Skalerbar og QoS-bevisst arkitektur for innholdsbasert distribusjon av store mengder data over WAN

av
André Andersen
Jørgen de Lange
Tomas Skøie

Masteroppgave i
informasjons- og kommunikasjonsteknologi

Høgskolen i Agder
Fakultet for teknologi

Grimstad, mai 2005

Sammendrag

Vi ser stadig at data som overføres på Internett øker, både i form av antall og størrelse. Ettersom uønskede pakker er blitt et økende problem, er det utviklet systemer som ruter pakkene kun til ønskelig mottaker ved hjelp av pakkens innhold. Innholdsbaserte distribusjonssystemer blir ofte brukt i disse tilfellene. Utviklingen av slike systemer er ofte beregnet for distribusjon av små data pakker hvor rask overføring er et krav.

Ettersom størrelsen på data som overføres øker i takt med den teknologiske utviklingen, settes det større krav til et innholdsbasert distribusjonssystem. Data kan eksempelvis være en høyoppløselig spillefilm eller en stor programvareoppdatering som skal distribueres til et antall sluttbrukere. Når disse skal overføres raskt, kan dette overbelaste det underliggende nettverket.

Spillefilmer og programvarer har som regel en lengre tidsfrist (utgivelsesdato) for når disse kan videreformidles. Ut i fra dette og problemer nevnt over har vi kommet frem til en ny innovativ metode for å spre belastningen jevnt over det underliggende nettverket.

Vi presenterer to løsninger på problemene over hvor vår metode er implementert. Disse løsningene er både innholdsbaserte, har fokus på distribusjon av data av stor størrelsesorden og tar tiden til hjelp for å spre belastningen jevnt over nettverket.

Et viktig mål for oppgaven har vært å belaste det underliggende nettverket minst mulig ved å ta tiden til bruk. Vi oppnår gode resultater i vår testing og viser at begge løsningene reduserer belastningen på underliggende nettverk betydelig.

Forord

Denne hovedoppgaven ble gitt av Infotain AS og skrevet som en del av mastergradtittelen innenfor informasjons– og kommunikasjonsteknologi. Oppgaven ble utført ved Høgskolen i Agder, i perioden fra januar 2005 til mai 2005.

Vi ønsker å takke P.hD. Granmo for veiledning og konstruktiv kritikk i prosjektperioden.

Grimstad, mai 2005
André Andersen, Jørgen de Lange & Tomas Skøie

Innholdsfortegnelse

1 Innledning 1
1.1 Bakgrunn 1
1.2 Siena 1
1.3 Oppgavedefinisjon 2
1.4 Relatert arbeid 2
1.5 Vårt Arbeid 3
1.6 Rapportens struktur 3
2 Teoretisk bakgrunn 4
2.1 Innholdsbasert ruting 4
2.2 Siena 5
2.3 Epidemisk distribusjon 9
2.4 Tjenestekvalitet 11
2.5 Avstandsvektor 13
2.6 Oppsummering 15
3 Arbeid utført 16
3.1 Simulering 16
3.2 Tidsfristbasert distribusjon 20
3.3 Siena utvidet 21
3.4 Epidemisk Siena 25
4 Resultater 34
4.1 Siena 36
4.2 Den utvidede Siena 41
4.3 Siena Epidemisk 45
5 Drøfting 50
5.1 Siena 50
5.2 Den utvidede Siena 51
5.3 Den epidemiske Siena 52
5.4 Sammenligning av løsningene 53
6 Konklusjon 55
7 Referanser 56
8 Vedlegg 59

Liste over figurer og tabeller

Figur 2-1: Siena Event Service 5
Figur 2-2: Siena Event Service med eventservers 6
Figur 2-3: Hierarkisk klient/server 7
Figur 2-4: Asyklisk peer-to-peer 8
Figur 2-5: Generisk peer-to-peer 8
Figur 3-1: Eksempel på et filter 27
Figur 3-2: Eksempel på et filter 27
Figur 3-3: Transittnettverk for over scenario 32
Figur 3-4: Gjennomsnitlig redundans sett over tid 33
Figur 4-1: Nettverket simuleringen ble utført over. 34
Figur 4-2: Siena: Transittnett med liten belastning 37
Figur 4-3: Siena: Transittlinkenes belastning (%) ved liten belastning 37
Figur 4-4: Siena - Transittnett med stor belastning 38
Figur 4-5: Siena: Transittlinkenes prosentvis belastning ved stor belastning 38
Figur 4-6: Siena - Belastning på hver av transittlinkene ved liten belastning 39
Figur 4-7: Siena: Entropi over transittnett med liten belastning 39
Figur 4-8: Siena - Belastning på hver av transittlinkene ved stor belastning 40
Figur 4-9: Siena - Entropi over transittnett med stor belastning 40
Figur 4-10: Siena Utvidet - Transittnett med liten belastning 41
Figur 4-11: Siena Utvidet - Transittlinkenes belastning (%) ved liten belastning 41
Figur 4-12: Siena Utvidet - Transittnett med stor belastning 42
Figur 4-13: Siena Utvidet - Transittlinkenes belastning (%) ved stor belastning 42
Figur 4-14: Siena Utvidet - Belastning på hver av transittlinkene ved liten belastning 43
Figur 4-15: Siena Utvidet – Entropi over transittnett med liten belastning 43
Figur 4-16: Siena Utvidet – Belastning på hver av transittlinkene med stor belastning 44
Figur 4-17: Siena Utvidet - Entropi over nettet med stor belastning 44
Figur 4-18: Siena Epidemisk - Transittnett med liten belastning 45
Figur 4-19: Siena Epidemisk - Transittlinkenes belastning (%) ved liten belastning 46
Figur 4-20: Siena Epidemisk - Transittnett med stor belastning 46
Figur 4-21: Siena Epidemisk - Transittlinkenes belastning (%) ved stor belastning 47
Figur 4-22: Siena Epidemisk - Belastning på hver av transittlinkene ved liten belastning 47
Figur 4-23: Siena Epidemisk - Entropi over transittnett med liten belastning 48
Figur 4-24: Siena Epidemisk - Belastning på hver av transittlinkene med stor belastning 48
Figur 4-25: Siena Epidemisk - Entropi over nettet med stor belastning 49

Tabell 3-1: Eksempel på en hendelse med forklaring 19
Tabell 3-2: Eksempel på en notification i Siena 22
Tabell 3-3: Eksempel på en notification i utvidet Siena 22
Tabell 3-4: Eksempel på en fragmentert notification 23
Tabell 3-5: Notification med flere destinasjonsadresser 28
Tabell 3-6: Første fragment 30
Tabell 3-7: Andre fragment 30
Tabell 4-1: Testdata for liten belastning 36
Tabell 4-2: Testdata for stor belastning 36
Tabell 8-1: Reserverte attributter 1

1 Innledning

1.1 Bakgrunn

I takt med den teknologiske utviklingen har det oppstått et økende behov for å overføre data av stadig høyere størrelsesorden. Eksempelvis kan det være ønskelig å distribuere data som høyoppløselige spillefilmer og store programvareoppdateringer til en rekke sluttbrukere spredt over store geografiske områder.

Siden ulike sluttbrukere har ulike behov settes det krav til at informasjonssystemet er innholdsbasert i den grad at hver sluttbruker individuelt kan bestemme innholdet på data som mottas. For eksempel er en sluttbruker bare interessert i å motta spillefilmer innen sjangeren Action og Drama, mens en annen bruker ønsker å motta programvareoppdateringer for operativsystemet Linux.

Overføring av data over store geografiske områder innebærer gjerne at deler av overføringen skjer over WAN. Det betyr at båndbredden kan være kraftig begrenset. Videre vil latenstiden være lengre enn ved overføring av data i LAN. Til tross for disse begrensningene er det ofte ønskelig at data overføres raskt, eventuelt innen en viss tidsfrist. Rask overføring av data ved begrenset båndbredde vil nødvendigvis belaste underliggende nettverksressurser. I mange tilfeller ønsker man å begrense belastningen på nettverksressursene, spesielt fordi slike ressurser ofte deles av flere brukere. Under slike konkurrerende betingelser blir det spesielt vanskelig å designe effektive dataoverføringssystem.

Siena (Scalable Internet Event Notification Architectures) angir en lovende tilnærming til problemet skissert over. I korte trekk støtter Siena innholdsbasert og skalerbar utveksling av datameldinger over WAN, men informasjonsutvekslingssystemet er primært konstruert for distribusjon av små hendelsesmeldinger.

1.2 Siena

Siena er et publish/subscribe hendelsesmeddelselsessystem (eng. event-notification service) konstruert til å skalere over WAN. Systemet distribuerer hendelser (eng. events) som børsmeldinger, sportsnyheter, værmeldinger, leserinnlegg etc.

Systemet består i hovedsak av produsenter, konsumenter og en tjenestemodell. Produsentene deklarerer (advertise) hva slags type data de skal produsere til tjenestemodellen. Konsumentene kan deretter abonnere (subscribe) på dataen som er tilgjengelig.

Siena har tatt i bruk en innovativ type nettverkstjeneste kalt content-based networking. Dette går ut på at det dannes et multicast-tre basert på advertise og subscribe som hendelsene sendes over.

En detaljert beskrivelse av Siena er å finne i kapittel 3.

1.3 Oppgavedefinisjon

Definisjonen på oppgaven er:

> *I denne master-oppgaven skal studentene undersøke om Siena egner seg til innholdsbasert distribusjon av data av høy størrelsesorden (video, etc.) over WAN. Dette innebærer blant annet å avgjøre i hvilken grad man kan overføre data innen gitte tidsfrister med minst mulig belastning på det underliggende nettverket. Studentene skal foreslå forbedringer av Siena der det er nødvendig. Slike forbedringer evalueres ved å simulere overføring av data før og etter at endringer er innført.*

Tittelen på oppgaven er:

> *Skalerbar og QoS-bevisst arkitektur for innholdsbasert distribusjon av store mengder data over WAN.*

Eventuelt:

> *Hvis det er tid kan også en prototyp implementeres.*

1.4 Relatert arbeid

Det er utført en del grundig forskning innenfor Sienaog innholdsbasert distribusjon generelt. For å utvide Siena til å takle data av stor størrelsesorden og for å tilføye tjenestekvalitet har vi funnet følgende artikler som har vært nyttig under arbeidet med oppgaven:

Antonio Carzaniga: Architectures for an Event Notification Service Scalable to Wide-area Networks, [1]: Denne doktorgradsavhandlingen tar for seg arkitekturen til Siena samt nyttige forslag til forbedringer. Hovedmålene til Siena er forestillingen av en allestedsnærværende tjenestemodell som er tilgjengelig fra hvert eneste administrativt domene på WAN. Siena skal være spesielt tilrettelagt for å støtte høyt distribuerte applikasjoner som krever en finkornet samhandling. En slik tjenestemodell utfyller andre mellomvare tjenester som punkt-til-punkt kommunikasjons mekanismer ved å tilby mange-til-mange kommunikasjon og integrasjon fasilitet. Doktorgradsavhandlingen nevner at det ikke er implementert noen form for tjenestekvalitet i arkitekturen.

I vår utvidelse av Siena vil vi legge til tjenestekvalitet i form av at pakker skal leveres til mottaker akkurat på tidsfristen. Doktorgradavhandlingen er skrevet med fokus på distribusjon av små datapakker mens i vår oppgave skal det distribueres data av stor størrelsesorden.

Paolo Costa, Matteo Migliavacca, Gian Pietro Picco, Gianpaolo Cugola: Indroducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms, [18]: Denne artikkelen kombinerer innholdsbasert distribusjon og Epidemisk algoritmer for å gjøre systemet mer robust. Vårt studie kommer ikke inn på akkurat dette emnet om robusthet, men vi har sett en del fordeler med epidemiske algoritmer som gjør det interessant å utvide Siena med dette.

Carsten Griwodz: Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure, [35]: Denne doktorgradsavhandlingen tar for seg studie av en desentralisert caching-orientert infrastruktur for wide-area video-on-demand. Avhandlingen består blant annet av å evaluere ulike distribusjonssystemer, hovedsakelig innenfor streaming av video. Vår oppgave kommer ikke inn på dette, men avhandlingen beskriver en del ulike distribusjonsmekanismer som er utviklet for å optimalisere leveranse av data. Vi har ut i fra dette fått en del interessante ideer som har vært nyttig for oppgaven.

1.5 Vårt Arbeid

Arbeidet i denne oppgaven startet med å tilegne oss kunnskaper om Siena. Dette ble gjort ved henvendelse til Sienas dokumentasjon, studering av kildekode og ved testing av Sienas API i et lite miljø.

Basert på opparbeidede kunnskaper utviklet vi vår egen simulator. Under utviklingen la vi spesielt vekt på at simulatoren skulle være mest mulig autentisk. Etter å ha konkludert med at simulatoren ga et sant bilde av systemet, startet vi å simulere med de satte krav. Ut i fra resultatene bemerket vi oss mangler ved systemet.

Ut i fra Siena sine negative sider kom vi frem til to forskjellige løsninger. Den ene kalte vi Siena Utvidet som er en utvidelse av den genuine Siena. Den andre løsningen er Siena Epidemisk som er en forenkling av den genuine Siena. Her er nettverkets ansvar flyttet fra kjernenettverket ut til endenodene.

Under simulering av løsningene valgte vi å generer topologien til internettverket for å gi et realistisk nettverk. Av den grunn måtte vi utvikle vår egen nettverksgenerator. Oppførselen som simuleres er generert på tilsvarende måte. Simulering av løsningene ble gjort med forskjellige scenarioer og på forskjellige nettverk. Ut i fra dette fikk vi et troverdig resultat. Det ble lagt stor vekt på simuleringen for å se etter nye sider som kunne forbedres. Når løsningene så ut til å svare til det vi hadde forventet, drøftet vi resultatene opp mot hverandre og kom frem til en konklusjon.

1.6 Rapportens struktur

Resten av rapporten beskriver teknologiene og løsningene til problemene beskrevet over. Kapittel 2 gir teoretisk innblikk i de ulike fagområdene vi vil komme inn på i resten av rapporten. I det kapittelet gir vi en generell beskrivelse av de ulike fagområdene som er viktig å ha til grunn for å forstå resten av oppgaven. I kapittel 3 beskriver vi arbeidet som har blitt utført. Vi beskriver hvordan vi har utvidet Siena og hvordan arbeidet rundt epidemisk utvidelse er gjort. Kapittel 4, 5 og 6 presenterer henholdsvis resultater, drøfting, og konklusjoner.

2 Teoretisk bakgrunn

Dette kapittelet gir et teoretisk innblikk i de ulike forskningsområdene som er gjennomgått i denne hovedoppgaven. Påfølgende underkapitler gir en innføring i innholdsbasert distribusjon, tjenestekvalitet, og avstandsvektor. Et innholdsbasert distribusjonssystem baserer seg på at mottakere av pakker har vist sin interesse for innholdet. Pakker vil med andre ord sendes kun dit den er ønsket. Vår løsning vil ta utgangspunkt i arkitekturen til Siena. Siena er et innholdsbasert distribusjonssystem som dekker deler av kravene til vår løsning. Det som Siena avviker fra vårt system er at den er beregnet for nyhetsskripts og har ingen form for tjenestekvalitet implementert. Siena er evaluert og testet for å se om den egner seg for distribusjon av store pakker. Vi har også sett på Epidemiske algoritmer som en utvidelse av Siena. Epidemiske algoritmer er tatt med i systemet for å kunne spre belastningen over nettverket, noe som vil gjøre løsningen mer robust. Unngåelse og kontroll av metning i et nettverk er et viktig tjenestekvalitetsspørsmål. Vi vil derfor beskrive hva som menes med tjenestekvalitet. En viktig del under dette temaet er hvordan pakkene i køene behandles. Ulike køingsmekanismer kan være med på å hindre metning og pakketap. Ved distribusjon av media som f.eks. en musikkvideo eller spillefilm vil pakketap medføre at mediet blir ubrukelig, det er derfor viktig for vårt system at det tilbys en form for tjenestekvalitet.

Å sende en pakke fra A til B er ikke like enkelt som det kanskje høres ut som. Hvordan skal A vite hvor B befinner seg i nettverket? Kanskje B er på motsatt side av nettverket, hvis det er tilfelle, hvilken vei er den korteste? Avstandsvektor som avsluttende underkapittel beskriver hvordan A får vite det den trenger for å sende en pakke til B. Avstandsvektor er en viktig brikke for simuleringsdelen av oppgaven vår. Den inneholder all informasjon de ulike nodene trenger for å gjøre ulike beregninger som er viktige for at systemets krav skal bli opprettholdt.

2.1 Innholdsbasert ruting

Det som skiller innholdsbasert ruting fra tradisjonell ruting, hvor rutingen baserer seg på en destinasjonsadresse [2], er at pakkene blir rutet gjennom nettverket basert på innholdet.

For at rutingen skal kunne finne sted må mottakerne av data fortelle nettverket om hva de er interessert i å motta, og avsenderne må fortelle nettverket om hva som kan sendes. Innholdsbasert ruting er derfor ofte plassert på toppen av en publish/subscribe[1] middelvare tjeneste som inneholder semantikk for å uttrykke behov. Disse behovene genereres hos mottakerne og avsenderne. Etter hvert propageres behovene gjennom nettverket.

Akkurat som for et tradisjonelt nettverk er adresseringsskjemaet realisert med forwarding[2] og ruting. I forwarding-tabellen ligger informasjon om hvor pakkene skal sendes videre. Ruting seansen oppdaterer forward-tabellen ved å utveksle behovene til nærliggende noder.

[1] Publish/subscribe kommunikasjon: utgivere legger inn meldinger med spesifikke "temaer" I stedet for å sende meldinger til spesifikke mottagere. Meldingssystemet kringkaster så innlagt melding til alle interessenter [34]

[2] Forwarding = fremsending/ videresending.

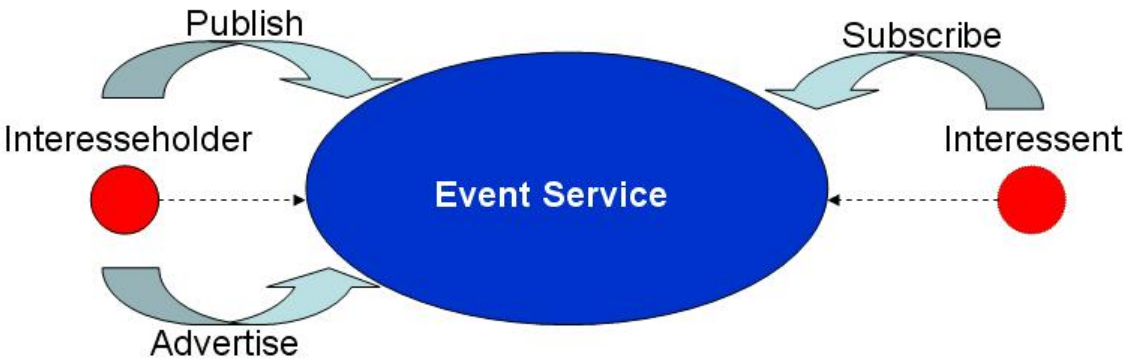
Det finns en rekke forskjellige innholdsbaserte distribusjonssystemer, men det var spesielt en som fanget vår oppmerksomhet. Siena dekket viktige deler av kravene til systemet vårt. Vi så det derfor naturlig for oss å evaluere og utvide Siena til å dekke alle kravene systemet vårt har. Videre så vi det som interessant å teste om Epidemiske algoritmer kan bidra med å spre belastningen over nettet.

2.2 Siena

Siena er et forskningsprosjekt med opphav i en doktorgradavhandling[3] skrevet av Antonio Carzaniga ved University of Colorado. Målet med prosjektet var å designe og konstruere en skalerbar publish/subscribe event service[3] med innholdsbasert ruting.

2.2.1 Siena Event Service

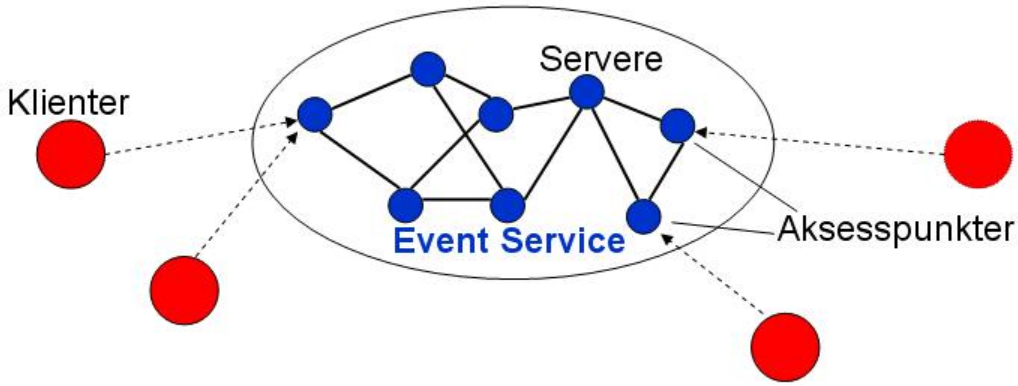
En *event service* er en sentral kontrollenhet for *event notifications*. Noder som bruker *event service* kan være av typen interessenter (eng. *Interested parties*) eller interesseholdere (eng. *object of interest*), eller begge tilfellene samtidig. Interesseholderne forteller hva de har tenkt å publisere ved hjelp av annonseringer, mens interessenter forteller hva de er interessert i ved hjelp av abonnementer. Dette kan for eksempel være en børsmelding som skal formidles. Interesseholderne kan sende tre typer meldinger: advertise, unadvertise og publish. Interessenter kan sende to typer meldinger: subscribe og unsubscribe (se fig. 2-1).



Figur 2-1: Siena Event Service

I Siena består en såkalt *event service* av mange sammenkoblede *event*servere (se fig. 2-2). En eventserver kan ses på som en ruter som har som oppgave å videresende pakker dit de skal. Meldinger kan komme i form av advertisements, subscriptions, eller publications. Eventserverne har som oppgave å utføre forskjellige opperasjoner avhengig av hvilken type melding som blir sent.

[3] Hendelses-meddelelse tjeneste.



Figur 2-2: Siena Event Service med eventservers

2.2.2 Interface

Siena har til sammen fem ulike meldinger som interessenter og interesseholdere benytter:

1. *Publish* (*notification n*)
2. *Subscribe* (*URI subscriber, pattern p*)
3. *unSubscribe* (*URI subscriber, pattern p*)
4. *Advertise* (*URI subscriber, filter f*)
5. *unAdvertise* (*URI subscriber, filter f*)

Publish består av selve dataen. Dette blir kun sendt til interessenter som har sendt en subscribe melding for å abonnere på denne dataen. Før verken publish eller subscribe kan sendes, må det bli sendt ut en advertise melding i nettverket som forteller hva en interesseholder har som hensikt å sende ut i fremtiden, hvis og bare hvis det er noen som er interessert i dette.

En event notification er selve dataen i en publish melding. I Siena kan dette være f.eks. en børsmelding som forteller at kursen på Oslo Børs har steget med 5%. En *event notification* består av et sett med attributter. I Siena inneholder hvert attributt navn, type og verdi. I en *event notification* blir attributter unikt identifisert av navnet. Hvis β er et attributt for en *event notification* da er navnet, typen og verdien henholdsvis β.name, β.type og β.value, der β.name identifiserer attributtet. Publiseringsmeldingen inneholder selve rådataen som skal ut til interessentene.

Et *event* filter er en konjunksjon av atributter som til sammen definerer *event notifications. Advertise*-meldingene inneholder filtere for å angi hva interessenter kan abonnere på.

Et *pattern* av *events* er en kombinasjon av et sett med *event* filtre. Abonnement meldingene inneholder *patterns*. Dette gjør det mulig for en interessent å abonnere på flere ting med en melding.

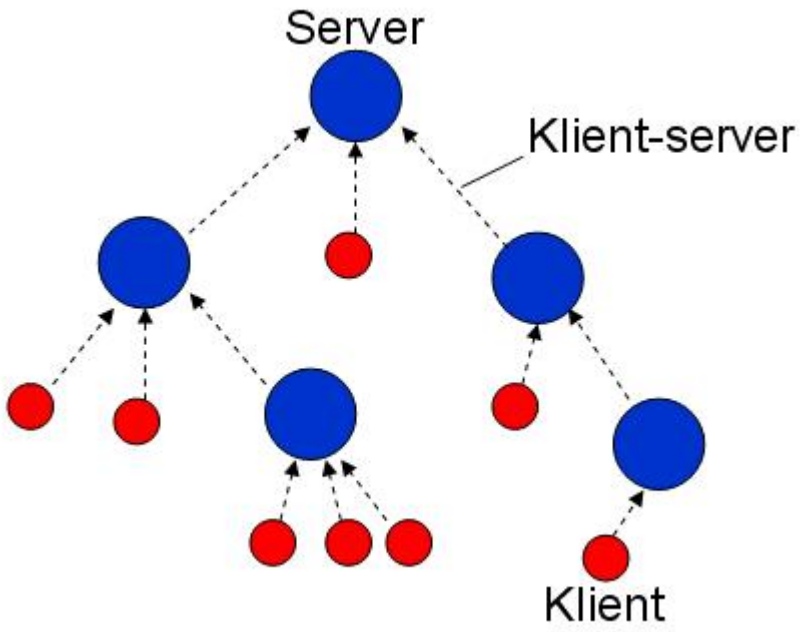
2.2.3 Servertopologi

Siena distribuerer data gjennom en sentral kontrollenhet bestående av sammenkoblede servere. Disse serverene må knyttes sammen ved hjelp av en sammenkoblingstopologi, som for Siena er tilgjengelig i tre varianter: hierarkisk

klient/server, asyklisk peer-to-peer og generisk peer-to-peer. I tillegg finnes det også en hybrid arkitektur som kombinerer to eller flere av de andre arkitekturene.

Hierarkisk klient/server

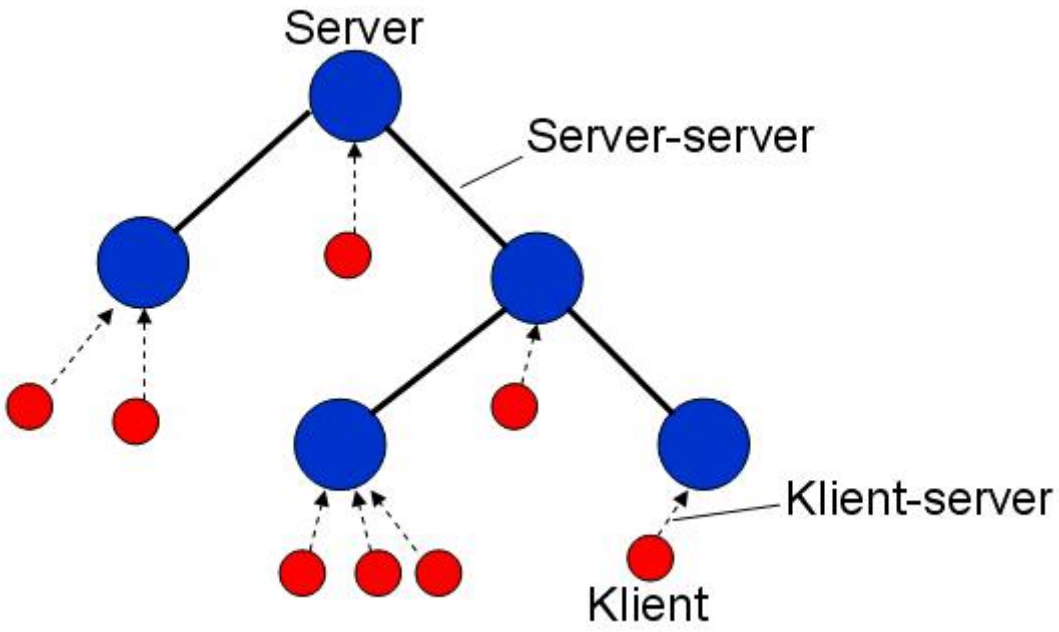
I denne type topologi har hver server et antall klienter som kan være interessenter, interesseholdere, eller *event* servere. I denne topologien ser serverne ikke forskjell på klienter og andre servere, den ser på alle som klienter. Det er da mulig for en forelder server å mota *notifications*, abonnementer, og annonser fra alle, men den vil bare sende *notifications* til sine klienter.



