**UNIVERSITETET I OSLO**
**Institutt for informatikk**

# Heterogene lenker og båndbredde-aggregering

Masteroppgave

Øystein Gyland

# Innhold

**Innholdsfortegnelse** **i**

**Figurer** **iii**

**Sammendrag** **v**

**Forord** **vi**

**1 Introduksjon** **1**

1.1 Motivasjon . . . . . 1

1.2 Problemstilling . . . . . 4

1.3 Oppgavens struktur . . . . . 4

**2 Bakgrunn og relatert arbeide** **5**

2.1 Bakgrunn . . . . . 6

2.1.1 Det fysiske laget og datalinklaget . . . . . 6

2.1.2 Nettverkslaget . . . . . 7

2.1.3 Transportlaget . . . . . 9

2.1.4 User Datagram Protocol . . . . . 9

2.1.5 Transmission Control Protocol . . . . . 10

2.2 Relatert arbeide . . . . . 16

2.2.1 Båndbreddeaggregering på datalinklaget . . . . . 16

2.2.2 Båndbreddeaggregering på nettverkslaget . . . . . 18

2.2.3 Båndbreddeaggregering på transportlaget og høyere 19

2.3 Sammendrag . . . . . 24

**3 Design** **25**
3.1 Eksisterende implementasjon . . . . . 25
3.1.1 Metningskontroll . . . . . 25
3.2 Våre endringer . . . . . 26
3.2.1 Oktetter . . . . . 27
3.2.2 Båndbreddeestimering . . . . . 30
3.2.3 Metningsvindu og ssthresh . . . . . 31
3.3 Tilstander i Westwood . . . . . 32
3.4 Vår tunnelimplementasjon . . . . . 33
3.4.1 NettverksTUNnel . . . . . 33
3.5 Sammendrag . . . . . 37

**4 Implementasjon** **39**
4.1 Implementasjonsspesifikke detaljer . . . . . 39
4.1.1 Tilstander i Westwood . . . . . 40
4.2 Sammendrag . . . . . 43

**5 Vurderinger** **45**
5.1 Vår Westwood- og Linux-implementasjon . . . . . 46
5.2 Westwood sammenlignet med CCID2 . . . . . 49
5.3 Sammendrag . . . . . 52

**6 Oppsummering og videre arbeid** **53**
6.1 Oppsummering . . . . . 53
6.2 Videre arbeid . . . . . 54
6.2.1 Dynamisk MTU . . . . . 54
6.2.2 IPv6-støtte . . . . . 54

**Appendiks** **55**

**Referanseliste** **57**

# Figurer


1.1 Typisk scenario for klient med flere nettverksforbindelser . . 2
1.2 Klient med flere nettverksforbindelser som bruker MULTI . . 3

2.1 De fire nederste lagene i OSI-modellen . . . . . 6
2.2 Virkemåte for Network Address Translation . . . . . 8
2.3 Virkemåte for AIMD . . . . . 12
2.4 Enkapsulering av pakker i IP-in-IP . . . . . 18

3.1 ACK-pakke etter modifisering . . . . . 27
3.2 Dataramme etter modifisering . . . . . 29
3.3 Vår tunnelimplementasjon . . . . . 34
3.4 Nettverksmessig oppsett av tunnelen . . . . . 35

4.1 Flytdiagram for vår Westwood-implementasjon . . . . . 41

5.1 Westwood på Linux vs egen Westwood-implementasjon. . . 47
5.2 Linux-impl. mot egen implementasjon, tap av konnektivitet 48
5.3 Linux-impl. mot egen implementasjon, pakketap 5% . . . . . 48
5.4 Linux-impl. mot egen implementasjon, pakketap 10% . . . . 49
5.5 Westwood og CCID2 10 Mbit strøm, 0% pakketap . . . . . 50
5.6 Westwood og CCID2 8.5 Mbit strøm, 5% pakketap . . . . . 50
5.7 Egen Westwood og CCID 8.5 Mbit strøm, 15% pakketap . . . 51

# Sammendrag

Bærbare datamaskiner, nettbrett, smarttelefoner o.l. har gradvis fått støtte for flere nettverkstilkoblinger med stor båndbredde. Bærbare maskiner har muligheter for tilkoblinger via ethernet og trådløse nett (WLAN). De mindre enhetene som smarttelefoner og lesebrett har støtte for trådløse nett og høyhastighetsmobiltelefoninett, som «High-Speed Downlink Packet Access» (HSPDA) og «Long Term Evolution» (LTE). I tråd med den teknologiske utviklingen, har leverandørene gitt oss overlappende dekning av disse nettverksteknologiene samtidig.

Denne oppgaven har som hovedfokus å undersøke hvordan man kan utnytte båndbredden, som kan muliggjøres ved å bruke flere av disse nettverksteknologiene samtidig. Selv om problemstillingene rundt redundante nettverkstilkoblinger er gamle, er det fortsatt vanskeligheter som må addresseres. Programmer, operativsystemer, rutere og andre nettverksenheter, støtter i liten grad fordeling av nettverkstrafikken mellom flere linker samtidig.

For å gjøre dette, så har vi utviklet et plattformuavhengig verktøy. Vi har gjennomgått hvordan vi kan håndtere problemstillinger som man må forvente å støte på som sluttbruker som f.eks: «Network address translation» (NAT) og ulik latens/båndbredde ved bruk av flere linker samtidig.

# Forord

Denne oppgaven er skrevet på Simula Research Laboratory på IT-Fornebu. Takk til guttene og jenten på labben for et kreativt, støttende og hyggelig arbeidsmiljø.

Takk til Kristian R. Evensen for hjelp gjennom hele oppgaven, med veiledning, inspirasjon og innspill.

Takk til min arbeidsgiver, Universitetets senter for informasjonsteknologi ved Universitetet i Oslo som gav meg studiepermisjon et semester slik at jeg kunne skrive denne masteroppgaven.
Sist men ikke minst, takk til familie, venner, og Therese.

Fornebu
2012-05-30
Øystein Gyland

# Kapittel 1

# Introduksjon

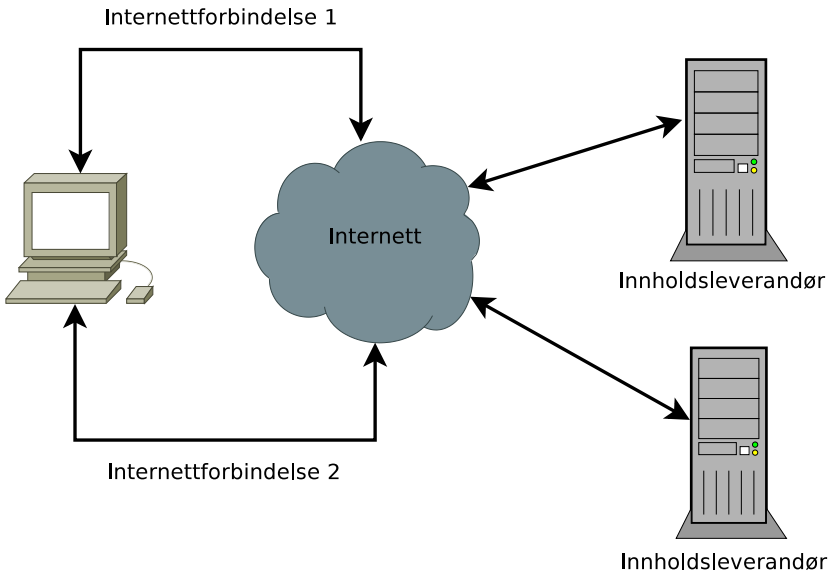
## 1.1 Motivasjon

Videostreaming, nettverkslagring og andre båndbreddekrevende tjenester er av de mer populære tjenestene på internett i dag. Den tilgjengelige båndbredden blant sluttbrukere øker, og dertil kommer det forventninger til bruksopplevelsen av disse båndbreddekrevende tjenestene.

De fleste bærbare enheter som har mulighet for å koble seg til nett, har også muligheter for å benytte seg av forskjellige typer nettverkstilkoblinger samtidig. Lesebrett og smarttelefoner har støtte for både trådløse nett (WLAN) og høyhastighets mobiltelefoninett. Siste utviklingen av mobiltelefoninett som «High-Speed Downlink Packet Access» (HSPDA) og «Long Term Evolution» (LTE) har et stort båndbreddepotensiale som kan utnyttes.

Bærbare datamaskiner har mulighet for å koble seg til via både ethernet[1] og WLAN. Økt utbygging av nettverksinfrastruktur, har gjort at både WLAN og høyhastighetsmobiltelefoninett i økende grad er tilgjengelig samtidig. Særlig veksten og den teknologiske utviklingen på smart-

[1] Med ethernet mener vi her de kablede variantene som er standardisert av IEEE under 802.3 paraplyen.



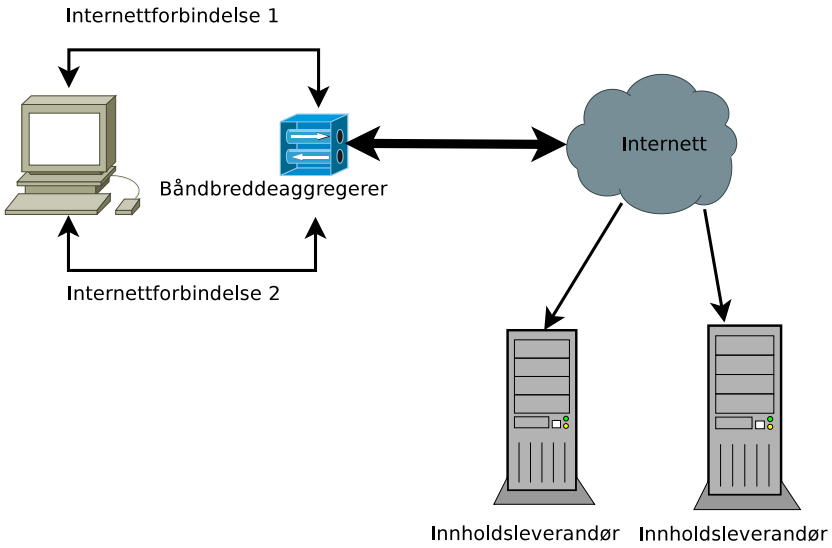

Figur 1.1: Typisk scenario for klient med flere nettverksforbindelser

telefoner, og senere lesebrett, har gjort at mobiltelefonileverandørene satser på høyhastighetsmobiltelefoninett. Trådløse nett og ethernet finner man hos bedrifter, hoteller, flyplasser, private hjem m.m.

I denne oppgaven har vi sett på hvordan man kan utnytte potensialet i den båndbredden som oppstår når man aggregerer flere nettverksforbindelser samtidig. Programvaren vi har utviklet, har hatt som designkriterie å fungere på tross av problemer og begrensninger, som man må forvente at sluttbrukere opplever.

Det var flere forskjellige utfordringer som måtte håndteres da vi utviklet båndbreddeaggregerings-programvaren i oppgaven. Det er få transportlagsprotokoller som ble utviklet med tanke på redundante forbindelser og aggregering av båndbredde. Vår implementasjon, hvor vi bruker nettverkstunnel for å fordele trafikken mellom flere nettverksforbindelser, måtte overkomme de problemene som oppstod som en konsekvens av dette designvalget. Tilsvarende gjelder løsningen vi implementerte for å få den mest hensiktsmessige pararelliseringen av nettverkstrafikk over nettverksforbindelsene som inngår i nettverkstunnelen.

Enheter som er tilkoblet flere nett har dertil flere unike nettverksidentifikatorer (i praksis, IPv4/IPv6-adresser). Det er flere forskjellige typer tilkoblinger, denne oppgaven omhandler enheter som i figur 1.1 som er tilkoblet flere typer nett samtidig, i motsetning til f.eks. rutere som også har multiple nettverkstilkoblinger.




Figur 1.2: Klient med flere nettverksforbindelser som bruker MULTI

Samtidig bruk av multiple nettverksforbindelser er en forutsetning for å benytte seg av båndbreddeaggregering. Programvaren vi har utviklet, har tatt høyde for at den skal kunne brukes uten å stille spesielle krav til typen nettverkstilkobling eller nettverksinfrastruktur. Dette innebærer f.eks. støtte for «Network Address Translation» (NAT), at vi ikke bruker eller introduserer nye nettverkslagsprotokoller, samt støtte for ulik latens og båndbredde mellom de forskjellige nettverkstilkobligene i bruk. Tanken bak valgene vi har tatt, har vært å evaluere mulighetene for å senere kunne videreutvikle programvaren til et sluttbrukervennlig produkt.

## 1.2 Problemstilling

Oppgaven er basert på Kristian R. Evensen sin doktorgradsavhandling fra 2012 [1]. Evensen har utviklet et program som aggregerer båndbredde over flere linker, og som derav implisitt støtter migrering mellom ulike IP-nett. Metningskontrollen «Datagram Congestion Control Protocol» (DCCP) «Congestion Control Identifier 2» (CCID2), har blitt implementert for UDP i programmet. Ettersom vi bruker UDP, så har vi måttet implementere metningskontrollen selv i applikasjonen, på TCP håndterer operativ-systemet metningskontrollen.

Applikasjonen som er oppdelt i tjener- og klientdel, utfører båndbredde-aggregeringen ved å slå sammen de involverte nettverksforbindelsene på klientsiden til et TUN-grensesnitt.

Hovedoppgaven vår har vært å vurdere en egenutviklet variant av TCP-metningskontrollen Westwood mot CCID2 på UDP.

## 1.3 Oppgavens struktur

- I andre kapittel har vi skrevet om bakgrunn og relatert arbeide.
- Tredje kapittel omhandler designet bak vårt arbeide.
- Fjerde kapittel omhandler implementasjonen vår.
- Femte kapittel er vurderingen av vår løsning.
- Sjette kapittel er konklusjon og fremtidig arbeide.

