Datakommunikasjon Eksamens Notater

Datakommunikasjon (Universitetet i Agder)

Introduksjon

Vi har en modell som kalles for TCP/IP modellen. Den består av 5 såkalte lag som data sendt fra en applikasjon til mottaker må igjennom trinn for trinn før data' en kan mottas og benyttes.

Modellen består av følgende fag:

1. **Applikasjonslaget**
   ➔ Dette er laget brukeren/hosten befinner seg i. Data fra applikasjonen blir sendt som message.
2. **Transportlaget**
   ➔ Dette laget sørger for å transportere data til applikasjonen eller neste lag under ved å benytte seg av to transportprotokoller kalt UDP*(User datagram protocol)* og TCP *(transmission Control protocol)*. Det som skiller disse protokollene fra hverandre, er hvordan de håndterer overføring og pålitelighet. Datapakker fra dette laget blir sendt som et segment.
3. **Nettverkslaget**
   ➔ Dette laget befinner IP protokollen seg i. IP blir mye brukt når pakker oppgir adressen den blir sendt fra og adressen den skal til. Pakker sendt fra dette laget blir kalt for datagram.
4. **Ethernet**
   ➔ Dette laget kan relateres til rutere og dens adresser. Adresser her blir kalt for MAC adresser, dette er ikke det samme som IP adresse. Dette laget kan både lese IP og MAC adresser mens lag 3 kun leser IP adresser. Pakker sendt fra dette laget kalles for frame.
5. **Fysisk**
   ➔ Sender data ut som en bitstrøm.

Generelt kan man si at en protokoll er sett med regler som må følges når enheter kommuniserer med hverandre over nett.

3 Transportlaget

Transportlaget protokollen får dataene til å tro de er direkte koblet sammen. Vi har to typer transportprotokoller: UDP og TCP. Den største forskjellen på disse to er påliteligheten ved sending av data. TCP er den protokollen som har feilfri og pålitelig overføring, mens UDP kan gjøre feil uten engang å si ifra verken til mottaker eller til sender. TCP er **forbindelsesorientert**, det vil si at før selve overføringen skjer, har TCP og mottakeren opprettet en forbindelse/snakket sammen før overføring. På denne måten har sender og mottaker hele tiden kontroll på om alt kommer og om feil skulle oppstå. TCP ber om ny data om data skulle gå tapt på veien. Oppgaven til disse protokollene er å samle og distribuere data til/fra host/applikasjoner.

Tilleggsfunksjoner som kan være med i laget:

- Feilkontroll (gjelder UDP og TCP).
- Feilretting (kun TCP)
- Kvitteringer for å kommunisere om eventuell feil eller alt er bra (kun TCP)
- Tilpasse senderraten avhengig av nettkapasiteten (TCP).
- Tilpasse senderrate til mottakerkapasitet – sørge for at mottaker ikke blir overbelastet(TCP)

TCP trenger portnummer og IP adresse for å lage en unik forbindelse. Portnumre fra 0 til 1013 er «well known» som vil si faste porter.

3.2. Multiplexing og demultiplexing + litt UDP og TCP

Socket: Socketen befinner seg utenfor en prosess kjørt av applikasjonen. Man kan si at socket er en slags dør som er inngangen til applikasjonens prosess. Når en pakke sendes til en host, vil pakken inneholde en socket id bestående av source og destinasjon port + IP for å være unik. Hver socket inneholder sender/mottakers IP adresse, det er dette som sørger for at riktig data blir sendt til riktig host da det kan være flere socket hos mottaker. UDP og TCP har hver sine formater for å identifisere socket id. **Demultiplexing** er å levere riktig data i et transportlag segment to riktig socket. Mottakeren kan få mange data fra mange socket, og må se på feltet som bestemmer hvilket socket en skal sendes til. **Multiplexing** har med nettverkslaget og gjøre. Denne metoden går ut på å samle datapakker hos source hosten fra forskjellige socket, pakke dataene med header informasjon for å lage segmenter, og deretter sende segmentene til nettverkslaget.

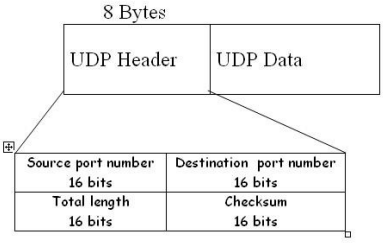
Et typisk transportlag segment består av source port nummer felt, destinasjon port nummer felt, other header fields, applikasjon/data/melding. Det er disse feltene som blir evaluert når den skal sendes til socket. UDP og TCP segmentene har i tillegg til disse også flere felter som er med på å indikere hvor pakken skal.

Multiplexing/demultiplexing med UDP og TCP: En UDP socket får automatisk et portnummer når socket blir opprettet. UDP socket er identifisert med destinasjon IP adresse, destinasjon port nummer. En TCP socket er identifisert med source IP adresse, source port nummer, destinasjon IP adresse og destinasjon port nummer.