Figur 2-3: Hierarkisk klient/server

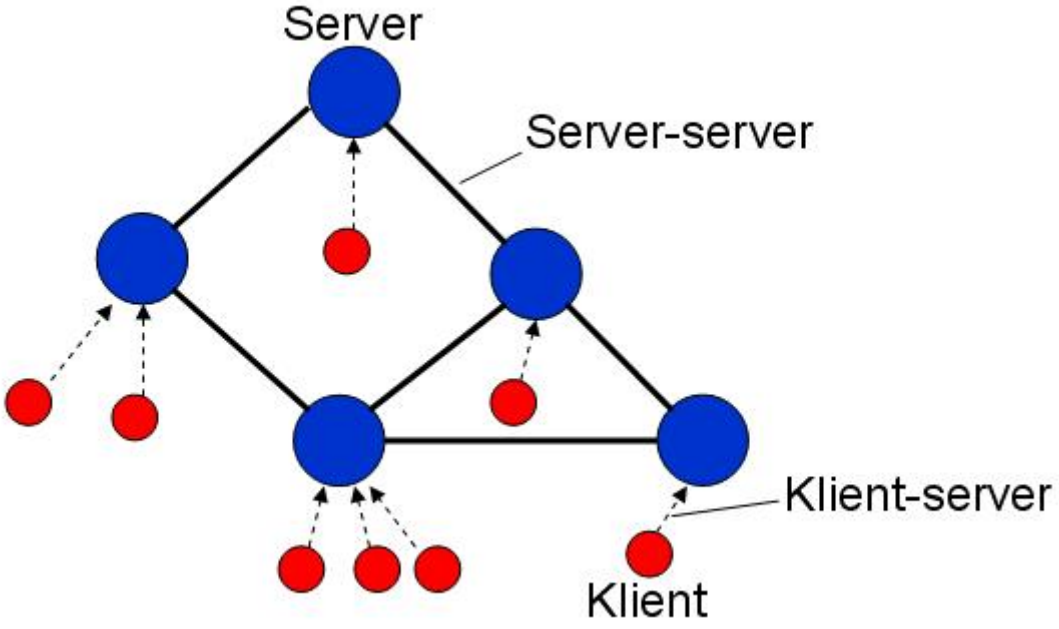
Asyklisk peer-to-peer

I denne topologien kommuniserer serverne med hverandre som likemenn (eng. peer). Det gjør det mulig for *notifications*, abonnementer, og annonsser å gå begge veier. Er klienter koblet mot en server (klient-server kommunikasjon) går meldingene kun den ene retningen, fra klient til server.



Figur 2-4: Asyklisk peer-to-peer

Generisk peer-to-peer
Generisk peer-to-peer topologi har samme type kommunikasjon som asyklisk peer-to-peer, men serverne kan her kobles mot flere servere samtidig.



Figur 2-5: Generisk peer-to-peer

2.2.4 Ruting og prosessering

For å kunne sende *notifications* til riktig mottaker må det rutes på en spesiell måte. Siena presenterer to algoritmer som sprer korrekt informasjon gjennom nettverket: *subscription forwarding*[4] og *advertisement forwarding*[5].

[4] Abonnement fremsending/ videresending.

Subscription forwarding
Denne algoritmen benytter abonnementer for å sette opp stier hvor *notifications* skal sendes. Hvert abonnement blir lagret og videresendt fra initiativtager til alle serverne i nettverket. Det blir hermed dannet et tre fra abonnenten til alle serverne i nettverket. Dette treet benyttes ved publisering, og *notification* rutes til de som abonnerte ved å følge stien i motsatt retning.

Advertisement forwarding
Som navnet antyder brukes advertisements i denne algoritmen for å sette opp stier for subscriptions. Interesseholder sender en annonse. Denne rutes til alle nodene I nettet slik at det dannes et *span-tre*. Nodene som ønsker å motta data abonnerer på dette. Abonnement meldingen rutes i motsatt retning tilbake til interesseholder slik at det dannes et optimalt *span-tre* fra interesseholder og til alle interessenter. Meldingen som sendes I motsatt retning sendes bare videre dersom den aktuelle noden mener den er på den korteste stien mellom interesseholder og interessent. Abonnement meldingen sendes tilbake for å aktivere en sti mellom noder. *Notifications* kan nå videresendes gjennom de aktiverte stiene.

2.2.5 Implementasjon

Implementering av Siena tilbyr et programmeringsgrensesnitt (API) for både Java og C++. Begge implementeringene er implementert i henhold til resultater fra studiene utført at Carzaniga[1].

API for C++ er basert på den asykliske peer-to-peer arkitekturen, mens API for Java baserer seg på hierarkisk klient/server arkitektur. De to implementeringene er kompatible med hverandre slik at det er mulig med en hybrid løsning som inneholder servere fra begge arkitekturene.

2.3 Epidemisk distribusjon

Epidemiske algoritmer er ikke i seg selv et innholdsbasert distribusjonssystem, grunnen til at vi har plassert kapittelet om epidemiske algoritmer under innholdsbasert distribusjon er fordi vi så på disse algoritmene som en mulig løsning for å spre ressursbruken jevnt over hele nettet.

Epidemiske algoritmer er en populær løsning for å spre oppdateringer på en skalerbar og effektiv måte. Algoritmene ble utviklet for å effektivt håndtere konsistens i replikerte databaser[3]. I senere tid har algoritmene blitt brukt for å løse mange forskjellige problemer med å propagere informasjon i internettverk eller i ad hoc nettverk pga dens enkle implementering, robusthet, høy feiltoleranse og at den er lite kostbar å kjøre[4,6,6].

> ***epidemi*** *(epidemi´) m1 (fra gr, av epi- og demos 'folk') (smittsom) sykdom som sprer seg hurtig, farsott, jf endemi og pandemi*[7]

Epidemiske algoritmer er inspirert av teorien bak epidemier, som omhandler spredning av smittsom sykdom. I vårt tilfelle er det nodene som prøver å ”smitte” så

[5] Annonse fremsending/ videresending.

mange andre noder med så få meldinger som mulig. Noden som skal spre oppdateringer vil videre bli omtalt som *smittsom*. Noder som ikke har blitt oppdatert ennå, omtales som *mottakelig*.

Det finnes tre måter for nodene å spre oppdateringene sine: *push, pull* eller *push-pull.*

Push: Ved push sender en smittsom node oppdateringen til en tilfeldig annen node. Hvis noden er mottakelig, overføres oppdateringen. Noden som er smittsom sender så videre til en ny node.

Pull: Ved pull metoden kontakter de mottakelige nodene en tilfeldig annen node i håp om å få mottatt oppdatering. Treffer noden en smittsom node, overføres oppdateringen.

Push-Pull: Ved push-pull metoden sendes oppdateringer fra bade smittebærer og den noden den kommuniserer med.

Det finnes mange variasjoner av epidemiske algoritmer, men de fleste har tatt utgangspunkt i de to hovedmetodene: *anti-entropi* og *ryktespredning*.

2.3.1 Anti-Entropi

Anti-entropi er en populær metode for å holde databaser svakt (eng. weakly consistent) oppdatert. I anti-entropi er det en eller flere noder som er smittsom. Dette er fast og kan ikke endres underveis. Mottakelige noder som blir smittet kan med andre ord ikke gå over til å bli smittsom ved oppdatering. Anti-entropi bruker både push, pull eller push-pull. Brukes en av disse viser det seg at oppdateringene vil spres til alle nodene hvis bare en node er smittsom i starten[8]. Anti-entropi garanterer at alle noder får oppdateringen, men er ikke en rask metode for å spre oppdateringer.

2.3.2 Ryktespredning

Ryktespredning er en mye brukt metode for å spre oppdateringer på en hurtig måte. Ved ryktespredning sendes oppdateringer fra smittsom node til en tilfeldig annen mottagelig node. I motsetning til anti-entropi hvor antall smittsome noder var fast, omformes noden som mottok oppdateringen fra mottakelig til smittsom. Spredningen av oppdateringer vil dermed øke eksponentielt.

Man kan ikke være sikker på at alle nodene mottar oppdateringene med denne metoden. Dette kan forklares med at en smittsom node mister interessen for å spre oppdateringer videre med en sannsynlighet på $1/k$ hvis den møter på en node som har fått oppdateringen fra en annen smittsom node. Prosentvis antall noder som forblir mottakelige når spredningen er avsluttet kan oppfylles (utregnes) med den rekursive likningen[8]:

$$s = e^{-(k+1)(1-s)}$$

Hittil i teoretisk bakgrunn kapittelet har vi beskrevet viktige deler ved Innholdsbasert distribusjon, og at Siena er en arkitektur vi skal utvide. En videre utvidelse vi har sett på er ved bruk av epidemiske algoritmer. Vi nevnte også at Siena ikke har noen form for tjenestekvalitet implementert i arkitekturen. Oppgaven vår går blant annet ut på å lage en QoS-bevisst arkitektur, det vil med andre ord si at systemet krever at en form

for tjenestekvalitet blir implementert. Videre i rapporten vil vi beskrive generelt hva tjenestekvalitet er og hvilken mekanisme innen tjenestekvalitet som trengs for at kravet skal bli oppfylt.

2.4 Tjenestekvalitet

Tjenestekvalitet (QoS – Quality of Service) er et fagord som brukes i forbindelse med nettverk. Tjenestekvalitet sier noe om hva et nettverk kan garantere av tjenester[9]. Å tilby tjenestekvalitet i ethvert nettverk starter med å kontrollere og hindre metning i nettet.

> **Tjenestekvalitet***: en mye brukt betegnelse som refererer til ytelsesatributter på en ende til ende tilkobling. En tjenestekvalitets definisjon for data vil adressere atributter som feilrate, pakketap rate, gjennomstrømning, og forsinkelse.*[10]

Et nettverk med tjenestekvalitet har mulighet til å levere data trafikk med en minimum mengde forsinkelse i et miljø hvor mange brukere deler samme nettverksressurser. Blir tjenestekvalitet innlemmet i et nettverk, er dette for å forsikre at nettverket kan levere forventet trafikkmengde.

Det er ikke lett å oppnå tjenestekvalitet i et IP nettverk. Ved bruk av rutere kan pakker velge forskjellige stier, noe som skaper uforutsigbare forsinkelser. IP er koblingssløs, den kan ikke avsette eller garantere båndbredde. IP bruker variabel pakkestørrelse, noe som gjør trafikkmønster uforutsigbare. Pakker fra mange kilder går igjennom delte linker og kan overbelaste rutere, noe som forårsaker metning, pakke tap, rettransmisjon, forsinkelse, etc. [10]

Eksempler på ulike typer tjenestekvalitet:

- Metningskontroll – for å redusere metning når det skjer eller for å forhindre at metning skal skje.
- Klassifisering og køingsmekanismer – klassifiserer trafikk, køer eksisterer for hver klassifisering, og køer med høyest prioritet får sende først.
- Båndbredde reserveringsteknikker – for å forsikre pakke leveranse.
- Pakke merking og etikett svitsjing – for å spesifisere stien gjennom nettverket.

Tjenestekvaliteten vi har utvidet Siena med er at pakker skal komme frem til mottaker akkurat på tidsfristen. For at dette skal kunne oppfylles må pakker som plasseres i køene hele tiden kunne omrokeres slik at den pakken med kortest tidsfrist blir sendt ut først. Dette krever at det finnes en køingsmekanisme implementert i nodene. Vi vil i påfølgende kapittel fortelle om ulike køingsmekanismer, hvilken vi har valgt for oppgaven blir beskrevet i kapittel 3. Neste kapittel er tatt med for å gi en smakebit på noen av de ulike køingsmekanismer som finnes.

2.4.1 Køingsmekanismer

Pakker som skal sendes fra en node blir gjerne plassert i køer, disse køene har ofte en fast lengde. Køingsmekanismer brukes da for å bestemme hvordan pakker skal håndteres i køen. Problemet med køer er at når en kø er full har det oppstått en metning, og ankommende pakker vil bli kastet. Køingsmekanismer brukes for å

bestemme hvilke pakker som skal sendes ut fra køen(e) først. Enkelte køingsmekanismer kontrollerer hvilke pakker som skal kastes i tilfelle en metning oppstår. Hvordan dette blir gjort er forskjellig for hver køingsmekanismene.

Det finnes en rekke køingsmekanismer å velge mellom. Noen av de mest sentrale er: *FIFO (first-in, first out), prioritetskøing, rettferdig køing, og vektet rettferdig køing (WFQ -eng. weighted fair queuing).* Hvilken som passer best i de ulike tilfellene kommer helt an på hvordan systemet er oppbygd og hva pakkens innhold har av betydning for systemet.

Generell informasjon om køingsmekanismer er å finne i [11]. Denne boken har blitt brukt for å kvalitetssikre informasjonen nedenfor.

FIFO
I FIFO (First-in, first-out) blir pakker behandlet i den rekkefølgen de ankommer. Problemet med denne mekanismen er at den bryr seg ikke om hvilken betydning (prioritet) pakken har for systemet når en metning inntreffer. Ankommer en høytprioritert pakke, og metning har oppstått, kastes denne pakken. FIFO er avhengig av at endesystemet kontrollerer metningen vha metningskontrollmekanismer.

Prioritetskøing
Prioritetskøing benytter seg av flere køer med forskjellige prioriteter. Pakker som befinner seg i den køen som har høyest prioritet blir sendt ut først. Er denne køen tom, fortsetter den med den køen som har nest høyest prioritet osv. De ulike prioritetskøene følger prinsippet til FIFO. Problemet med denne mekanismen er at det kan forekomme at pakker som ligger i de køene med lav prioritet aldri blir sendt ut. Dette er noe som forekommer hvis det er stor pågang med høyt prioriterte pakker. Ved metning i nettet blir pakker kastet fra køene med lavest prioritet.

Rettferdig køing
Denne mekanismen løser problemet ved at pakker som ligger i lavt prioriterte køer risikerer å aldri bli sendt. Dette problemet blir løst ved at alle køene får sendt etter tur (round-robin[6]). Det kan oppstå andre problemer med denne mekanismen. Hvis pakker har variabel størrelse og hver kø kan sende en pakke om gangen, vil noen køer ta lang tid. Noen køer kan også være fullere enn andre og vil derfor trenge mer prosesseringstid. Denne mekanismen tar ikke hensyn til at noen køer trenger mer prosesseringstid. Ut i fra navnet vil mekanismen behandle hver kø rettferdig, dvs lik prosesseringstid vil bli fordelt.

Vektet rettferdig køing
Vektet rettferdig køing kan ses på som en kombinasjon av prioritetskøing og rettferdig køing. I denne mekanismen fordeles prosesseringstid til alle køene slik at ingen kø sulter, dvs at noen køer kan få mer prosesseringstid enn andre. Ut i fra navnet blir det gitt en vekt til køene for å gi noen køer høyere prioritet enn andre. For eksempel en kø kan få halvparten av tilgjengelig båndbredde og andre køer vil dele resterende båndbredde. Vektet rettferdig køing vekter trafikken slik at lav-

[6] En metode for å garantere at et antall enheter vil få mulighet til å få sin tur til å sende. Round-robin metoden krever at hver eneste enhet får sende etter tur. Etter den siste i listen har sendt får den første sende, og prosessen starter på nytt [x2].

båndbredde trafikk får en rettferdig prioritet. Hvis høyt prioriterte køer er tomme vil lavere prioriterte køer bruke dens ressurser.

Innholdsbasert distribusjon og herunder Siena og Epidemiske algoritmer, samt tjenestekvalitet er hittil beskrevet. Neste kapittel omhandler avstandsvektor. Avstandsvektor har blitt brukt for å formidle informasjon til alle nodene i nettverket. Informasjonen kan bestå av ting som nodene trenger for å gjøre forskjellige beregninger, eller ta forskjellige avgjørelser. Vi vil i dette kapittelet fortelle om hvordan avstandsvektor fungerer og virkemåten til algoritmen som er benyttet i forbindelse med dette. I kapittel 3 er det beskrevet mer om hvordan arbeidet rundt dette er gjort.

2.5 Avstandsvektor

Avstandsvektor (eng. Distance Vector) er den rutingmetoden som har lengst tradisjon på Internett. Avstandsvektor er en distribuert adaptiv algoritme for å finne korteste avstand (shortest-path) mellom alle nodepar i et dynamisk nettverk[12]. Navnet avstandsvektor kommer fra måten rutinginformasjonen distribueres på. Den distribueres omtrent på samme format som rutingtabellen som en rekke med (destinasjonsnett, avstand)-par[13].

Oppførselen til avstandsvektor i korte trekk:

1. Hver node/ruter vet i starten kun kostnadden til sine nærmeste naboer. Dette står oppført i rutingtabellen til noden/ruteren. Noder som ikke er direkte tilknyttet får tildelt en kostnadd på ∞.

2. Hver node/ruter sender en melding til sine naboer. Meldingen inneholder dens personlige liste med avstander (kostnadder) til sine naboer. Si at node A vet den kan nå node B med 1 hopp og node C med ∞ hopp (C er ikke nabo til A). I B sin melding står det at den kan nå node C med 1. A vet dermed at den kan nå C med 2 hopp via B. Nodene/ruterne oppdaterer sine rutingtabeller hver gang det kommer en ny melding. Hvis det skulle komme en melding til A fra node D at den kan nå C med 2 hopp (3 hopp til sammen) vil A ignorere denne opplysningen ettersom den har en kortere sti til C lagret i rutingtabellen sin. Nodene vil dermed vite korteste vei til alle nodene i nettverket.

3. Hver node/ruter vil etter hvert få oppdatert rutingtabellene sine til å inneholde kostnadden og nest hopp for å nå alle nodene i nettverket. Hvis det ikke er noen forandringer i topologien til nettverket, er dette gjort på noen få meldinger.

Det positive med en distribuert algoritme som denne er at den gjør det mulig for alle nodene å ha et konsistent bilde av nettverket uten noen form for sentralisert kontroll.
En detaljert beskrivelse av ruting algoritmen beskrevet over er å finne i [14] på side 274 – 278.

Det er to forskjellige omstendigheter hvor en gitt ruter bestemmer seg for å sende en oppdatering av rutinginformasjon til naboene sine: *periodisk oppdatering og trigger oppdatering* (hver gang det skjer en endring)[14].

2.5.1 Periodisk oppdatering

Hver node sender automatisk en oppdatering i perioder. Perioden kan variere fra noen få sekunder til flere minutter avhengig av ruting protokollen. Periodisk oppdatering er til nytte av to grunner:

- Forteller andre noder at den fortsatt er i live. Dette kan brukes for å oppdage noder som er nede.
- Forsikrer at nodene får informasjon som trengs hvis dens stier blir utilgjengelig.

2.5.2 Trigger oppdatering

Trigger oppdatering sendes når det skjer en forandring i rutingtabellen til en node. Hvis en node A mottar rutinginformasjon fra en nabo, som kan forårsaker at A må endre ruting tabellen sin. Hvis dette skjer må A sende oppdatering til sine naboer slik at disse kan oppdatere sine rutingtabeller.

2.5.3 Dijkstra's Shortest Path

Dijkstra's algoritme for å beregne korteste vei mellom noder er en mye brukt algoritme fra graf teori. Den brukes blant annet i linktilstandsruting (eng. Link State) og avstansvektor (eng. distance vector).

Algoritmen fungerer slik (hentet fra [14]):
N – er alle nodene i grafen.
l(i,j) – er positiv kostnadden til linken mellom node *i, j* ε *N. H*vis det er ingen link mellom *i* og *j* er *l(i,j)* = ∞.
s – noden som starter algoritmen for å finne korteste vei til alle andre noder i *N*.
M – er antall noder som så langt har blitt tatt med i beregningen av korteste vei.
C(n) – er kostnadden av veien fra *s* til hver node *n*.

M = {*s*}
for hver *n* i *N* – {*s*}
    C(n) = *l(s,n)*
while (*N* != *M*)
    M = *M U* {*w*} slik at *C(w)* er minimum for alle *w* i (*N* - *M*)
    for hver *n* i (*N* - *M*)
        C(n) = MIN(*C(n), C(w)* + *l(w,n))*

M inneholder noden *s* og setter opp tabell med kostnadden (*C(n)s*) til andre noder ved bruk av dens kjente kostnadd til sine direkte tilknyttede nabo noder. Det blir så sett etter den noden som er innen rekkevidde med lavest kostnadd (*w*), dette legges til *M*. Til slutt oppdateres tabellen med kostnadder ved å ta kostnadden for å nå noder via *w* i betraktning. I siste linje i algoritmen, velges en ny vei til node *n* som går via *w.* Dette gjøres hvis total kostnadden ved å gå fra kilden til *w* og linken fra *w* til *n* er mindre enn den opprinnelige veien til *n*. Dette blir gjentatt helt til alle noder er inkorporert i *M*.

Dijksra's algoritme har flere metoder/algoritmer for å beregne ruting tabellen til nodene (f.eks. *forward-search* og *in-depth).* Til vår oppgave valgt vi *in-depth* Metoden/algoritmen. Virkemåten og hvordan vi har implementert denne er beskrevet i kapittel 3.

2.6 Oppsummering

I dette kapittelet har vi gitt en smakebit av de ulike forskningsområdene vi har kommet inn på under arbeidet med oppgaven. Det som kjennetegner et innholdsbasert distribusjonssystem er at pakkene blir rutet gjennom nettverket basert på innholdet i pakken, ikke en adresse slik som det gjøres i tradisjonell ruting. Siena ble evaluert bl.a. pga. at arkitekturen går inn under innholdsbasert distribusjon og er skalerbar. Siena har en rekke Eventservere, som har i oppgave å utføre forskjellige opperasjoner ut i fra hvilken pakke de mottar. Pakker de mottar kan være enten en advertise, unadvertise, subscribe, unsubscribe eller publish. Både subscribe- og advertisemeldinger kan brukes for å sette opp stiene melding skal følge. Eventservere benyttes her for å formidle meldingene videre til riktig sted. Dette blir kalt subscription forwarding og advertisment forwarding. Vi har også beskrevet de tre ulike servertopologiene som henholdsvis er hierarkisk klient/server, asyklisk peer-to-peer, og generisk peer-to-peer.

Epidemiske algoritmer er inspirert av teorien bak epidemier, som går ut på å spre smittsom sykdom. Vi har beskrevet dens to hovedmetoder: anti-entropi og ryktespredning. Disse metodene kan benytte seg av tre ulike måter for å spre oppdateringene sine, som er push, pull, eller push-pull.

Køingsmekanismer er en tjenestekvalitet som vi kommer innom i dette prosjektet. Enkelte køingsmekanismer kontrollerer hvilke pakker som skal kastes i tilfelle en metning oppstår. Vi har i dette kapitelet beskrevet disse mekanismene: FIFO, prioritetskøing, rettferdig køing, og vektet rettferdig køing

Helt til slutt tok vi for oss Avstandsvektor og algoritmen den bruker for å kartlegge avstanden mellom alle nodene. Avstandsvektor blir brukt for å finne den korteste avstand mellom de forskjellige noder.

I neste kapittel beskriver vi forskningsmetodikken som er brukt ved denne oppgaven. Her beskrives materialer, verktøy, instrumenter, metoder, prosedyrer, formler og algoritmer som er benyttet.

3 Arbeid utført

I dette kapitlet vil vi gi en nærmere presentasjon av det arbeidet som er utført i løpet av studiet. Vi vil først gi et innblikk i hvordan simulatoren er konstruert. Deretter vil vi forklare hvordan tidsfristbasert distribusjon fungerer, og hvilke løsninger vi har kommet frem til innen dette problemområdet. Følgende delkapitler vil greie ut om de forskjellige informasjonsutvekslingssystemene som vi har foreslått, og i den anledning vil vi også se på muligheten for å integrere tidsfristbasert distribusjon for hver enkel løsning.

3.1 Simulering

Evaluering av et skalerbart system over WAN i praksis krever en rekke noder spredt over et stort geografisk område. Internettverk som støtter disse kravene er ofte dyre og vanskelig å kontrollere, noe som gjør at slike internettverk sjeldent er tilgjengelig for eksperimentelle formål. Studiet vil derfor evaluere algoritmene og arkitekturene i simulatorer utviklet av studentene.

Simulatorene trenger i hovedsak to modeller: (1) en modell av en internettverkstopologi som gjenspeiler virkeligheter, og (2) en modell av oppførselen til nodene i informasjonsutvekslingssystemet. Metodikken for generering av disse to modellene er utledet i de neste to delkapitlene. Deretter vil de viktiste momentene ved simulatorene bli belyst.

3.1.1 Generering av internettverkets topologi

Utfordringen bak generering av en internettverkstopologi ligger i å få en realistisk topologi som i størst mulig grad gjenspeiler et virkelig internettverk. Denne utfordringen kan deles inn i tre deler: (1) generering av nodenes plassering, (2) generering av linker mellom nodene, og (3) generering av internettverkets egenskaper som båndbredde og latenstid. I vårt tilfellet er det også valgt en tjenestemodell (kalt ”Event Service Topology” i Siena) som skal ligge på toppen av internettverket.

Vi sjekket først muligheten for å bruke allerede eksisterende generatorer for å generere et internettverk med tilhørende tjenestemodell. Blant annet ble *Inet Topology Generator (Inet)* [28] og *Georgia Tech Internet Topology Models (GT-ITM)*[27] evaluert i den sammenhengen. Siden disse generatorene var lite fleksible og derfor vanskelige å tilpasse vårt forhold lagde vi vår egen generator.

Det finnes en rekke forskjellige tjenestemodeller som kan legges på toppen av et internettverk. I mange tilfeller blir flate grafer brukt, mens i andre tilfeller blir det brukt hierarkiske grafer som Transit-Stub eller N-Level [20]. Siden vår forskning baserer seg på distribusjon over WAN er det naturlig å velge en Transit-Stub tjenestemodell. Transittnettverket vil dermed bestå av tynne transittservere sammenkoblet i en graf som har som eneste formål å rute trafikk over WAN, mens stubbnodene blir interessentene.

Plasseringen til transittnodene er tilfeldig generert innenfor et gitt rektangulært område. Til dette er det brukt Microsoft .Net Framework Random Number Generator[32] som baserer seg på en matematisk algoritme. Siden generatoren er

matematisk kan man ikke garantere fullstendig tilfeldighet, men nummerene er tilstrekkelig tilfeldig for praktiske formål[32].

Plasseringen til stubbnodene er også tilfeldig generert med samme generator[32], men nodene er definert til å være innen området

$$A_{stub} = A_{total} \cap \overline{A_{transit}}$$

hvor $A_{transit}$ er området hvor transit-nodene er definert, og A_{total} er det totale området hvor det kan defineres noder innenfor. A_{total} har dobbelt så stort areal som $A_{transit}$.

For generering av linker (også kalt kanter) mellom transittnodene ble det brukt metoden Waxman 1[20]. Denne metoden gjenspeiler i stor grad realistisk generering av grafer ved at sannsynligheten for at det skal bli opprettet en link fra *u* til *v* er

$$P(u, v) = \alpha e^{-d / (\beta * L)}$$

hvor $0 < \alpha,\beta \leq 1$, og hvor verdien *d* er den aktuelle linkens avstand (Eucledian Distance) fra *u* til *v*:

$$d(u,v) = \sqrt{(u_1-u_2)^2+(v_1-v_2)^2}$$

Verdien *L* representerer den maksimale avstanden som er mulig mellom noden *u* og *v* innenfor rektangelet hvor grafen er definert. Den lengste avstanden innen et rektangel er diagonalen, noe som kan regnes ut med Pythagorean Theorem ved å betrakte *L* som en hypotenus:

$$L^2 = K^2 + K^2$$

En økning i *α* vil øke den generelle sannsynligheten for at linken skal bli opprettet, mens en økning i *β* reduserer sannsynligheten for opprettelse av linker hvor avstanden mellom nodene er stor. I vårt tilfellet har vi satt *α=0.5* og *β=0.2*, noe som vil si at det er en liten sannsynlighet for at linker mellom noder med stor avstand skal bli opprettet.