# Kapittel 2

# Bakgrunn og relatert arbeide

Båndbreddeaggregering er ikke et nytt forskningstema, og det eksisterer løsninger på de fleste nivåer i OSI-modellen.

De eksisterende løsningene har i liten grad tatt høyde for nettverksforbindelser med ulik båndbredde og latens, problemstillinger som er særegne for trådløse nett og støtte for tilkoblinger via tredjepartsnett hvor man er vanlig sluttbruker.

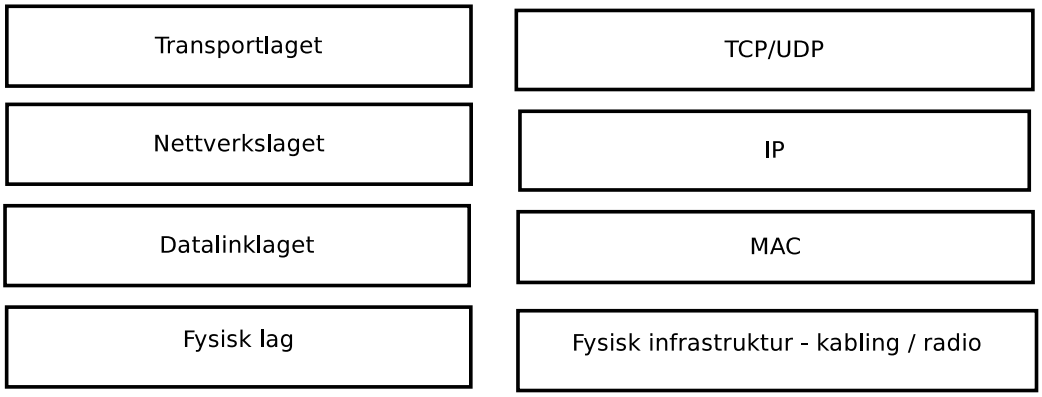
Løsninger som forutsetter nye eller modifiserte protokoller, har en stor jobb med å bli standardisert og få implementert standardene sine. Dette vil gjelde både hos standardiseringsorganene, som f.eks. «Internet Engineering Task Force» (IETF), og hos leverandørene av både nettverksutstyr og endeutstyret.

I denne oppgaven har vi fokusert på transparent båndbreddeaggregering, et område hvor det såvidt oss bekjent ikke foreligger noe som tilsvarer vår løsning fra før av. Det har foregått arbeid innenfor relaterte områder, som f.eks. pararellisering av nedlasting og andre typer applikasjonsspesifikke båndbreddeaggregeringer.

I dette kapittelet så har vi også gjennomgått de elementene som er viktige for å kunne forstå hva oppgaven dreier seg om.

## 2.1 Bakgrunn

De fire nederste lagene i OSI-modellen er illustrert i figur 2.1. I illustrasjonen er navnet på laget til venstre, og til høyre står det oppført hva man typisk forholder seg til på det aktuelle laget. Vi vil her gå igjennom de fire nederste lagene, med fokus på det som er viktig for videre forståelse av oppgaven.




Figur 2.1: De fire nederste lagene i OSI-modellen

### 2.1.1 Det fysiske laget og datalinklaget

Det er ingen mulighet for båndbreddeaggregering på det fysiske laget, derfor vil vi ikke gjennomgå dette nærmere. Det fysiske laget består av kabling og radiosignaler, samt andre former for trådløs bruk av elektromagnetisk stråling.

Datalinklaget overfører data mellom enhetene, og sørger for feilhåndtering og tilgang til det fysiske laget. I miljøer som støtter halv dupleks så håndterer datalinklaget kollisjoner på mediet. Ettersom kun en part kan sende hvis mediet bruker halv dupleks må man sjekke for kollisjoner. Dette skjer ved å sjekke om mediet er ledig før sending, slik at enheten kan sende data til en annen enhet, uten at de «snakker i munnen på hverandre».

Over de senere årene har ethernet-standardene fra «Institute of Electrical and Electronics Engineers» (IEEE) 802.3, og standardene for WLAN fra IEEE 802.11, blitt de dominerende teknologiene i bruk.

### 2.1.2 Nettverkslaget

Nettverkslaget håndterer adressering av trafikk, slik at pakkene kommer riktig frem, alternativt sier ifra at mottaker er utilgjengelig. På nettverkslaget er det «Internet Protocol» versjon 4 (IPv4), og i økende grad versjon 6 (IPv6), som blir brukt på nettverkslaget. «Alle» nettverksgrensesnitt har i praksis en IPv4- og/eller IPv6-adresse.

Det er også nettverkslaget som håndterer fragmentering og sammensetning av pakker ovenfor datalinklaget, som tradisjonelt har en maksimumsstørrelse (Maximum Transmission Unit(MTU)) på 1500 [1] oktetter pr. pakke.

#### Network Address Translation

Network Address Translation (NAT) som opprinnelig beskrevet i RFC1631 [2] fra 1994, var ment som en midlertidig løsning på problemet med begrenset tilgang på IPv4-adresser: [2]

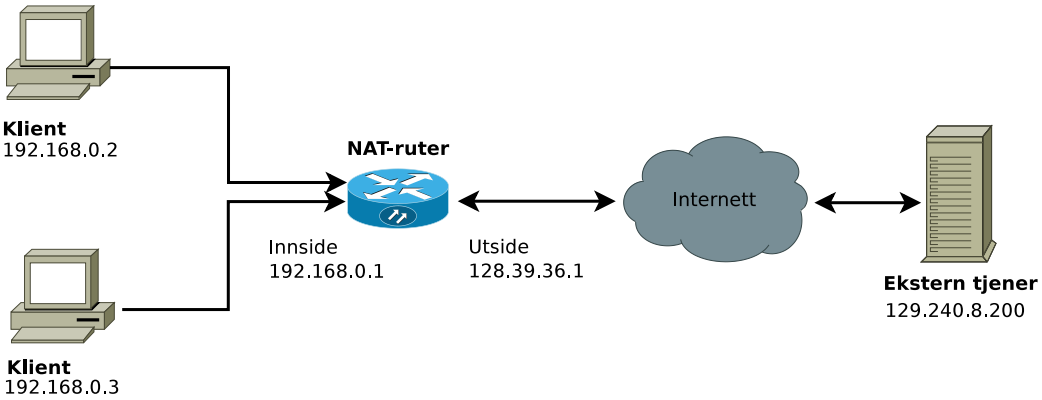
> This memo proposes another short-term solution, address reuse, that complements CIDR or even makes it unnecessary. The address reuse solution is to place Network Address Translators (NAT) at the borders of stub domains.

Det finnes flere typer NAT, men vi vil her gjennomgå den vanligste varianten som er utbredt for sluttbrukere.

---

[1] Sett fra IETF sin side, IEEE vil inkludere ethernet-rammen slik at det blir 1518 oktetter

[2] CIDR som beskrevet i RFC1517 er «Classless Inter-Domain Routing» som gjorde det mulig å rute nett av annen størrelse enn den historiske inndelingen i klasse A, B og C (/8, /16 og /24)




Figur 2.2: Virkemåte for Network Address Translation

NAT fungerer ved at en ruter manipulerer IP-pakkene, slik at hele nettverket på innsiden fremstår med én enkelt IP-adresse utad, som illustrert i figur 2.2. IP-adressene man bruker på innsiden av nettverket er fortrinnsvis RFC1918-adresser [3], som ikke ligger i den globale rutingtabellen.

For trafikk som utelukkende er initiert fra klientsiden, så er dette stort sett overkommelig. Trafikk som forutsetter at man kan opprette forbindelser fra utsiden av nettet vil NAT blokkere, da man ikke kan sende trafikk som er initiert fra utsiden av nettverket til en klient på innsiden. Ettersom ruteren som utfører NAT bruker portnumre for å sette opp en oversettingstabell, så er støtten for transportprotokoller utover TCP og UDP begrenset, fra RFC3022 [4]:

> Sessions other than TCP, UDP and ICMP query type are simply not permitted from local nodes, serviced by a NAPT router.

NAT er et fenomen man må regne med å møte på, og man må forvente at det blir enda mer utbredt. Etterhvert som tilgjengeligheten av ledige IPv4-adresser går mot null, så må man også regne med flere lag av NAT.

Ettersom vår programvare må ta høyde for bruk av NAT, så har det innsnevret mulighetene våre noe, særlig med tanke på at transportlags-

| Klient-IP | Mottaker-IP | Klientport | Mottakerport | NAT-port |
|---|---|---|---|---|
| 192.168.0.2 | 129.240.8.200 | 1025 | 80 | 2025 |
| 192.168.0.3 | 129.240.8.200 | 1025 | 80 | 2026 |

Tabell 2.1: Manipulering av IP-pakker med NAT

protokollene vi kan bruke er begrenset til TCP og UDP.

Tabellen viser hvordan ruteren som utfører NAT manipulerer IP-pakkene mellom innsiden og utsiden av nettet. I dette eksempelet så etablerer begge klientene en HTTP-sesjon til 129.240.8.200. Klientene setter selv source port til 1025 og 1026, disse portene blir manipulert av ruteren som endrer de til 2025 og 2026. Det motsatt skjer når det kommer svar fra 129.240.8.200, nå blir portene manipulert fra 2025 og 2026 til 1025 og 1026.

### 2.1.3 Transportlaget

Transportlaget sørger i henhold til RFC1122 [5] for ende-til-ende kommunikasjon for applikasjoner på høyere nivå. Kommunikasjonen foregår ved at programmet oppretter en socket, hvor de viktigste argumentene er; port, IP-adresse og type transportprotokoll. Programmet kan kommunisere med en annen node via denne socketen, uten å forholde seg til; adressering, fragmentering o.l., som håndteres på de lavere lagene.

### 2.1.4 User Datagram Protocol

«User datagram protocol» (UDP) ble først standardisert i 1980 med RFC768 [6]. Sammen med TCP utgjør UDP de to definitivt mest utbredte transportprotokollene. UDP skiller seg fra TCP på endel områder:

- UDP setter ikke opp en forbindelse før trafikken overføres.
- UDP har ingen funksjonalitet for å bekrefte av pakker kommer frem.
- UDP gir ingen garanti for at pakkene kommer frem i riktig rekkefølge.

- UDP har ingen form for metningskontroll.

Disse egenskapene gjør UDP velegnet til endel scenarier, tradisjonelt tidskritiske formål, som f.eks. IP-telefoni, videostreaming og spill, samt andre formål hvor pakketap kan aksepteres på bekostning av tidsforsinkelsen det vil medføre å retransmitere pakken. At UDP ikke har noen ACK-funksjonalitet gjør den velegnet til multicast-trafikk, og andre områder hvor sender ikke støtter retransmitering.

### 2.1.5 Transmission Control Protocol

Transmission Control Protocol (TCP) som først beskrevet i RFC793 [7], som stammer fra 1981. Vi har ikke brukt TCP i oppgaven vår, men vi har lånt flere ideer fra TCP. Følgende egenskaper kjennetegner denne protokollen:

- TCP etablerer en forbindelse før data kan overføres.
- TCP har funksjonalitet for å bekrefte av pakker kommer frem (ACK).
- TCP sørger for at pakkene ankommer og blir levert i riktig rekkefølge til applikasjonen.
- TCP støtter metningsprotokoll på transportlaget slik at nettverket ikke blir overbelastet.
- TCP støtter flytkontroll slik at mottaker ikke blir overlesset med trafikk.
- TCP er strøm-orientert, TCP deler selv opp data fra høyere lag opp i segmenter før de sendes til nettverkslaget.

Disse egenskapene har gjort TCP til den mest utbredte transportlagsprotokollen. TCP sin innebygde garanti for at pakker ikke er korrupte, metningskontrollen, at pakker ankommer og blir levert i riktig rekkefølge, gjør TCP til et populært valg som protokoll til filoverføring. Dette

inkluderer f.eks. «file transfer protocol» (FTP), «secure copy» (SCP), «rsync», «network file system» (NFS) og filoverføring i Windowsnettverk.

Andre formål som TCP brukes til er WWW/HTTP(S), peer2peer-protokoller (som f.eks. «BitTorrent»), epostrelaterte protokoller som SMTP, POP3, IMAP. Pakkeleveransen er viktig for disse protokollene, og metningskontroll er viktig for å utnytte båndbredden med p2p-protokoller.

### Metningskontroll i TCP

Formålet med metningskontrollen i TCP er todelt; nettverket skal ikke overbelastes, og fordelingen av båndbredderessursen til de forskjellige strømmene skal være jevn. Milepælen innenfor metningskontroll i TCP innførte Van Jacobson [8] i 1988, her ble de viktige elementene «slow start» og «congestion avoidance» innført.