Det som kan problematisk med UDP socket er at hvis de har forskjellige source port nummer eller source IP adresse, kan de allikevel sende til samme destinasjons port og socket vil da motta data fra to forskjellige sendere på samme socket. TCP socket løser dette problemet ved at **hver forbindelse har en unik socket** slik at det ikke oppstår slike situasjoner som med UDP socket. At segmenter med samme destinasjons port ikke havner i samme socket. På denne måten har TCP segmenter forskjellige source IP/port, noe som også vil være med på å hindre at de havner i samme sted. Når et TCP segment ankommer en host, blir alle de fire feltene brukt for å sende(demultiplexe) segmentet til tilhørende socket.

Litt om TCP og UDP protokollen

TCP: Ved TCP trenger man både portnummer og IP adresse. Når mottaker skal sende tilbake til samme avsender, har den også et portnummer og IP adresse. Dette gir en unik tilkobling. Som det har blitt nevnt, opprettes det en forbindelse mellom sender og mottaker før selve overføringen skjer. På denne måten får vi en sikker og feilfri overføring. TCP segment er mer omfattende og inneholder flere felter enn et UDP segment. Dette vil vi komme nærmere inn på litt senere.
UDP: Ved denne protokollen må applikasjonen selv ta ansvaret for feilretting eller om feil skulle oppstå. Vi har heller ingen relasjon til mottaker slik i TCP. UDP må selv legge inn en timer slik at applikasjonen spør hvis den ikke har fått svar innen en viss tid. UDP gir multiplexingsmulighet ved hjelp av portnummer og ha en enkel feilkontroll (ikke retting). UDP kan ikke garantere at dataen kommer frem til prosessen når data går inn i en UDP socket. UDP får applikasjonsdata via socket og legger til en **header** med source + destinasjon portnummer, feilkontroll og lengde indikator. **En UDP header består av følgende felt:**



3.4 Prinsipper for sikker overføring

Sender: Sender står klar-> sender->tar imot dataene og legger til en header (lager en pakke) og sender det -> så går den tilbake til den tilbake til den tilstanden var i begynnelsen.

Mottaker: Klar->mottar pakker->pakker ut dataene->overfør data til laget over->Klar.

Pakker som forsvinner: Det blir lagt til en timer som spør sendere hvis en viss tid har gått. Hver pakke har et sekvensnummer som sjekker om pakken kommer to ganger.

ACK: Positiv kvittering for mottatt pakke. **NACK:** Negativ kvittering for pakke tap/ikke mottatt pakke.

Pipeline protokoll: En pipeline protokoll gjør det mulig for sendere å sende flere pakker av gangen samtidig som mottaker kan kvittere for flere pakker av gangen. På denne måten slipper vi å måtte vente til en pakke er kvittert før neste pakke kan sendes. **Dette medfører følgende konsekvenser for sikker overføring protokoller**: Antall sekvensnumre må økes i og med at flere pakker vil bli sendt samtidig og må ha et unikt sekvensnummer. Sender og mottaker må bufre mer enn en pakke, sender må i hvert fall bufre pakker som har blitt sendt men ikke blitt kvittert for.

3.4.3 Go-Back-N(GBN)

GBN er en av to metoder som blir brukt ved pipeline error. Metoden går ut på at sender kan sende flere pakker om mulig uten å måtte vente på en kvittering for en pakke. Men GBN har en viss grense på hvor mange N ukvitterte pakker i pipelinen.

Sender: Sendt N påfølgende pakker før man får kvittering – Vent på kvittering for eldste pakker – når kvittering mottas, sendes neste pakke.

Svakhet: Er det en feil på en pakke, må alle sendes på nytt.

Mottaker: Vent på første pakke-ved mottak av pakke, kvitter hvis viktig – vent på neste.

3.4.4 Selective repeat

Denne protokollen sender som GBN. I motsetning til GBN så driver ikke SR å måtte re-transmittere alle pakkene i pipelinen på nytt om en pakke i pipelinen skulle ha feil. Denne protokollen unngår unødvendige re-transmisjoner ved at senderen re-transmitterer kun pakker som den mistenker har en feil eller mistet hos mottakeren. Dette krever at mottakeren kvitterer kun pakker som er korrekte.

Mottaker har også et vindu som sender. Pakker med feil sekvensnummer, aksepteres så lenge de er innenfor vinduet. Alle pakker kvitteres individuelt, og kvittering gjelder ikke foregående pakker. Mottaker ber spesifikt om re-transmisjon over en pakke.