Serverne i transittnettverket er knyttet sammen på en måte som avbilder den underliggende nettverkstopologi, og på den måten forenkles simuleringen ved å redusere antall attributter. Man trenger med andre ord ikke simulere både tjenestemodellen og internettverket.

Hver av stub-nodene er koblet til transittnettverket ved å opprette en link til den nærmeste transit-noden. Forholdet mellom en stub-node og transittnode er 1-til-1, det vil si at hver stubbnode bare har én link til én transittnode.

Vi genererer to attributter for linkene: latenstid(n_1, n_2) og båndbredde(n_1, n_2). Latenstid er tiden det tar for å sende data på linken mellom nodene n_1 og n_2, og den er målt i tid (s). Båndbredde på linken mellom nodene n_1 og n_2 er målt i bits/s, som er definert som datamengde (bits) overført per tidsenhet (s). Vi går ut i fra at linkene er

symmetriske, det vil si at symmetri eksisterer dersom følgende logiske utsagn er sann:

$$\text{latenstid}(n_1, n_2) = \text{latenstid}(n_2, n_1) \text{ ^ båndbredde}(n_1, n_2) = \text{båndbredde}(n_2, n_1).$$

Latenstid på en link varierer etter type materiell linken er laget av og fysisk lengde på linken[14]. Vi har forutsatt at linkene er av typen OC-1 (Optical Carrier level 1), og linkens latenstiden er dermed satt til å være proporsjonal med aucledian distance med et gjennomsnitt på 10 ms. En link av typen OC-1 har en båndbredde på 51.85 Mbps, men siden vi bare er interessert i linkens tilgjengelige båndbredde blir verdien redusert med en verdi som baserer seg på en kvantitativ undersøkelse av lediger ressurser på internettverk.

Etter at alle linker og noder med tilhørende attributter er generert blir det sjekket om grafen som utgjør nettverket henger sammen (med dette menes at det går en sti mellom hvert nodepar). For å sjekke dette ble det implementert en modifisert og rekursiv utgave av algoritmen Dept-First Search (DFS). Denne algoritmen brukes vanligvis til å travasere grafer på en systematisk måte, men vi har utvidet den til å kontrollere konnektivitet ved å sjekke om antall noder den er innom under travaseringen tilsvarer antall genererte noder.

Se vedlegg B for en demonstrasjon av nettverksgeneratoren.

3.1.2 Generering av hendelser

Hendelsesgeneratoren er et program utviklet av studentene som på en realistisk måte genererer hendelser for informasjonsutvekslingssystemer. Hendelsesgeneratoren er laget i to versjoner: en som genererer hendelser som etterligner tiltenkt bruk av Siena, og en som genererer hendelser av data av stor størrelsesorden.

Oppførsel til Siena

Siena bruker fem ulike type meldinger: *publish(notification n)*, *subscribe(string identity, pattern expression)*, *unsubscribe(string identity, pattern expression)*, *advertise(string identity, filter expression)* og *unadvertise(string identity, filter expression)*.[1]

En stubbnode som averterer en kategori, må etter hvert publisere denne. Forholdet mellom disse to meldingstypene er lik Siena.

Vi har satt følgende forutsetning for generering av hendelser for Siena:

- Ingen stubbnode kan abonnere på en kategori som ikke har blitt annonsert tidligere.
- Ingen stubbnode kan publisere en kategori som den ikke har annonsert tidligere.
- En stubbnode kan ikke annonsere en kategori som den tidligere har annonsert.
- En stubbnode kan ikke abonnere på noe som den har abonnert på tidligere.
- Ut i fra dette er det naturlig at en *advertise*-melding kommer først.

Tabell 3-1: Eksempel på en hendelse med forklaring

Navn	Eksempelverdi	Forklaring
Tick	1675	Tidspunktet hendelsen skal skje
NodeID	18	Stubbnoden som utfører hendelsen
Action	publish	Hendelsen stubbnoden skal utføre
Filter	Thriller18	Kategorien som hendelsen bruker.
Par	3050	Størrelsen på pakken (kun ved publisering)
Tidsfrist	3115	Tidsfristen for når pakken skal være fremme til mottaker

Oppførsel store mengder data
I Siena sendes det ut ca. en *advertise*-melding etterfulgt av ca. ti publish meldinger. For å tilpasse generatoren til å generere modell for store mengder data, endret vi forholdet mellom *advertise*- og *publish* meldingene til 1-1.

Generatoren har følgende begrensningene:

- Advertise blir alltid sendt først.
- Subscribe blir så sendt fra et tilfeldig antall noder.
- En stubbnode kan kun subscribe på noe som er avertert tidligere.
- Blir det sendt ut en publish, kan ikke noen node subscribe på denne ved en senere anledning.
- Ingen stubbnode kan publisere en kategori som den ikke har avertert tidligere.
- En stubbnode kan ikke avertere en kategori som den tidligere har avertert og ikke publisert ennå.
- En stubbnode kan ikke abonnere på noe som den har abonnert på tidligere.

3.1.3 Simulator

I løpet av studiet har vi laget simulatorer for følgende informasjonsutvekslingssystemer: Siena, utvidet versjon av Siena og epidemisk versjon av Siena. Dette delkapitlet tar for seg de viktigste aspektene ved disse simulatorene.

Det foreligger en rekke rammeverk for simulering av informasjonsutvekslingssystemer, men vi har valgt å lage vår egen simulator fra bunnen av. Grunnen til dette er at vi da er mer fleksible med tanke på implementasjon av særegne mekanismer, samt at vi får et grensesnitt som er tilpasset vårt behov.

Hovedessensen i hver av simulatorene er funksjonaliteten, metodene og protokollene som utgjør informasjonsutvekslingssystemet som skal simuleres. Denne kjernen krever modell for både nettverkstopologi og hendelser. Med tanke på modell for hendelser vil simulatoren bare kunne bruke vårt eget, men for modell av nettverkstopologi er det implementert parser for både vårt eget format, *Inet Topology Generator (Inet)* [28] og *Georgia Tech Internet Topology Models (GT-ITM)*[27].

Simulatoren blir styrt av modellen for hendelser. Alle hendelsene er hentet inn fra denne modellen, og lagt i en liste som er sortert stigende etter tidspunkt for når hendelsen skal inntreffe. Følgende hendelser kan inntreffe:

- Advertise
- Unadvertise
- Publish
- Subscribe
- Unsubscribe

Simulatorene er drevet av en timer som illustrerer tidsforløpet. Timeren inkrementerer verdien med et sekund av gangen, og når dette gjøres blir listen over hendelser sjekket for å se om en hendelse skal utføres. Noder som har data i køen vil også overføre en mengde data som tilsvarer et sekund.

Når en hendelse inntreffer utfører simulatoren en rekke prosedyrer, og hvilke prosedyrer simulatoren utfører er avhengig av hva slags type hendelse som inntreffer. Dersom for eksempel en *Advertise* står for tur i simulatoren for Siena vil det bli sendt en broadcast-melding til alle noder for å annonsere at det nå er mulig å abonnere på en ny type data. En annen hendelse inntreffer dersom en node sender *Subscribe*, noe som medfører at multicast-treet må utvides for å inkludere den aktuelle noden.

Hver gang timeren inkrementeres blir det tatt et øyeblikksbilde (eng. snapshot) av tilstanden. Øyeblikksbildet inneholder tidspunkt, belastning på linker, inn- og utkø hos nodene, redundans og antall pakker i omløp. Denne informasjonen kan eksporteres til en rekke formater, deriblant XML (fork. eXtensible Markup Language), (XLS) Microsoft Excel dokument og CSV (fork. Comma Separated Values).

Simulatorene har et grafisk visualiserer som viser hvordan pakkene blir sendt rundt i informasjonsutvekslingssystemet. For å få et godt bilde av hva som skjer er det også mulig å stille hastigheten for hvor ofte timeren skal inkrementeres. I tillegg kan man sette simulatorene på pause for å studere øyeblikksbildet for det aktuelle tidspunktet.

I alle våre foreslåtte informasjonsutvekslingssystemenr er det brukt en aller annen form av distance-vector for å propangere data om nettverkstopologien. Denne dataen vil deretter bli brukt for å finne korteste sti mellom to noder, eller korteste spenntre (eng. minimal spanning three) fra node *n* på grafen *G*. I simulatorene har vi brukt Dijkstra's Shortest Path Algorithm for å løse disse problemene.

Se vedlegg C for en demonstrasjon av simulatoren.

3.2 Tidsfristbasert distribusjon

For å fordele belastningen i transittnettverket over tid har vi innført en metode som vi har kalt tidsfristbasert distribusjon. I dette kapitlet vil vi forklare de grunnleggende prinsippene rundt denne metoden, mens hvordan denne metoden kan benyttes i Siena og foreslåtte informasjonsutvekslingssystemer vil bli beskrevet i respektive kapitler.

Tidsfristbasert distribusjon går ut på at objekter som skal overføres blir markert med et tidspunkt som indikerer når objektet senest må være fremme hos mottaker. Ved å kombinere denne informasjonen med objektets størrelse kan informasjonsutvekslingssystemet finne den minste overføringshastigheten som er

mulig før tidsfristen brytes. Med andre ord vil objektets bruk og belastning av transittnettverket bli fordelt over tid.

Det er viktig å merke seg at tidsfristen ikke skal brukes til å fremskynde en overføring. Tidsfristen skal i stedet brukes i tilfeller hvor det går greit at objektet bruker lengre tid enn hva det ellers ville brukt med best-effort.

Vi har funnet ut at en overføring av et objekt fra node *a* til node *b* vil få overføringshastigheten (*R*):

$$R(a,b) = \frac{L}{T_d - T_p}$$

hvor *L* er objektets størrelse, T_d er objektets tidsfrist og T_p er nåværende tidspunkt.

Dersom et objekt ikke er markert med en tidsfrist har vi betraktet det som best-effort. Det vil si at informasjonsutvekslingssystemet vil gjøre sitt beste for å få overført objektet, men det vil ha lavere prioritet enn objekter med tidsfrist. Objekter som under overføring passerer tidsfrist ($T_d < T_p$) blir også betraktet som best-effort. Grunnen til dette er at informasjonsutvekslingssystemet da skal prioritere objekter som fortsatt kan nå tidsfristen, i stedet for å la objekter som allerede har brutt fristen bli en propp i systemet.

Dersom en node har flere objekter som skal overføres vil disse bli overført parallelt med dens respektive overføringshastighet. I tilfeller hvor summen av overføringshastigheten til objektene overstiger linkens kapasitet ($\Sigma T(a,b) > T_{link}$) vil informasjonsutvekslingssystemet måtte velge hvilke objekter som skal overføres. Objektene som har en høy overføringshastighet er også objektene som har det travlest, og de blir derfor høyt prioritert. For å forhindre at objekter som allerede har overført deler av data skal bli nedprioritert og stoppe opp er det også et avgjørende kriterium hvorvidt objektet allerede har startet overføringen eller ikke.

Såframt objektet bare skal overføres mellom to noder settes det ingen krav til at klokkene er synkronisert. I mange informasjonsutvekslingssystemer rutes objektene via flere noder, og det settes da større krav til at klokkene er synkrone. Siden det i vårt tilfellet gjelder store mengder data vil tidsfristene gjerne være langt frem i tid, og dermed vil et lite avvik i synkroniserting være ubetydlige. Informasjonsutvekslingssystemet bør uansett inneholde en synkroniseringsmekanisme, for å rette opp tilfeller hvor det forekommer stor differanse mellom klokkeslett. Vi har ikke tatt for oss dette i vårt studium.

3.3 Siena utvidet

For at Siena skal være bedre egnet til å distribuere data av stor størrelsesorden har vi innført en rekke mekanismer. Disse mekanismene er beskrevet i dette delkapitlet.

I den geniune Siena blir dataen som distribueres kalt notifications, og den består av et sett med attributter som beskriver en inntruffet hendelse. Det kan for eksempel være en hendelsesmelding innen kategorien "børs/marked" om at aksjen MSFT på børsen NYSE har hatt et fall på 0.5 prosentpoeng (se tabell 3-2), eller en

hendelsesmelding innen kategorien "spill/moro" som forteller at Blizzard Entertainment har annonsert spillet World of Warcraft.

I vårt tilfelle består ikke dataen av attributter som utgjør en hendelsesmelding, men den består av kodet data av stor størrelsesorden (videofilmer, audioklipp, programfiler etc.). Dataen har også en meldingstopptekst (eng. message header) som består av samme typen attributter som Siena. Disse attributtene er metadata som beskriver hva slags type innhold dataen er. På samme måte som Siena bygger opp rutingtabellene av attributtene i hendelsesmeldinger, vil vårt system bygge opp rutingtabellene basert på attributtene i metadataen.

Tabell 3-2: Eksempel på en notification i Siena

Type	Navn	Verdi
string	*kategori*	*børs/marked*
time	*tidspunkt*	*2005-05-01 07:00*
string	*børs*	*NYSE*
string	*aksje_symbol*	*MSFT*
string	*aksje_navn*	*Microsoft Corp.*
float	*forandring*	*-0.5*

Tabell 3-3: Eksempel på en notification i utvidet Siena

Type	Navn	Verdi
string	*Navn*	*The Lord of the Rings: The Fellowship of the Ring*
string	*Sjanger*	*fantasy*
int	*spilletid*	*242*
string	*språk*	*engelsk*
byte[ ]	*data*	

Siena bruker en applikasjonslags implementasjon av distance-vector for å propangere informasjon som trengs til rutingen. Normalt sett inneholder en avstandsvektor avstanden til andre noder i form av antall hopp, men Siena har utvidet denne til å inneholde latenstiden. Det vil si at den korteste vegen mellom to noder ikke bestemmes av antall hopp, men av summen av latenstid.

Ved overføring av små mengder data vil latenstiden være viktig, men for overføring av store mengder data er det båndbredden som er avgjørende. I vårt tilfellet har vi derfor utvidet distance-vector til å inneholde båndbredde i tillegg til antall hopp. Betydningen av å velge båndbredde fremfor latenstid ved overføring av store mengder data er illustrert i kommende scenario.

Tenk et tilfellet hvor 1 byte skal sendes over en link. Det vil da være stor forskjell i ytelse om linken har en latenstid på 1 ms eller om den har en latenstid på 100 ms. Dersom man i tillegg ser på hvor lang tid det vil ta å overføre denne byten på en link med en båndbredde på 1 Mbps og deretter på en link med båndbredde på 100 Mbps vil man se at overføringstiden er minimal i forhold til latenstiden. I det første tilfellet vil overføringstiden bli 8 µs, mens for det andre tilfellet vil overføringstiden bli 0,08 µs.

I et tilfellet hvor en pakke på 25 Mbytes skal sendes over en link vil resultatet bli motsatt. Dersom linkens båndbredde er 10 Mbps vil overføringstiden være 20

sekunder. Det har derfor liten betydning om latenstiden er på 1 ms eller 100 ms, noe som vil gi en neglisjerbar forskjell på 20.001 sekunder og 20.1 sekunder (noe som utgjør en differanse på > 0.5%).

I den genuine Siena blir bare én pakke sendt av gangen. I vårt tilfellet ønsker vi å kjøre en prosess per tilkoblede link slik at linkene parallelt kan overføre data. Dette gjør at informasjonsutvekslingen skjer jevnere uten at store pakker holder på ressursene og blokkerer de andre linkene. Siden linkene til tjenestemodellen gjenspeiler underliggende linklag, vil parallelle overføringer også gi en bedring i utnyttelse av båndbredden.

3.3.1 Fragmentering

Fragmentering går ut på å dele store pakker opp i mindre biter (fragmenter). Fragmenteringsprosessen gir hvert av fragmentene sin egen meldingstopptekst slik at transittnettverket kan rute segmentene individuelt mot destinasjonene.

Fragmenteringen skjer hos produsentene hvor fragmentenes størrelse bestemmes av MTU (fork. Maximum Transmission Unit). Denne verdien kan settes individuelt hos produsentene, og den bestemmer hva som skal være maksimal segmentstørrelse. Fragmentene forblir de samme mens de rutes gjennom transittnettverket, og de blir først satt sammen til opprinnelig innhold når de kommer frem til endelig mottaker.

Merk at underliggende nettverkslag og linklag kan ha sine egen fragmenteringsmetoder med sin egen MTU. Foreslåtte fragmentering gjelder bare applikasonslaget, og den har ingen inflytelse på underliggende lag.

En av fordelene med å fragmentere store pakker er at man ikke trenger å retransmittere hele pakken ved feil, men bare de aktuelle segmentene. Konsekvensene ved forekomster av små bittfeil er derfor minimerte. En annen fordel knyttes til at segmentene rutes individuelt i transittnettverket, og at de derfor kan være på forskjellige steder på et gitt tidspunkt. Dette gjør at belastningen blir jevnet ut over flere transittlinker, noe som igjen resulterer i færre tilfeller av overfylte køer.

For å innføre fragmentering i Siena må vi utvide meldingstoppteksten med tre nye attributter: *fragident*, *fragnum* og *fragtotal*. Disse attributtene vil ligge sammen med resten av attributtene i meldingstoppteksten, men navnene vil være reservert av informasjonsutvekslingssystemet slik at de ikke kan brukes til andre formål enn fragmentering.

Tabell 3-4: Eksempel på en fragmentert notification

Type	Navn	Verdi
int	*fragident*	*1005*
int	*fragnum*	*5*
int	*fragtotal*	*23*
string	*navn*	*The Lord of the Rings: The Fellowship of the Ring*
string	*sjanger*	*fantasy*
int	*spilletid*	*242*
string	*språk*	*engelsk*
byte[ ]	*data*	

Som vi ser i tabell 3-4 er pakken fra tidligere eksempel nå segmentert. *fragident* er en unik identifikator som entydig definerer segmentene som tilhører samme pakke. Denne identifikatoren består av en sammenslåing av verdiene SienaURI (hvert objekt i Siena er entydig identifisert), tallet 0 og et løpenummer (hver produsent har sitt eget løpenummer som inkrementeres hver gang en ny pakke publiseres). I eksempelet ovenfor er SienaURI=10 og løpenummeret er 5.

Grunnen til at det legges på en null (0) mellom SienaURI og løpenummeret er for å sikre at verdien er unik for hele informasjonsutvekslingssystemet. Dersom ikke kan følgende forekomster oppstå: node med SienaURI=2 sender ut melding med løpenummer 12, noe som gir en identifikator med verdien 212. Deretter sender node med SienaURI=21 ut sin andre melding, noe som også gir en identifikator med verdien 212. Ved å innføre sammenkobling med null vil verdiene henholdsvis bli 2012 og 2102.

Attributtene *fragnum* og *fragtotal* forteller hvilket fragmentnummer det aktuelle fragmentet er ut i fra hvor mange fragmenter som det finnes totalt. I eksempelet ovenfor vises fragmentnummer 5 av totalt 23 fragmenter. Denne informasjonen brukes når sluttnoden skal sette sammen fragmentene til en pakke (eng. reassembly).

Sluttnoden kan sette sammen fragmentene fortløpende som de kommer inn, men dette forutsetter at de kommer inn i riktig rekkefølge. Dersom rekkefølgen brytes, og det blir hull i fragmentnumrene, blir fragmentene midlertidig lagt i en kø frem til manglende fragmenter dukker opp.

3.3.2 Tidsfristbasert distribusjon

Siden arkitekturen til Siena ikke har støtte for tjenestekvalitet har vi utvidet med tidsfristbasert distribusjon. Som tidligere nevnt vil tidsfristbasert distribusjon spre overføringen over tid, og dermed redusere belastningen i nettverket. Dette er derfor en naturlig utvidelse for at Siena skal være i stand til å kunne distribuere data av stor størrelsesorden.

For å innføre tidsfristbasert distribusjon i Siena må vi utvide tidligere nevnte løsning. Grunnen til dette er at Siena har en del mekanismer som det må tas hensyn til. Blant annet skjer distribusjonen over et transittnettverk hvor det settes opp et multicast-tre.

Multicast-treet blir dannet ved hjelp av *Subscription* og *Advertisement*. Siden Siena bruker innholdsbaser ruting er det ikke adresser som propangeres med distance-vector, men hva slags type innhold som produseres eller konsumeres (dette uttrykkes med *filters* og *patterns*). Rutingtabellen vil i vårt tilfellet inneholde både antall hopp til et gitt type innhold, men også båndredden (denne brukes av distance-vector til å regne ut beste veg).

En gitt transittnode kan ha flere linker ut fra seg, og siden dette er multicast kan det også skje at en og samme pakke må sendes ut på to eller flere av linkene. Lengre ut i treet kan det være nye transittnoder som forgrener seg, og slik kan det fortsette. Det er derfor viktig at overføringshastigheten fra en gitt node tar hensyn til det verste tilfellet slik at alle konsumentene får pakken innen tidsfristen. Dette er også ivaretatt

av rutingen ved at antall hopp frem til et gitt type innhold er fra den stien som har flest hopp.

Linken mellom en stubnode og inn til transittnettverket brukes bare av den ene noden. Det er derfor liten hensikt i å begrense belastningen på denne linken. Pakkene blir derfor sendt fortest mulig inn til transittnettverket slik at transittnettverket tidligst mulig får dataen og kan deretter planlegge videre overføring.

Vi har funnet ut at overføringshastigheten (*R*) for å overføre et objekt fra transittnoden *a* til interessenter av datainnhold *b* er gitt ved

$$R(a,b) = \frac{L_{total} * (D + F) - L_{progress}}{T_d - T_p}$$

T_d og T_p er som tidligere tidsfristen og det nåværende tidspunkt. *D* er avstanden i form av antall hopp frem til interessenten som er lengst borte. Grunnen til at det må tas hensyn til antall hopp er fordi et fragment ikke kan sendes videre til neste node før hele fragmentet er kommet inn.

L_{total} er fragmentets størrelse, og verdien finnes ved å telle antall bits som det fragmentet består av. $L_{progress}$ er et mål for hvor mange bits av den totale datamengden som frem til nå er overført. Grunnen til at denne attributten inkluderes er i tilfellet utregningen vil skje etter at overføringen har begynt.

Den siste attributten i formelen er *F*. Denne indikerer hvor mange etterkommende fragmenter som gjenstår, og den er definert ved

$$F = fragtotal – fragnum$$

hvor *fragtotal* er antall fragmenter total, og *fragnum* forteller hvilket fragmentnummer den aktuelle pakken har. Begge disse attributtene er lagt ved meldingstoppteksten til pakken.

Vedlegg D demonstrerer formelen ved å ta for seg et scenario.

3.4 Epidemisk Siena

Vi har tatt utgangspunkt i Siena, og deretter utvidet arkitekturen med prinsippene bak den epidemisk protokollen. Løsningen som vi kom frem til er beskrevet i dette delkapitlet.

Den epidemisk utvidelsen går i korte trekk ut på å spre dataen ved å smitte andre noder. Nodene som blir smittet kan igjen smitte andre noder, noe som gjør at spredningen skjer eksponentielt. Utvelgelsen av hvem som skal bli smittet, og dermed også få data, skjer i vårt tilfelle tilfeldig.

Det første vi gjorde var å erstatte den innholdsbaserte rutingen med en vanlig adressebasert ruting. Ruting-tabellene inneholder nå antall hopp frem til destinasjonene, og den korteste stien baserer seg på båndbredde.

Siden rutingen er adressebasert må hver node ha en unik adresse. Hver transittnode får en adresse som er et heltall, og siden det sjeldent er endringer i transittnettverkets topologi kan disse adressene settes statisk. Stubbnodene vil få en adresse på formatet x.y, hvor x er adressen til transittnoden som den er koblet opp mot. Verdien x er også et heltall som er unik i forhold til de andre nodene som er koblet opp mot samme transittnode.

Adressen 0 er reservert av informasjonsutvekslingssystemet, og den brukes til kringkastning (eng. broadcast). Kringkastningen fungerer på samme måte som i Siena, ved at en meldingen som kommer til en transittnode bare sendes videre til alle andre tilkoblede transittnoder, dersom den ikke har mottat den samme meldingen tidligere.

På grunn av at stubbnodene skal kunne koble seg av og på informasjonsutvekslingssystemet, og for at informasjonsutvekslingssystemet skal være fleksibelt med å koble på noder for første gang, bør tilegnelse av adresse skje automatisk. Dette er realisert ved at transittnodene deler ut adresser til sine tilkoblede stubbnoder når de kobles til. Transittnodene verifiserer også at adressene som blir utstedt ikke er i bruk av en annen node. Når en stubbnode kobler seg av vil adressen bli ledig og etterhvert utdelt til andre stubbnoder.

Siden informasjonsutvekslingssystemet skal være skalerbart med tanke på antall tilkoblede noder har vi innført noen optimaliseringer i rutingen. Vi utnytter i dette tilfellet at stubnodenes adresse inneholder hvilken transittnode de er koblet opp mot. Det holder derfor at rutingtabellene inneholder informasjon om neste steg til transittnodene, i stedet for å ha oppføring for alle stubbnoder. Med tanke på at det som regel er mange flere stubbnoder enn transittnoder (gjerne et 1/10-forhold) i et informasjonsutvekslingssystem vil dette kraftig redusere antall oppføringer i rutingtabellen, samt redusere størrelsen på vektoren som sendes rundt med distance-vector.

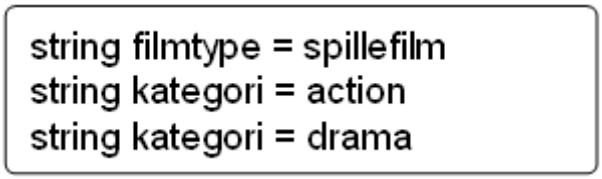
Ettersom informasjonsutvekslingssystemet skal være innholdsbasert må det implementeres noen mekanismer som gjør at nodene individuelt kan bestemme hva slags type informasjon de vil abonnere på. Dette utrykkes på samme måte og med samme meldinger som i Siena, men informasjonen om hva som produseres og konsumeres er flyttet fra transittnettverket og inn i stubbnodene.

En stubbnode deklarerer sin intensjon om å sende data som samsvarer med attributtene til filteret *f* ved å sende meldingen *Advertise(f)*, og effekten av den handlingen blir opphevet ved å sende *Unadvertise(f)*. Akkurat som for Siena kringkastes disse meldingene slik at de når ut til alle noder. Vedlagt i meldingstoppteksten som attributt er også avsenderens adresse (attritbuttet *srcadr*). Som vi snart skal se er denne vedlagt for at noder skal kunne sende unicast meldinger tilbake til stubbnoden.

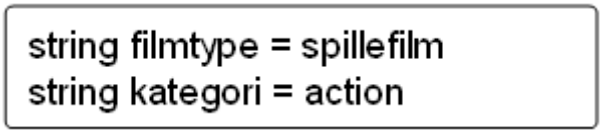
Når en stubbnode får en *Advertise*-melding blir den lagt i en liste slik at stubbnoden vet hva slags type informasjon som det er mulig å abonnere på. På et hvilket som helst tidspunkt kan nå stubbnoden abonnere på denne informasjonen. Dette gjøres ved at den sender en *Subscription*-melding tilbake til stubbnoden som annonserte informasjonen.

For at nye noder som kobler seg på informasjonsutvekslingssystemet skal få vite hva som publiseres vil *Advertise*-meldingene bli send ut periodevis. Bare nye noder som ikke har abonnert tidligere svarer på denne ved å sende en *Subscribe*-melding tilbake til produsenten. Siden utsendelsen av *Advertise*-meldinger skjer periodevis kan dette også brukes av interessentene for å detektere om en produsent har falt ut.

Vi har innført en mekanisme for at interessentene i størst mulig grad bare skal få informasjonen de er interessert i, og dette gjøres ved at *Subscription*-meldingen som sendes tilbake til produsenten inneholder et filter. Dette filteret brukes til å ytterligere spesifisere hva slags type innhold det er ønskelig å motta, men det må ha mindre eller likt omfang enn filteret som ble sendt med *Advertise*-meldingen. Det vil si at dersom filteret som kom med *Advertise*-meldingen inneholder attributtene som vist i figur 3-1, kan stubbnoden ytterligere avgrense dataen som den vil motta ved å legge med filteret som er vist i figur 3-2 i *Subscription*-meldingen. Figur 3-1 viser en produsent som har en intensjon om å sende data av typen *spillefilm* for kategoriene *action* og *drama*. Figur 3-2 viser en interessent som avgrenser omfanget av hva han ønsker å motta ved å spesifisere at den bare er interessert i å motta data for kategorien *action*.