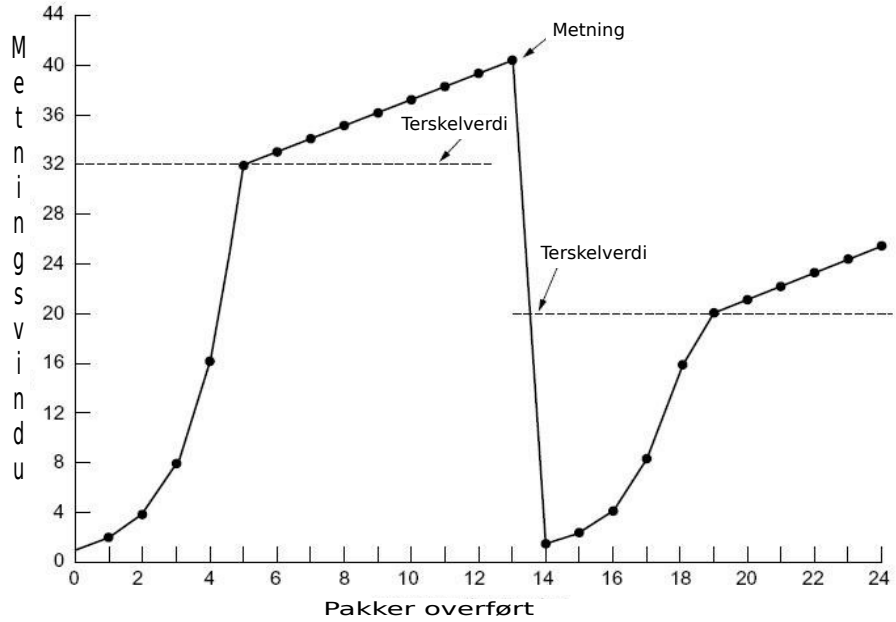
«Slow start» er ref. Welzl «Network Congestion Control» [9] den initielle fasen TCP inntar. I fasen «slow start» øker metningsvinduet eksponentielt, fra 1 inntil metningsvinduet når terskelverdien «ssthresh». Deretter øker metningsvinduet etter formelen: [3]

$$metningsvindu = metningsvindu + (\text{MSS} * \text{MSS}/metningsvindu)$$

Denne fasen som inntreffer når metningsvinduet har nådd «ssthresh» kalles «Congestion avoidance». TCP merker at nettverket er mettet ved at ACK-pakkene signaliserer omstokking, huller i sekvensen eller mangel på ACK-pakker. Når dette skjer vil «ssthresh» settes til halvparten av metningsvinduet og forbindelsen inntar «slow start»-fasen igjen.

Virkemåten er illustrert i figur 2.3, hvor man kan se hvordan

[3] MSS er «Maximum Segment Size»




Figur 2.3: Virkemåte for AIMD

metningsvinduet øker før og etter terskelverdien «ssthresh» er nådd.

Denne funksjonaliteten med lineær vekst av metningsvinduet og halvering av «ssthresh» etter metning kalles «additive increase/multiplicative decrease» (AIMD). Denne funksjonaliteten går igjen i mange metningskontroller.

### Kalkulasjon av tidsavbrudd for retransmitering og rundreisetid

«Retransmission timeout» (RTO) er et viktig element til metningskontrollen for å detektere pakketap. Når en pakke blir sendt vil sender beregne hvor lenge man skal vente på en ACK-pakke (ACK-pakken trenger ikke å være for det segmentet). Hvis det ikke ankommer noen ACK-pakke innenfor tidsintervallet antar man at pakken er tapt.

Rundreisetid («Round trip time» - RTT) som vi bruker til beregningene

i Westwood, er tiden det tar fra en pakke blir sendt til vi får en ACK-pakke til svar.

Formlene for å regne ut både RTO og RTT, ble opprinnelig skissert i RFC793 [7], sist oppdatert med RFC6298 [10]. Vi har ikke implementert støtte for retransmisjon i vår Westwood-implementasjon, men vi har implementert funksjonaliteten til RTO for å detektere pakketap via tidsavbrudd.

### New Reno

Westwood baserer seg på den eldre metningskontrollen «New Reno», hvor oppførselen på sendersiden er modifisert.

«New Reno» ble definert i 1999 med RFC2582 [11] [4]. Denne metningskontrollen ble utviklet for å forbedre «fast retransmit» og «fast recovery», ved å handle basert på ACK-pakker som signaliserer at mottaker ikke har mottatt den forventede sekvensen med data.

«Fast retransmit» er ref RFC2001 [12] en tilstand i metningskontrollen hvor sender først venter for å avklare om DUPACK-pakkene signaliserer pakketap, eller omstokking av segmenter. Gitt tre DUPACK-pakker på rad, så vil det tapte segmentet retransmiteres uten å vente på tidsavbrudd.

«Fast recovery» er tilstanden som inntreffer etter «fast retransmit». Her vil ikke senderen innta tilstanden «slow start», da det er utestående segmenter i nettverket. Hadde sender inntatt tilstanden «slow start» istedenfor så ville det ha redusert flyten ytterligere ettersom metningsvinduet settes til 1 i «slow start».

Ref. Welzl: «Network Congestion Control» [9, s. 79] så er endringene i New Reno i forhold til RFC2581 [13] følgende:

- Høyeste overførte segmentnummer blir lagret, for å skille mellom ACK som verifiserer overføring av hele sekvenser, og ACK-pakker

[4] RFC2582 ble senere erstattet av RFC3782

som verifiserer overføring av ukomplette sekvenser.

- Gitt at ACK-pakkene signaliserer at hele sekvensen er overført uten tap, så kan metningsvinduet settes til «ssthresh». Ettersom dette innebærer at det kan være mindre data underveis enn størrelsen på metningsvinduet, så må tjenersiden unngå å sende byger med data. Alternativt så kan sender sette metningsvinduet til verdien av «ssthresh» pluss data underveis, pluss «maximum segment size». Uansett hvilken fremgangsmåte man velger så, vil vindusstørrelsen minskes, og tjenersiden vil avslutte «fast recovery».

- Gitt at ACK-pakkene signaliserer hull i den overførte sekvensen, vil det første manglende segmentet bli overført. Samtidig vil metningsvinduet minskes til mengden data bekreftet overført, pluss et segment. Et segment vil sendes hvis metningsvinduet er stort nok til å tillate dette. Målet med denne prosedyren er å forsikre seg om at mengden data underveis er omtrent lik verdien til «ssthresh», når tilstanden «fast recovery» blir avsluttet. Tjenersiden forblir så i «fast recovery»-tilstanden.

#### Westwood og Westwood+

Westwood er en modifisert utgave av «New Reno», hvor kun oppførselen til sender har blitt endret. Disse modifiseringene, skal håndtere pakketap forårsaket av overføringsfeil bedre enn i «New Reno».

Westwood ble utviklet av Saverio Mascolo, Mario Gerla, Claudio Casetti og Medy Senadidi [14]. I 1999 ble en implementasjon av Westwood til nettverkssimulatoren «ns2» publisert.

Mascolo modifiserte senere Westwood, denne videreutviklingen ble hetende Westwood+. Det er Westwood+ som har blitt utbredt, den har bl.a. blitt implementert i Linux.

Denne navngivingen kan være forvirrende. F.eks. heter Westwood+ kjernemodulen til Linux «tcp_westwood», selvom det altså er snakk om

Westwood+. I denne oppgaven, som i annen litteratur så har Westwood+ blitt referert til som Westwood.

Hverken Westwood eller Westwood+ er definert i noen RFC, implementasjonene er de-facto løsninger basert på artikler, hvor forfatteren oppgir [14–16] som hovedreferanser.

## 2.2 Relatert arbeide

Båndbreddeaggregering går ut på å slå sammen båndbredden til flere nettverkslinker til en logisk link. Dette foregår ved å pararellisere nettverkstrafikken over linkene som inngår i den logiske linken. På denne måten blir den tilgjengelige båndbredden høyere enn hva en enkelt fysisk link kan tilby. Dette kan foregå på flere av lagene i OSI-modellen, vi vil her presentere eksisterende løsninger på de forskjellige lagene.

### 2.2.1 Båndbreddeaggregering på datalinklaget

Det finnes mange protokoller for båndbreddeaggregering og feiltoleranse på datalinklaget:

- Link Aggregation, standardisert av IEEE som 802.1AX. [5]
- EtherChannel er en propertiær Cisco-løsning som var grunnlaget for IEEE 802.3ad.
- Propertiære derivater av 802.1AX / 802.3ad fra andre leverandører.
- Propertiære løsninger for trådløse nett.

For trådløse nett finnes det flere propertiære løsninger, som f.eks. Atheros «Super G» og Broadcom «125 High Speed Mode» [17].

Linkaggregering med IEEE 802.1AX og derivater av denne, forutsetter støtte i nærmeste enhet (svitsjen), samt støtte i operativsystemet/nettverkskortdriveren. Det forutsetter også at man har samme hastighet og dupleksinnstillinger på alle nettverkslinkene i aggregeringsgruppen. Antallet linker som kan inngå i en aggregeringsgruppe, er gjerne begrenset fra to til åtte. To, fire og åtte er det antallet nettverkslinker som gir

[5] 802.1X var tidligere standardisert som 802.3ad

en hensiktsmessig utnyttelse av båndbredden ved gitte lastbalanseringsalgoritmer [18].

Aggregering på datalinklaget gir båndbreddeaggregering basert på en rekke forskjellige kriterier, f.eks. lastbalansering basert på IP- og MAC-adresse til sender og mottaker. Men det gir ingen båndbreddeaggregering på en enkelt strøm mellom to IP-adresser. Følgelig vil linkaggregering på en tjener med 2*100Mbit ethernetkort ha kapasitet til å sende 200 Mbit med nettverkstrafikk, men øvre båndbredde pr. strøm er fremdeles begrenset til hastigheten pr. ethernetkort.

Båndbreddeaggereging på det trådløse datalinklaget er rene propertiære løsninger, som bl.a. forutsetter at alle parter har identisk maskinvare. Ettersom frekvensbåndet til WLAN er begrenset, så vil aggressiv bruk av radiofrekvenser kunne medføre negative konsekvenser for trådløs bruk av andre innenfor samme område.

Det er også flere propertiære alternativer til IEEE 802.1AX / 802.3ad. Avaya (tidligere Nortel), Huawei og ZTE tilbyr propertiære løsninger som man må anta at kun fungerer mellom leverandørens egne produkter.

### Multi-Link Point-to-Point protocol

Multi-Link Point-to-Point protocol (MLPPP), standardisert i RFC1990 [19], støtter båndbreddeaggregering over flere linker. Dette skjer ved å splitte pakkene opp i mindre segmenter som fordeles over de forskjellige linkene, og så sette sammen pakkene igjen ved mottak.

Denne løsningen egner seg dårlig for heterogene nettverkslinker grunnet pakkefragmenteringsalgoritmen. Algoritmen utnytter ikke potensialet til den raskeste linken når lenkene har ulik hastighet og latens. PPP er en egen datalinkslagprotokoll, så bruken av protokollen forutsetter at nettverksutstyret mellom sender og mottaker er satt opp til å støtte denne typen trafikk. Dette er urealistisk å få til som sluttbruker, alternativet er at MLPPP tunneleres over en høyere lags protokoll.

**Oppsummering av løsninger for aggregering på datalinklaget**

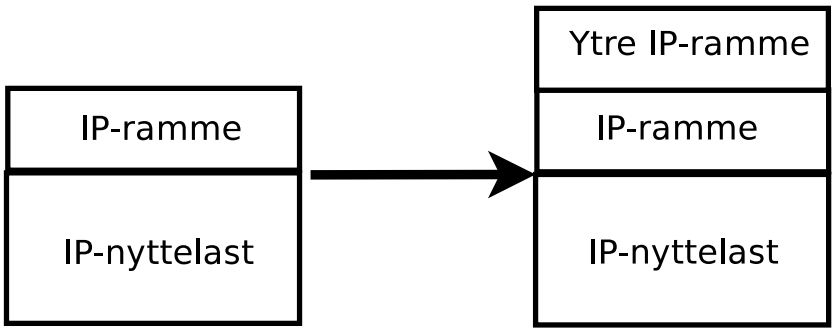
Aggregering på datalinklaget er uegnet for denne typen båndbreddeaggregering som er implementeret i denne oppgaven. Enten så setter løsningene krav til at man har adgang til å konfigurere nettverksinfrastrukturen til nettleverandøren (noe som er et urealistisk krav som vanlig sluttbruker). Eller så har løsningene spesifikke krav til hvor og hvordan nettverkstilkoblingen kan foretas.

### 2.2.2 Båndbreddeaggregering på nettverkslaget

Fordelen med å implementere båndbreddeaggregering på dette laget, er at det i teorien kan skje sømløst ovenfor eksisterende nettverksinfrastruktur, og uten endringer i IPv4/IPv6-protokollene.

Tunnelering på nettverkslaget skal i utgangspunktet være kurant, da den mellomliggende nettverksinfrastrukturen kun forholder seg til IP-rammen, for å adressere trafikken til mottaker.

**IP-in-IP**




Figur 2.4: Enkapsulering av pakker i IP-in-IP

Tunnelering for å oppnå båndbreddeaggregering via IP-in-IP har blitt forsøkt, f.eks. i artikkelen [20]. IP-in-IP som beskrevet i RFC2003 [21], fungerer ved at en IP-pakke blir enkapsuleret i en annen IP-pakke. Den store ulempen med IP-in-IP som tunneleringsmekanisme, er at i nettverk

hvor det brukes Network Address Translation (NAT), vil man ikke ha noen ende-til-ende konnektivitet. Ettersom IP-in-IP ikke bruker porter som TCP og UDP, vil følgelig ikke ruteren som utfører NAT ha noen mulighet til å rute IP-in-IP.

#### Equal-Cost Multi Path

Equal-Cost Multi Path (ECMP) som beskrevet i RFC2992 [22], fungerer ved at en node med flere nettverksforbindelser fordeler trafikken ved å vekte de forskjellige forbindelsene. Dette fungerer bra i et statisk scenario, men det er ikke spesielt velegnet med et dynamisk scenario, ref. [1]. ECMP fungerer bra for programmer som oppretter et stort antall unike nettverksstrømmer, som f.eks. «peer2peer» (P2P) trafikk. ECMP gir ingen båndbreddeaggregering for en sesjon, så trafikk mellom en tjener og node har ikke noe utbytte av ECMP. En stor fordel med ECMP, er at den ikke krever noen endringer, utover å endre rutingtabellen til noden.

### 2.2.3 Båndbreddeaggregering på transportlaget og høyere

Båndbreddeaggregering på dette laget har stort sett dreid seg om å modifisere «stream control transmission protocol» (SCTP) og «transmission control protocol» (TCP), ref. [1]. Disse modifiseringene har medført at man har måttet endre operativsystemet på både klient og tjener. Ingen av disse løsningene kan brukes direkte i vår oppgave, men de har løsninger som vi kan trekke på.