3.5 Connection-Oriented Transport: TCP

At TCP er forbindelsesorientert vil si at før en applikasjons prosess kan begynne å sende data til en annen enhet, må de to først «handshake» med hverandre. Altså at de oppretter en forbindelse før selve transmitteringen skjer. Begge sider av overføringen vil initialere mange TCP variabler som vil bli presentert litt senere (3.7). TCP er en såkalt VC, Virtual Circuit. TCP protokollen kjøres bare i endesystemer. Rutere ser ikke TCP forbindelser, men datagram. TCP forbindelser sørger for en full **full-duplex service.** Det vil si at en forbindelse mellom to enheter, så kan data sendes fra begge ender på samme tid. TCP forbindelse er også **point-to-point**, det vil si at det er en single sender og en single mottaker. TCP kjører ikke multicasting som er en host som sender data til mange mottakere.

TCP er en **logisk forbindelse**: Det vil si at en TCP forbindelse etableres med tre meldinger eller **three-way handshake:**

1. Etableringsønske-> SYN (ønsker å etablere en forbindelse
2. Respons <- SYNACK (Kvitterer for forbindelsen)
3. Bekreftelse->ACK (Kan begynne å sende data)

Pakker som sendes inneholder et sekvensnummer for å oppdage eventuelle feil og holde styr på hvilke pakkes som sendes/mottas. Disse sekvensnumrene må bli bestemt før sending av data. Her er sekvenstellerne synkronisert og unik socket etablert og man er klar til å sende data.

Applikasjonen kan levere stor mengde data i TCP, hvor TCP deler opp data i segmenter. Typisk 1500-40 = 1460 byte. 40 bytene blir reservert av IP header og TCP header som hver tar 20 byte. TCP setter sammen nyttelasten fra segmentene på mottakersiden og leverer til applikasjon i riktig rekkefølge. Blant annet ved bruk av fragmenter som vil bli forklart litt senere.

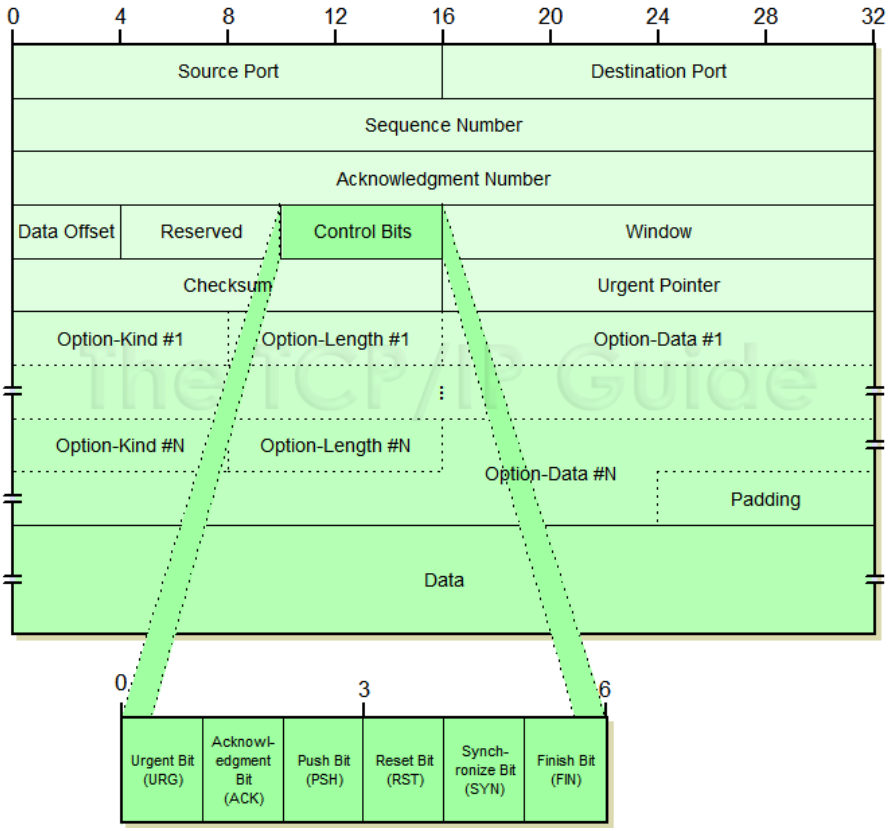
Sekvens og ACK nummer: TCP ser på data som en strøm av bytes. Sekvensnumrene teller byte og ikke segmenter. For eksempel hvis det sendes et segment med 30 bytes, økes sekvenstelleren med 30. TCP sender dataen inn i et sende buffer før det skal sendes. Det maksimale størrelsen data som kan puttes i et segment er begrenset av **maximum segment size.**

Når TCP skal sette sammen data, så følger det med en TCP header og blir formet til et segment. Segmentet blir deretter sendt ned til nettverkslaget hvor den deretter blir pakket inn som et IP datagram og sendt inn i nettet. Når TCP mottar segmentet, vil den bli lagt inn i mottaker bufferet og applikasjonen vil til slutt lese data strømmen fra bufferet.