Figur 3-1: Eksempel på et filter



Figur 3-2: Eksempel på et filter

Når *Subscription*-meldingen blir mottatt av noden som produserer data blir vedlagte filter og avsenders adresse lagt i en liste. Denne listen inneholder informasjon om alle abonnenter, og det er ut i fra denne det avgjøres hvem som skal motta dataen. Når en pakke skal sendes vil pakkens egenskaper bli sammenlignet med filterene i listen. I de tilfellene hvor filtrene i listen er dekket av pakkens attributter vil adressen bli lagt i meldingstoppteksten til pakken. Tabell 3-5 viser en notification som har flere destinasjonsadresser. Adressene er lagt med som en vanlige attributter i meldingstoppteksten, og attributtens navn (*destadr*) er reservert av informasjonsutvekslingssystemet.

Tabell 3-5: Notification med flere destinasjonsadresser

Type	Navn	Verdi
string	*destadr*	*2.9*
string	*destadr*	*1.8*
string	*destadr*	*2.2*
int	*destcount*	*3*
string	*nextdestadr*	*2.3*
int	*sequence*	*1*
string	*Navn*	*The Lord of the Rings: The Fellowship of the Ring*
string	*Sjanger*	*fantasy*
int	*spilletid*	*242*
string	*språk*	*engelsk*
byte[ ]	*data*	

To andre obligatoriske og reserverte attributter er *destcount* og *sequence*. *destcount* inneholder opprinnelig antall destinasjonsadresser, det vil si det antallet destinasjonsadresser som pakken inneholdt ved starten. Grunnen til at denne verdien ikke kan oppdrives ved å telle antall forekomster av *dest*-attributtet i meldingstoppteksten er fordi adressene fjernes fra meldingstoppteksten fortløpende ettersom adressenten får data. Det andre attributtet, *sequence*, er et løpenummer som forteller hvor mange stubbnoden den samme dataen har vært innom tidligere. Denne verdien inkrementeres hver gang pakken kommer frem til en stubbnode. Som vi senere skal se blir disse attributtene brukt for å innføre tidsfristbasert distribusjon.

Etter at meldingstoppteksten er generert er pakken klar for utsending. Pakken vil da bli sendt som unicast til én av adressentene. Hvilken interessent som først får dataen bestemmes ved å trekke en tilfeldig adresse fra meldingstoppteksten. For at transittnettverket skal vite hvilken av adressene i meldingstoppteksten som pakken skal rutes etter, vil navnet på attributtet til utvalgte adresse bli omdøpt til *nextdestadr*.

Når pakken kommer frem til stubbnoden med adresse *nextdestadr* vil det være to noder som har denne informasjonen. Det skjer nå en epidemisk spredning ved at hver av disse to nodene kan spre dataen videre til to nye noder. Før dataen sendes videre gjøres det noen små operasjoner. Først fjernes attributtet *nextdestadr* fra meldingstoppteksten. Grunnen til dette er at noden som denne adressen peker til har nå fått dataen, og det er derfor unødvendig å la denne adressen være med i fremtidige uttrekninger. Deretter trekkes det ut en ny adresse fra meldingstoppteksten hvor navnet på atributtet blir omdøpt fra *destadr* til *nextdestadr*. Pakkene kan nå sendes videre.

Slik som metoden for spredning av data er fremtstilt til nå vil nodene fortsette å pushe data helt til det ikke er flere adresser igjen i meldingstoppteksten. Dette har en negativ side ved at adressenten som er valgt i fra meldingstoppteksten kan ha mottatt samme data av en annen node tidligere. I et slikt tilfelle vil det være forgjeves å sende dataen nok en gang. En løsning på dette problemet vil være å spørre den utvalgte noden om den allerede har mottat den aktuelle dataen, før dataen sendes.

I enkelte tilfeller vil svært mange noder være interessert i å motta samme datapakke. Dersom hver node som allerede har fått pakken skal spørre alle andre noder som gjenstår, vil dette resultere i en omfattende meldingsutveksling som kan gå på

bekostning av skalerbarheten. I den genuine epidemiske algoritmen er denne problemstilligen tenkt på ved at sannsynligheten for at en node skal fortsette å spre data blir redusert med 1/K dersom den spør en node som allerede har fått samme data fra en annen node. I vårt tilfellet har vi bygget videre på denne tankegangen ved å innføre en mekanisme som garanterer at minimum halvparten av nodene vil få dataen, og dermed øker den generelle sannsynligheten for at alle nodene vil få dataen.

Vår mekanisme utnytter teorien om at epidemisk spredning øker eksponentielt (2^x), det vil si at nodene som får data blir smittet og kan spre data videre til andre noder. Når man vet spredningen (eksponentiell) og antallet interessenter (*destcount*) kan man finne ut av hvor mange hele runder (*S*) distribusjonen vil ta:

$$S = \lceil \mathrm{Log}_2 (\mathrm{destcount}) \rceil$$

Ved å sammenligne *sequence* (beskrevet tidligere) og *S* vet man hvor langt ut i prosessen distribusjonen er kommet. Dersom *sequence* er 2 og *s* er 5 vet man at dette er runde 2 av 5, noe som også betyr at 4 (2^2=4) av nodene skal ha fått dataen. Når *S* og *sequence* er kjent kan man også finne ut av når distribusjonen er på nest siste runde (*sequence* + 1 = *S*). Når nest siste runde er avsluttet skal minimum halvparten av nodene ha fått dataen, og halvparten eller færre av nodene gjenstår. Vi sikrer derfor at minimum halvparten av nodene får dataen ved at nodene aldri skal slutte å spørre så lenge *sequence* < *S*.

Når siste runden inntreffer (*sequence* = *S*) vil det være færre eller like mange noder som mangler dataen i forhold til noder som sprer data. Dersom 2^S = *destcount* vil det være like mange av hver. En smittsom node vil da fortsette å spørre helt til den har fått positivt svar fra en node som fortsatt mangler data. Når overføringen mellom dem er ferdig vil begge bli uvirksomme. I tilfellet hvor $2^S \neq$ *destcount* vil det være flere smittsomme noder enn noder som mangler data. Det vil derfor bare være de første som får positivt svar som får spre data videre. Resten av nodene vil etterhvert stoppe å spørre videre ved at sannsynligheten for å fortsette reduseres med 1/K for hver negative respons.

3.4.1 Fragmentering

Vi har også valgt å innføre fragmentering i den epidemiske utgaven av Siena. Fragmentering går ut på å dele store pakker opp i mindre biter (fragmenter). I dette delkapitlet er det bare lagt vekt på å forklare mekanismer som skiller seg ut fra tidligere forklarte implementasjon av fragmentering i Siena.

Som tidligere nevnt skjer fragmenteringen hos produsenten hvor en pakke deles i størrelser bestemt av MTU (fork. Maximum Transmission Unit). I den epidemiske utgaven av Siena vil samme data bli sendt innom en rekke interessenter, og i løpet av denne prosessen vil segmentene være uforandret. Segmentene blir først satt sammen (eng. reassembly) til én stor pakke når den aktuelle noden slutter å spre data.

For å redusere redundans i meldingstoppteksten vil det være forskjell på de ulike fragmentenes innhold. Det første fragmentet (*fragnum*=1) vil inneholde alle ønskelige attributter, deriblant metadata (*sjanger*, *språk,* etc.), fragmenteringsinformasjon

(*fragnum*, *fragident* og *fragtotal*) og adressenter (*destadr*, *destcount*, *nextdestadr* og *sequence*). Derimot vil de neste fragmentene (1 < *fragnum* ≤ *fragtotal*) bare inneholde informasjon som er nødvendig for å prosessere fragmentet. Dette inkluderer *nextdestadr* som transittnettverket bruker for videresending, *deadline* og *starttime*, som vi senere skal se brukes til å innføre tidsfristbasert distribusjon, og fragmenteringsinformasjon som trengs for å sette sammen fragmentene til en pakke. Vi har dermed fjernet metadata og deler av adresseinformasjonen (*destadr* og *destcount*) som utgjør mesteparten av meldingstoppteksten. Tabell 3-6 viser attributtene i det første fragmentet, mens tabell 3.7 viser attributtene i etterkommende fragment.

Tabell 3-6: Første fragment

Type	Navn	Verdi
int	*fragident*	*908*
int	*fragnum*	*1*
int	*fragtotal*	*19*
string	*destadr*	*1.8*
string	*destadr*	*2.2*
int	*destcount*	*3*
string	*nextdestadr*	*2.3*
int	*sequence*	*1*
string	*navn*	*Pooh's Heffalump Movie*
string	*sjanger*	*family*
int	*spilletid*	*68*
string	*språk*	*engelsk*
byte[ ]	*data*	

Tabell 3-7: Andre fragment

Type	Navn	Verdi
int	*fragident*	*908*
int	*fragnum*	*2*
int	*fragtotal*	*19*
string	*nextdestadr*	*2.3*
byte[ ]	*data*	

3.4.2 Tidsfristbasert distribusjon

Vi har også innført tidsfristbasert distribusjon i den epidemiske utgaven av Siena. Årsaken til det er todelt: vi ønsker at løsningen skal ha mulighet til å redusere belasningen i transittnettverket. I tillegg vil vi også vise at prinsippene bak tidsfristbasert distribusjon er enkle og kan integreres i ulike informasjonsutvekslingssystemer. I dette delkapitlet vil hovedfokus være på forskjellene i forhold til tidligere forklarte implementasjoner av tidsfristbasert distribusjon.

Som tidligere nevnt er tidsfristbasert distribusjon en metode som reduserer belastningen i nettverket ved at overføringen hales ut i tid. Dette gjøres ved at objektet som skal overføres markeres med en tidsfrist, og ut i fra det kan informasjonsutvekslingssystemet finne den minste overføringshastigheten som er mulig før tidsfristen brytes.

Den største egenskapen som man må ta hensyn til ved innføring av tidsfristbasert distribusjon i den epidemiske versjonen er at overføringen av data går over runder (også kalt sekvenser). Dette har vi løst ved å gi hver runde en egen tidsfrist (T_{ds})

$$T_{ds} = T_s + \frac{T_d - T_s}{S} * seqence$$

T_s er en ny attributt som er innført. Denne er vedlagt som attributt i meldingstoppteksten (*startttime*), og den forteller når det første fragmentet ble sendt ut fra produsenten. Attributtet er uforanderlig gjennom hele distribusjonen.

T_d er som tidligere en benevnelse for tidsfristen. *S* og *sequence* er også nevnt tidligere, hvor *S* er totalt antall runder som distribusjonen vil ta, mens *sequence* er den aktuelle runden.

Når tidsfristen for en aktuell runde (T_{ds}) er gitt kan vi regne ut overføringshastigheten (*R*). Formelen har store likhetstrekk med formelen til den utvidede versjonen av Siena, hvor eneste forskjell er at T_d er erstattet med T_{ds}:

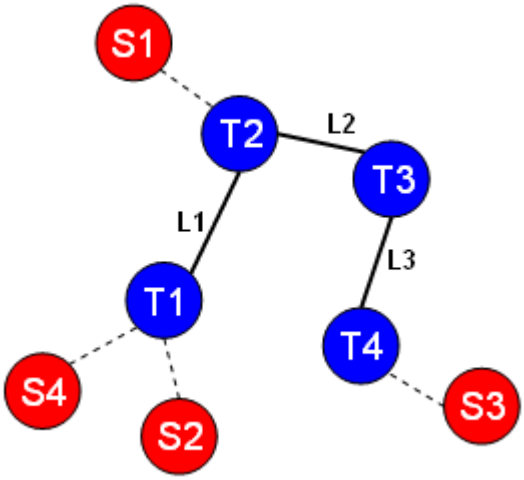
$$R(a,b) = \frac{L_{total} * (D + F) - L_{progress}}{T_{ds} - T_p}$$

Dersom $T_p \geq T_{ds}$ er dette et tegn på at tidsfristen er blitt brutt i det aktuelle intervallet. I dette tilfellet vil informasjonsutvekslingssystemet prioritere objekter som fortsatt kan nå tidsfristen, og det gjeldende objektet vil derfor bli betraktet som best-effort. Når objektet kommer frem til en stubbnode vil det bli sendt videre til neste stubbnode, og denne overføringe kan gå over en annen sti hvor det ikke er metning. Hvorvidt objektet skal være best-effort eller ikke vil derfor bli reevaluert hos hver stubbnode. En avgjørende faktor er da om objektet kan klare å komme frem til neste interessent uten at tidsfristen for det gitte intervall brytes.

3.4.3 Temping

Dersom samme data blir overført to eller flere ganger over samme link kan vi si at det har oppstått redundans. I enkelte tilfeller er redundans positivt, men bakdelene er motstridende med vår målsetning. Vi har derfor innført temporær lagring (temping) av data i transittnettverket for å fjerne redundansen.

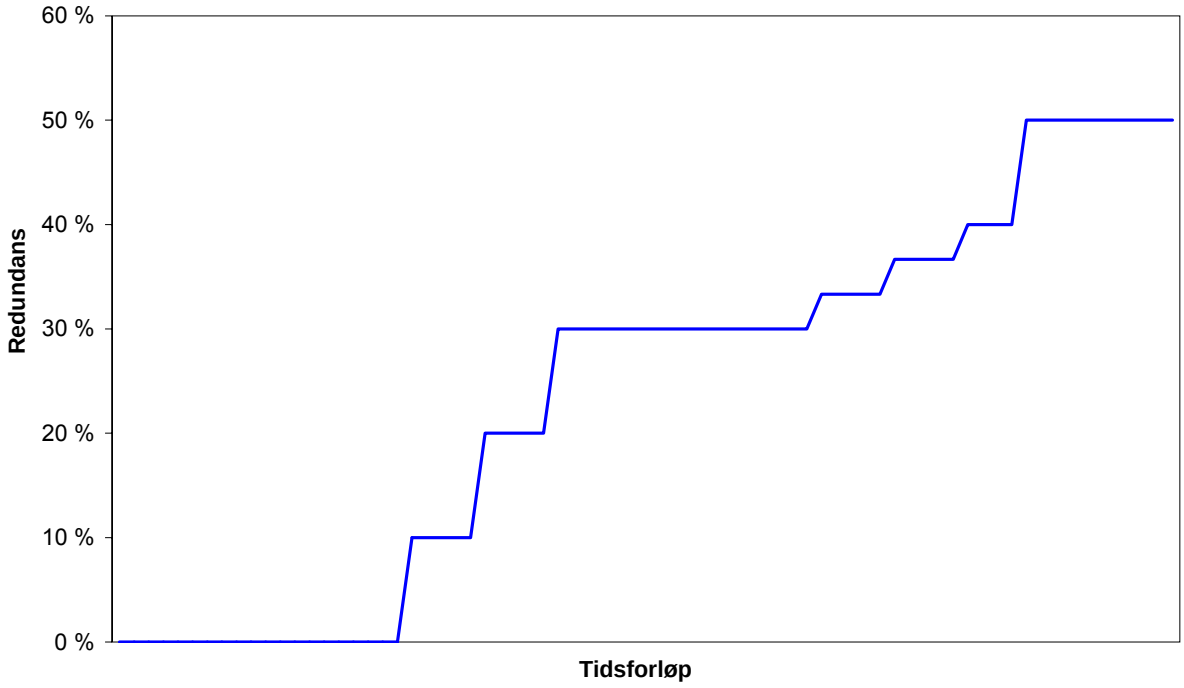
Grunnen til at den epidemiske protokollen medfører redundans er fordi distribusjonen består av en rekke overføringer som skjer ende til ende. Figur 3-3 viser nettopp dette. Det første som skjer er at node *S2* smitter node *S3.* Dataen blir da overført over link *L1*, *L2* og *L3*. Deretter smitter noden *S3* node *S1*, og dataen overføres da over linkene *L3* og *L2*. Som vi ser ut i fra gitte scenario har den samme dataen blitt overført over linkene *L2* og *L3* hele to ganger.



Figur 3-3: Transittnettverk for over scenario

Temping som vi foreslår går ut på at data som overføres gjennom transittnettverket midlertidig blir lagret hos transittnodene. Ved enkelte tilfeller vil det da være enklere å sende data fra tempen i stedet for at data overføres hele vegen fra stubbnoden. Dette kan illustreres ved å se på samme scenario som tidligere (figur 3-3). Data overføres først fra *S2* til *S3*, og på vegen frem til *S3* blir dataen lagt i tempen til transittnodene *T1*, *T2*, *T3* og *T4*. Deretter skal dataen overføres fra *S3* til *S1*. Tempen til transittnodene som ligger på stien mellom *S3* og *S1* blir nå sjekket for å se om den aktuelle dataen ligger der. I vårt tilfellet ligger dataen i tempen til alle transittnodene, og det startes derfor en overføring fra den transittnoden som ligger nærmest *S1* målt i antall hopp.

For å ytterligere poengtere viktigheten av tempingen har vi logget redundansen i informasjonsutvekslingssystemet. Vi tok utgangspunkt i samme topologi som vist i figur 3-3, og konstruerte en situasjon hvor noden *S3* produserer og sender ut én datapakke, og nodene S1, S2 og S4 abonnerer. Gjennomsnitlig redundansen som oppstod i informasjonsutvekslingssystemet er vist i figur 3-4. Som grafen viser vil det på det meste være 50% gjennomsnitlig redundans.

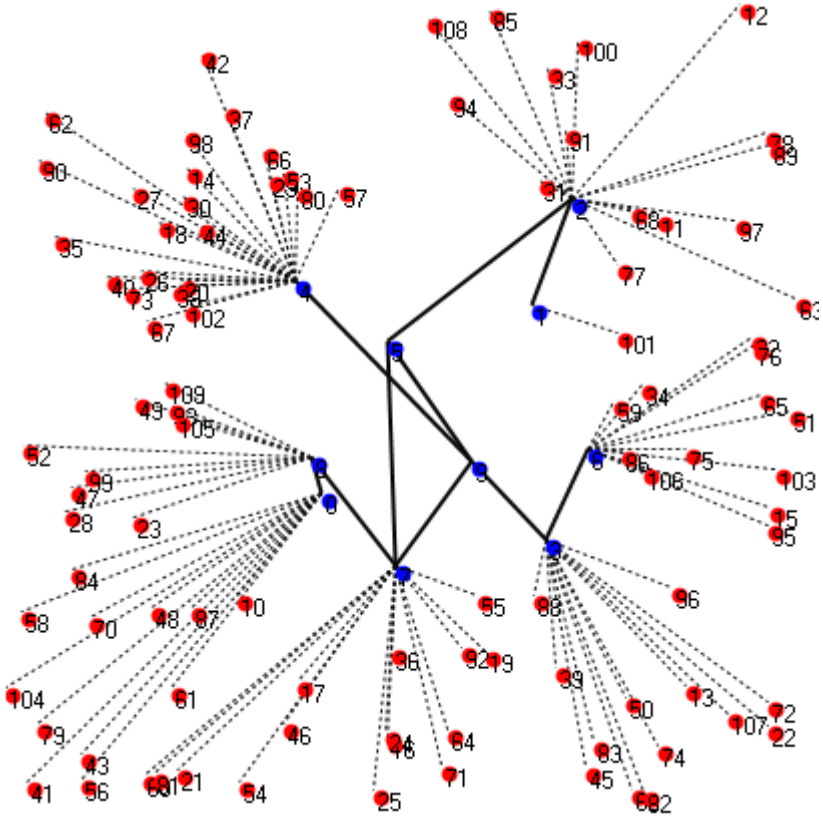


Figur 3-4: Gjennomsnittlig redundans sett over tid

4 Resultater

Resultatene i dette kapittelet er hentet fra ulike simuleringer av de ulike løsningene vi har forklart tidligere i rapporten [*ref. kapittel 3*]. De forskjellige løsningene vi har simulert er: Den opprinnelige Siena, den utvidede *Siena* og den epidemiske *Siena*. Forklaringer av de ulike løsningene er å finne i de respektive underkapitlene.

Nettverkstopologien er blitt generert av egenutviklet nettverksgenerator (*ref. kapittel 3*). Vi genererte et nettverk bestående av 10 transittnoder og 100 stubbnoder (gj.f. figur 4-1). Linkene mellom transittnodene har en båndbredde mellom 2000 og 2500 kb/s. Linkene mellom transittnodene og stubbnodene har en båndbredde mellom 500 og 1500 kb/s. Båndbredden til linkene viser ikke til linkenes totale båndbredde i virkeligheten, men det vi går ut i fra som deres ledige kapasitet. Metoden som er brukt ved generering av nettverket er valgt for å gjenspeile et realistisk nettverk.



Figur 4-1: Nettverket simuleringen ble utført over.

Det som skal skje under simuleringen er generert av egenutviklet generator for hendelser (*ref. kapittel 3*). Her har vi satt ønsket tidsintervall til 24 timer. Det blir gjennomsnittlig hvert 15 minutt tilfeldig valgt en ny operasjon som skal utføres.

Ved simulering har vi målt hvor mye data som har blitt sendt ut på hver av transittlinkene hvert sekund og hvor stor prosent av transittlinkenes kapasitet som har blitt brukt. Ved hjelp av disse data har vi regnet ut *Linkenes belastning per tid, Nettverkets belastning totalt per tid*, *og Nettverkets entropi.*

Vi har valgt å ikke bruke noen målenhet for annet enn tiden på grafene. Siden grafene i første rekke er laget for å sammenligne løsningene er ikke målenhet nødvendig.

Siden denne oppgaven tar for seg trafikken som foregår over WAN er det først og fremst transittnettet vi ser på. Dette er et kritisk nett som blir delt av mange brukere, hvor avstander og resurser på de forskjellige linker kan variere stort. Grafene vi viser vil derfor bare ta for seg trafikken i transittnettet.

Linkenes belastning per tid forteller hvor mye data som blir sendt på hver av transittlinkene til en hver tid. Alle grafene for hver av linkene blir vist på samme figur. Det blir vist to forskjellige figurer som viser *Linkenes belastning per tid.* Den ene viser i prosent hvor mye av hver link sin totale kapasitet som blir brukt til en hver tid. Den andre viser hvor stor datamengde som passerer over hver av linkene til en hver tid. Hver av grafene tar for seg trafikken som til sammen går i begge retninger.

Total belastning per tid forteller hvor mye data som til sammen blir sendt over alle transitt linkene til en hver tid. Ved å summere alle linkenes belastning får man rede på hvor mye nettet totalt sett blir belastet. Ut i fra denne grafen ser man også hvor jevnt det totale nettverket blir belastet over tid. Grafen tar for seg trafikken som til sammen går i begge retninger.

Nettverkets Entropi forteller noe om hvor bra belastningen er fordelt over hele nettverket til en hver tid[31]. For å regne ut nettverkets entropi brukes formelen:

$$\text{Entropi}(t) = \sum_i - a_i\,(t) * \log a_i(t)$$

t – tidspunktet a er hentet fra.
i – indikerer hvilken link.
a – hvor stor del av all data som ble sendt på hver link på tidspunktet t, ble sendt på link *i.*

Entropiverdien for et tidspunkt får man altså ved å summerer verdien ($-a_i(t)*\log a_i(t)$) hver av linkene får. Tallet man kommer frem til kan ha verdien mellom [0,∞>. Siden det ikke har noen beneving, eller maks verdi, gir ikke entropiverdien noe mål alene. Den kan bare brukes til å sammenligne forskjellige løsninger, for å komme frem til hvilken av dem som er best. Resultatene forteller noe om hvor bra linkene i nettverket blir utnyttet til en hver tid. Det som blir spesielt interessant i denne oppgaven er den gjennomsnittlige entropiverdi for hver av løsningene. Gjennomsnittsverdien forteller hvor bra systemet utnytter linkene i nettverket under simuleringsperioden.

Et av målene med denne oppgaven er å belaste det underliggende nettet minst mulig. Dette målet gjelder uansett om det er store eller små mengder data som blir sendt. Siden mengden data som skal bli sendt kan avgjøre hvordan nettet blir utnyttet har vi valgt å simulere to forskjellige belastninger på nettet. Den ene tar for seg nettverket med liten belastning totalt sett over tid, hvor det kun er en stor pakke som blir sendt (se tabell 4-1). Den andre tar for seg nettverket med stor belastning totalt sett over tid, som består av seks store pakker (se tabell 4-2). Et punkt å merke seg her er at den ene pakken som blir sendt ved liten belastning blir sendt med

nøyaktig de samme kriterier ved simulering med stor belastning. Ved å sammenligne hvordan denne data blir sendt på nettet i hver av tilfellene får man opplysninger om hvor bra nettverket blir benyttet. Man får se om dataen belaster nettverket like jevnt i begge tilfellene, som vil være det ideelle.

For å kunne sammenlikne de forskjellige løsningene må hver av dem bli testet med samme test data. Testdata vi har brukt er vist i tabell 4-1 og 4-2.

Tabell 4-1: Testdata for liten belastning

	Starttid	Deadline	Størrelse	Ant. Mottakere
Pakke 1	1675	3115	2325 Mb	3

Tabell 4-2: Testdata for stor belastning

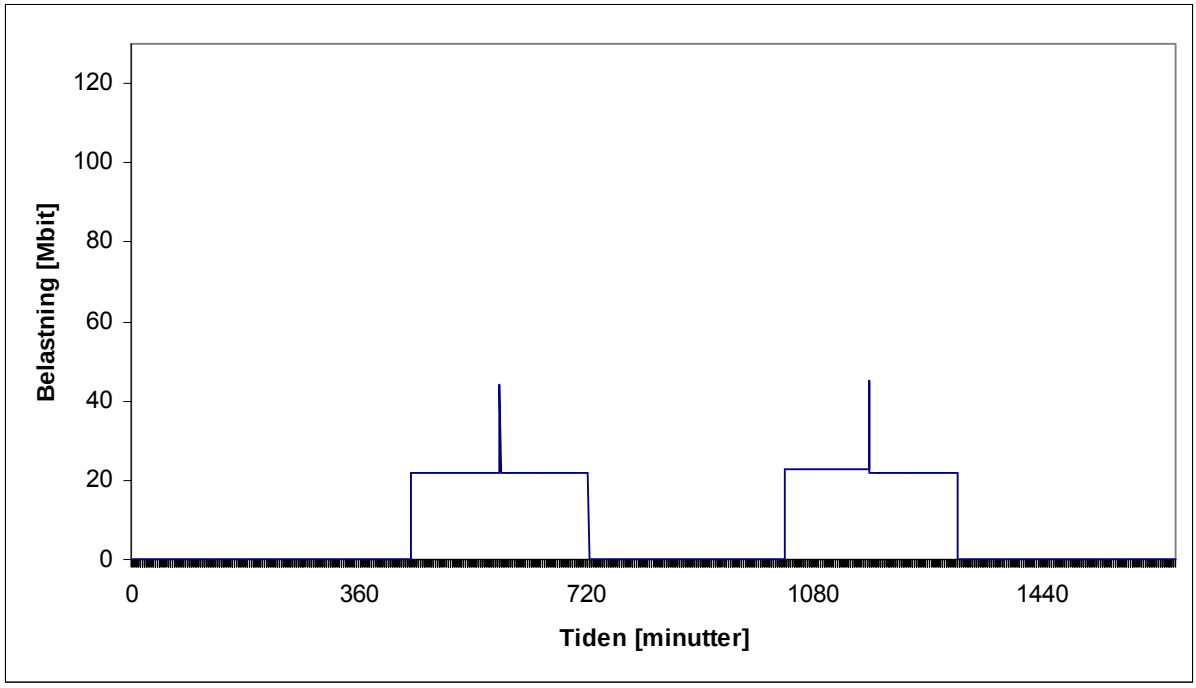
	Starttid	Deadline	Størrelse	Ant. Mottakere
Pakke 1	1675	3115	2325 Mb	3
Pakke 2	1681	2623	688 Mb	11
Pakke 3	1800	3006	768 Mb	15
Pakke 4	1986	2498	637 Mb	7
Pakke 5	2256	2897	647 Mb	2
Pakke 6	2554	3115	723 Mb	3

Vi valgte å utføre ti simuleringer ved hver av løsningene. Dette forsikrer at simuleringen ikke er et enkelt tilfelle. Vi valgte å plukke ut den simuleringen som hadde verdier nærmest gjennomsnittsverdien til simuleringene, i stedet for å ta for oss gjennomsnittsverdien. Grunnen til dette er fordi gjennomsnittsverdien kan gi et feil bilde av hvordan systemene fungerer. Under noen av simuleringene må det nemlig forekomme tidspunkt hvor transittnettet ikke blir brukt. Siden dette tidspunktet kan variere for hver gang, kan det ut i fra en gjennomsnittsverdi se ut som transittnettet har blitt brukt under hele simuleringen. De utvalgte simuleringene vil nå bli vist i de påfølgende underkapitler.