#### Stream Control Transmission Protocol

Stream Control Transmission Protocols (SCTP) som beskrevet i RFC4960 [23], og senere RFC6096 [24], er en egen transportlagsprotokoll som har lånt noen ideer fra metningskontrollen til TCP. SCTP støtter også multiple nettverksforbindelser, en forbindelse kan bruke flere veier via forskjellige forbindelser. Men, i utgangspunktet brukes kun de redundante veiene for

feiltoleranse, ikke for båndbreddeaggregering. Båndbreddeaggregering i SCTP er kjent som «concurrent multipath transfer» (CMT), implementasjonen ble introdusert i artikkelen [25]. Båndbreddeaggregeringsfunksjonaliteten som har blitt implementert i denne oppgaven har lånt noen ideer herfra, som et metningsvindu og buffer pr. link.

Som med IP-in-IP og andre transportprotokoller utover TCP og UDP, må man anta at SCTP ikke fungerer overhodet i nettverk hvor det brukes NAT. «Stream control transmission protocol network address translation» [26] prøver å løse dette problemet, men det er problemstillinger som ikke har noen åpenbar løsning enda.

### Transmission Control Protocol

Designvalget som medfører at kommunikasjon over TCP skulle skje mellom to noder med hver sin IP-adresse, har medført at man ikke kan ta ibruk båndbreddeaggregering uten fundamentale endringer i protokollen.

Eksempler på slike endringer er: mTCP [27], pTCP [28] og Multipath TCP (MPTCP) [29, 30]. mTCP er en forbedret utgave av pTCP. Felles for de begge er gode resultater fra simulering, men ingen av de har bevist at de har noen god løsning for hvordan omstokking av pakker skal håndteres [1].

MPTCP er et aktivt prosjekt under IETF sine vinger, det slekter på SCTP CMT. MPTCP fordeler datastrømmen over flere TCP-strømmer, vektet etter metningsvinduet til hver strøm. MPTCP er det prosjektet som har størst fremdrift, men det er uløste problemer rundt valget av metningskontroll som må løses før man kan lansere en produksjonsklar løsning.

PRISM [31] har også implementert båndbreddeaggregering på transportlaget, ved å modifisere TCP på sendersiden, og å sette opp en nettverkslags-proxy. Løsningen er tiltenkt grupper av mobile enheter i umiddelbar nærhet av hverandre, hvor enhetene kan slå sammen nettverksressursene sine. I motsetning til de fleste andre implementasjonene, som er nevnt, så bruker PRISM den raskeste nettverksforbindelsen (typisk

den trådløse) til å dele data med andre noder i umiddelbar nærhet. Den tregeste forbindelsen (til mobilnettet) brukes for å laste ned data fra Internett. Tunnelering finner sted på nettverkslaget med IP-tunneller, og man har lånt ideer fra metningskontrollen til TCP for å unngå metning.

Løsningen med å etablere nettverksforbindelser til andre noder via WLAN gjør at PRISM forutsetter at de andre nodene er i umiddelbar nærhet. Dette medfører at PRISM blir en nisjeløsning for spesielle situasjoner, det bærer mer preg av å være et ad-hoc-nettverk enn en løsning for båndbreddeaggregering.

### TCP som tunneleringsprotokoll

TCP som tunneleringsprotokoll er utbredt på sesjonslaget, som f.eks. som SOCKS-proxy for HTTP-trafikk. TCP-tuneller kan brukes som generell tunneleringsmekanisme for enkeltporter, ved bruk av SSH-tunneller. Men for tunnelering på nettverkslaget, er det mindre utbredt da TCP-pakkene har større rammer enn andre protokoller (spesielt UDP). Bruk av TCP innebærer at man får retransmitering og flytkontroll av pakker på to lag hvis man tunnelerer TCP-på-TCP, eller en annen protokoll som har innebygget støtte for retransmitering og flytkontroll. Til virtuelle private nettverk (VPN) så er ikke retransmitering og flytkontroll på transportlaget enkelt å forholde seg til når det kommer kryptografi inn i bildet, så UDP er mer utbredt i VPN.

### UDP som tunneleringsprotokoll

UDP er utbredt i VPN og andre former for tunneleringsprogramvare, særlig i de tilfellene hvor klienten er bak et nett som bruker NAT. UDP brukes f.eks. i NAT-T som er IPsec tunnelert over UDP, som beskrevet i RFC3948 [32]. Fordelen med UDP i VPN er problemstillinger som oppstår når kryptografi skal innblandes, f.eks. vil omstokking av pakker og levering av duplikate pakker forårsake problemer. Det kan derfor være enklere å bruke UDP hvor VPN-programvaren selv foretar metningskontroll på

applikasjonslaget.

### Andre/nye protokoller

Gjennomgående problemer med bruk av transportprotokoller utover de veletablerte, er standardisering og støtte i nettverksinfrastruktur.

For standardiseringen sin del er det begrenset med felter ledig i IP-protokollfeltet, på IPv4 er dette feltet en oktett som setter en øvre begrensning på antallet IP-protokoller til 256. Gitt hvor begrenset denne ressursen er, kan man anta at det er problematisk å få «The Internet Assigned Numbers» Authority (IANA) til å dele ut av den.

Når det gjelder nettverksinfrastruktur må man påbegregne å støte på brannmurer, rutere med aksesslister og andre enheter som proxy-tjenere som potensielt kan droppe ukjente typer trafikk. Selvom protokollen man bruker er standardisert, må man også ta høyde for utstyr som enten ikke er oppdatert, er malkonfigurert eller mangler støtte for protokollen.

Når det gjelder støtte for nye protokoller, er ruterindustrien ofte konservative, og dertil er støtten for protokoller ofte begrenset. Vi har testet protokollstøtten på en Cisco-ruter som kjører IOS12.4 med den mest innholdsrike programvarelisensen. Selv med denne nye rutingprogramvaren, er protokollstøtten til aksesslister begrenset til de tradisjonelle og mer utbredte protokollene. Aksesslister er tradisjonelt oppbygd sekvensielt hvor man hvitelister trafikk man vil slippe igjennom, før man tilslutt blokkerer den man ikke eksplisitt har godtatt. Med slike aksesslister må man påregne at protokoller som er lite utbredt blir blokkert.

Når det gjelder maskinvarebaserte rutere og brannmurer, er det ikke urimelig å anta at behandling av uvanlige protokoller skjer i programvare. Dette innebærer at lasten på ruteren øker, og dertil er det også en grunn til at systemeier ikke slipper trafikken igjennom.

Et eksempel på problematikk som kan oppstå når man tar i bruk nye standarder, er Cisco-bug CSCds23698: «PIX sends RSET in response to TCP connections with ECN bits set». Explicit Congestion Notification

(ECN, standardisert i RFC2481 og RFC3168 [33, 34]) ble aktivert i Linux 2.4, dette medførte at Cisco PIX-brannmurer droppet TCP-handshaken, da ECN-bitene [35,36] tidligere var definert i RFC793 [7] til å være 0.

Problemene som vi har beskrevet er både vanskelig å feilsøke når de oppstår, og vanskelig å ta høyde før man implementerer protokollen.

## 2.3 Sammendrag

Båndbreddeaggregering på datalinklaget utgjøres primært av IEEE 801.AX, eller tilsvarende propertiære løsninger, disse er fullstendig uegnet til kravspesifikasjonen til denne oppgaven.

På nettverks- og transportlaget er det et større utvalg av forskjellige løsninger. Men alle de eksisterende løsningene faller igjennom på ulike punkter. De har enten forutsetninger som er urelistiske å oppfylle, eller er de designet til å oppfylle andre krav enn våre egne. Det er særlig NAT som setter begrensninger på transportlagsprotokoller. Å utvikle en egne protokoll, spesielt på nettverkslaget medfører endel problemer, men det blir enklere høyere opp på OSI-modellen.

I kapittel 3 vil vi gjennomgå hvilke designvalg vi gjorde, og hva som måtte fjernes/implementeres i vår variant av Westwood.

# Kapittel 3

# Design

For å få implementert og evaluert Westwood i programvaren vi har utviklet, måtte vi skissere opp designet for hvordan implementasjonen skulle gjennomføres. CCID2-implementasjonen og Westwood-algoritmen måtte analyseres, slik at vi kunne spesifisere hva vi måtte endre for å implementere Westwood. Vi måtte også ta et designvalg på andre områder, som f.eks. valg av tunneleringsløsning.

Westwood er implementert i Linux 2.4 og senere utgaver, samt nettverkssimulatoren «ns2». Vi har valgt å bruke Linux-versjonen som referanse. Ved å sammenligne med en implementasjon fra et kjørende system istedenfor med en nettverkssimulator så har vi bedre muligheter til å lage fornuftige sammenligningsgrunnlag.

## 3.1 Eksisterende implementasjon

### 3.1.1 Metningskontroll

Båndbreddeaggregering-programvaren brukte i utgangspunktet metningskontrollen CCID2. DCCP er en serie med metningskontroller som tilbyr UDP støtte for egenskaper man finner i TCP, som f.eks. selective ack (SACK) og explicit congestion notification (ECN). DCCP er ment å bli implementert

på transportlaget, slik at man slipper å implementere metningskontroll på et høyere lag i hver applikasjon. DCCP kommer i tre forskjellige versjoner; CCID2, CCID3 og CCID4, standardisert i henholdsvis: RFC4341 [37], RFC4342 [38] og RFC4828 [39]. CCID2 som var implementert i programvaren vår er tiltenkt ref RFC4341:

> CCID 2, TCP-like Congestion Control, is appropriate for DCCP flows that would like to receive as much bandwidth as possible over the long term

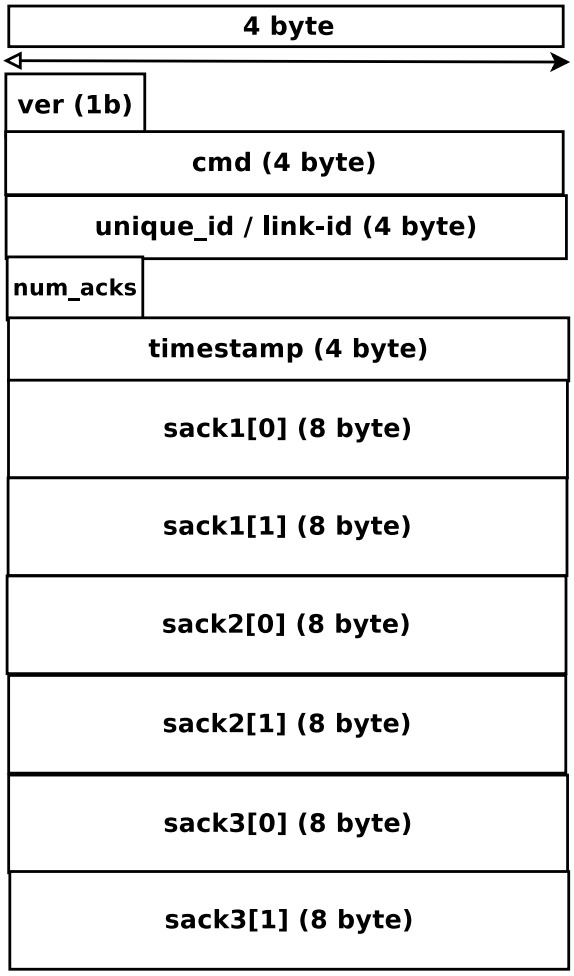
Til sammenligning så er CCID3 og CCID4 tiltenkt strømmer med lik størrelse på pakkene, og mindre behov for båndbredde. CCID2 er altså det åpenbare valget av metningskontroll i DCCP-serien til vår bruk.

Som de fleste TCP-baserte metningskontroller, baserer CCID2 seg på en «additive increase multiplicative decrease» (AIMD) basert fremgangsmåte. I likhet med Westwood, så har CCID2 implementert fremgangsmåten fra RFC2581 [13] for å detektere metning i nettverket.

## 3.2 Våre endringer

Endringer som måtte implementeres, eller endres for å kunne bruke Westwood var følgende:

- Sekvensnummer ble erstattet med pakkestørrelse. CCID2 forholder seg til sekvensnummer uten noe begrep om hvor store pakkene er.
- Sendersiden måtte lagre hva som hadde blitt sendt. CCID2 bruker sekvensnummer som gjør det trivielt å ha en oversikt over hvilke pakker som har blitt sendt, og i hvilken rekkefølge.
- Selektiv ACK ble endret. CCID2 har en selektiv ACK-funksjonalitet som vi utvidet.
- Algoritmen for båndbredde-beregning måtte utvikles. SACK støtter ingen form for beregning av båndbredden.




Figur 3.1: ACK-pakke etter modifisering

Vår implementasjon bruker Westwood-kjernemodulen fra Linux 3.2.2 som referanse. Ettersom Westwood-kjernemodulen kjører som en integrert del av TCP/IP-stakken i Linux-kjernen måtte vår kode tilpasses noe. Linux-implementasjonen bruker funksjoner og elementer som ikke uten videre kunne videreføres i vår implementasjon.