3.5.2 TCP segment struktur

Et TCP segment består av flere felter enn et UDP. Dette fordi TCP sikrer feilfri overføring, og da må flere betingelser følge med det TCP segment for å sikre at dette opprettholdes.

TCP header er vanligvis på 20 bytes (12 bytes mer enn UDP sin)



- TCP header inneholder **source og destinasjons porter** slik som UPD header. Disse blir brukt til multi-/de-multiplexing av data fra/til høyere lag appl.
- TCP har også **sjekksum** felt akkurat som UDP header.
- 32 bits sekvensnummer og ACK feltet ble implementert for sikre sikker overføring.
- 16 bits **recieve window** blir brukt til flytkontroll. Dette feltet blir brukt for å indikere antall bytes som mottakeren er kapabel til å motta.
- 4 bits **header lengde** feltet spesifiserer lengden av TCP header i 32-bits ord.
- **Options feltet** er vanligvis tomt, så TCP header er typisk 20 bytes.
- **Control bits eller flag field** består av 6 bits. Blant annet brukes ACK flagget til å indikere om ACK er gyldig eller ikke. RST, FIN og SYN blir brukt til oppkobling av forbindelser. PSH indikerer at mottakeren skal sende data opp til høyre lag umiddelbart. URG indikerer om data er urgent.

3.5.3 Round-Trip Estimation and Timeout

RTT er tiden fra et segment blir sent til den blir kvittert igjen. **Sample RTT** er tiden det tar fra et segment blir sent (sendt til IP) og får en kvittering for mottatt data.

EstimatedRTT regnes ut på følgende måte:

EstimatedRTT = 0.875*OldEstimated RTT (ACK mottatt-tid sendt) *0.125*SampleRTT (den vi er på).

Så første gangen vil EstimatedRTT kun være ACK mottatt – tidspunkt sendt

Andre gangen vil EstimatedRTT = 0.875*RTTOld (den forrige sin RTT) +0.125*SampleRTT (Den vi er på)

3.5.4 Sikker overføring av data + PFtransmit

Nettverkslaget og IP har ikke sikker overføring. IP garanterer ikke at datagrammet blir levert og garanterer ikke at dataen kommer i den rekkefølgen den ble sendt. TCP sørger for at dette ikke oppstår. TCP sørger for at buffer ikke blir overbelastet, hindre duplikasjon og at dataen kommer inn den samme rekkefølgen som de ble sendt. TCP mottar data fra applikasjonen, pakker den inn som et segment og sender den videre til IP som nevnt tidligere. Hvert segment inneholder et sekvensnummer som er byte strøm nummeret til det første data byten i segmentet.

TCP starter sin timer når segmentet har blitt passert til IP. TCP sin timer responderer ved å re-transmittere segmentet som forårsaket timeouten, for så å restarte timeren igjen. Noen eksempler ved bruk av TCP timer:

Situasjon 1: Host A sender et segment med sekvensnummer 92 med en payload på 8 byte. Host B svarer med ACK, men den går tapt på veien. Host A venter til timeren blir trigget og sender det samme segmentet på nytt.

Situasjon 2: Host A sender et segment med sekvensnummer 92 og payload på 8. Host B sender ACK men forsvinner på veien. Samtidig har Host A sendt et nytt segment med sekvensnummer 100 og payload på 20 bytes, host B svarer med ACK og den kommer frem. **Da vet host A at host B har mottatt all data opp til og med byte 119, og trenger ikke å sende segment 1 og segment 2 på nytt.**

Etter hvert som timeren blir trigget gjentatte ganger, vil timer perioden øke og vi kan til slutt ende opp med at timeren bruker veldig lang tid før den trigges. På denne måten blir det veldig mye «dødtid» fordi ingenting skjer fra da senderen sender til den venter på kvittering eller at timeren skal trigges. **Fast retransmit** er en metode som sørger for mer effektivitet slik at det kan skje andre ting samtidig som senderen venter på en kvittering.

Fast retransmit går ut prinsippet «triple ACK». Denne går ut på at om sendere skulle miste data på veien, i stedet for å vente på en eventuell lang timer tid, så trenger mottakeren kun å sende **tre** kvitteringer om at samme segment ikke har kommer frem. Da vil senderen re-transmittere den savnende pakken *FØR timeren har utløpt*. På denne måten sparer vi tid og øker effektiviteten da senderen kan sende pakker fortløpende.