4.1 Siena

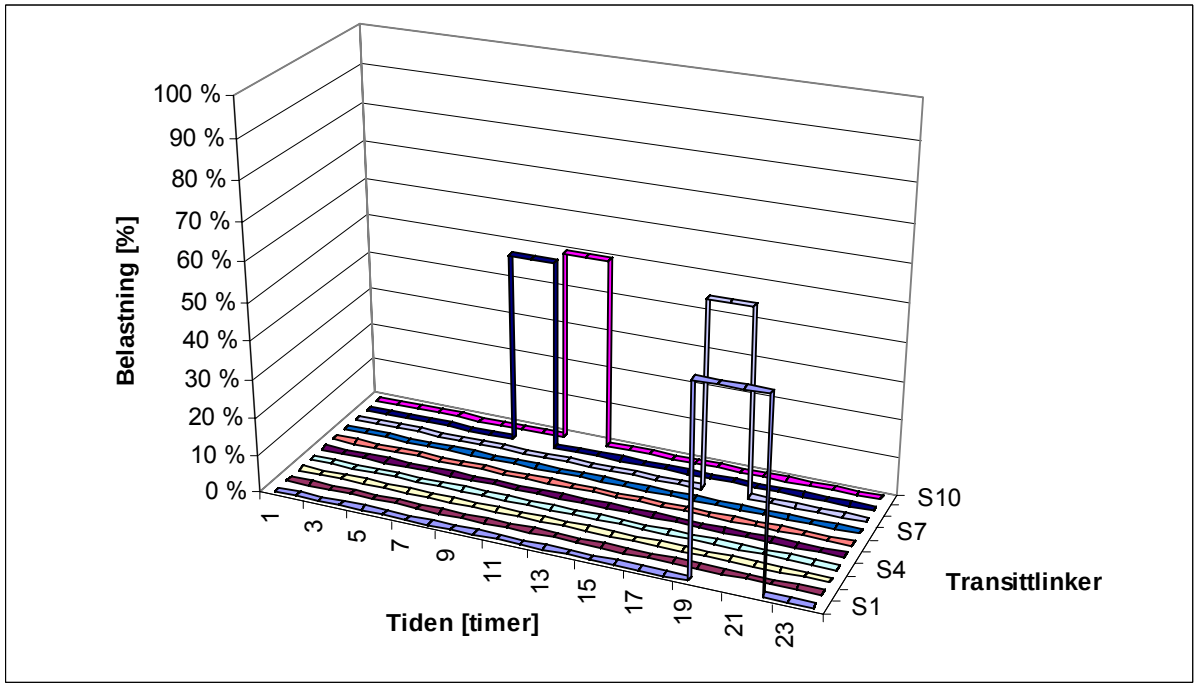
Den genuine Siena er utviklet for distribusjon av relativt små pakker (vanligvis på ca. 166 byte). Det er derfor spesielt interessant for oss å se hvordan oppførselen blir når opptil ca. 14000000 ganger så store pakker blir sendt gjennom nettverket.

4.1.1 Belastning



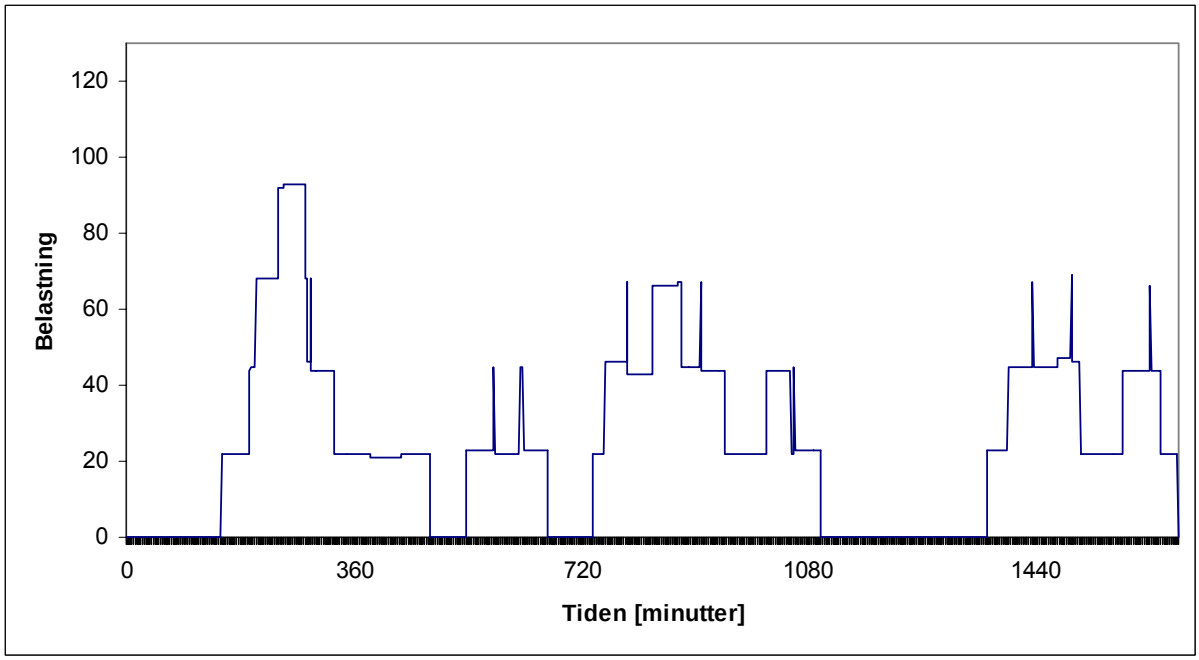
Figur 4-2: Siena: Transittnett med liten belastning

På Figur 4-2 blir transittnettet målt med en liten belastning. Ved denne simuleringen var pakken fremme til siste mottaker etter ca. 27,5 timer.



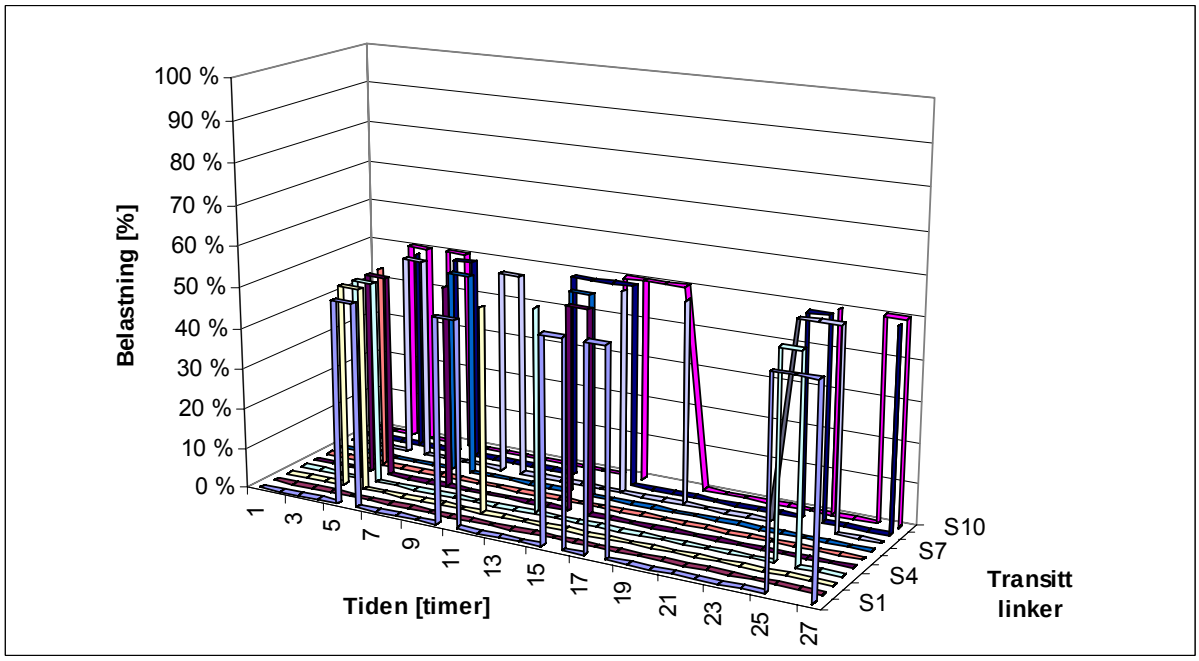
Figur 4-3: Siena: Transittlinkenes belastning (%) ved liten belastning

Figur 4-3 viser hvor stor prosent av linkenes kapasitet som blir brukt hver time. Ut i fra grafen ser man at når noe blir sendt på en link så benyttes det 50% av linkens kapasitet. På linken kan pakkene sendes i begge retninger, 50% hver vei.



Figur 4-4: Siena - Transittnett med stor belastning

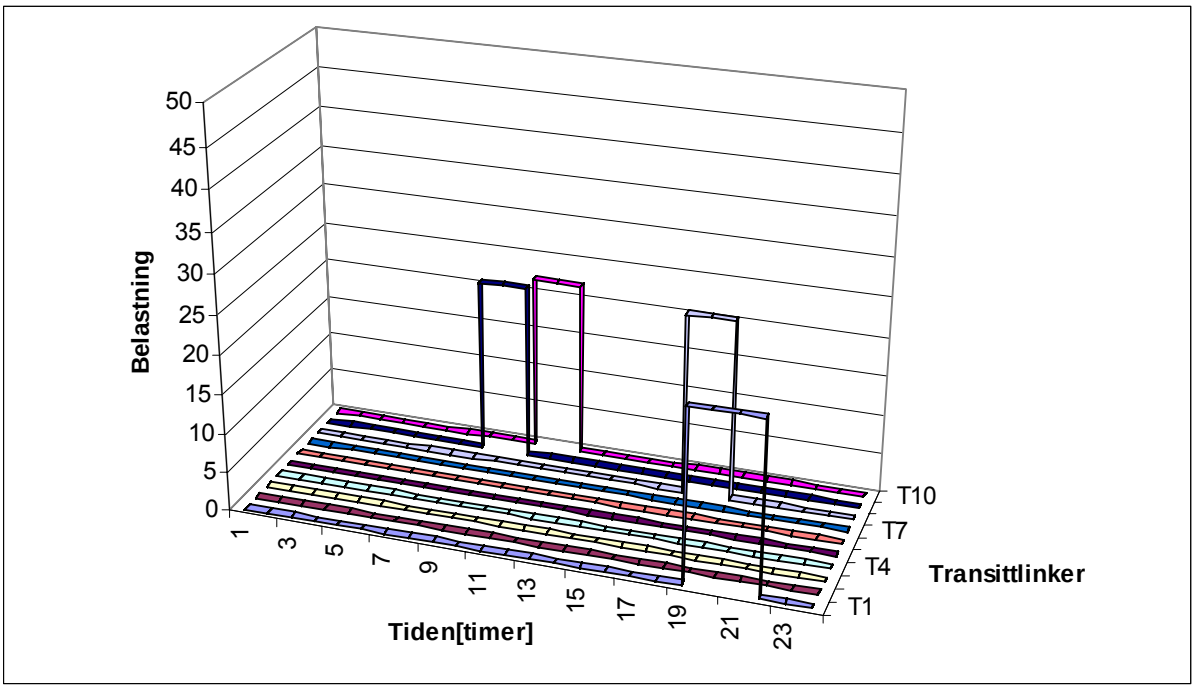
Figur 4-4 viser hvor stor belastning nettverket hadde totalt til en hver tid. På denne simuleringen måtte det brukes nærmere 41.5 timer før all data nådde frem. På grunn av Siena sine transittnoder bare kan sende data over en link om gangen må trafikken på transittnettet vente hvis overføring til en stubbnode blir prioritert. Belastningen på transittnettet varierer derfor mye fra ingen til en belastning på mer enn 90 Mbit per minutt. Grafen viser bare over tiden transittnettverket blir belastet.



Figur 4-5: Siena: Transittlinkenes prosentvis belastning ved stor belastning

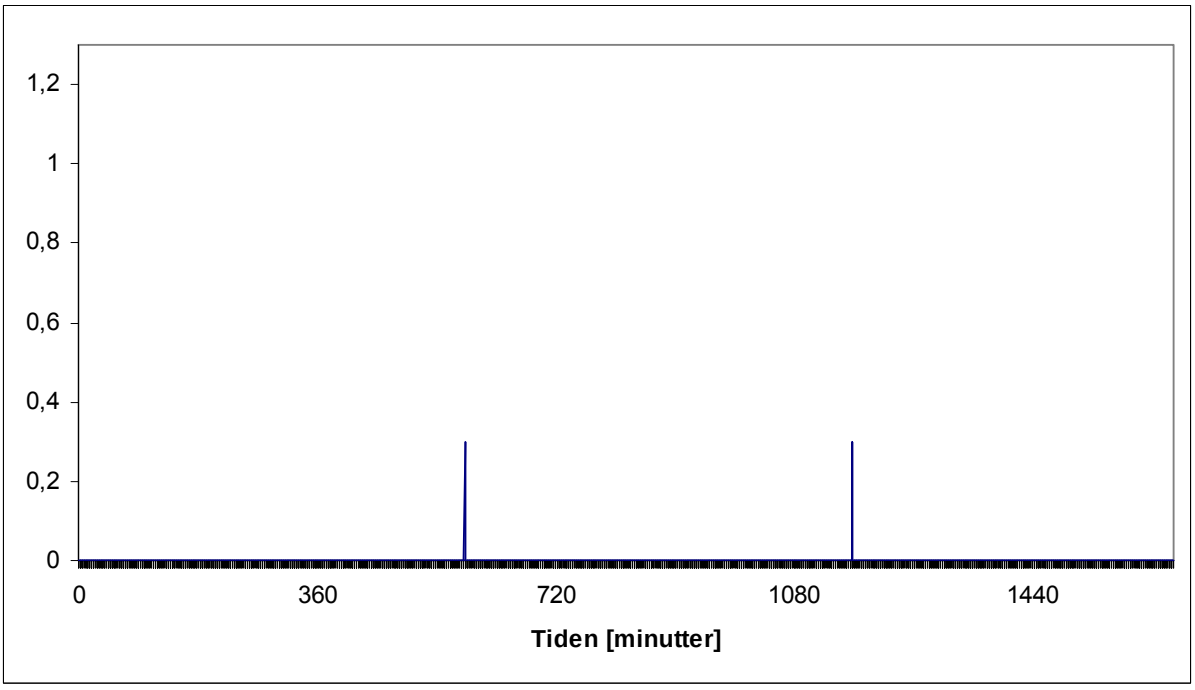
Figur 4-5 viser hvor mange prosent av hver transittlink som benyttes til enhver tid. Linkene kan sende i begge retninger, 50% hver vei.

4.1.2 Spredningsmål



Figur 4-6: Siena - Belastning på hver av transittlinkene ved liten belastning

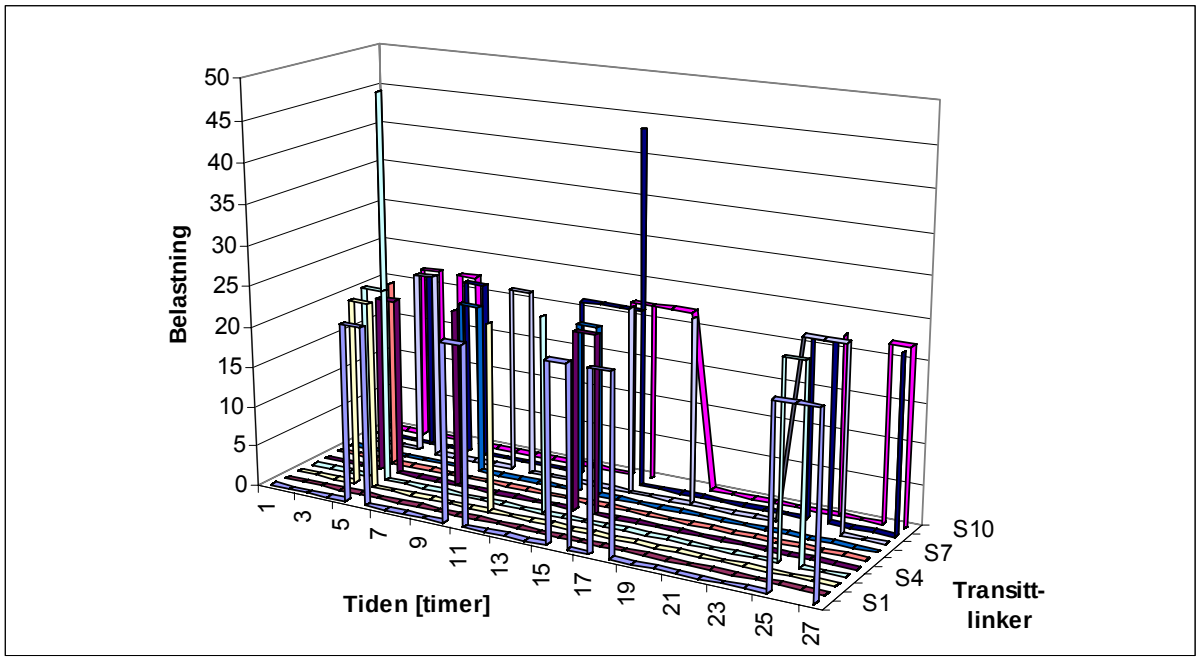
Figur 4-6 viser belastningen på hver av transittlinkene for hver av de 24 timene. Sammenliknet med figur 4-7 ser vi at når trafikken går fra en link over til en annen så får vi et utslag i entropi grafen.



Figur 4-7: Siena: Entropi over transittnett med liten belastning

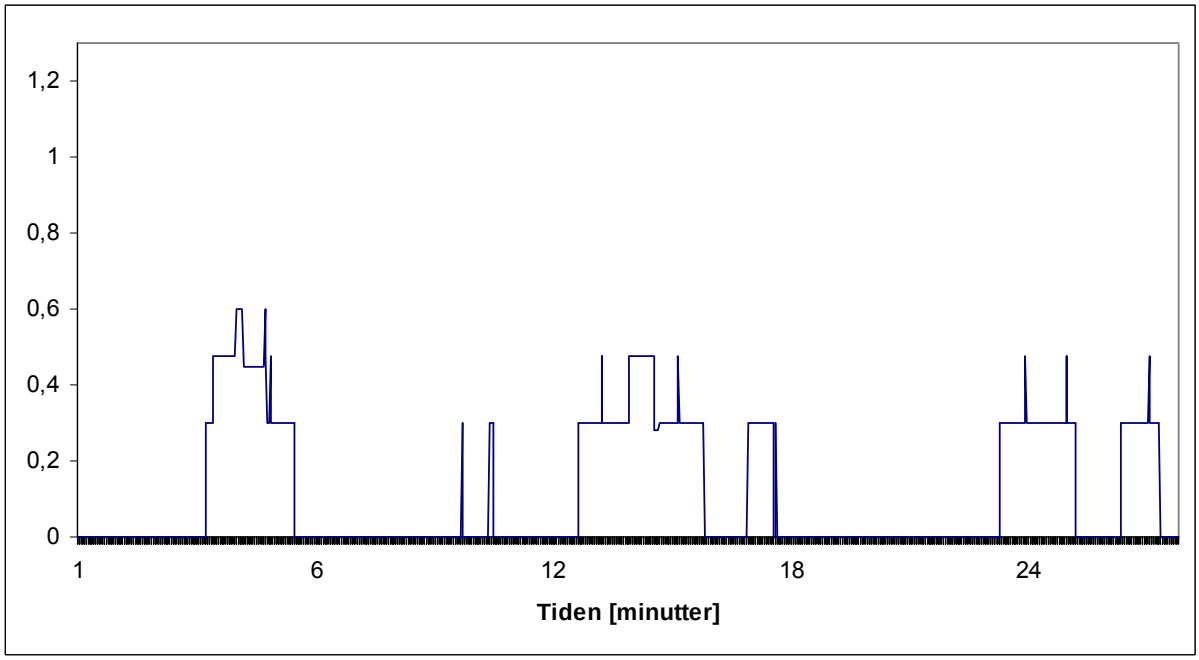
Figur 4-7 viser entropiverdien nettet har der begrenset mengde data blir distribuert. Ut i fra grafen ser man at all distribusjon foregår bare på en link av gangen, noe som gir entropi verdi null. De tynne søylene som har dukket opp indikerer tidspunktet dataoverføringen går over fra en link til en annen. I den genuine Siena er dette

strengt tatt ikke mulig, men ettersom pakkene transporteres så raskt, har en transittnode mottatt pakken og klart å videresende pakken før minuttet på grafen er over.



Figur 4-8: Siena - Belastning på hver av transittlinkene ved stor belastning

Figur 4-8 viser belastning for hver av transittlinkene i løpet av tiden alle de seks pakkene blir distribuert over nettet.



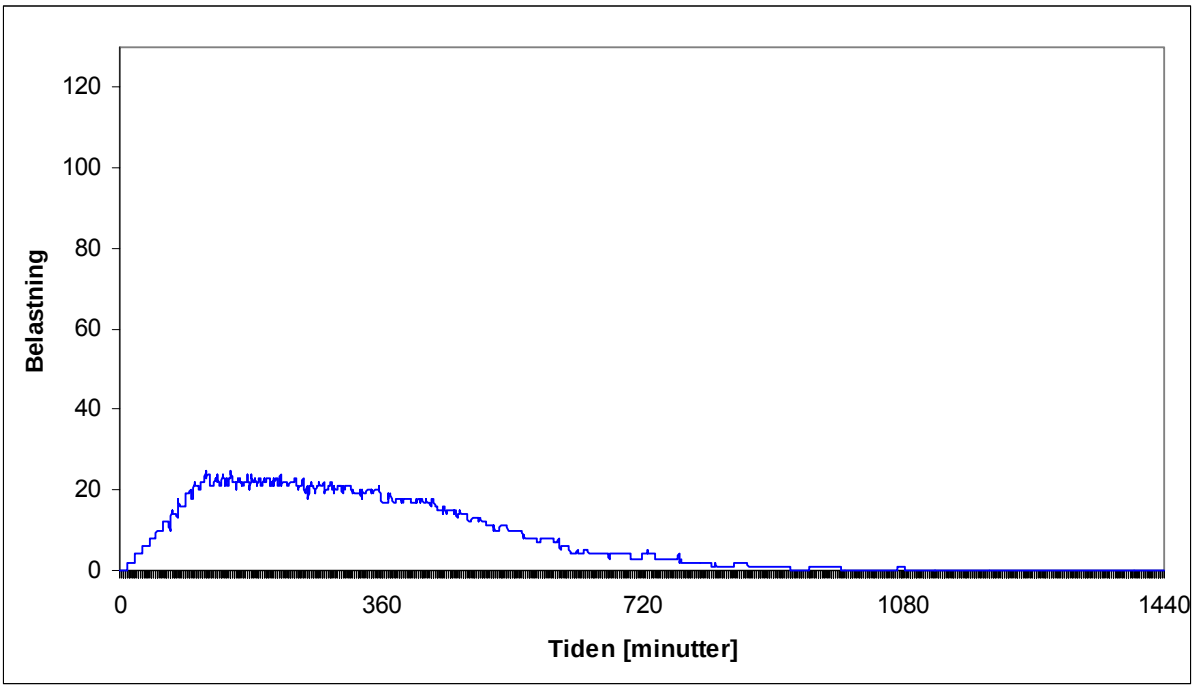
Figur 4-9: Siena - Entropi over transittnett med stor belastning

Figur 4-9 viser entropien over transittnett der store mengder data blir distribuert. Vi ser her, som ved liten belastning, at grafen store deler av tiden har en entropiverdi lik 0. Som tidligere nevnt tyder dette på at bare en link ofte blir belastes om gangen, eller at det her ikke finnes noe data som transporteres i nettet.

4.2 Den utvidede Siena

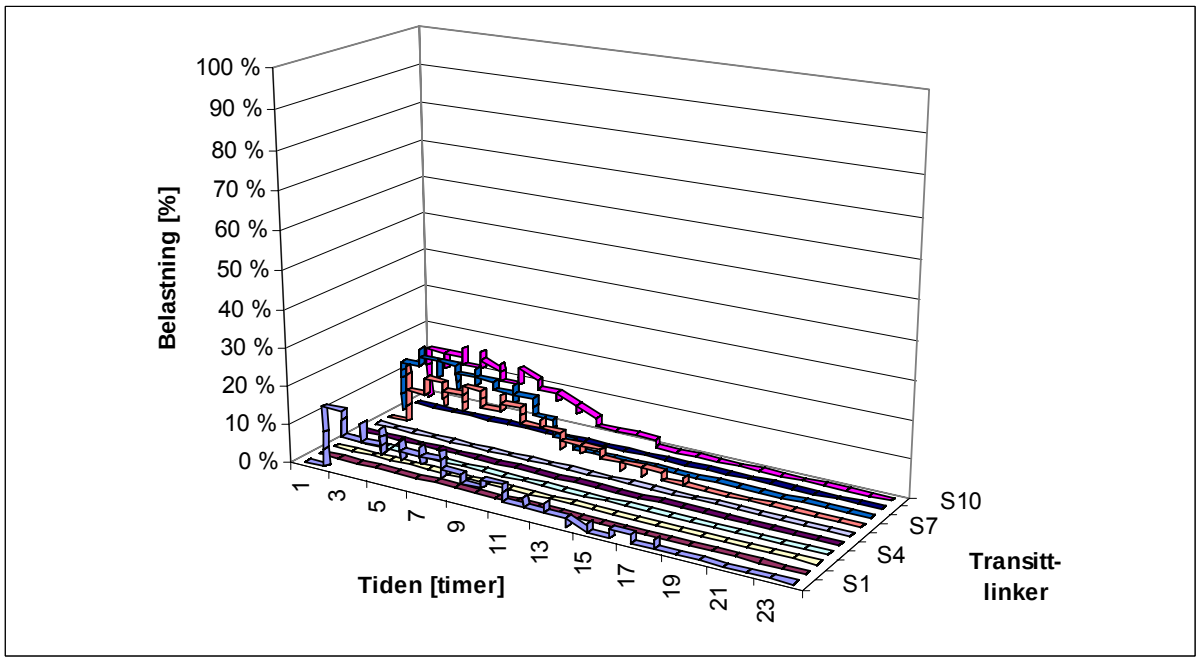
Vi skal her se på de samme målingene for den utvidede versjon av Siena som vi gjorde for den genuine Siena.