### 3.2.1 Oktetter

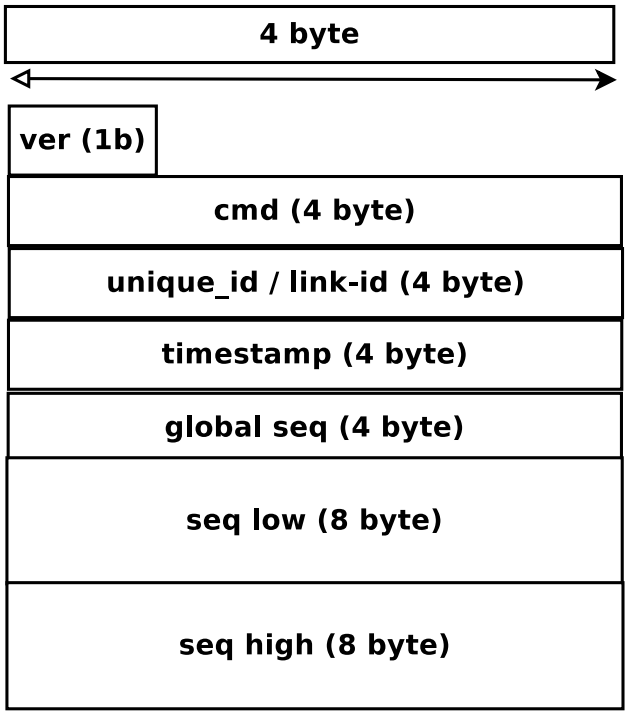
Sekvensnummer måtte erstattes med å ha to oktetter som markerer starten og slutten på pakkene. Dette fordi algoritmen i Westwood og de

underliggende andre standardene som RFC2581 [13], forholder seg til størrelsen på pakkene istedenfor sekvensnummer. Med bruk av oktetter fremfor sekvensnummer i utregningene til metningskontrollen, tar vi hensyn til at pakkestørrelsen kan variere. TCP har sin Nagle-algoritme (ref. RFC896 [40]) som prøver å slå sammen flere små pakker til en større pakke. UDP har ingenting tilsvarende Nagle-algoritmen, spredningen i pakkestørrelse er potensielt større enn hvis man hadde brukt TCP. Sekvensnummer kan være en bra løsning, gitt at pakkestørrelsen er uniform, og det er trivielt å detektere pakketap og omordning av pakker, da ACK-pakkene burde komme i numerisk rekkefølge. Ulempen med å bruke oktetter fremfor sekvensnummer, er at man må ha en oversikt over hvilke segmenter man har sendt, for å kunne detektere pakketap og omordning av pakker ved å se på ACK-pakkene.

Den orginale tunneleringsprogramvaren med CCID2 brukte en ramme hvor det var to sekvensnumre; et globalt og et lokalt for linken. Det globale sekvensnummeret brukes for å håndtere pakkeflyten i de fysiske linkene i nettverkstunellen, slik at pakkene kommer i riktig rekkefølge. Det lokale brukes av metningskontrollen på linken. Det lokale seksvensnummeret ble erstattet med to felter for henholdsvis laveste og høyeste oktett i datapakken som ble overført. Det globale sekvensnummeret ble beholdt.

Oversikten over sendte oktettintervaller ble lagret i et ringbuffer, dette passet bra til «sliding window» data. Ulempen med ringbuffer er at ved pakketap må sender iterere seg gjennom ringbufferet for å finne ut hvor mange pakker som har gått tapt. Ringbufferet vi bruker for å holde styr på oktettintervallene som er sendt har en begrenset størrelse. Vi har derfor måttet sette en begrensning på metningsvinduet for å forhindre at køen av utestående pakker blir større enn hva det er plass til i ringbufferet.

Rammen for ACK-pakker ble utvidet som illustrert i figur 3.1. Modifiseringen medførte at ACK-rammen ble utvidet fra 28 til 72 oktetter. Ytelsesmessig spiller dette liten rolle i et tradisjonelt trafikkmønster med asymmetrisk trafikk mellom tjener og klient, da det sjeldent er




Figur 3.2: Dataramme etter modifisering

båndbredden fra klient til tjener som er flaskehalsen.

Rammen for data-pakker måtte også utvides, som illustrert i figur 3.2. Pakken fikk tilført to uint64 for henholdsvis start og sluttoktett, det lokale sekvensnummeret ble fjernet. Dette innbar at rammen økte i størrelse fra 24 til 40 oktetter.

### Selective acknowledgement

CCID2 har en enkel variant av «selective acknowledgement» (SACK). I ACK-pakkene så var det en matrise, *ack_vector*, med plass til to sekvensnumre. Slik kan CCID2 slå sammen to ACK-pakker til en. *Num_acks*, som er størrelsen på *ack_vector* eksisterte i pakkerammen fra CCID2. *Ack_vector* ble erstattet til fordel for tre matriser med plass til to 64-bit heltall. Dette for å kunne aggregere opptil tre ACK-pakker til en. Til forskjell fra implementasjonen i TCP-stakken til Linux, er plassene til SACK fast allokert

i rammen.

For å optimalisere overføringen, har vi valgt å videreføre SACK-funksjonaliteten fra CCID2. Dette innebærer at bruken av SACK er noe annerledes enn beskrevet i RFC2018 [41]. Sender vil i utgangspunktet prøve å utnytte de tre ledige SACK-feltene i datarammen for å spare antallet ACK-pakker, slik som CCID2 gjør. SACK som beskrevet i RFC2018 bruker SACK-feltene for å markere huller i segmentene som ble mottatt. Dette vil medføre at mottaker bruker mindre båndbredde for å bekrefte at pakker mottas, samt at i et halvt-dupleks miljø vil mottaker blokkere mediet sjeldnere.

Ettersom maksverdien til en «unsigned int32» er 4294967295, så vil vi få overskrivingsproblematikk hver gang nettverkstrafikken passerer 4,3GiB. Ved å bruke «unsigned int64» som datatype istedenfor så slapp vi denne problematikken.

### 3.2.2 Båndbreddeestimering

De fleste metningskontrollene på TCP baserer seg på å finne ut når det inntreffer metning i nettverket ved å øke metningsvinduet inntil man opplever metning, for å halvere metningsvinduet når metning oppstår. Denne fremgangsmåten er kjent som «additive increase/multiplicative decrease» (AIMD).

Westwood skiller seg fra disse ved at metningskontrollen beregner båndbredden (BWE - «bandwidth estimate»), ved hvert RTT-vindu. Westwood «sampler» båndbredden ved å beregne mengden data $D_k$ som har blitt bekreftet mottat utifra ACK-pakker, deretter beregnes båndbredden $B_k$ i hvert RTT-vindu $T_k$ som $B_k = D_k / T_k$.

For å ikke få ekstreme verdier ved for liten RTT, så har Westwood en minimumsverdi på 50 millisekunder på et RTT-vindu. Båndbreddesamplingen går gjennom et «low-pass filter» uttrykt som:

$$BWE_k = \frac{7}{8} BWE_{k-1} + \frac{1}{8} B_k$$

BWE er båndbreddeestimat, k er nåværende RTT-vindu og k-1 er forrige vindu.

Mottatte DUPACK-pakker kalkuleres med i båndbreddeestimatet, men vi endrer ikke metningsvinduet når ACK-pakkene kommer i omstokket rekkefølge.

Hensikten ved å beregne båndbredden er å unngå den tradisjonelle fremgangsmåten, ved å halvere metningsvinduet ved metning. Da metningskontrollen vet hvor stor båndbredde nettverksforbindelsen støtter. Dette er en fordel hvis man har en trådløs nettverksforbindelse som er utsatt for kortvarig tilfelle av pakketap. Ettersom forstyrrelsen i nettverkstrafikken ikke skyldes metning, men støy så er det ikke nødvendig å halvere metningsvinduet.

### 3.2.3 Metningsvindu og ssthresh

Westwood følger RFC2581 [13] når det gjelder å finne ut når det inntreffer metning i nettverket, dvs. ved å øke metningsvinduet og sette terskelverdien «ssthresh». Algoritmen er beskrevet i [16] på følgende måte:

- Ved tre duplikate ACK-pakker så settes
  *ssthresh* = *(*BWE * RTT*min)* / MSS.
  Og *metningsvindu* = *ssthresh*.

- Gitt forventet mottak av ACK-pakker øker metningsvinduet som i RFC2581 [13].

- Gitt ingen ACK-pakker mottat innen RTO settes
  *ssthresh* = (BWE * RTTmin) / MSS.
  og *metningsvindu* = 2. [1]

[1] Linux-implementasjonen setter metningsvinduet i denne tilstanden til 2. [16] sier at den skal settes til 1.

Flere metningskontroller setter den initielle verdien til «ssthresh» til maksverdien til datatypen, denne verdien forblir høy inntil metningskontrollen eventuelt opplever metning. Tilsvarende har vi gjort det i vår implementasjon.

#### Maximum segment size og UDP

RFC2581 bruker «maximum segment size» (MSS) for å beregne metningsvinduet. Siden UDP ikke har noe begrep om segmenter, har heller ikke MSS noen mening. UDP-rammen har et 16 bits felt for lengde som altså gir en teoretisk grense på 65535bits minus UDP-ramme minus IP-ramme. På TCP settes MSS normalt utifra MTU på nettverkskortet minus TCP-ramme minus IP-ramme. UDP har heller ingen «path MTU» funksjonalitet for å kalkulere optimal pakkesegmentstørrelse, basert på hva rutere mellom sender og mottaker støtter. Ettersom det virtuelle TUN-grensesnittet har en MTU på 1400 oktetter, bruker vi denne verdien som MSS.

## 3.3 Tilstander i Westwood

Westwood-modulen til Linux inneholder følgende tilstander:

- CA_EVENT_FAST_ACK
- CA_EVENT_COMPLETE_CWR
- CA_EVENT_FRTO
- CA_EVENT_SLOW_ACK

Da vår implementasjon ikke inkluderer støtte for retransmisjon for å ikke forandre oppførselen til UDP, er ikke alle disse tilstandene tatt i bruk. I vår implementasjon som har tilstander basert på RFC2581 [13], kaller vi tilstanden CA_EVENT_FAST_ACK når ACK-pakker mottas som forventet. Tilstanden CA_EVENT_FRTO kalles når tjeneren mottar

tre duplikate ACK-pakker uten mellomrom fra andre pakker. Tilstanden CA_EVENT_SLOW_ACK bruker vi når tjenersiden detekterer pakketap via manglende ACK-pakker med mindre omstokking enn tre duplikate ACK-pakker. CA_EVENT_COMPLETE_CWR bruker vi ikke, da dette er en tilstand som brukes av Linux-implementasjonen til retransmisjon.

I Linux-kjernen opptrer Westwood som en integrert del av TCP/IP-stakken, Linux har god støtte for modulære metningskontroller via flere «hooks» [42].

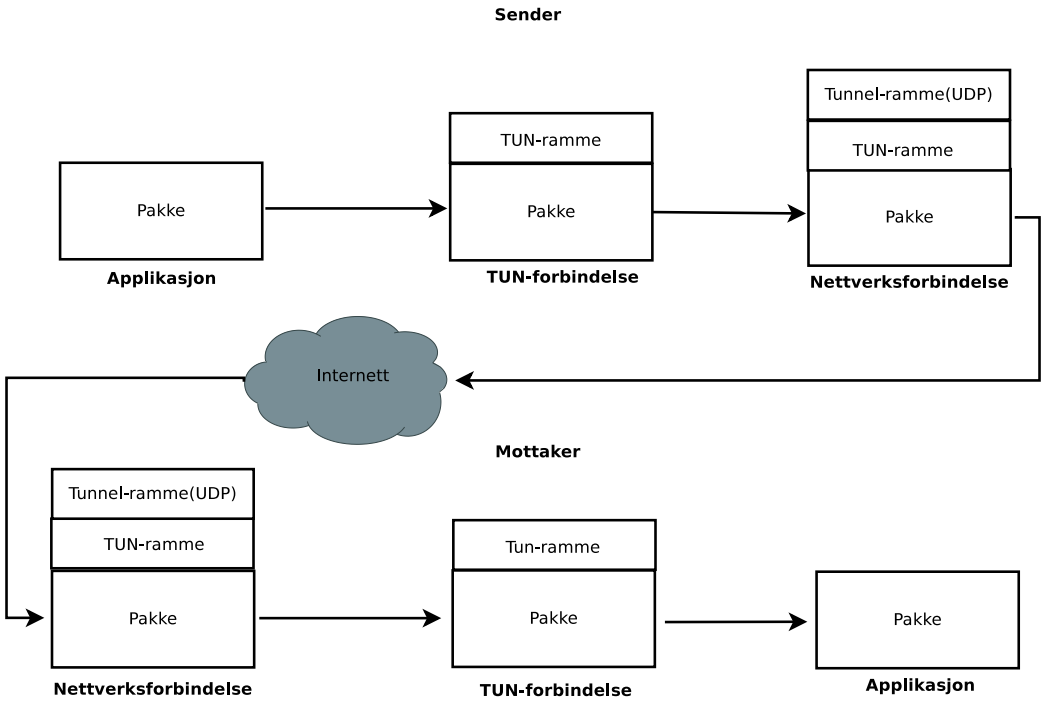
Westwood håndterer selv alle funksjonene i metningskontrollen, unntatt «slow start» og «congestion avoidance». Disse håndteres av standard-funksjonene, som er definert i `/net/ipv4/tcp_cong.c`.
At Westwood selv ikke håndterer all funksjonalitet i metningskontrollen har gjort det vanskelig å få et totalt overblikk over hvordan ting henger sammen. Debugging av kjernemodulen gjorde det enklere å få oversikten.

## 3.4 Vår tunnelimplementasjon

### 3.4.1 NettverksTUNnel

TUN (som i nettverksTUNnel) og TAP (som i network TAP) er virtuelle nettverksforbindelser. Begge to eksisterer kun i kjernen på maskinen, og de har følgelig heller ingen fysisk maskinvare tilordnet. TAP er virtuelle grensesnitt på datalinklaget («pseudo wire»). TUN, som vi har brukt i programvaren i oppgaven er en virtuell nettverksforbindelse på nettverkslaget. TUN fungerer som en regulær forbindelse, sett fra operativsystemet og brukeren sin side. Det eneste som skiller en TUN-forbindelse fra en vanlig nettverksforbindelse, er at TUN-grensesnittet ikke har noen ethernet-rammer. TUN er støttet på flere plattformer.