3.5.5 Flow Control

Flytkontroll sørger for at mottaker bufferet hele tiden ikke blir overbelastet. Flytkontroll er en hastighets tilpassende mekanisme som matcher senderaten med raten mottakeren klarer å lese data. TCP' s flytkontroll sørger for at senderen holder på en variabel kalt **recieve window (rwnd). Rwnd** blir brukt for gi senderen en pekepinn om hvor mye ledig plass det er i bufferet hos mottakeren. **Rwnd** befinner seg hos mottakeren mens senderen har **sendevindu** som alltid skal være mindre enn rwnd. Sendere får info om rwnd gjennom kvittering fra mottakeren, der den vil indikere om hvor mye ledig plass det igjen. Hvis mottaker bufferet skulle bli fult, kan ikke mottakeren si ifra når bufferet tømmes da den ikke har noen pakker å kvittere for på daværende tidspunkt. Dette problemet løses ved at senderen sender segmenter på 1 byte for hele tiden sjekke om det er ledig i mottakerens buffer.

3.6 Congestion Control (metningskontroll)

Hovedprinsippet med metningskontroll er å regulerer trafikken inn i nettet slik at dette ikke blir overbelastet. Rutere i nettet har også buffere som eventuelt kan bli overbelastet. Metningskontroll skal øke eller senke hastigheten i nettet avhengig av opphopning i trafikken. TCP har ende-ende metningskontroll, det vil si at avsender tolker nettilstanden ut ifra kvitteringer fra mottaker (tap, forsinkelse). ATM har en nettassistert metningskontroll, det vil si at rutere melder sin tilstand til endepunktene.

3.6.3 Nettassistert metningskontroll med ATM ABR

Som tidligere nevnt er ATM ABR en protokoll som er nettassistert metningskontroll. Som vil si at det er rutere som hele tiden melder sin tilstand ved eventuelle forsinkelse, overbelastning osv. ATM er VC orientert, det vil si at en pakke følger en bestemt rute gjennom bestemte svitsjer som holder på tilstanden til overføringen. VC tillater svitsjer å finne forbindelsesinformasjonen til hver enkelt sending som for eksempel senderate. ATM ABR fungerer på den måten at data er dataceller som går igjennom svitsjer på fra start til destinasjon. Mellom de datacellene, kan det befinne seg en annen type celle kalt **resource-managemnt cells (RM cells).** RM celler blir brukt til å gi informasjon til hoster og svitsjer fortløpende. Når RM celler ankommer svitsjer, så kan svitsjene gi RM cellene informasjon om svitsjens tilstand og sende den tilbake til sender.

3.7 TCP Congestion Control

TCP metningskontroll har tre funksjoner: slow start- congestion avoidance-fast recovery. Senderen har variablene cwnd som er congestion window. Sendervindu er den minste av recieve window og cwnd. TCP merker at det er opphopning mellom sender og mottaker ved for eksempel timeout eller trippel – ACK. Cwnd økes for hver positive kvittering. Dette kan tyde på at ting går bra og ingen feil i overføring. Vi sier TCP er self-clocking. TCP må bruke ende-ende metningskontroll enn nettassistert fordi IP laget ikke tilbyr tilbakemelding to endesystemer ved nettopphopning. TCP sin metningskontroll gjør det mulig for senderen å tilpasse sin senderate til mottaker avhengig om mottakeren er overbelastet eller begynne å bli overbelastet. Cwnd angir hvor mye TCP kan sende inn i nettet.

TCP congestion-Control algorithm

Den algoritmen har tre hovedkomponenter: slow-start, congestion avoidance, fast recovery. **Slow-start:** Cwnd starter med 1MSS (1460 byte) i begynnelsen. For hver mottatt ACK økes cwnd med 1 MMS. Vi får da en eksponentiell økning. Ved pakketap/timeout, settes cwnd til 1, og prosessen starter på nytt igjen. Samtidig blir variabelen ssthreshold satt til å være cwnd/2. Når verdien av cwnd er lik ssthreshold sin verdi, vil cwnd øke lineært til timeout eller trippel ACK oppstår. Vi går da fra **slow-start til congestion avoidance**. Hvis trippel ACK trigges, vil slow-start utføre **fast retransmit og gå i fast recovery modus. Congestion avoidance:** I denne tilstanden økes cwnd mer varsomt og forsiktig enn i slow-start. Verdien til cwnd er nesten halvert siden sist gang congestion ble behandlet. Når vil cwnd ikke øke eksponentielt men lineært. Cwnd økes med 1MSS for hver mottatte kvittering. Denne lineære økningen går helt opp til ssthreshold eller timeout eller trippel ACK. Hvis timeout oppstår så vil cwnd bli satt til 1 MSS og ssthreshold blir halvparten av cwnd ved pakketap (trippel ACK) - > fast recovery modus. **Fast recovery:** I fast recovery blir verdien til cwnd økt med 1 MSS for hver duplikat ACK mottatt for den savnede segmentet som forårsaket TCP til å i denne modusen. Hvis en ACK til slutt kommer frem for det savnede segmentet, går TCP i congestion avoidance. Hvis en timeout oppstår, går fast recovery til slow-start. Cwnd blir satt til 1 MSS og ssthreshold blir cwnd/2.