4.2.1 Belastning



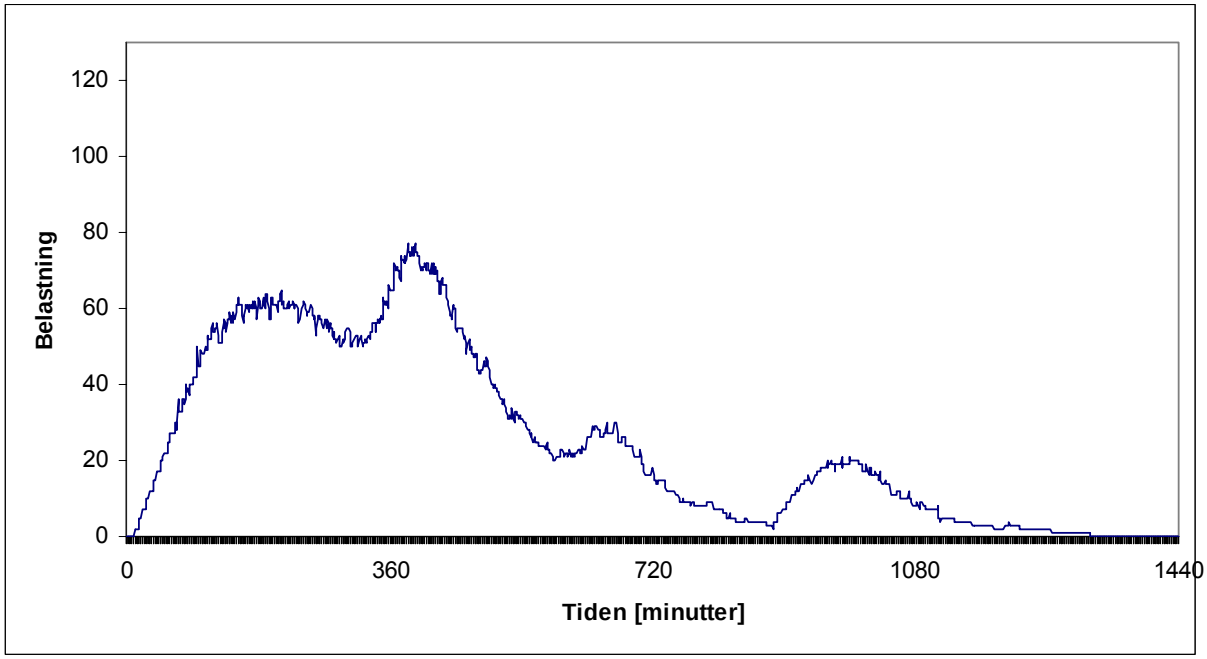
Figur 4-10: Siena Utvidet - Transittnett med liten belastning

Figur 4-10 viser hvor mye trafikk som går igjennom transittlinkene. Grafen viser summen av belastningen for alle transittlinkene for hvert sekund.



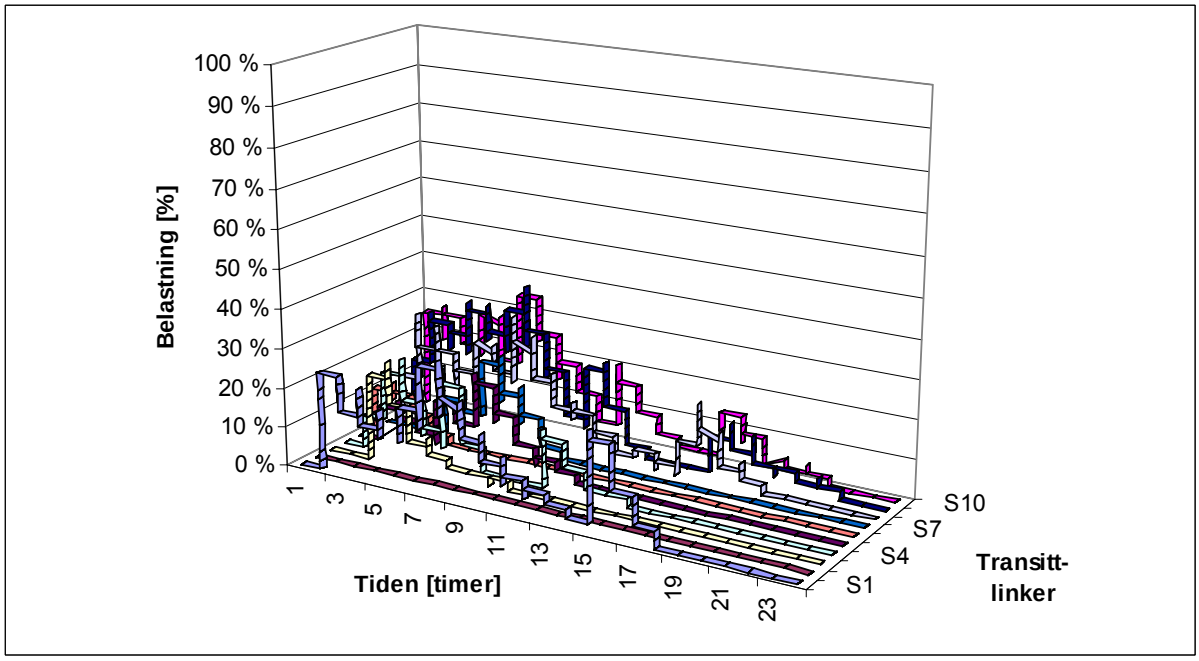
Figur 4-11: Siena Utvidet - Transittlinkenes belastning (%) ved liten belastning

Figur 4-11 viser hvor stor prosent av linkens kapasitet som brukes i løpet av testperioden. Vi ser at linkene har en del kapasitet å gå på. Grunnen til at den benytter liten kapasitet er fordi det ikke kreves mer for at pakken skal komme frem til tidsfristen. Også her er linkenes totale kapasitet begge veier 100%, 50% hver vei.



Figur 4-12: Siena Utvidet - Transittnett med stor belastning

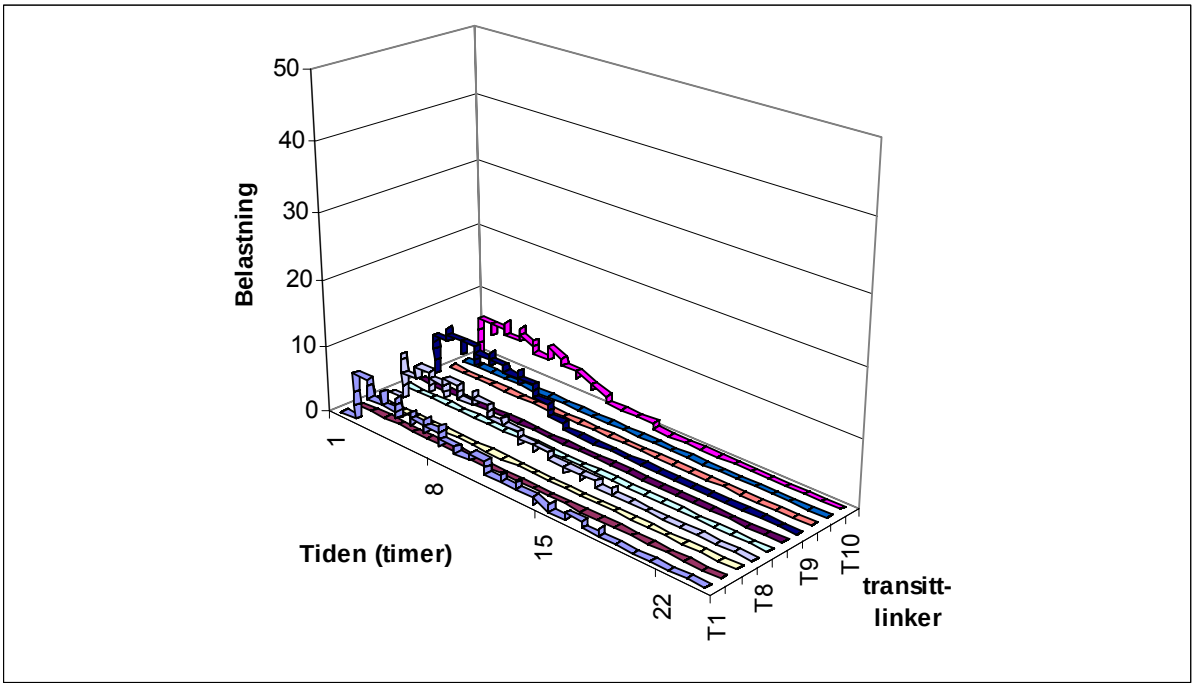
Figur 4-12 viser belastningen på alle linkene totalt hvert minutt i testperioden når det sendes seks pakker over transittnettet. Hensikten med målingen er i tillegg til å se hvor mye data som kan sendes over nettet, også om den greier å sende pakkene til mottakerne innen tidsfristen. I tillegg ser vi om nettet blir jevnt belastet over tid i forhold til mengden data som sendes.



Figur 4-13: Siena Utvidet - Transittlinkenes belastning (%) ved stor belastning

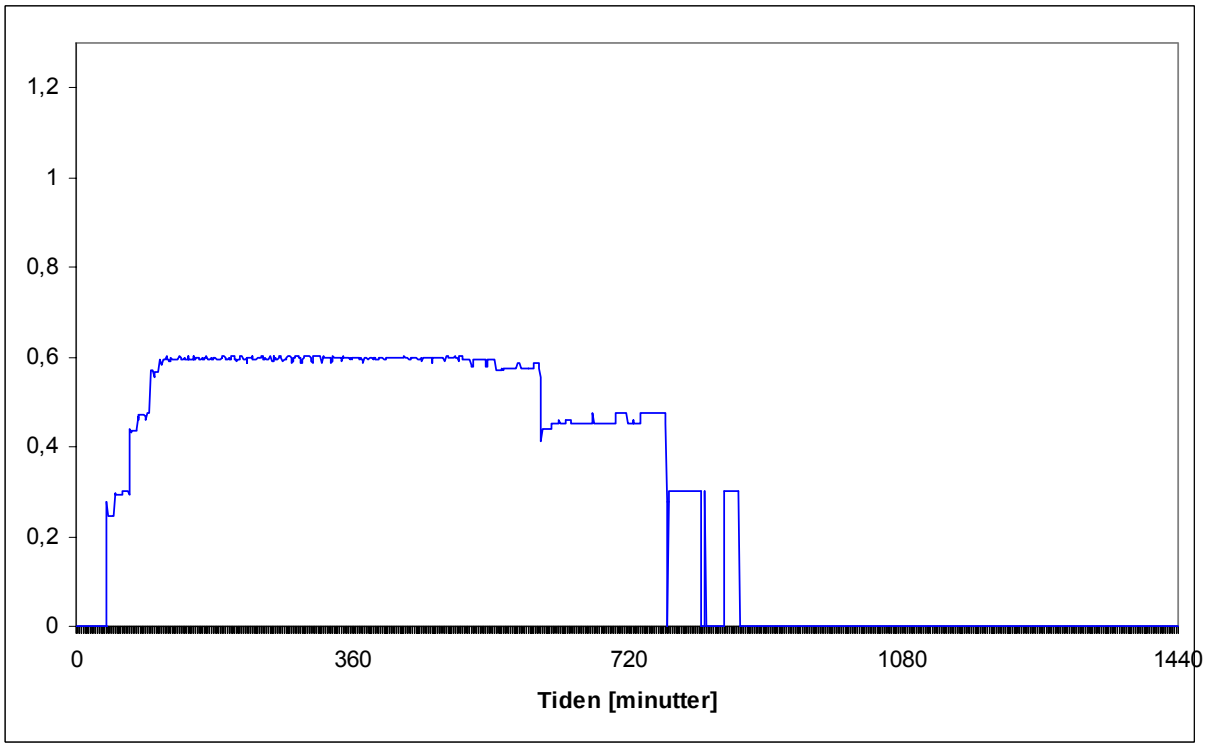
Figur 4-13 viser hvor stor prosent av transittlinkenes kapasitet som brukes for hver time i testperioden. Vi ser ut i fra grafen her at linkene er langt i fra overbelastet.

4.2.2 Spredningsmål



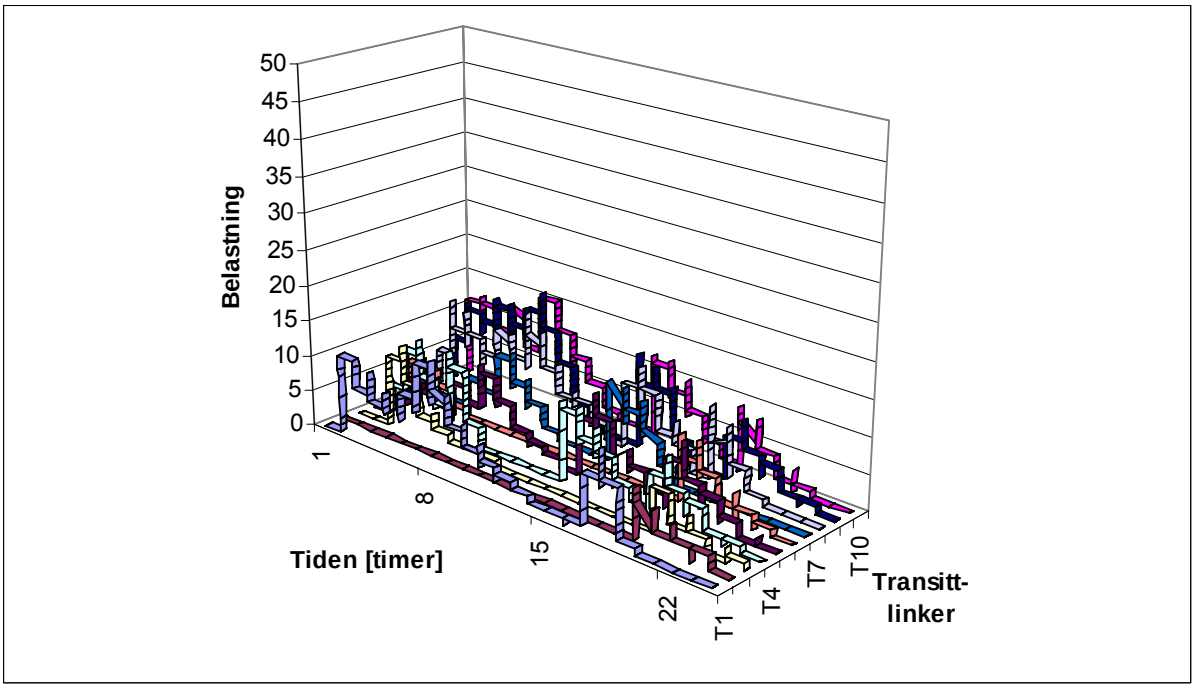
Figur 4-14: Siena Utvidet - Belastning på hver av transittlinkene ved liten belastning

Figur 4-14 viser belastning på hver av transittlinkene i løpet av det døgnet pakken ble sendt over nettet. Denne grafen og figur 4-13 viser at linkene, i motsetning til den genuine Siena, benytter kun den transmisjonsraten de behøver for å få levert pakken til tidsfristen.



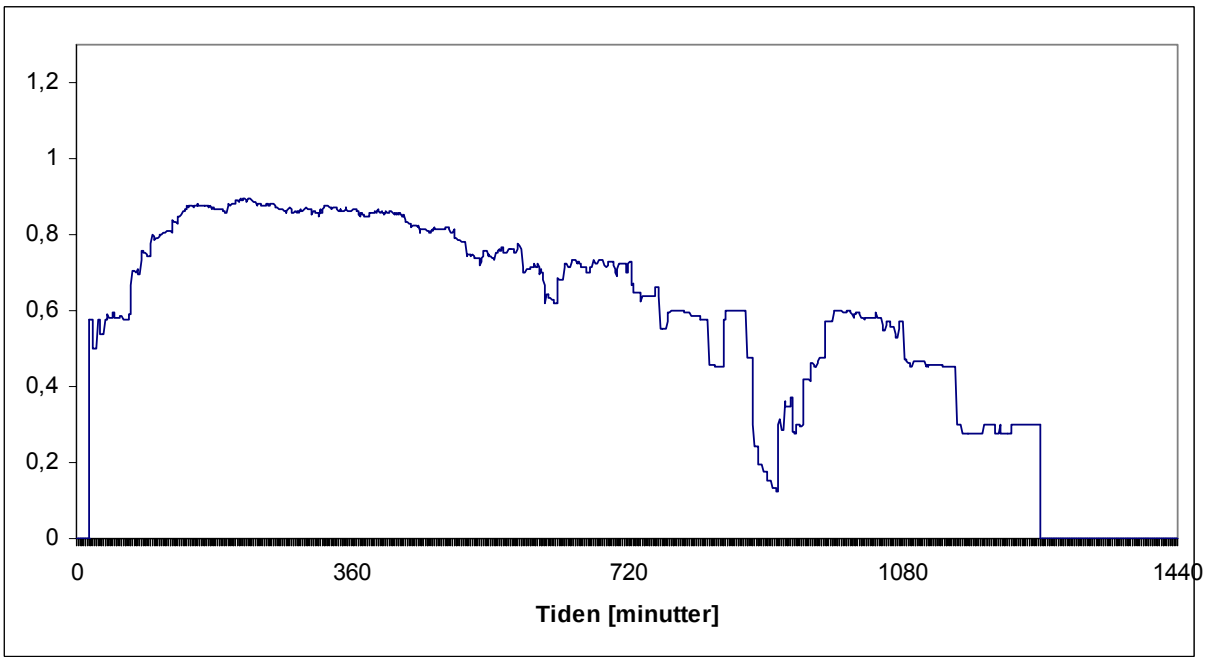
Figur 4-15: Siena Utvidet – Entropi over transittnett med liten belastning

Figur 4-15 viser entropien for transittnettet når det blir sendt en pakke over nettet. Grafen viser til tider ingen verdi. Dette vil som sagt ikke nødvendigvis si at det ikke blir sendt noe trafikk over noen linker på tidspunktet. Hvis bare en link benyttes ved et tidspunkt, vil det si at Entropien er dårlig, og den vil gi en null verdi på grafen.
Den gjennomsnittlige entropiverdi ble her utregnet til å være *0,29* i løpet av 24 timer.



Figur 4-16: Siena Utvidet – Belastning på hver av transittlinkene med stor belastning

Figur 4-16 viser belastning på hver av transittlinkene for hver time i et døgn når seks pakker blir sendt over transittnettet.



Figur 4-17: Siena Utvidet - Entropi over nettet med stor belastning

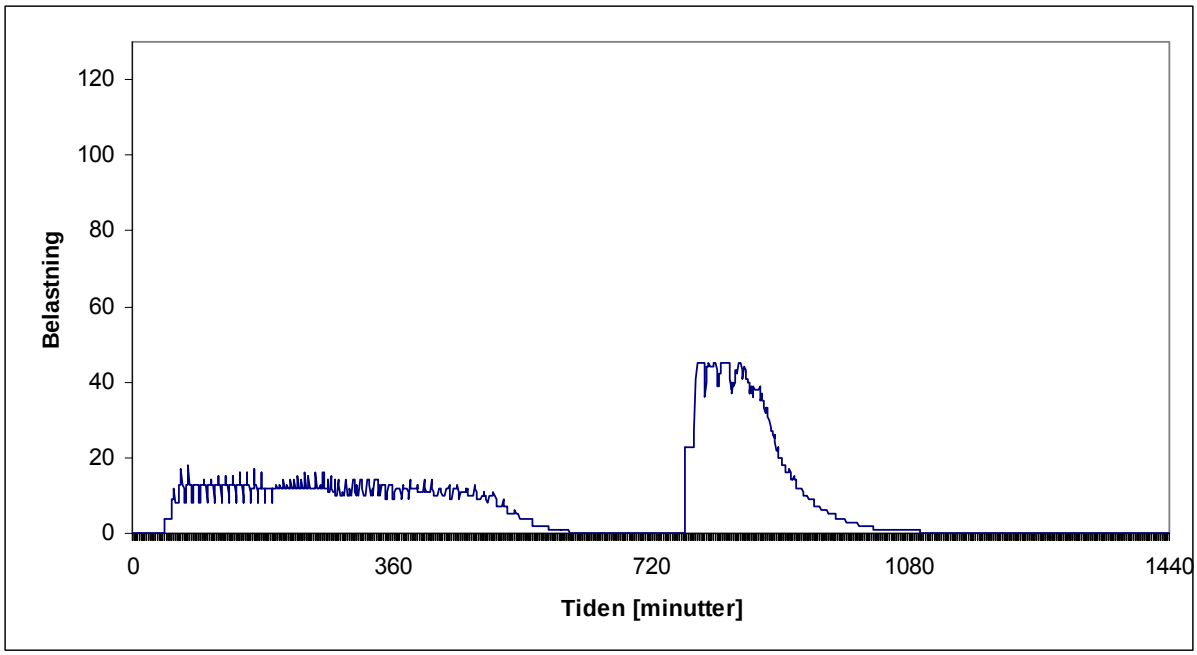
Figur 4-17 viser entropien for transittnettet når det sendes seks pakker over nettet. Den gjennomsnittlige entropiverdi ble her utregnet til å være *0,56* i løpet av 24 timer.

4.3 Siena Epidemisk

En videre utvidelse vi har foretatt på den utvidede Siena er å benytte *Epidemisk* (gj.f. kapittel 2). Grunnen til at vi valgte å utvide Siena med denne algoritmen er for å se om det er mulig å spre belastningen jevnt over hele nettverket. Den epidemiske *Siena* har de samme utvidelsene som den utvidede Siena, men definisjonen av tidsfristen er forandret.

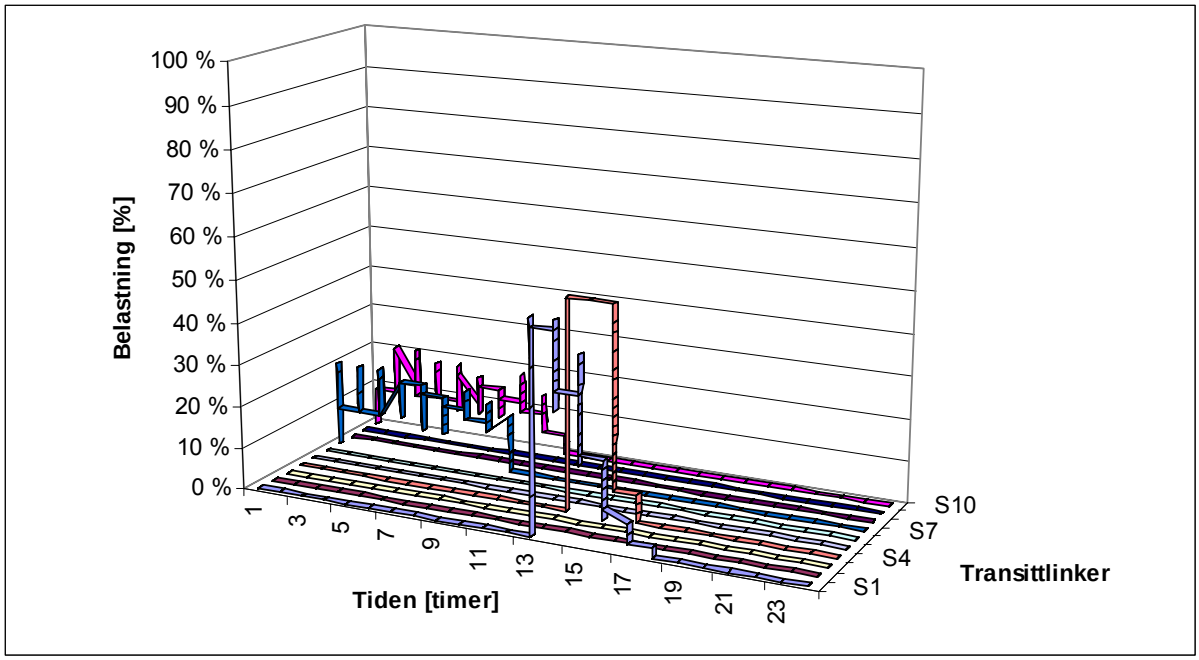
Tidsfrist: Prinsippet bak tidsfrist er den samme som for den utvidede Siena. Det som skiller disse er at ved denne utvidelsen vil ikke alle, men en stor andel av mottagerne få pakken akkurat på tidsfristen. I motsetning til den utvidede Siena vil alle mottagere få pakken akkurat på tidsfristen, forutsatt at det er mulig.

4.3.1 Belastning



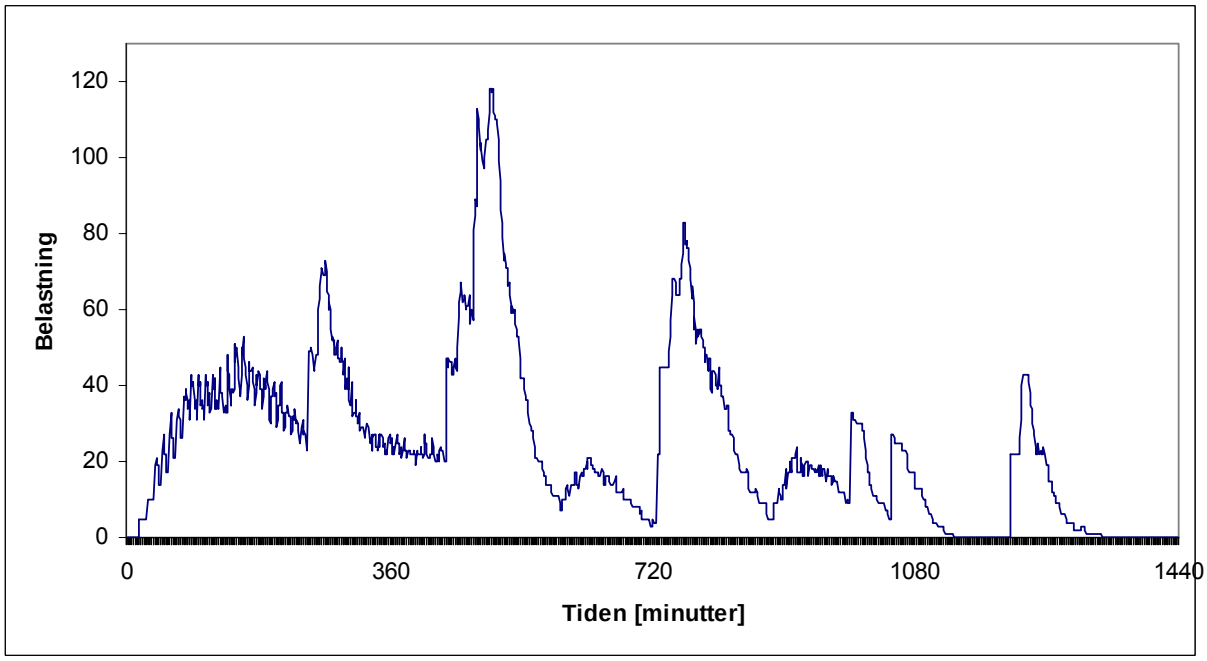
Figur 4-18: Siena Epidemisk - Transittnett med liten belastning

Figur 4-18 viser belastningen over transittnettet når en pakke distribueres.



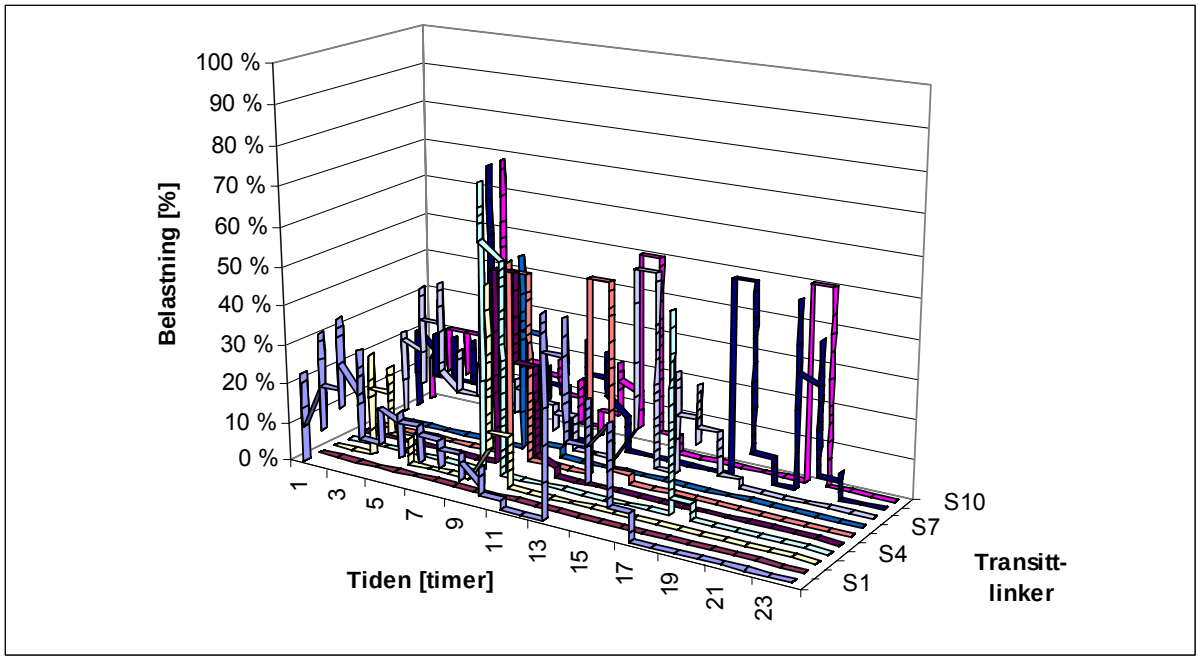
Figur 4-19: Siena Epidemisk - Transittlinkenes belastning (%) ved liten belastning

Figur 4-19 viser hvor stor prosent av transittlinkenes kapasitet som blir brukt hver time. Som tidligere nevnt har hver link totalt 100% kapasitet til sammen begge veier.