TUN-rammeverket gir enkel bruk av tunneller, det er dette rammeverket vi har brukt i programvaren som har blitt utviklet i oppgaven, illustrert i figur 3.3. I programvaren vi har utviklet vil alle pakkene få to lag med



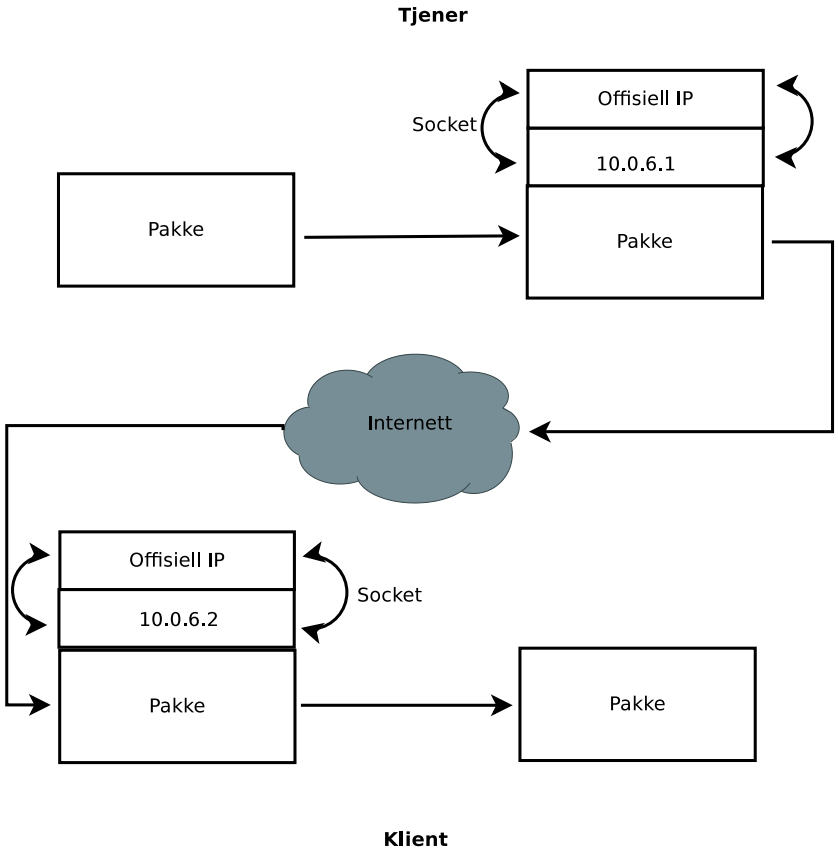

Figur 3.3: Vår tunnelimplementasjon

rammer. En for tunnelen vi oppretter, og en for nettverkssocketen vi bruker for å sende trafikk mellom tjener og klient. Utgående trafikk får først en TUN-ramme, før TUN-rammen legges på tilslutt.

Tunnellen fungerer ved at programvaren tunnelerer all trafikk mellom en socket på tjenersiden og en socket på klientsiden. Dette innebærer noen fordeler, siden socketene vi oppretter bruker UDP som tunneleringsprotokoll, kommer vi rundt mye av problematikken nevnt i kapittel 2, særlig med NAT.

Fra et nettverksmessig synspunkt blir TUN-forbindelsen opprettet med en ikke-rutbar RFC1918-adresse som er hardkodet i programmet.

Oppsettet med socket-er er identisk hos både klient og tjener. Socketen som blir opprettet binder seg til IP-adressen til TUN-forbindelsen og den/de globale IPv4-adressen(e) som man velger å bruke til tunnelen. Slik får man fungerende ruting mellom tjener og klient, som illustrert i figur 3.4. Tunnelen gjør at tjener og klient får en nettverksforbindelse som




Figur 3.4: Nettverksmessig oppsett av tunnelen

fremstår som om de står på samme subnett.

For å benytte oss av flere nettverksforbindelser, har vi på Linux brukt `iproute` for å sette opp klientmaskinen med «N» antall standardruter, vektet etter båndbredde.

Når tunnelen opprettes på henholdsvis tjener- og klientsiden vil det opprettes et TUN-grensesnitt som fjernes når programvaren avsluttes.

På Linux må man i utgangspunktet ha root-tilgang for å opprette TUN-grensesnitt, men tilgangen kan deles ut til vanlige brukere ved å dele ut privilegiet «CAP_NET_ADMIN».

### Maximum transmission unit og nettverkstunnellering

Ulempen med bruk av tunneler på et generelt nivå, er ytelsestap i form av dobbelt lag med rammer, i vårt tilfelle så får alle pakkene et ekstra lag med IP-rammer som brukes internt i tunnelen. «Maximum transmission unit» (MTU) er grensen for hvor store segmenter nettverkslaget kan håndtere. På ethernet er grensen historisk sett 1500 oktetter (ekskludert ethernet-rammen). «Jumboframes» som støttes på gigabit og raskere ethernet har en høyere, ikke-standardisert størrelse på opptil ca. 9000 oktetter.

Den økte rammestørrelsen som kommer som en konsekvens av tunneleringen, vil gjøre at MTU blir lavere. I vårt tilfelle medfører MTU-rammeverket et tap på 20 oktetter. Mindre MTU kan også generere pakke-fragmentering, i de tilfellene hvor applikasjonene ikke ser på MTU før pakkene sendes til nettverket. Ulempen med pakkefragmentering, er at en pakke må deles opp i to pakker. Følgelig får man en lite optimal bruk av nettverksressursene. UDP har heller ingen funksjonalitet for å slå sammen mindre pakker til en stor som TCP har med Nagle. Ved å implementere en løsning på nettverkslaget, får man en løsning som er transparent for høyere lag slik at applikasjoner på enheten fungerer uten tilpassninger.

Båndbreddeaggregerings-programvaren krever ikke krever endringer i kjernen, mengden avhengigheter av systembibloteker er marginal og administratortilgang for å installere og eksekvere programmet ikke er nødvendig så er det en lav terskel for å ta programvaren i bruk.

## 3.5 Sammendrag

I dette kapittelet har vi listet opp det vi måtte utvikle for vår Westwood-implementasjon. Vi har analysert den eksisterende metningskontrollen CCID2, for å peke ut de punktene som måtte endres, og det som måtte implementeres fra bunnen av. Basert på analysen har vi designet de endringene som måtte implementeres for å innføre Westwood som metningskontroll.

Designet av en løsning for hvordan båndbreddeaggregeringen skal foretas på nettverksnivå er en viktig del av oppgaven. Vi har i dette kapittelet gjennomgått tunneleringsløsningen vår med de ulemper og fordeler TUN medfører.

Neste kapittel omhandler hvordan designet ble implementert i praksis.

# Kapittel 4

# Implementasjon

Kapittel 3 omhandlet designvalgene vi måtte ta, for å få implementert de forskjellige elementene i programvaren vi utviklet. Dette kapittelet handler om hva som ble gjort for at planleggingen fra forrige kapittel kunne implementeres i vår programvare.

## 4.1 Implementasjonsspesifikke detaljer

Både CCID2 og Westwood har «slow start» og «congestion avoidance», som beskrevet i RFC2581 [13]. Denne funksjonaliteten håndteres ikke av Westwood-modulen, men bruker funksjonene fra `/net/ipv4/tcp_cong.c`. Vår implementasjon har brukt koden i denne filen, samt `/net/ipv4/tcp_westwood.c` som utgangspunkt.

Funksjonen «tcp_reno_cong_avoid», som er definert i `tcp_cong.c`, forholder seg ikke til RFC2581, da funksjonen har implementert «TCP congestion control with appropriate byte counting (ABC)», som beskrevet i RFC3465 [43]. Vi har valgt å utelate funksjonaliteten som stammer fra RFC3465 fra vår implementasjon, da det er endringer som har kommet i etterkant av RFC2581. Det har kommet flere RFC-er utover RFC3465 som er relatert til RFC2581. RFC5681 [44] erstatter hele RFC2581. Såvidt vi kan se, har vi heller ikke tatt med funksjonalitet i vår implementasjon som har

kommet med RFC5681.

Båndbredde-beregningen til Westwood er basert på at ved hvert RTT-vindu vil Westwood rekalkulere båndbredden i bruk. Dette ble implementert i vår versjon, ved å beregne tidsforskjellen mellom nåværende tid, og når pakken ble mottatt.

Som vi skrev om i kapittel 3, bruker Westwood «maximum segment size» (MSS) for å beregne både båndbredde, og kalkulasjon av «ssthresh». UDP har ikke noe begrep om MSS, vi har brukt 1400 som er MTU på TUN-forbindelsen som MSS.

### 4.1.1 Tilstander i Westwood

Westwood-implementasjonen i Linux inneholder flere tilstander enn i vår implementasjon, dette skyldes primært at vår implementasjon ikke har støtte for retransmitering. Retransmitering støttes av TCP, hvor det er garanti for at pakkene kommer frem og blir levert i riktig rekkefølge. UDP støtter ikke dette, og vi har ikke ønsket å endre oppførselen til UDP. Pakketap må således håndteres av høyere lag.

I Linux-kjernen foregår flytkontrollen pr. TCP-forbindelse, mens i vår implementasjon skjer flytkontrollen pr. nettverkstunnel.

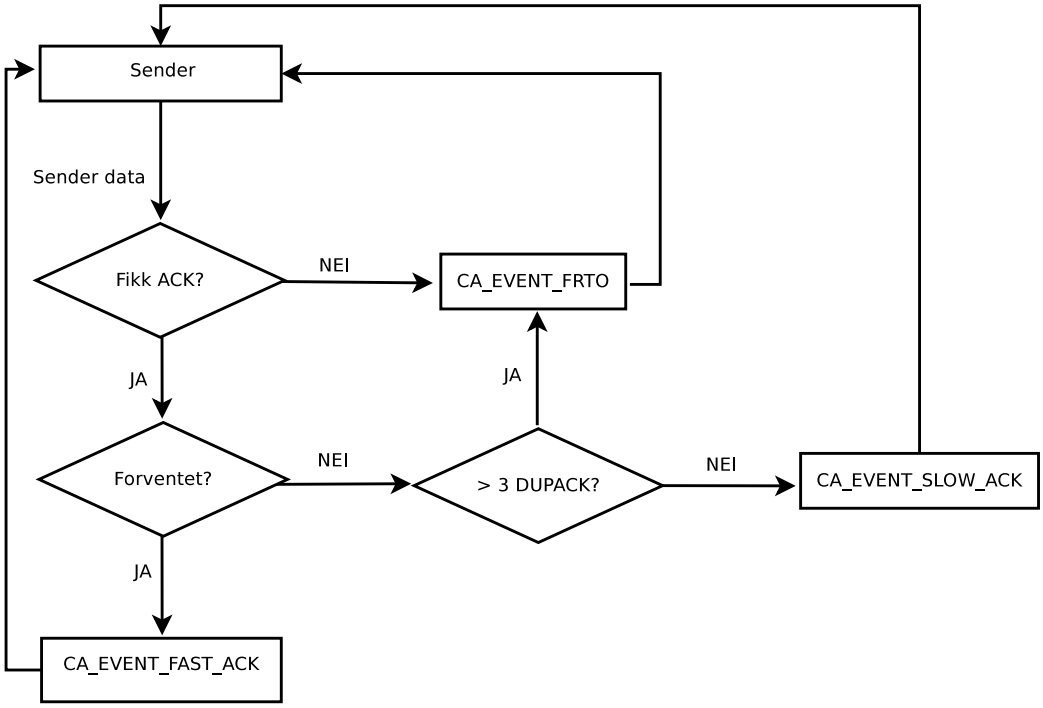
Linux-kjernemodulen til Westwood har tilstanden CA_EVENT_COMPLETE_CWR, i tillegg til de som har blitt implementert i vår kode. Tilstandene CA_EVENT_TX_START og CA_EVENT_CWND_RESTART, som brukes i TCP/IP-stakken, håndteres ikke av Westwood-kjernemodulen.

De forskjellige tilstandene som er tilgjengelige for metningskontrollene i Linux, er definert i `/include/net/tcp.h`.

Flyten i vår Westwood-implementasjon er illustrert i figur 4.1.

I vår implementasjon settes tilstanden til CA_EVENT_FAST_ACK, hvis flyten av ACK-pakker ankommer som forventet. I denne tilstanden øker metningsvinduet som beskrevet i RFC2581 [13].

Hvis flyten av ACK-pakker ikke går som forventet, skyldes dette en




Figur 4.1: Flytdiagram for vår Westwood-implementasjon

av to ting. Metningskontrollen havner i tilstanden CA_EVENT_SLOW_ACK når det ankommer ACK-pakker hvor flyten er omstokket. I denne tilstanden forblir metningsvinduet uendret, men båndbreddekalkulasjonen finner fortsatt sted sålenge det ankommer ACK-pakker.

Hvis det ikke ankommer ACK-pakker i det hele tatt, settes tilstanden til CA_EVENT_FRTO. Denne tilstanden brukes også hvis det ankommer tre eller flere DUPACK-pakker, uten at det er andre pakker innimellom. Hvis dette inntreffer, settes «ssthresh» til en båndbredde-beregnet verdi, og metningsvinduet til «ssthresh».

Gitt ingen mottatte ACK-pakker innenfor tidsintervallet satt av RTO-funksjonaliteten, settes metningsvinduet til 2, og «sshthresh» til en båndbreddeberegnet verdi.

Tilstandsmaskinen vi har implementert, baserer seg på to funksjoner, «westwood_acked_count» som setter neste tilstand basert på ACK-pakkene som ble mottatt, og RTO-funksjonalieten i koden som kaller

CA_EVENT_FRTO når det ikke kommer noen ACK-pakke innenfor det fastsatte tidsintervallet.

Vi kunne ha implementert en tilstand til. Vi havner i tilstanden CA_EVENT_FRTO fra RTO-funksjonaliteten ved tidsavbrudd, og når vi får tre sammenhengende DUPACK-pakker. Dette kunne ha blitt implementert med to forskjellige tilstander istedenfor, men vi valgte å bruke en felles tilstand, da oppførselen til de to tilstandene hadde blitt tilnærmet identisk.