Fast recovery har to versjoner, TCP Tahoe og TCP Reno. TCP Tahoe setter cwnd til 1 MSS og går i slow-start ved timeout eller trippel ACK, vokser eksponentielt til den når ssthreshold og vokser lineært etter det. TCP Reno setter cwnd lik cwnd/2 og vokser deretter lineært ved trippel ACK.

TCP Reno gjør det mulig å ha høy senderrate om pakketap/timeout skulle oppstå ved at den setter cwnd lik cwnd/2 i stedet for å sette vinduet lik 1 MSS.

3.7.1 TCP rettferdighet

UDP rettferdighet:

- Ved UDP så unngår man at overføringshastigheten blir forsinket selv om nettet er overbelastet.
- UDP har ikke congestion Control. Applikasjonene kan sende lyd og video over nett med konstant hastighet og tillate og miste pakker i ny og ne i stedet for å senke hastigheten for å oppnå rettferdig slik som i TCP.

Oppsummering av kapittel 3

TCP tilbyr sikker overføring av data, forsinkelses-garanti og båndbredde garanti til applikasjonen. UDP derimot tilbyr ikke sikker overføring slik som i TCP. Begge transport protokollene har sine fordeler og ulemper. TCP passer ypperlig ved overføring av viktig data der ingen data kan gå tapt. UDP derimot passer bra til blant annet streaming, der pakketap ikke har så mye å si.

En pipeline er en protokoll som gjør det mulig å sende mange segmenter uten å få kvittering. På denne måten unngår vi stop and wait prosedyren, men at det hele tiden holdes i gang. En ikke-pipeline protokoll gjør det motsatte av pipeline. Her må sendere vente på kvittering på sendte segment før ny sending kan ta sted.

Sendervinduet er den minste av cwnd (congestion control) og recieve window (flow control). Grunnen til at sendervinduet er den minste er for å sikre at senderen ikke sender mer data enn det mottakeren klarer å ta imot.

Ved sending av data mellom to hoster, så vil hver pakke inneholde et sekvensnummer. Sekvensnummeret blir brukt for å sjekke om pakken har ankommet eller ikke. Nummeret er byte orientert. Mottaker kvitterer med neste forventede sekvensnummer (byte)

Timeout blir brukt hvis en kvittering for en pakke ikke blir sendt innen et bestemt tidsrom. Dette kan for eksempel indikere at en pakken aldri kom eller forsvant på veien tilbake. Andre ting kan være overbelastning i nettet og mottakeren rekker ikke å sende innen timeouten trigges. Timeout tiden beregnes fortløpende fra sendingen sendes til den mottas, basert på måling av RTT.

En mer effektiv timeout metode er trippel ACK. Den trigger en mekanisme kalt fast retransmit hvis mottakeren kvitterer for samme segment tre ganger. På denne måten slipper vi å måtte vente helt timer tiden går ut, og flere segmenter kan sendes av gangen også.

Flow Control er en metode brukt for å regulerer trafikken for å hindre at mottaker bufferet ikke overbelastes. Mottakeren har en rwnd som viser hvor mye ledig plass det er i bufferet. Når mottaker kvitterer benytter den feltet «rwnd» i TCP header som viser hvor mye ledig plass det er.

Metningskontroll er også en mekanisme for å regulerer trafikken, men nå er det i nettet. Basert på enten nettassistert metningskontroll der ruteren har som jobb å melde sin tilstand for regulering av trafikk. Den andre metoden er ende-ende prinsippet der nettilstanden blir tolket utifra kvitteringer fra mottaker (pakketap, timer utløpt). Metningskontroll for TCP har tre mekanismer: slow-start som sørger for rask oppstart ved eksponentiell økning av cwnd, congestion-avoidance som sørger for optimal utnyttelse av nettkapasitet ved lineær vekst av cwnd, og fast recovery som gjenoppretter en høy senderate dersom trippel ACK eller pakke tap skulle oppstå. Det som trigger «slow start» er timer utløp. Random early detection (RED) gir mykere regulering av senderate ved at rutere kaster tilfeldige segmenter ved for høy bufferfylling.

En TCP forbindelse opprettes ved en trippel handshake ->SYN->SYNACK->ACK. For å avslutte en TCP forbindelse benyttes variablene FIN og ACK i begge retninger.

Kapittel 4 The Network Layer (Nettverkslaget)

Videresending (forwarding): Overføring av en pakke fra innkommende link til en utgående link innenfor én ruter. **Ruting (routing):** Involverer videresending av pakke mellom alle ruterne i nettverket. Kapittelet vil også ta for seg nettverks adresse oversetting (NAT), datagram fragmenter, IMCP, IPv4 og IPv6, ruting protokoller.