Figur 4-20: Siena Epidemisk - Transittnett med stor belastning

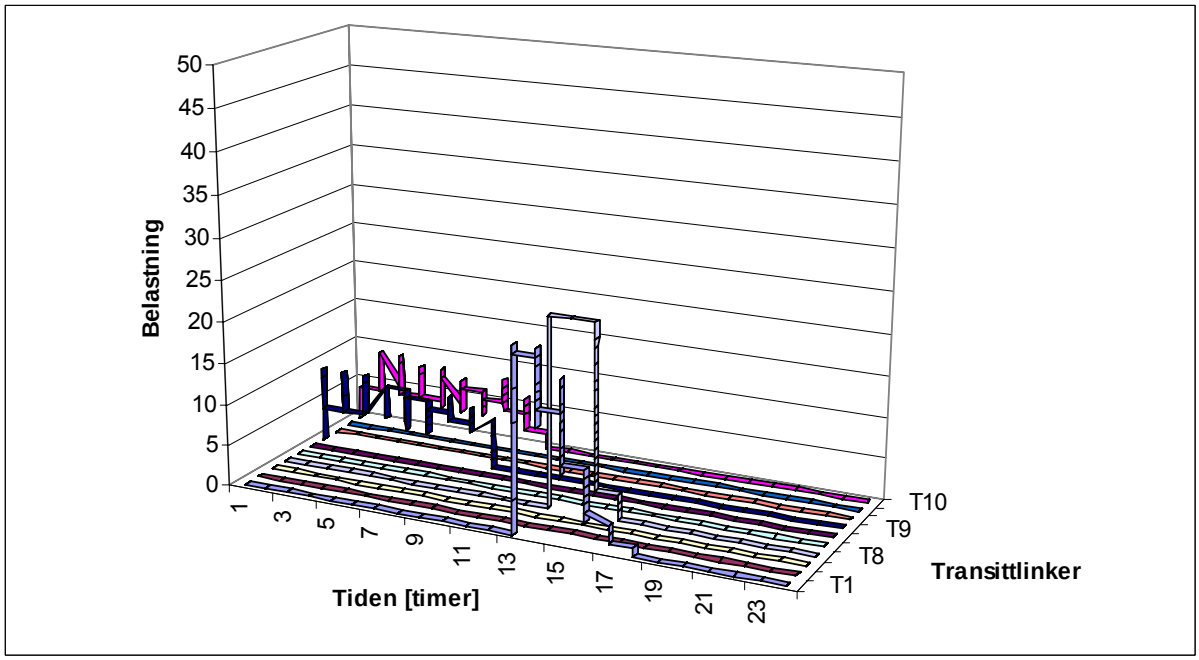
Figur 4-20 viser hvor stor belastning transittnettet har totalt for hvert minutt i testperioden.



Figur 4-21: Siena Epidemisk - Transittlinkenes belastning (%) ved stor belastning

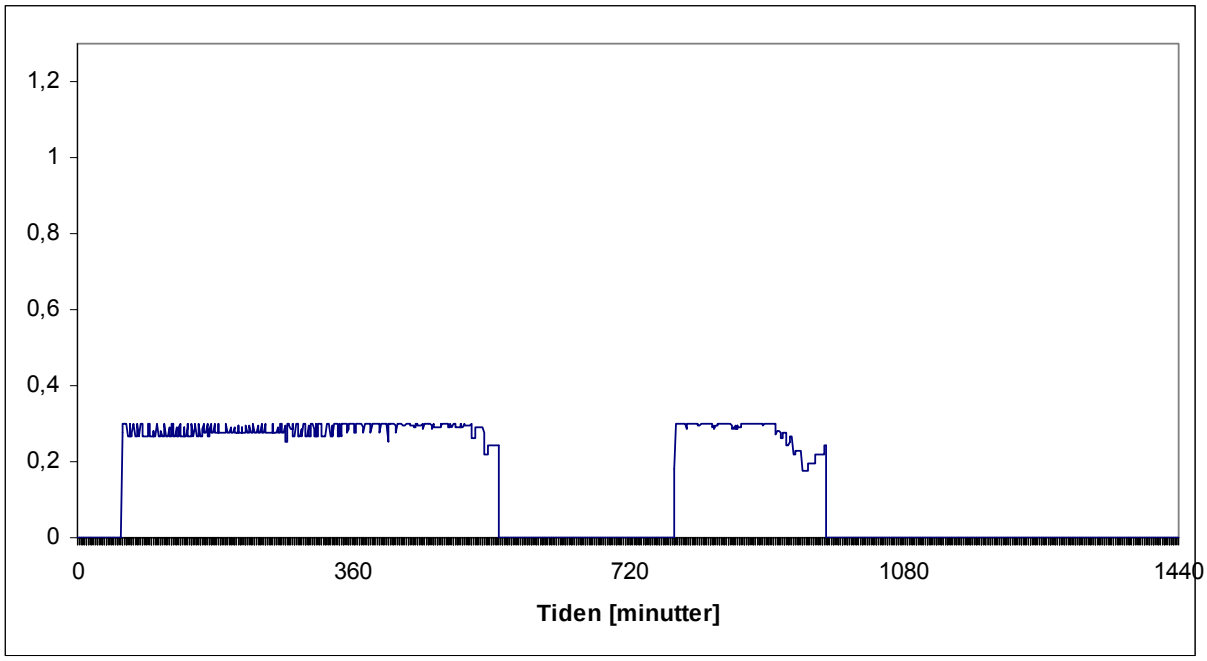
Figur 4-21 viser hvor mange prosent hver av transittlinkene benytter til enhver tid. I grafen ser vi at det til tider sendes pakker begge veier på linken. Dette vises ved at grafen overstiger sin 50% kapasitet per vei.

4.3.2 Spredningsmål



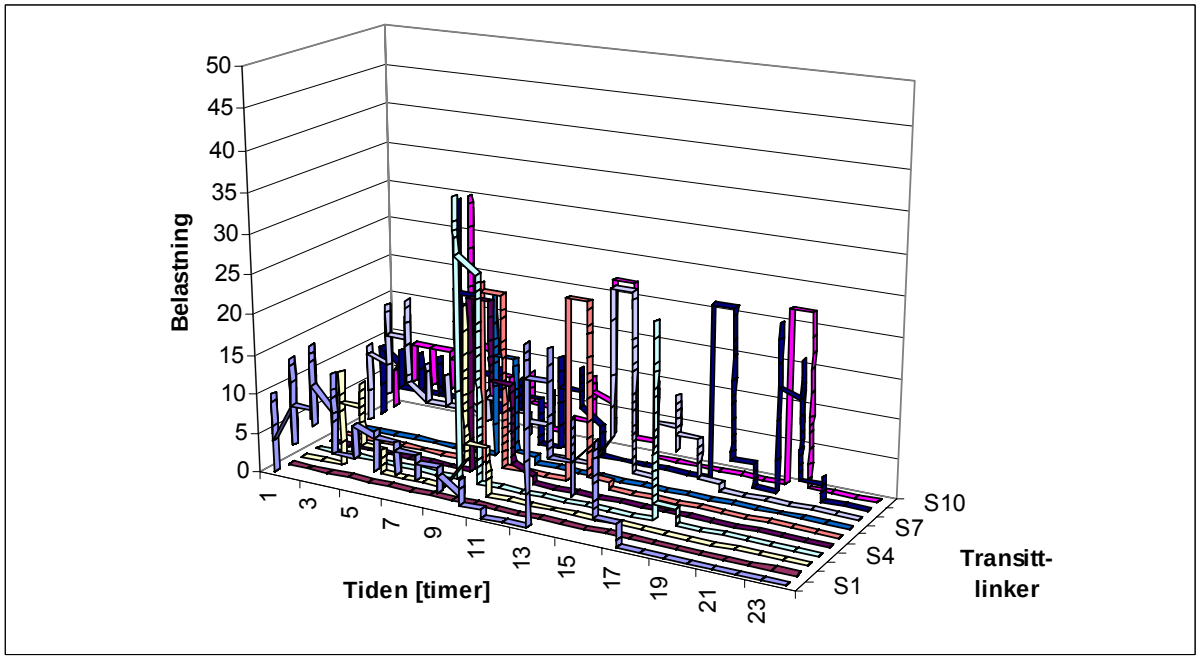
Figur 4-22: Siena Epidemisk - Belastning på hver av transittlinkene ved liten belastning

Figur 4-22 viser belastningen på hver av transittlinkene hver time når en pakke sendes over nettet. Ut i fra grafen ser vi at spesielt to linker på et tidspunkt er høyt belastet.



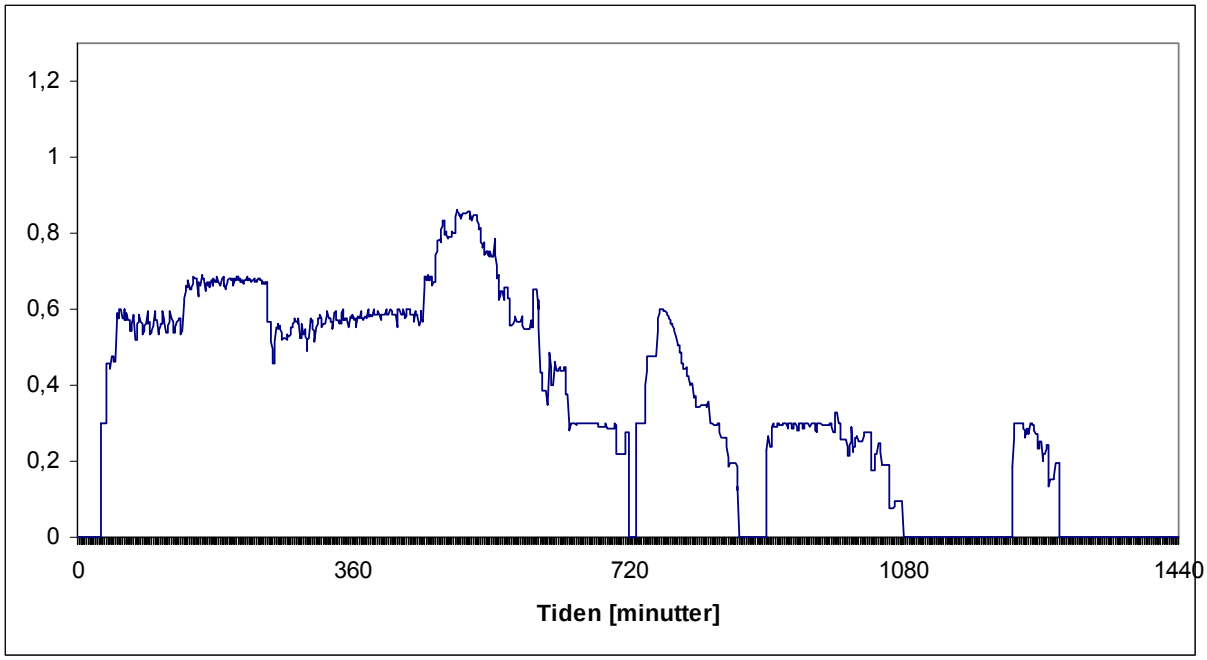
Figur 4-23: Siena Epidemisk - Entropi over transittnett med liten belastning

Figur 4-23 viser entropien for transittnettet når en pakke sendes over nettet.
Den gjennomsnittlige entropiverdi ble her utregnet til å være *0,14* i løpet av 24 timer.



Figur 4-24: Siena Epidemisk - Belastning på hver av transittlinkene med stor belastning

Figur 4-24 viser hver av transittlinkenes belastning for hver time i testperioden når seks pakker distribueres over nettet.



Figur 4-25: Siena Epidemisk - Entropi over nettet med stor belastning

Figur 4-25 viser entropien for transittnettet når seks pakker sendes over nettet. Den gjennomsnittlige entropiverdi ble her utregnet til å være *0,35* i løpet av 24 timer.

5 Drøfting

I denne oppgaven skulle vi se hvor vidt Siena egner seg for distribusjon av store mengder data, og eventuelt komme med forbedringer. Vi startet med å forklare problemstillingen før vi tok for oss de forskjellige teoretiske bakgrunn man må ha kjennskap til. Vi kom frem til to forskjellige løsninger som ble beskrevet i kapittelet *Arbeid utført*. I dette kapittelet skal vi nå drøfte resultatene vi fikk i forrige kapittel. Vi vil starte med å se på resultatene fra hver av testmetodene, før vi runder av med å sammenligne dem. Vi ser ikke på hver av grafene, men bare de som fremhever spesielle sider ved metodene.

5.1 Siena

En sterk side ved Siena er at det ved hjelp av distance-vector blir satt opp et multicast-tre før dataen blir sendt på nettverket. Ved å sette opp et tre på denne måten forhindrer den at redundans oppstår. Den prøver å sette opp et minimalt spenntre mellom interesseholder og interessenter. For å komme frem til denne blir det tatt hensyn til latenstiden mellom nodene og antall noder som passeres. Siena er i første rekke beregnet for små pakker. Siden pakker av liten størrelse har minimal innvirkning på tiden en overføring tar den ikke hensyn til pakkens størrelse. Derimot i denne oppgaven, hvor vi tar for oss distribusjon av store data pakker vil dette bli annerledes. Her sendes det så mye data at overføringstiden blir betydelig lenger, og den totale distribusjonstiden vil øke. Grunnen til dette er fordi rutingen til Siena foregår på applikasjonslaget, noe som gjør at data ikke kan videresendes før hele pakken er mottatt. Dette er en av årsakene til at Siena er så lang tid på å overføre data under simuleringene vi foretok.

Ser vi nærmere på Figur 4-2 (*Siena - Transittnett med liten belastning)* ser det ut som transittnettet blir dårlig utnyttet. Grafen viser at Transittnettet har en lang pause midt under distribusjonen. Grunnen er en svakhet ved den genuine Siena. Hver av nodene kan bare sende på en link om gangen. Siden denne grafen bare viser trafikken mellom transittnodene blir det ikke synlig når data blir sendt fra transittnode til stubnode. Dette er altså hva som skjer i begynnelsen og midt under denne simuleringen. Dette kommer også tydelig frem i Figur 4-4 (*Siena - Transittnett med stor belastning)*. Selv om det her skal være stor belastning over nettet mesteparten av tiden, er det flere steder som transittnettet ikke blir brukt i det hele tatt. Antall abonnenter kan derfor være med på å øke den totale overføringstiden betraktelig.

Når vi ser på Siena sine *s*predningsmål er det et nytt bevis på at nettverkets resurser blir dårlig utnyttet. Dens *Entropi over transittnett med liten belastning* på Figur 4.7 viser en verdi lik null nærmest hele tiden, mens *Entropi over transittnett med stor belastning* på Figur 4.9 viser verdi lik null rundt halvparten av tiden. Siden null er den laveste entropi verdi som er mulig tyder dette på liten spredning over geografisk avstand. Dette kommer også klart frem når man ser på Figur 4.6 (*Siena - Belastning på hver av transittlinkene ved liten belastning*) som viser at bare én link belastes til enhver tid.

Det er tydelig at maksimal overføringshastighet blir brukt mellom transittnodene hele veien. Dette kommer frem på figur 4.3 (*Siena – Transittlinkenes belastning (%) ved liten belastning*) som viser hvor mye av hver links kapasitet som blir brukt til enhver tid. Siden figuren tar for seg store mengder data som en node sender vet vi at ingen

av linkene blir belastet i begge retninger samtidig. Vi kan dermed konkludere at maks båndbredde blir brukt så fort noe blir sendt, siden hver av grafene viser en belastning på enten 0% eller 50%. Dette kan få konsekvenser hvis en link blir opptatt over lenger tid samtidig som annen data venter på å bli sendt.

5.2 Den utvidede Siena

Hovedforandringen som er gjort med Siena Utvidet i forhold til Siena er at dataen som sendes blir delt opp i et antall fragmenter bestemt etter dataens størrelse. I tillegg benytter pakkene seg av tiden den har tilgjengelig før den må være fremme til tidsfristen. Vi skal her drøfte denne løsningen.

Siden den utvidede Siena er en forbedring av den genuine Siena med tanke på distribusjon av store pakker, er selvsagt den genuine Siena sine positive sider beholdt. Sammenliknet med den genuine Siena er den like skalerbar og den belaster nettet minimalt med tanke på redundans. Den sørger også for å finne den mest egnede spenntreet, men dette gjøres på en annen måte.

Ved distribusjon av små pakker er det først og fremst latenstiden som avgjør hvor lang tid det tar å overføre en pakke over en link. Hvis man øker størrelsen på pakken minsker latenstidens betydning samtidig som pakkens størrelse får mer og mer innflytelse på overføringstiden. I den utvidede Siena tar distance-vectoren hensyn til linkenes båndbredde fremfor dens latenstid, noe som kan være med på å redusere overføringstiden av data.

En viktig forbedring med den utvidede Siena er at dataen som sendes blir fragmentert. I stedet for at transittnodene må vente med å videresende til de har mottatt alt, kan de starte overføringen så fort et nytt fragment blir mottatt. Dette kommer tydelig frem i figur 4.10 (*Siena Utvidet - Transittnett med liten belastning)*, hvor man ser transittnoden begynner å sende data kort tid etter t = 0, som var tidspunktet stubbnoden begynte å sende denne data. Dette er en av hovedårsakene til at pakkene som ble sendt under simuleringen nådde frem innen tidsfrist. En annen viktig årsak til at tidsfrist ble nådd er at transittnodene kan sende data på alle linkene parallelt.

Parallell overføring og fragmenteringen er med på å bedre modellens spredningsmål. Dette kommer klart frem på grafene som tar for seg spredningsmålene for transittnettet. Ut ifra Figur 4.15 (*Siena Utvidet – Entropi over transittnett med liten belastning*) øker entropiverdien fra verdien 0 til 0,6 på 3 intervaller i løpet av kort tid ved begynnelsen. Vi vet på forhånd at det bare er en stor pakke som blir sendt under denne simuleringen. På grunn av at pakken er fragmentert belaster den tre forskjellige linker i løpet av kort tid, noe som øker nettets entropiverdi og gir det totale nettverket en jevnere belastning. Dette kommer også tydelig frem på Figur 4.14 (*Siena Utvidet – Belastning på hver av transittlinkene ved liten belastning*), som viser belastningen på hver av linkene.

En spesiell side ved den utvidede Siena er at den prøver å spare på nettverkets resurser til enhver tid. Dataen kommer ikke frem til mottaker før ved eksakt tidsfrist. Dette fører til at pakkene bruker den minst mulige båndbredde på hver av linkene. Figur 4.14 (*Siena Utvidet - Belastning på hver av transittlinkene ved liten belastning*) er et klart eksempel på dette. Her viser grafen at overføringshastigheten gradvis

varierer til enhver tid i stedet for å være enten 0 eller maksimum, som den ellers ville vært.

I tillegg til å spare på resursene tar Siena Utvidet hensyn til konkurrerende pakker på andre måter også. Det som blir sendt på hver av linkene avgjøres av hvem som har det mest travelt. Det blir jevnlig utregnet nødvendig båndbredde med hensyn på tidsfrist. Dette gjøres for hver av pakkene. Hvilke pakker som blir prioritert avgjøres av hvem som trenger størst båndbredde. Utregningen av båndbredden sier altså noe om pakkens prioritet, ønsket overføringshastighet og hvor god tid pakken har. Egenskapene for denne metoden kommer ikke godt nok frem i test resultatene, siden belastningen aldri blir stor nok under simuleringen. For å vise alle egenskapene ved denne metoden måtte for eksempel en stor pakke med kort tidsfrist bli sendt ut på et normalt belastet nettverk. For denne pakken ville det dermed bli utregnet en høy ”ønsket” båndbredde og den ville fått høyeste prioritert på alle linker helt til den kom frem. Overføringshastigheten til de andre pakkene kan i slike tilfeller stoppe helt opp, mens den nye pakken blir behandlet. Siden pakkenes nødvendige båndbredde må utregnes til enhver tid, vil pakkene som har hatt en pause eller lavere hastighet trenge høyere båndbredde etter denne hendelsen.

Når vi ser på Figur 4.12 (*Siena Utvidet - Transittnett med stor belastning*) ser det ut som nettverket får en sterkt stigende belastning når ny data blir sendt ut fra en stubbnode. Dette kan man se stemmer når man sammenlikner tidspunktet på grafens toppunkt med tidspunkt stubbnodene publiser data. Grunnen til at det danner seg toppunkt er fordi stubbnodene alltid sender data ut med maksimum hastighet. Slik er det valgt å gjøre fordi det aldri vil komme noe uforutsigbar trafikk som bør prioriteres på denne linken. Siden pakkene nesten ankommer transittnoden på samme tidspunkt vil et stort antall av dem belaste den første linken samtidig. Dette er altså grunnen til at belastningen stiger så mye. Kort tid senere synker belastningen gradvis. Dette er fordi pakkene som kom først får tildelt en høyere båndbredde enn de som ankommer senere. Dette er altså grunnen til at de markerte toppunkter oppstår. Denne belastningen unngår man ved å styre overføringshastigheten fra stubbnodene også, men dette er som sagt unødvendig. Metning av denne grunn er ikke noe å bekymre seg for, siden de siste pakkene vil ha en lav prioritet, og kan lett stoppe opp for andre med større behov.

5.3 Den epidemiske Siena

Vi valgte å utvikle en epidemisk versjon av Siena. Tanken bak denne var å forenkle Siena og i tillegg forbedre dens skalerbarhet. Det som gjør en epidemisk løsning skalerbar er dens enkle og desentraliserte oppbygning. Av den grunn flyttet vi mye av logikken i Siena sitt transittnett over til stubnodene. I tillegg reduserte vi antall meldinger som ble sendt over transittnettet vesentlig. På grunn av denne fordelingen av ansvar har vi gjort systemet mer skalerbart.

Måten den epidemiske Siena behandler store datapakker er mye likt den utvidede Siena. Pakkene blir også her fragmentert. Linker kan da belastes samtidig av fragmenter fra samme opprinelige pakke. Dette kommer klart frem i 4.18 (*Siena Epidemisk - Transittnett med liten belastning*). Ved tidspunkt 0 blir data sendt ut fra stubbnoden, og kort tid senere blir først en og så to transittlinker belastet. Figur 4.19 (*Siena Epidemisk – Transittlinkenes belastning (%) ved liten belastning*) viser også dette. Her kommer det tydeligere frem at det er to linker som blir belastet. Denne

figuren forteller også at maksimum båndbredde ikke blir brukt. Grunnen er at den regner ut den laveste overføringshastighet på hver av linkene på samme måte som Siena Utvidet gjør. Dette systemet finner også den korteste og mest lønnsomme stiene med hensyn på båndbredde.

Den epidemiske Siena sine transittnoder har også mulighet til å sende data ut på flere linker samtidig. Siden hver avsender behandler en interessent om gangen, kommer denne egenskapen bare til nytte hvis flere interesseholdere sender data gjennom samme transittnode samtidig. Det er vanskelig å bevise denne egenskapen ut i fra grafene.

En spesiell side ved de epidemiske Siena er dens utnyttelse av tilgjengelig tid før tidsfrist. Før data blir sendt ut på nettverket vet en interesseholder på forhånd hvor mange tidsintervaller som må til for at dataen skal nå frem til alle interessenter. Den kan dermed fordele tilgjengelig tid på antall intervaller slik at data kommer frem til alle. I tillegg til disse tidsintervallene blir dataen midlertidig lagret i transittnodene den passerer. Av den grunn oppstår ingen redundans. Siden antall avsendere av denne data fordobles for hvert tidsintervall, er tempingen med på å forhindre at en metning i nettet skal oppstå. Dette kommer frem i figur 4.20 (*Siena Epidemisk – Transittnett med stor belastning*). Grafen viser tre spisse søyler med korte tydelige trappetrinn. Grunnen til at disse har dukket opp er fordi pakkene har lang vei å gå i løpet av kort tid. Dette er grunnen til at en høy båndbredde er nødvendig. Antall trappetrinn indikerer hvor mange linker denne dataen belaster på samme tidspunkt. Dataen som blir sendt på hver av søylene blir altså lagret i transittnodene, og deres sjansen for belaste nettet senere reduseres. På denne måten utnytter Siena epidemisk tiden den har tilgjengelig, i tillegg til at en stor del av interessentene får dataen akkurat ved tidsfrist. Tidspunktet abonnentene får dataen kommer som sagt ikke frem på noen av grafene siden de bare tar for seg trafikken mellom transittnodene.

5.4 Sammenligning av løsningene

Ved sammenlikning av de tre metodene viser det seg at Siena Utvidet og Siena Epidemisk behandler store datapakker bedre enn den genuine Siena. Selv om den genuine Siena anvender maksimal hastighet over hver link, klarer den ikke å overføre data før de andre løsningene. Det er først og fremst to viktige årsaker til dette. For det første blir ikke fragmentering foretatt, og en transittnode kan dermed ikke videresende data før den har mottatt alt. Den andre grunnen er fordi nodene bare kan sende over en link om gangen, noe som fører til ekstra tid hvis en transittnode må sende over flere linker. Vi vil av den grunn bare sammenlikne de to nye løsningene videre.

En sterk side ved den epidemiske Siena er dens skalerbarhet. I forhold til Siena Utvidet er denne metoden mer desentralisert. Årsaken til dette er fordi den har mindre logikk i transittnodene, og funksjonaliteten er lagt mer over på stubbnodene. Dette medfører at færre meldinger er nødvendig for å opprettholde rutingen. I tillegg er antall meldinger rundt topologien og hvert av formålene som skal gjøres redusert.
Som vi vet oppstår det ingen redundans når data blir sendt med disse to løsningene. I tillegg vet vi at de finner den mest egnede sti på samme fremgangsmåte. Dette gjør at metodenes entropiverdi kan sammenliknes og fortelle hvem av dem som utnytter det underliggende nett jevnest over areal. Siden resultatene til den epidemiske Siena vil variere for hver gang vil også dens entropiverdi variere. Sammenlikning med en

verdi fra Siena epidemisk gir derfor ikke et klart svar på forholdet mellom dem, noe vi har tatt hensyn til. Vi gjorde derfor ti simuleringer av den epidemiske Siena. Den utvalgte simuleringen ga en verdi som lå midt i mellom den laveste og høyeste verdi. Av den grunn er ikke resultatet bare en tilfeldighet.

Under sammenlikningen av metodenes entropiverdi var det Siena Utvidet som kom best ut med en gjennomsnittsverdi lik 0,56, mens den epidemiske Siena fikk en verdi lik 0,35. Det kan også nevnes at ingen av de ti simuleringene av den epidemiske Siena klarte å få noen entropiverdi bedre enn Siena Utvidet. Årsaken til at Siena Utvidet kom bedre ut er fordi en utsendelse av data forgrener seg ut til alle interessentene over så lang tid som mulig. Hos den epidemiske Siena derimot starter utsendelsen med at en node får denne data. I neste intervall sender disse to til hver sine noder, og slik fortsetter det. Ettersom samme data sannsynligvis ikke vil passere samme link flere ganger på grunn av temping, reduserer dette antall transitt linker som blir belastet samtidig. Dette er grunnen til at den epidemiske Siena får en dårligere entropiverdi.