## 4.2 Sammendrag

I dette kapittelet har vi beskrevet hvordan designet fra forrige kapittelet ble implementert i praksis, og hvordan vi tilpasset oss referanseimplementasjonen fra Linux. Videre beskriver vi hvordan vår implementasjon av Westwood henger sammen, hvilke tilstander vi har tatt med i koden, og hvordan flyten mellom de forskjellige tilstandene er. I neste kapittel har vi vurdert hvordan implementasjonen vår fungerte sammenlignet med CCID2 og Linux-implentasjonen av Westwood.

# Kapittel 5

# Vurderinger

Forrige kapittel omhandlet hvordan vi implementerte Westwood, i dette kapittelet har vi vurdert hvordan implementasjonen fungerer i ulike testscenarier. Fokus på testingen har basert seg på å sammenligne vår implementasjon mot den eksisterende løsningen, med CCID2 som metningskontroll, og mot Westwood-implementasjonen i Linux-kjernen. Vi har testet implementasjonen vår med et tradisjonelt asynkront trafikkmønster, dvs. at trafikken utelukkende går en vei mellom tjener og klient.

`iperf` som vi har brukt under testingen, er et multiplattformverktøy med godt støtte for ulike testscenarier. Verktøyet kan brukes på både TCP og UDP, og har støtte for å sette forskjellige testparametere. Verktøyet forutsetter at man har kjører `iperf` i daemon-modus på tjenermaskinen, og `iperf` i klientmodus på den andre maskinen. `iperf` har ikke implementert noen form for metningskontroll på UDP. All UDP-testing innebærer at man sender trafikk på en fast trafikkrate.

I tillegg til `iperf` har vi brukt et egenutviklet verktøy som kun støtter bruk av UDP. Funksjonaliteten og virkemåten forøvrig, er i stor grad identisk med `iperf`, men støtten for å kunne sette pakkestørrelsen og dertil unngå fragmentering av pakker er bedre enn med `iperf`.

Målingene i forsøkene er foretatt på datalinklaget, målt ytelse på transportlaget er naturlig nok høyere. Fordelen med å måle på datalink-

laget sammenlignet med på transportlaget, er at målingene inkluderer all overhead, noe som gjør det enkelt å sammenligne de ulike scenariene opp mot hverandre.

Den store forskjellen mellom målt båndbredde på datalinklaget, og transportlaget i forsøkene våre med 8.5 Mbit UDP bitstrøm, skyldes primært en lite optimal pakkestørrelse, men også overheaden som kommer fra nettverkstunnelen.

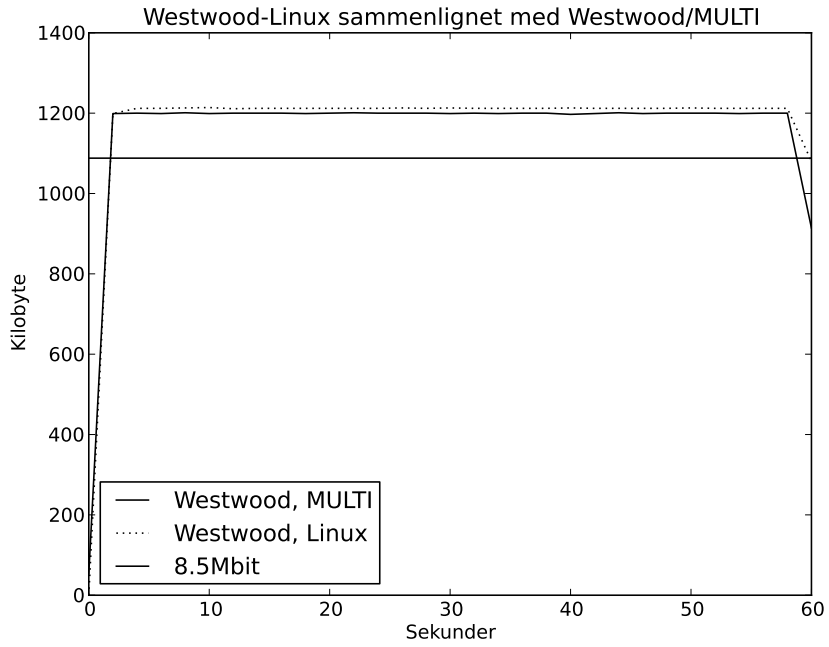
## 5.1 Vår Westwood- og Linux-implementasjon

Det kan by på problemer å sammenligne vår Westwood-implementasjon mot Linux sin Westwood-implementasjon, da den ene bruker TCP, og den andre bruker UDP. Det er også naturlig at Linux-implementasjonen ikke bruker nettverkstunnelen vi benytter oss av i programvaren vi har utviklet, så sammenligningsgrunnlaget blir ikke identisk. Målet med implementasjonen vår, er at de to implementasjonene skal ha lik ytelse, minus overhead fra tunneleringen.

I scenariet illustert i figur 5.1, har vi skapt nettverkstrafikk ved å sende UDP-data med fast bitrate på 8.5 Mbit i tilfellet med vår Westwood-implementasjon. Til sammenligning har vi brukt `iperf` over TCP mellom to Linux-maskiner, som begge brukte Westwood som metningskontroll. Nettverksoppsettet er identisk i alle forsøkene, klientmaskinen har en 10 Mbit ethernetforbindelse og det er heller ikke pakketap.

I scenariet illustrert i figur 5.2 er identisk med forrige forsøk med to unntak: Bitstrømmen er på 10 Mbit, ikke 8.5 Mbit som i forrige forsøk. 10 Mbit vil si at vi sender mer trafikk enn hva linken støtter, følgelig tar vi all kapasiteten til linken. Etter 10 sekunder oppstod det et tap av konnektivitet på nettverkslaget som varte i to sekunder. Dette scenariet skal forestille et typisk trafikkmønster, som inntreffer på trådløse nett når en klient hopper fra et aksesspunkt til et annet.

I eksempelet bak figur 5.3, har vi sammenlignet vår Westwood-

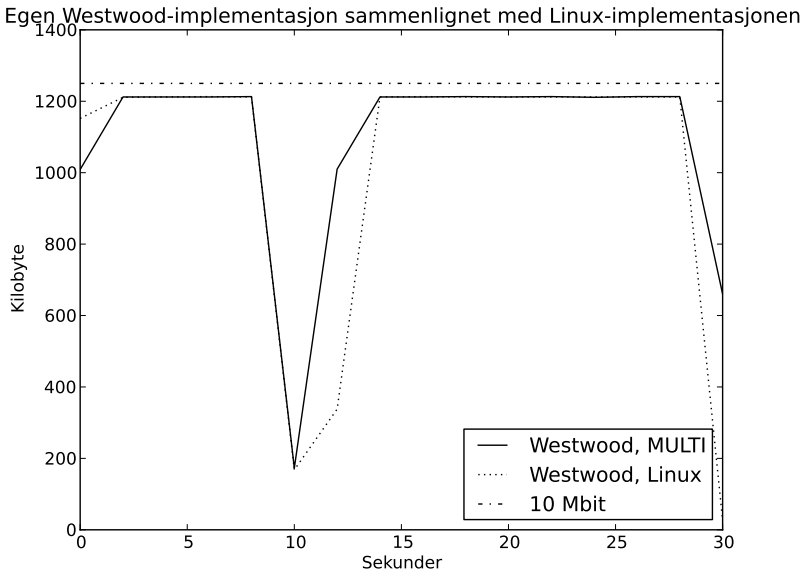



Figur 5.1: Westwood på Linux vs egen Westwood-implementasjon.

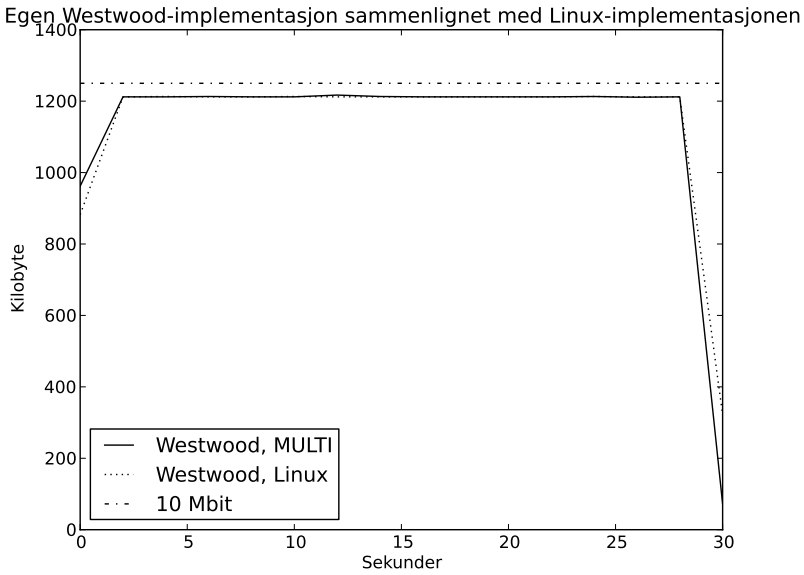
implementasjon med Westwood-implementasjonen til Linux i et scenario med 5% pakketap. Pakketapet er jevnt fordelt utover tidsaksen, men de kommer i «byger». For å generere trafikk til forsøkene med vår implementasjon, har vi brukt en UDP-strøm med fast bitrate på 10 Mbit. I eksemplene med Westwood på Linux, har vi brukt `iperf` over TCP direkte mellom to maskiner.

I 5.4 utførte vi et forsøk til som ble foretatt under identiske omstendigheter som forrige forsøk. Pakketapet i dette forsøket var på 10%, ikke 5% som i forrige forsøk.

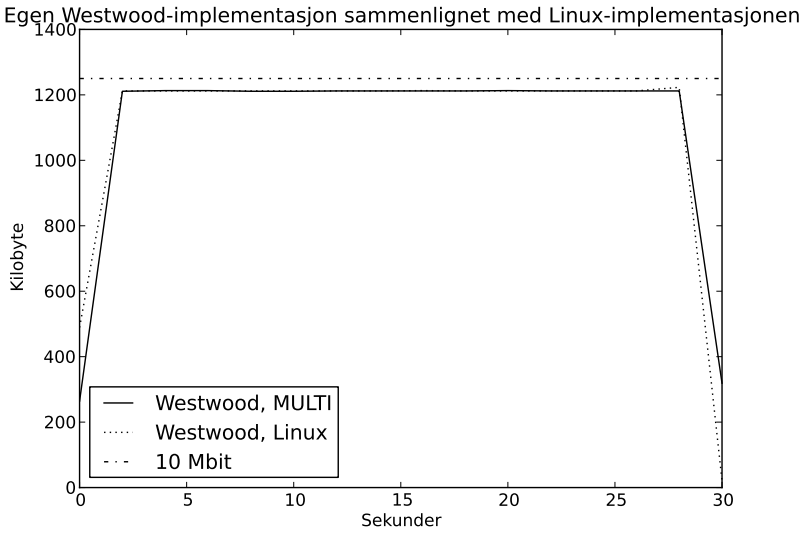
Vi ser av figurene våre at målingene på datalinklaget er tilnærmet identiske. Under alle scenariene hvor vi har sammenlignet Westwood-implementasjonene, har vår Westwood-implementasjon ytet likt som Linux-implementasjonen, fraregnet overheadet fra tunelleringen.




Figur 5.2: Linux-impl. mot egen implementasjon, tap av konnektivitet




Figur 5.3: Linux-impl. mot egen implementasjon, pakketap 5%




Figur 5.4: Linux-impl. mot egen implementasjon, pakketap 10%

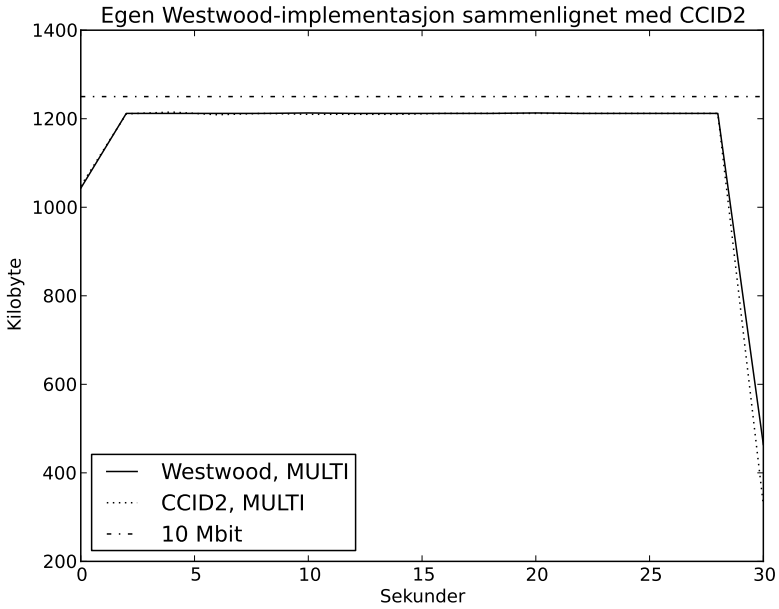
## 5.2 Westwood sammenlignet med CCID2

Det er enkelt å sammenligne CCID2 og Westwoodimplementasjonen vår, det er ingen vesensforskjeller mellom de to. Som med vår Westwood-implmentasjon i forrige avsnitt, så har testingen foregått ved å bruke nettverkstunnelen vi oppretter med programvaren vi har utviklet.