4.1 (tar for seg nettverkslagets funksjoner og tjeneste)

Hvis en host A skal sende data til host B: så tar først host 1 segmentet fra transportlaget og pakker den inn som et datagram (nettverkslagets datatype), og sender datagrammet til nærmeste ruter. Når host B mottar datagrammet fra dens nærmeste ruter, vil host B pakke ut datagrammet til et transportlag segment, og sende den videre opp til transportlaget. Hovedrollen til rutere er å videresende datagram from input link til output link. **Rutere kjører verken applikasjons- eller transport-lag protokoller, kun nettverkslag protokoller.** Rutere vet ikke om det er TCP eller UDP, den bare transporterer datagrammet. IP sørger for at data som kommer inn på en link, sendes ut til riktig link.

4.1.1 Forwarding and Routing

Rollen til nettverkslaget er enkelt og greit å flytte pakker fra sender host til mottaker host ved hjelp av to funksjoner: forwarding og routing.

Forwarding (videresending): Flytte segmenter (IP datagram) fra en innkommende link til riktig utgående link slik at segmentet kommer fram. Dette forutsetter at det fins en tabell slik at den kan utføre sendejobben. Routing (ruting): Nettverkslaget må finne ut hvor segmenter (IP datagram) skal sendes (ruten, veien) for å nå riktig mottaker. Algoritmen som kalkulerer disse veiene for pakken blir kalt for ruting algoritmer, som for eksempel ruten fra sender A til mottaker B. Routing har to hovedoppgaver: kartlegge nettet, bygge og vedlikeholde rutingtabeller. **Forwarding tabell (FT):** Hver ruter har en FT. Ruteren videresender en pakke ved å se på verdien i den innkommende pakkens header og sammenligner verdien med tabellen for å finne riktig output link. Ruteren mottar ruting protokoll meldinger som blir brukt for å konfigurere FT. **Pakke svitsj:** Generell pakke-svitsjing enhet som overfører a pakke fra input link interface til output link interface avhengig av verdien i headeren. Noen pakke-svitsjer kalt link-lags svitsjer baserer videresendingen på verdien i link-lags rammen. Disse blir kalt for lag 2 enheter. Andre pakke-svitsjer kalt rutere, baserer videresendingen på verdien i nettverks-lags feltet. Rutere er da lag 3 enheter, men må også i tillegg implementere lag 2 protokoller siden lag 3 krever tjenestene til lag 2 for å implementere deres funksjonalitet (lag 3). For eksempel kjenner ikke lag 2 (Link-lag) igjen IP adresse, men lag 3 (nettverks-lag) kan kjenne igjen IP adresser.

4.1.2 Network Service Models

Nettverks-laget tilbyr blant annet følgende tjenester:

- Garantert levering: Pakken vil nå fram til slutt
- Garantert levering med begrenset forsinkelse.

For en «flow» av pakker mellom source og destinasjon, tilbys følgende tjenster:

- Pakkene blir levert i riktig rekkefølge
- Garantert minimum båndbredde
- Garantert Maximum variasjon i forsinkelse (jitter)
- Sikkerhetsmekanismer: Autentisering, kryptering, dataintegritet.

ATM dekker de fleste av disse tjenestene. Nettverks-laget dekker ingen av tjenestene, men utfører «best effort», altså ikke garantert levering.

4.2 Virtual Circuit og Datagram Nettverk:

Forbindelsesløs: Send et datagram og glem at den er sendt (følger ikke samme vei tilbake). For eksempel datagram nettverk. **Forbindelsesorientert:** Etablerer en forbindelse (virtuell) før den sender data (følger samme vei). Ved hjelp av «handshake» mellom terminaler og alle mellomliggende rutere. Alle pakker tilhørende en forbindelse, får en merkelapp som brukes for å videresende i ruteren. For eksempel virtual circuit nettverk.

Disse to typene nettverk bruker forskjellig informasjon for å bestemme videresendingen.

4.2.1 Virtual Circuit Networks

VC består tre ting: (1) en path som er en serie av linker og rutere mellom source og destinasjon. (2) Et VC nummer som en pakke får tildelt for hver link langs ruten, (2) og plasser i videresendingstabellen i hver ruter langs ruten. En pakke i VC har et VC nummer i dens header, og får tildelt et nytt VC nummer på hver link fra videresendingstabellen. Når en ny VC blir etablert over en ruter, blir oppføringen lagt til i videresendingstabellen, tabellen må også slette VC ved sletting.