Grafene som tar for seg den total belastning per tid viser at Siena Utvidet belaster det totale nettverk jevnere per tid enn den epidemiske Siena. Årsaken er at den epidemiske Siena må sende dataen over kortere tidsintervaller, i stedet for å sende ut til alle samtidig slik den utvidede Siena gjør. Dette kan bli spesielt kritisk hvis antall interessenter er høyt, noe som gjør tidsintervallene korte, og dermed vanskeligere å rekke. I tillegg vil avstanden mellom interessentene og dataens størrelse være med på å øke problemet.

6 Konklusjon

Vår oppgave var å undersøke om Siena egnet seg til innholdsbasert distribusjon av store datafiler over WAN. Dette innbar blant annet at å se i hvilken grad man kunne overføre data innen gitte tidsfrister med minst mulig belastning på det underliggende nettverk. I tillegg skulle vi komme med en forbedring av Siena hvis det var nødvendig.

Under studiets periode satte vi oss dypt inn i distribusjonssystemet Siena sin oppbygning, og studerte hvordan den behandlet problemsområde til oppgaven. Vi registrerte dens svakheter og konkluderte at deler av Siena måtte forbedres. Vi kom frem til to nye løsninger; den utvidede Siena og den epidemiske Siena. Vi simulerte videre de to nye løsningene på samme måte, og drøftet deretter om deres resultater og oppbygning holdt til de satte kriterier.

Vi kom frem til at både den utvidede Siena og den epidemiske Siena holdt kravene som var satt under simulering, selv om deres oppførsel var forskjellig. Hovedforskjellen mellom dem var at den utvidede Siena belastet det underliggende nettverk jevnere enn den epidemiske Siena. Dette gjaldt spesielt hvis en pakke skulle nå frem til mange abonnenter innen en kort tid. En annen forskjell var at den epidemiske Siena var mer robust fordi mye av nettverkts ansvar var lagt over på hver av nodene. Forskjellen mellom dem var klart størst når det gjaldt hvem som belastet det underliggende nettverk jevnest. Vi kan derfor konkludere at den utvidede Siena vil egne seg best for den gitte problemstilling og dermed foreslå denne som en løsning. Det skal allikevel nevnes at tilfeller med særdeles mange tilkoplede noder kan kreve en annen løsning. Siden den utvidede Siena forlanger flere meldinger for å opprettholde systemet kan dette føre til et skalerbarhetsproblem. Vi vil i slike hendelser anbefale den epidemiske Siena, forutsatt at antall abonnenter og pakkenes tidsfrist er innenfor løsningens grenser.

7 Referanser

[1] Carzaniga, A.
"Architectures for an Event Notification Service Scalable to Wide-area Networks"
PhD Thesis. Politecnico di Milano. December, 1998.

[2] A. Carzaniga, A. L. Wolf.
"Content-Based Networking: A New Communication Infrastructure"
In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Lecture Notes in Computer Science n. 2538 p. 59-68, Springer-Verlag, Scottsdale, Arizona, October, 2001

[3] *J. Holliday, R. Steinke, D. Agrawal, A. E. Abbadi*
"Epidemic Algorithms for Replicated Databases"
IEEE transactions on knowledge and data engineering, Vol 15, No. 5, side1218-1238
published by the IEEE computer Society, 2003

[4] *R. Chandra, V. Ramasubramanian, K. Birman*
"Anonymous gossip: Improving multicast reliability in mobile ad-hoc networks"
In Proc. 21st Int. Conf. on Distributed Computing Systems, sider 275-283, 2001

[5] *J. Lou, P. Eugster, J. –P. Hubaux.*
"Route Driven Gossip: Probabilistic Reliable Multicast in Ad Hoc Networking"

[6] *P. T. Eugster, R. Guerraoui, A. –M. Kermarrec, L. Massoulié*
"Epidemic Information Dissemination in Distributed Systems"
Research feature, Vol. 37, No. 5, side 60-67
published by the IEEE Computer Society, 2004

[7] *Bokmålsordboka*
www.ordnett.no *(per 12.04.05)*

[8] *A. S. Tanenbaum, M. van Steen*
"Distributed Systems – principles and paradigms"
published by Prentice Hall, 2002

[9] Webopedia
www.webopedia.com *(per 11.04.05)*

[10] wi-lan
www.wi-lan.com/technology/glossary.html *(per 15.04.05)*

[11] T. Sheldon
"Encyclopedia of Networking & Telecommunications"
3rd edition 2001

[12] *J. Chen, P. Druschel, D. Subramanian*
"An Efficient Multipath Forwarding Method"
IEEE 17th Annual Joint Conference, side 1418-1425 vol. 3, 1998

[13] *F. Sørensen*
"Moderne IP-Nett"
Publisert av IDG Norge Books, 2004

[14] *L. L. Peterson, B. S. Davie*
"Computer Networks – A System Approach"
published by Morgan Kaufmann, 3rd edition, 2003

[15] Google
www.google.com – define:round-robin
definasjon hentet fra www.ioiscsi.com/library/terms/r.html *(per 25.04.2005)*

[16] A. Carzaniga, D. S. Rosenblum, A. L. Wolf
"**Design of a Scalable Event Notification Service: Interface and Architecture"**
Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, August, 1998

[17] *R. Chandra, V. Ramasubramanian, K. Birman*
"Anonymous gossip: Improving multicast reliability in mobile ad-hoc Networks"
In Proc. 21st Int. Conf. on Distributed Computing Systems, sider 275-283, 2001

[18] *P. Costa, M. Migliavacca, G. P. Picco, G. Cugola*
"Introducing Reliability in Content-Based Publish-Subscribe through Epidemic Algotithms"
Dip. di Elettronica e Informazione, Politecnico di Milano (År)

[19] P. Costa, M. Migliavacca, G. P. Picco, G. Cugola
"Indroducing Reliability in Content-Based Publish-Subscribe through Epidemic Algorithms"

[20] E. W. Zegura, K. L. Calvert, S. Bhattacharjee
"How to Model an Internetwork"
Proceedings of IEEE Infocom '96, San Francisco, CA

[21] Y. Yu, I. Cheng, A. Basu
"Optimal Adaptive Bandwidth Monitoring for QoS Based Retrieval"
IEEE Transactions on Multimedia, vol. 5, no.3, September 2003

[22] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, A. J. Demers
"Flexible Update Propagation for Weakly Consistent Replication"
Published by ACM 1997

[23] V. S. W. Eide, F. Eliassen, O. Lysne, O-C. Granmo
"Extending Content-based Publish/Subscribe Systems with Multicast Support"
Simula Research Laboratory, Technical Report 2003-03

[24] V. S. W. Eide, F. Eliassen, J. A. Michaelsen
"Exploiting Content-Based Networking for Fine Granularity Multi-Receiver Video Streaming"
Simula Research Laboratory, Oslo

[25] A. Carzaniga, M. J. Rutherford, A. L. Wolf
"A Routing Scheme for Content-Based Networking"
Technical Report CU-CS-953-03, Department of Computer Science, University of Colorado, June, 2003

[26] A. Carzaniga, A. L. Wolf
"Forwarding in a Content-Based Network"
Proceedings of ACM SIGCOMM 2003. p. 163-174. Karlsruhe, Germany. August, 2003

[27] Georgia Institute of Technology.
"Topology Models (GT-ITM)"
College of Computing. Georgia Tech Internet. http://www.cc.gatech.edu/projects/gtitm (per 23.02.05)

[28] University of Michigan.
"Internet Topology Generator"
Electrical Engineering and Computer Science. http://topology.eecs.umich.edu/inet/ (per 04.03.05)

[29] K. Calvert, M. Doar, E. W. Zegura.
"Modeling Internet Topology"
IEEE Communications Magazine, June 1997

[30] A. Carzaniga, A. L. Wolf
"A Benchmark Suite for Distributed Publish/Subscribe Systems"
Technical Report CU-CS-927-02, Department of Computer Science, University of Colorado, April, 2002

[31] C. E. Shannon
"A Mathematical Theory of Communication"
Published in 1948

[32] Microsoft
"Microsoft .Net Framework Referense for Random Class"

[33] A. Carzaniga, A. L. Wolf
"Fast Forwarding for Content-Based Networking"
Technical Report CU-CS-922-01, Department of Computer Science, University of Colorado, November, 2001. Revised, Semptember 2002

[34] Wikipedia
http://en.wikipedia.org
Hentet fra http://en.wikipedia.org/wiki/Publish/subscribe *(per 05.05.05)*

[35] C. Griwodz
"Wide-area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure"
Fachbereich Informatik, TU Darmstadt, Hochschulkennziffer D17, 2000

[36] A. Carzaniga, A. L. Wolf
"Content-Based Networking: A New Communication Infrastructure"
In NSF Workshop on an Infrastructure for Mobile and Wireless Systems. Lecture Notes in Computer Science n. 2538 p. 59-68. Springer-Verlag. Scottsdale, Arizona. October, 2001

[37] A. Carzaniga, A. L. Wolf
"Design and Evaluation of a Wide-Area Event Notification Service"
ACM Transactions on Computer Systems, vol. 19, no. 3, pages: 332-383, August 2001

[38] A. Carzaniga, D. S. Rosenblum, A. L. Wolf
"Interfaces and Algorithms for a Wide-Area Event Notification Service"
Technical Report CU-CS-888-99, Department of Computer Science, University of Colorado, October, 1999 (revised May 2000)

[39] K. K. Landes
"Scrutiny of the Abstract"
http://sep.stanford.edu/sep/prof/abscrut.html (per 09.03.05)

[41] J. Claerbout
"Scrutiny of the introduction"
http://sep.stanford.edu/sep/prof/Intro.html (per 09.03.05)

[42] J. H. Cowie, D. M. Nicol, A. T. Ogielski
"Modeling The Global Internet"
Published by IEEE, 1999

[43] Y. K. Dalal, R. M. Metcalfe
"Reverse Path Forwarding of Broadcast Packets"
Published by ACM, 1978

[44] M. Caporuscio, A. Carzaniga, A. Wolf
"An Experience in Evaluating Publish/Subscribe Services in a Wireless Network"
3rd International Workshop on Software and Performance. In conjunction with International Symposium on Software Testing and Analysis (ISSTA) Rome, July, 2002.

[45] M. Caporuscio, A. Carzaniga, A. Wolf
"Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications"
3rd IEEE Transactions on Software Engineering, 29(12):1059-1071, December 2003

[46] W. Willinger, V. Paxson
"Where Mathematics meets the Internet"
AT&T Labs-Research, 1998

[47] A. Carzaniga, D. S. Rosenblum, A. Wolf
"Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service"
19th ACM Symposium on Principles of Distributed Computing (PODC2000), Portland, Oregon. July, 2000

[48] A. Carzaniga, D. S. Rosenblum, A. Wolf
"Content-Based Addressing and Routing: A General Model and its Application"
Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, January, 2000

[49] A. Carzaniga, D. S. Rosenblum, A. Wolf
"Challenges for Distributed Event Services: Scalability vs. Expressiveness"
Engineering Distributed Objects (EDO '99), ICSE 99 Workshop, Los Angeles, California, May, 1999

[50] A. Carzaniga, E. D. Nitto, D. S. Rosenblum, A. Wolf
"Issues in Supporting Event-based Architectural Styles"
3rd International Software Architecture Workshop (ISAW3), Orlando, Florida, November, 1998

8 Vedlegg

Vedlegg A – Reserverte attributter
Vedlegg B – Demonstrasjon av nettverksgenerator
Vedlegg C – Demonstrasjon av simulator
Vedlegg D – Demonstrasjon av tidsfristbaserte formel for Siena

Vedlegg A - Reserverte attributter

Enkelte av attributtene i meldingstoppteksten vil være reservert av informasjonsutvekslingssystemet. Nedenfor er en liste over disse attributtene.

Kolonnen til venstre viser attributtene med en tilhørende beskrivelse i kolonnen til høyre for den. De to kolonnene til høyre forteller om attributtet er representert i Siena utvidet og Siena epidemisk.

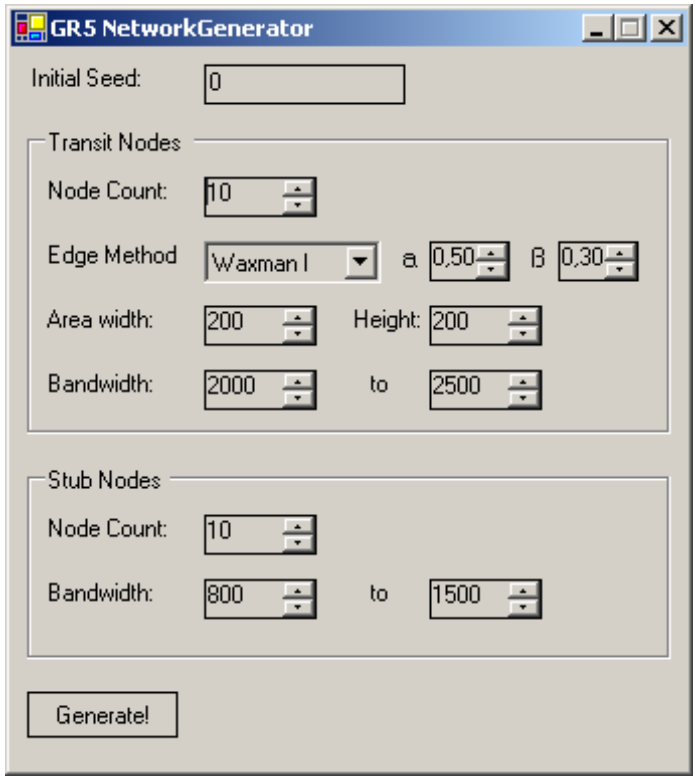
Tabell 8-1: Reserverte attributter

Attributt	Beskrivelse	Siena ut.	Siena epi.
fragident	Unik identifikator for fragment	○	○
fragnum	Fragmentummer	○	○
fragtotal	Totalt antall fragmenter	○	○
destadr	Destinasjonsadresse	○	○
nextdestadr	Neste hops destinasjonsadresse		○
destcount	Antall destinasjoner (abonnenter)		○
sequence	Den aktuelle runden		○
srcadr	Avsenders adresse. Brukes ved *Advertise*	○	○
deadline	Tidsfrist for når pakken må være fremme	○	○
starttime	Starttidspunktet for når pakken ble publishert		○

Vedlegg B - Demonstrasjon av nettverksgenerator

I dette vedlegget tar vi for oss hvordan nettverksgeneratoren virker.

Hovedvindu



Figur 8-1: Hovedvindu nettverksgenerator

Hovedvinduet til nettverksgeneratoren (se Figur 8-1) består av en rekke innstillinger som kan justeres for å få ønskede nettverk. Nedenfor er hver av innstillingene beskrevet.

Transit Nodes:

- Node Count – Antall transittnoder som nettverket skal bestå av.
- Edge Method – Her kan man velge hvilken metode som skal brukes for å regne ut kantene (linkene). Det er bare *Waxman I* som er tilgjengelig. Innstillingene for *α og β* kan bare velges mellom 0 og 1. En økning i *α* vil øke den generelle sannsynligheten for at linken skal bli opprettet, mens en økning i *β* reduserer sannsynligheten for opprettelse av linker hvor avstanden mellom nodene er stor
- Area Width og Height: Rektanglet som transittnodene er definert innenfor.
- Bandwidth: Bestemmer hvilken grense båndbredden skal være innenfor.

Stub Nodes:

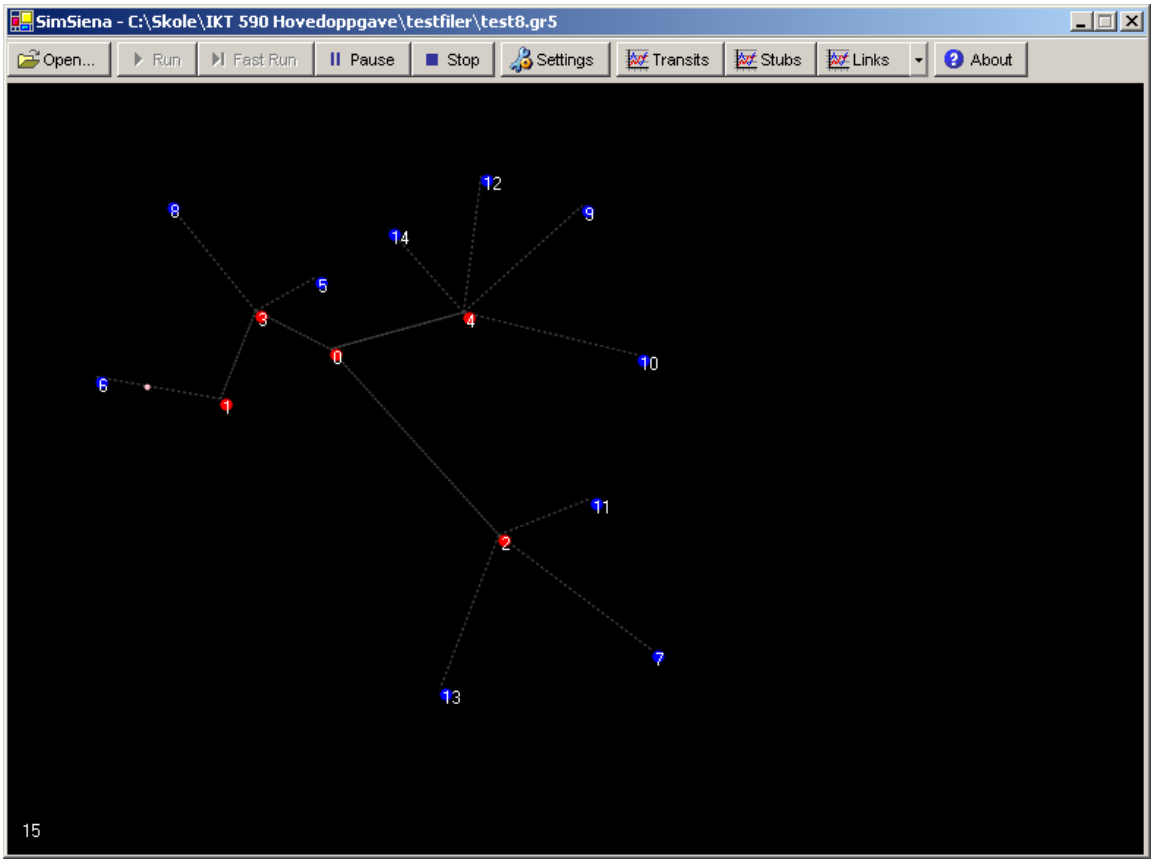
- Node Count: Antallet stubbnoder som nettverket skal bestå av.
- Bandwidth: Bestemmer hvilken grense båndbredden skal være innenfor.

Når attributtene er stilt inn kan man trykke *Generate!* for å få generert nettverket. Man vil da få frem et nytt skjermbilde som visuelt beskriver nettverket som er generert. Man kan deretter velge *Save* for å lagre nettverket på formatet Gr5.

Vedlegg C - Demonstrasjon av simulator

I dette vedlegger demonstrerer vi simulatorene. Siden våre tre simulatorer har store likhetstrekk har vi skrevet generelt om dem.

Hovedvindu



Figur 8-2: Hovedvindu simulator

Hovedvinduet til simulatoren (se figur 8-2) består i hovedsak av to elementer: en verktøylinje og en grafisk visualisering av nettverket.

Verktøylinjen består av følgende knapper:

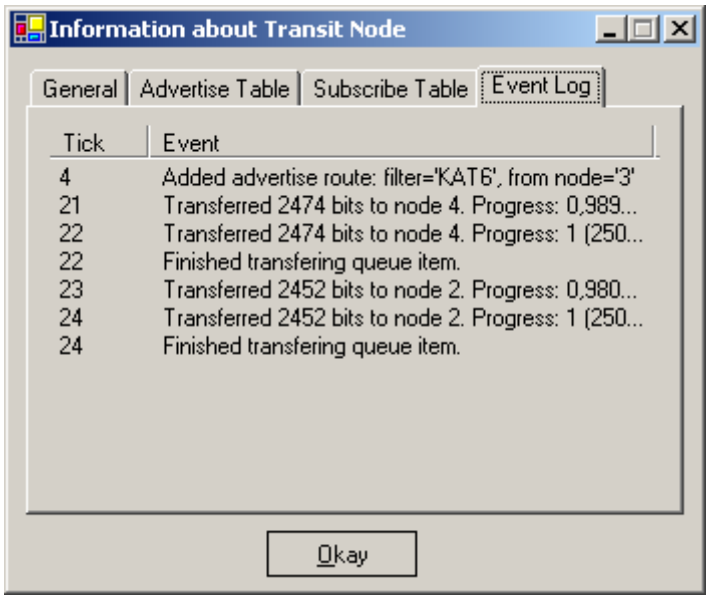
- Open – Åpner en fil som inneholder informasjon om toppologien til nettverket og hendelsene til tjenestemodellen.
- Run – Starter simulatoren i visualiseringsmodus.
- Fast Run – Kjører gjennom simuleringsmateriellet uten å vise det grafisk.
- Pause – Pauser simuleringen.
- Stop – Stopper simuleringen.
- Settings – Åpner skjermbilde for innstillinger.
- Transits – Åpner et skjermbilde hvor man kan se grafer over aktiviteten til transittnodene.
- Stubs – Åpner et skjermbilde hvor man kan se grafer over aktiviteten til stubbnoder.
- Åpner et skjermbidle hvor man kan se grafer over aktiviteten på linkene.
- About – Viser versjon og build-nummer.

Når simulatoren er i visualiseringsmodus kan man se hvordan pakkene blir sendt rundt i informasjonsutvekslingssystemet. Man kan når som helst ta simulatoren på pause for å studere det aktuelle øyeblikksbildet. Helt nederst til venstre i skjermbildet vises tidsforløpet.

Videre i dokumentasjonen vil noen av de mest interessante funksjonene til simulatorne bli eksponert.

Informasjon om nodene

Ved å klikke med musen på en node vil man få mer informasjon om den. Figur 8-3 viser hendelsesloggen for en node.



Figur 8-3: Informasjon om node

Figur 8-3 viser hendelsesloggen for en node. Det vil også være mulig å se hva som ligger i *Advertise* og *Subscribe*-tabellen for nodene, samt diverse informasjon under skillearket *General*.

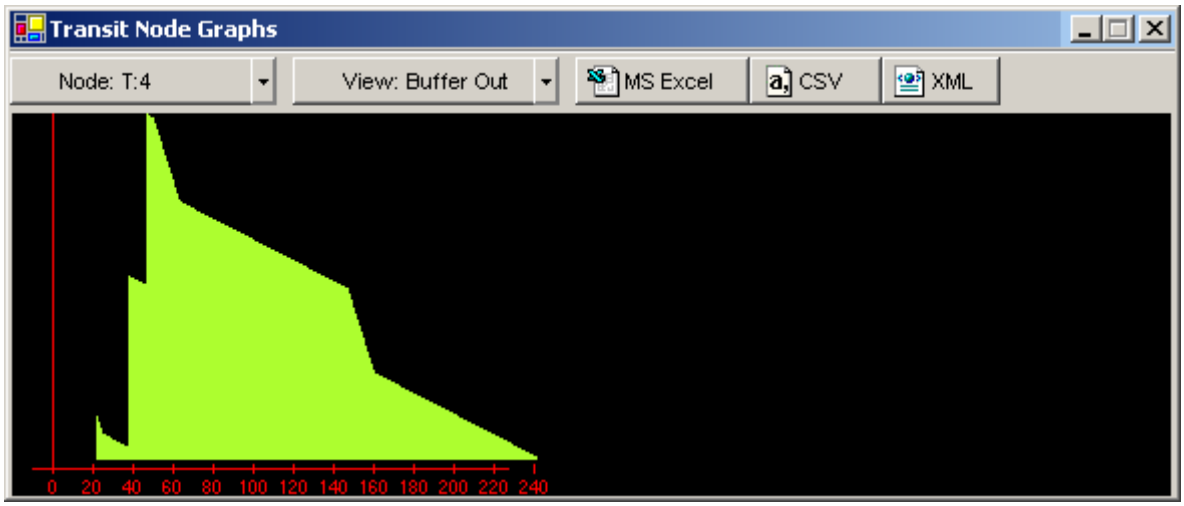
Settings

Ved å klikke på ikonet med tittel *Settings* fra hovedvinduet vil man komme til skjermbilde for innstillinger. Denne funksjonen vil være litt forskjellig avhengig av simulator.

Den mest nevneverdige innstillingen er *Speed*. Denne angir hvor raskt simulatoren skal gå.

Graffremvisning og eksport av data

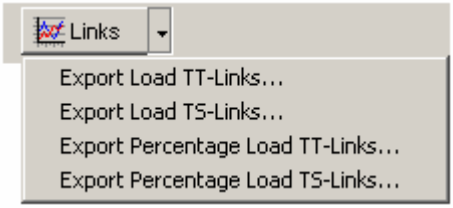
Man kan se grafer for transittnoder, stubbnoder og linker. Grafene kan man få frem ved å klikke på henholdsvis *Transits*, *Stubs* og *Links* knappene i verktøylinjen.



Figur 8-4: Graffremvisning av ut-buffer for en transittnode

Figur 8-4 viser skjermbilde av graffremvising for en transittnode. Det er mulig å se grafer for en rekke forskjellige data for hver av nodene. Dataen kan eksporteres til en XML (fork. eXtensible Markup Language), Microsoft Excel dokument og CSV (fork. Comma Separated Values).

Avhengig av simulator er det i tillegg mulig å eksportere andre typer data. Figur 8-5 viser eksporteringsmulighetene for linker i simulator for den genuine Siena.



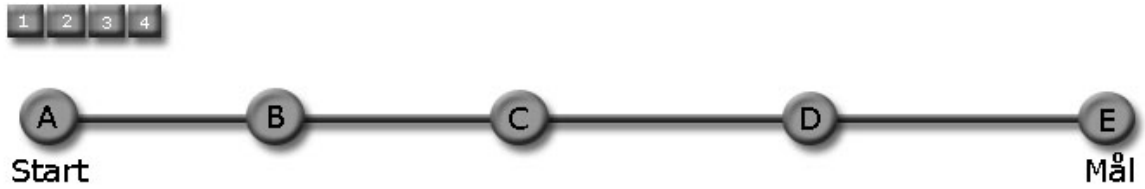
Figur 8-5: Eksporteringsmuligheter

Vedlegg D - Demonstrasjon av tidsfristbasert formel for Siena

I dette vedlegget illustrerer vi formelen for tidsfirstbasert distribusjon.

$$R(a,b) = \frac{L_{total} * (D + F) - L_{progress}}{T_d - T_p}$$

R – transmisjonsraten som pakken skal sendes over linken med.
S – Størrelsen på pakken som skal sendes.
T_d – Tidsfrist for når hele pakken skal være fremme hos mottaker.
T_p – Tidspunktet nå (time present), dvs når utregningen gjøres.
D – Antall linker fra der pakken befinner seg og frem til mottaker.
F – Antall resterende fragmentpakker bak pakken som skal sendes over linken.



S = 1, T_p = 0, T_d = 7 timer, dvs at hver pakke skal bruke 1 time per link

1.pakke – A til B:	1.pakke – B til C:	1.pakke – C til D:	1.pakke – D til E:
1/(7-0) * (4+3) = **1**	1/(7-1) * (3+3) = **1**	1/(7-2) * (2+3) = **1**	1/(7-3) * (1+3) = **1**
2.pakke – A til B:	2.pakke – B til C:	2.pakke – C til D:	2.pakke – D til E:
1/(7-1) * (4+2) = **1**	1/(7-2) * (3+2) = **1**	1/(7-3) * (2+2) = **1**	1/(7-4) * (1+2) = **1**
3.pakke – A til B:	3.pakke – B til C:	3.pakke – C til D:	3.pakke – D til E:
1/(7-2) * (4+1) = **1**	1/(7-3) * (3+1) = **1**	1/(7-4) * (2+1) = **1**	1/(7-5) * (1+1) = **1**
4.pakke – A til B:	4.pakke – B til C:	4.pakke – C til D:	4.pakke – D til E:
1/(7-3) * (4+0) = **1**	1/(7-4) * (3+0) = **1**	1/(7-5) * (2+0) = **1**	1/(7-6) * (1+0) = **1**

Ut i fra resultatene ser vi at transmisjonsraten blir jevnt fordelt over alle linkene.