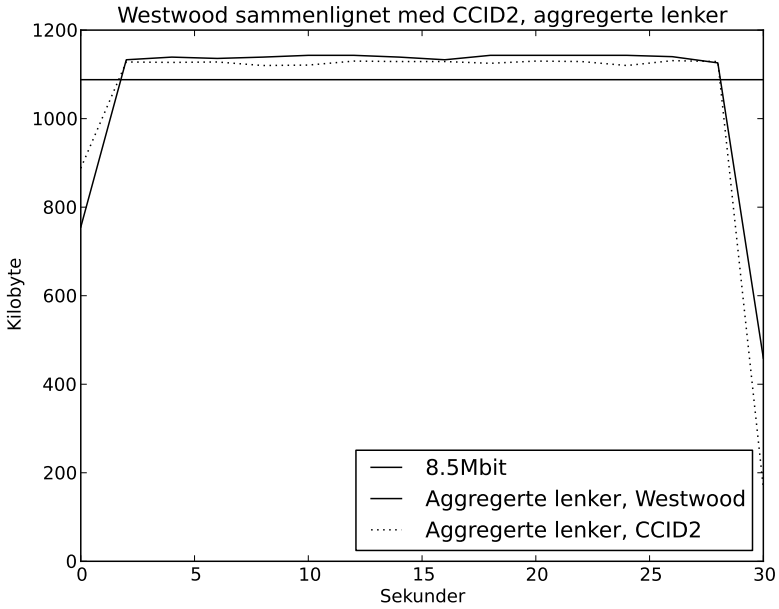
I figur 5.6 har vi sammenlignet CCID2 opp mot Westwood i et miljø med 5% pakketap. Som man kan se, er det ingen signifikante forskjeller mellom de to metningskontrollene. Vi gjorde et tilsvarende forsøk med 10% pakketap, i dette forsøket var også resultatene tilnærmet identiske som i forsøket med 5% pakketap.

Dette kan skyldes at pakketapet er såpass jevnt fordelt utover, at ingen av algoritmene behandlet pakketapet som metning. For å skape nettverkstrafikken har vi sendt UDP-trafikk med fast bitrate på 8.5 Mbit over nettverkstunnelenen.

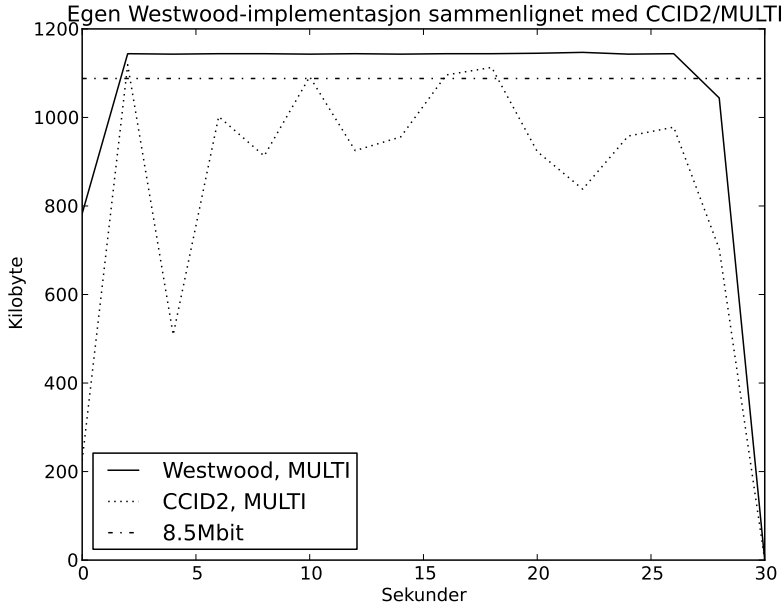
I figur 5.7 har vi sammenlignet CCID2 mot Westwood i et miljø med 15% pakketap. Som man kan se er det en tydelig forskjell mellom ytelsen til de to forskjellige metningskontrollene.




Figur 5.5: Westwood og CCID2 10 Mbit strøm, 0% pakketap




Figur 5.6: Westwood og CCID2 8.5 Mbit strøm, 5% pakketap




Figur 5.7: Egen Westwood og CCID 8.5 Mbit strøm, 15% pakketap

Westwood håndterer bedre situasjoner når det er stort pakketap. Dette skyldes at Westwood beregner båndbredden som nettverksforbindelsen har kapasitet til. Derfor vil metningsvinduet settes til en beregnet verdi vi vet nettverket har kapasitet til når metning inntreffer. Det er tydelig at CCID2, som halvverer metningsvinduet ved metning taper mot Westwood i et slikt scenario med stort pakketap. Westwood er beryktet for å være bra i scenarier med pakketap, og vi ser her at det stemmer.

## 5.3 Sammendrag

I dette kapittelet har vi vurdert vår Westwood-implementasjon opp mot metningskontrollene CCID2 og Linux-implementasjonen av Westwood. Da Westwood sin sterke side skal være god ytelse når det er pakketap, har vi sammenlignet vår Westwood-implementasjon med CCID2 under nettopp disse omstendighetene. For å vurdere hvordan vår Westwood-implementasjon oppfører seg sammenlignet med Westwood-implementasjonen i Linux, har vi sammenlignet oppførselen til disse to metningskontrollene under flere forskjellige scenarier. Under alle scenariene hvor vi har sammenlignet Westwood-implementasjonene har vår Westwood-implementasjon ytet likt som Linux-implementasjonen, frareg-net overheadet fra tunelleringen.

Vi har lagd grafer utifra de forskjellige testene, for å illustrere forskjellene i ytelse, og hvordan de forskjellige metningskontrollene oppfører seg i de forskjellige scenariene. Neste kapittel er også det siste. Her vil vi oppsummere hva oppgaven har dreid seg om. Vi vil og peke ut noen områder hvor det er potensiale for videre arbeid.

# Kapittel 6

# Oppsummering og videre arbeid

I denne oppgaven har vi implementert en Westwood-basert metningskontroll for UDP. I dette kapittelet vil vi oppsummere alt arbeidet vi har utført, og de viktigste resultatene fra evalueringen. Til slutt i kapittelet vil vi komme med forslag til videre arbeid.

## 6.1 Oppsummering

I denne oppgaven har vi designet, implementert og evaluert vår UDP-tilpassede utgave av metningskontrollen Westwood.

Vi sammenlignet ytelsen til metningskontrollen vår mot Linux-implementasjonen av Westwood, og mot metningskontrollen CCID2. Under evalueringen har vi identifisert noen scenarier hvor Westwood utmerker seg til fordel for CCID2. Som forventet, så var dette scenarier med høyt pakketap.

Vår Westwood-implementasjon har vist seg å yte identisk med Linux-implementasjonen til Westwood i flere forskjellige scenarier.

## 6.2 Videre arbeid

Programvaren vi har utviklet er naturlig nok ikke uten forbedringspotensiale. Vi har her kommet med noen ideer til fremtidig arbeid som vil gavne programvaren vi har utviklet.

### 6.2.1 Dynamisk MTU

Ettersom fragmentering av pakker påvirker ytelsen i negativ retning kan det være hensiktsmessig å finne den optimale MTU-verdien. Ved å sette MTU for lav øker man andelen pakker som blir fragmentert. Er den for høy må man ta høyde for at trafikk går tapt. I dag settes MTU statisk når nettverkstunnellen opprettes. Utregningen av MTU kunne med fordel ha blitt beregnet dynamisk, ved å basere seg på MTU-verdien til nettverksforbindelsene som inngår i tunnelen.

### 6.2.2 IPv6-støtte

Programvaren har ingen støtte for IPv6. Hvorvidt denne protokollen blir utbredt blandt sluttbrukere i overskuelig fremtid, er et åpent spørsmål. Ekte IPv6[1] vil løse endel problemer, som f.eks. at man ikke lenger må ta høyde for NAT. Dette åpner for nye muligheter for design av tunneleringsløsningen. Således blir IP-in-IP og andre nettverkslagsprotokoller mer aktuelle.

[1] I motsetning til IPv6-tuneller

# Appendiks

## Etterord

Begreper som f.eks. metningsvindu og metningskontroll er tatt fra nettsidene til norske utdanningsinstitusjoner. Endel andre begreper som f.eks. «additive increase/multiplicative decrease» og forkortelser som «ssthresh» er beholdt, da det ikke ser ut til å eksistere noen etablert norsk oversettelse av ordet, og jeg har ikke ønsket å introdusere mine egne.

# Forkortelser

Liste over forkortelser

| | |
|---|---|
| ACK | Acknowledment |
| AIMD | Additive increase/multiplicative decrease |
| CCID | Congestion Control ID |
| DCCP | Datagram Congestion Control Protocol |
| DUPACK | Duplicate ACK |
| ECN | Explicit Congestion Notification |
| ECMP | Equal-Cost Multi-Path |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| HSPDA | High-Speed Downlink Packet Access |
| ICMP | Internet Control Message Protocol |
| IP | Internet Protocol (implisitt IPv4 hvis ikke nærmere spesifisert) |
| LTE | Long Term Evolution |
| MAC | Media Access Control |
| MSS | Maximum Segment Size |
| MTU | Maximum Transmission Unit |
| SCTP | Stream Control Transmission Protocol |
| NAT | Network Address Translation |
| PPP | Point-to-Point Protocol |
| RFC | Request For Comments |
| RTO | Retransmission timeout |
| RTT | Round-trip time |
| SMTP | Simple Mail Transfer Protocol |
| TCP | Transmission Control Protocol |
| TUN | NettverksTUNnel |
| UDP | User Datagram Protocol |
| WLAN | Wireless Local Area Network |
| WWW | World Wide Web |

# Bibliografi


[1] K Evensen. Aggregating the Bandwidth of Multiple Network Interfaces to Increase the Performance of Networked Applications. 2012.

[2] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. Obsoleted by RFC 3022.

[3] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[4] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001.

[5] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379, 5884, 6093, 6298.

[6] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[7] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[8] V. Jacobson. Congestion avoidance and control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[9] Michael Welzl. *Network Congestion Control: Managing Internet Traffic (Wiley Series on Communications Networking & Distributed Systems)*. Wiley, 2005.

[10] V. Paxson, M. Allman, J. Chu, and M. Sargent. Computing TCP's Retransmission Timer. RFC 6298 (Proposed Standard), June 2011.

[11] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582 (Experimental), April 1999. Obsoleted by RFC 3782.

[12] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001 (Proposed Standard), January 1997. Obsoleted by RFC 2581.

[13] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[14] S Mascolo, C Casetti, M Gerla, M.Y Sanadidi, and R Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. 2001.

[15] L Grieco and S Mascolo. Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control.

[16] Linux 2.4 Implementation of Westwood+ TCP with rate-halving: A Performance Evaluation over the Internet. 2004.

[17] Broadcom. Broadcom Ships New 54g Technology That Delivers Best Real-World Wi-Fi Performance. `http://www.broadcom.com/press/release.php?id=507846`, 2004.

[18] Aaron Balchunas. Etherchannel. `http://www.routeralley.com/ra/docs/etherchannel.pdf`.

[19] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP Multilink Protocol (MP). RFC 1990 (Draft Standard), August 1996.

[20] Dhananjay S. Phatak, Tom Goff, and Jim Plusquellic. Ip-in-ip tunneling to enable the simultaneous use of multiple ip interfaces for network level connection striping. *Computer Networks*, 43:787–804, 2003.

[21] C. Perkins. IP Encapsulation within IP. RFC 2003 (Proposed Standard), October 1996. Updated by RFC 3168.

[22] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992 (Informational), November 2000.

[23] R. Stewart. Stream Control Transmission Protocol. RFC 4960 (Proposed Standard), September 2007. Updated by RFCs 6096, 6335.

[24] M. Tuexen and R. Stewart. Stream Control Transmission Protocol (SCTP) Chunk Flags Registration. RFC 6096 (Proposed Standard), January 2011.

[25] Janardhan R. Iyengar, Paul D. Amer, and Randall Stewart. Concurrent multipath transfer using sctp multihoming over independent end-to-end paths. *IEEE/ACM Trans. Netw.*, 14(5):951–964, October 2006.

[26] R. Stewart and M. Tuexen. Stream control transmission protocol (sctp) network address translation. `http://tools.ietf.org/html/draft-ietf-behave-sctpnat-06`.

[27] Ming Zhang, Junwen Lai, Arvind Krishnamurthy, Larry Peterson, and Randolph Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '04, pages 8–8, Berkeley, CA, USA, 2004. USENIX Association.

[28] Hung-Yun Hsieh and Raghupathy Sivakumar. ptcp: An end-to-end transport layer protocol for striped connections. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, ICNP '02, pages 24–33, Washington, DC, USA, 2002. IEEE Computer Society.

[29] C. Raiciu, M. Handley, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356 (Experimental), October 2011.

[30] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar. Architectural Guidelines for Multipath TCP Development. RFC 6182 (Informational), March 2011.

[31] Kyu-Han Kim and Kang G. Shin. Prism: Improving the performance of inverse-multiplexed tcp in wireless networks. *IEEE Transactions on Mobile Computing*, 6(12):1297–1312, December 2007.

[32] A. Huttunen, B. Swander, V. Volpe, L. DiBurro, and M. Stenberg. UDP Encapsulation of IPsec ESP Packets. RFC 3948 (Proposed Standard), January 2005.

[33] K. Ramakrishnan and S. Floyd. A Proposal to add Explicit Congestion Notification (ECN) to IP. RFC 2481 (Experimental), January 1999. Obsoleted by RFC 3168.

[34] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.

[35] Evi Nemeth, Trent R Hein, and Garth Snyder. *Linux Administration Handbook*. Prentice-Hall, Upper Saddle River, NJ, 2002.

[36] Cisco Systems Inc. . Cscds23698 bug details. `http://tools.cisco.com/Support/BugToolKit/search/getBugDetails.do?method=fetchBugDetails&bugId=CSCds23698`, may 2012.

[37] S. Floyd and E. Kohler. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control. RFC 4341 (Proposed Standard), March 2006.

[38] S. Floyd, E. Kohler, and J. Padhye. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC). RFC 4342 (Proposed Standard), March 2006. Updated by RFCs 5348, 6323.

[39] S. Floyd and E. Kohler. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. RFC 4828 (Experimental), April 2007.

[40] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, January 1984.

[41] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.

[42] David S. Miller. Linux TCP Developments. `http://vger.kernel.org/~davem/TCP_LCA2006.odp`, 2006.

[43] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental), February 2003.

[44] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681 (Draft Standard), September 2009.