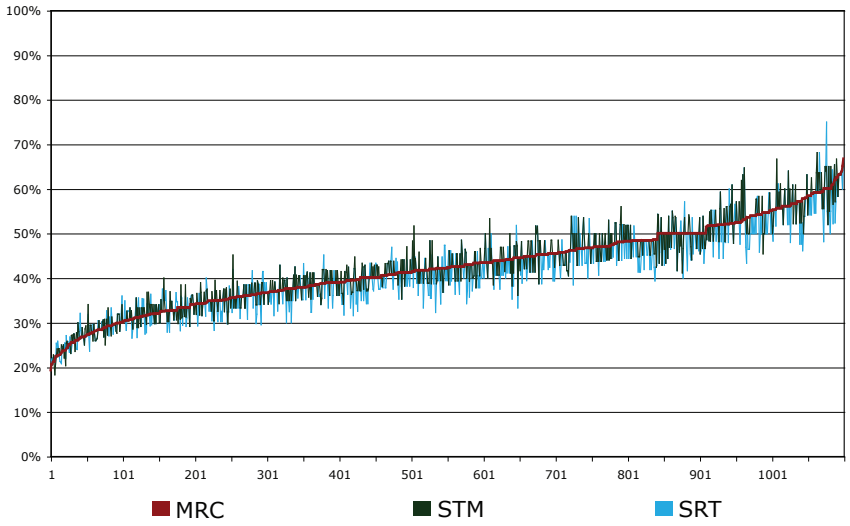
Vi tok også med disse to grafene, som er en tilfeldig valgt topologi, isolert 1100 ganger for både syntetiske og reelle nettverkstopologier. Disse grafene viser også en utpreget forskjell mellom syntetiske og reelle nettverkstopologier; figur 20 viser resultatet til ruting i reelle topologier. Legg merke til hvor mye bedre allokeringskapasiteten er for MRC i reelle topologier, sammenlignet med fig. 21.



Figur 19: Grafen viser 785 forskjellige topologier, 10 kjøringer av hver. Allokeringskapasiteten for hver enkelt kjøring er her ordnet horisontalt etter ytelsen til SRT (vertikalt) oppgitt i prosentandelen godkjente resursallokeringsanmodninger. MRC vises her i rødt, og SRT kombinert med MRC (STM) vises i gjennomsiktig grønt. Den mørke grønnfargen er resultatet av rødfargen under som skinner gjennom.



Figur 20: Denne grafen er representativ for de fire forskjellige isoleringsløsningene produsert til de reelle topologiene ved hjelp av MRC og SAK. Fordi vi her ønsker å se hvordan MRC yter i forhold til SRT (og STM) er samtlige av de tilsammen 1100 observasjonene ordnet horisontalt etter allokeringskapasiteten (y-aksen) til MRC. Grafen viser allokeringskapasiteten for store reservasjoner. Legg merke til hvor mye lavere og dermed dårligere SRT ligger i grafen i forhold til fig. 21, og at grafen viser tilsvarende mønster som nederste graf i fig. 17, som viser resultatet av 30 kjøringer av 36 forskjellige reelle topologier.



Figur 21: Denne grafen viser 1100 elefantreservasjoner, sortert etter allokeringskapasiteten til MRC langs x-aksen. Allokeringskapasiteten kan dermed leses av vertikalt og oppgis i prosent. Grafen er representative for de fire forskjellige isoleringsløsningene produsert med MRC og SAK. Legg også her merke til at grafen viser tilsvarende tendenser som nederste graf i fig. 18, hvilket styrker troen på at dette resultatet er representativt.