Eksempel med VC:

Incoming Interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	2	22
2	63	1	18
3	7	2	17

Grunnen til at pakker ikke beholder samme VC nummer hele veien er: (1) Redusere lengden av VC i pakke headeren. (2) Unngå duplikat, derfor velger hver link en tilfeldig VC. I et VC nettverk må ruterne opprettholde **connection state information.** Det vil si at for hver nye forbindelse som opprettes langs en ruter, må en ny forbindelses entry bli lagt til i ruteren videresendingstabell. Og samme hvis en forbindelse blir brutt, da må entry' en bli fjernet. Det er tre faser i VC: **VC setup:** Transportlaget kontakter nettverkslaget og spesifiserer mottaker adresse og venter på at nettverket skal sette opp VC. Nettverkslaget vil da bestemme ruten/veien mellom sender og mottaker, bestemmer i tillegg VC nummeret for hver link. Nettverkslaget legger også til en entry i videresendingstabellen i hver ruter langs veien. **Data transfer:** Når VC har blitt opprettet, kan pakker begynne å overføres **VC teardown:** Initieres når sender/mottaker sier ifra til nettverkslaget om at den vil terminere VC. Nettverkslaget vil da informere endesystemet på den andre siden og i tillegg oppdatere videresendingstabellen i ruterne pakken har vært innom. Initiate call->incoming call<-accept call<-call connected->begin dataflow->receive data

4.2.2 Datagram nettverk

I et datagram nettverk, hver gang et endesystem skal sende en pakke, så legger den til destinasjons adresse og sender den uti nettet.

Det er ingen VC og rutere trenger derfor ikke ha noe status informasjon. Rutere bruker bare pakkens destinasjons adresse for å videresende. Hver ruter har en forward tabell (FT) som «mapper» destinasjons adressen til riktig link interface. Alle pakker (datagram) som sendes på ha komplett adresse til mottakeren. I ruterne må i prinsippet alle adresser være oppført. I IPv4 er adressen 32 bit, ca. 4 millioner adresser.

Adresser må slåes sammen for å få enklere tabell: Et eksempel på hvordan en pakke velger riktig link:

Vi har en ruter bestående av tre link interfaces, alle pakker som starter med adresse xxxx går til matchende linken.

Interface	Adresse (Alle som starter med)
0	1100 1000 0001 0111 00010
1	1100 1000 0001 0111 0001 1000
2	1100 1000 0001 0111 00011

Hvis vi for eksempel har en pakke med destinasjonsadresse lik:

1100 1000 0001 0111 0001 0110 1010 0001 – Vi ser her at link 0 passer best, pakken blir sendt ut denne linken mot destinasjonen.

Men hva skjer hvis destinasjonsadressen matcher flere link interface? Jo vi gjør følgende:

1100 1000 0001 0111 0001 1000 1010 1010

1100 1000 0001 0111 0001 1000 1010 1010

Link interface 1 har 24 bit som matcher, link 2 har bare 21 bit som matcher. Da bruker vi en metode kalt **«longest match prefix»** som velger den som har mest matchende antall bit

Fordeler ulemper ved DN og VC

Fordeler ved DN: Følger ikke en bestemt rute, hvis ruteren er full eller linjen er ødelagt kan man bare finne en ny rute. Dette gjør det mer fleksibelt.

Ulemper DN: Forsinkelsen er uforutsigbar. DN må bruke TCP da pakkene kan komme i forskjellige rekkefølge fordi de kan velge forskjellige ruter. Sist og ikke minst må rutere ha større rutingtabeller, fordi de må vite neste steg + hvor den skal.

Fordeler VC: Pakkene kommer i riktig rekkefølge og alle har samme forsinkelse. Går raskere for switchen og videresende pakken fordi vi kan ha mindre tabeller. Trenger bare å se på hvor neste steg da pakken kun følger en bestemt rute.

Ulemper VC: Hvis en ruter får problemer, må en ny linje etableres, noe som krever en del. Bufferene til ruterne kan bli fulle fordi pakker fra forskjellig hoster kan få samme rute, da vil en stor forsinkelse oppstå og en ny linje må eventuelt etableres

4.3 What's Inside a Router (forward function)

En ruter inneholder en svitsjeenhet som tar imot innkommende datagram og sender til riktig output link ved hjelp av rutingtabell og datagram-adresse. **Svtisjen** kobler sammen ruterens input port og dens output port. Input porten består av tre deler: -

1	2	3	Svitsjeenhet

>

1: Overgang fra fysisk til databit (fysisk lag, radio, kobberkabel, fiberkabel).

2: Henter ut datainnhold og er typisk et IP datagram (Linklaget)

3: Innbuffer som har to oppgaver. (1) Legge IP datagram i innkøen til svitsjen. (2) Slå opp i rutingtabellen for å finne ut hvordan svitsjenheten skal styres og sendes ut på riktig output port.

1100 1000 0001 0111 0001 1000 1010 1010

Ruting prosessor er koblet til svitsjeenheten og kjører ruting protokollene (4.6), vedlikeholder rutingtabeller og beregner rutingtabellene. **Dette skjer i input port del 3.**

Svitsjing kan skje på flere metoder:

- Via minne, kopiere innhold fra et område til et annet ved hjelp av databuss.
- Via egen svitsjebuss (felles dedikert buss mellom inn og ut buffer).
- Via svitsje matrise (flere datagram kan sendes samtidig).

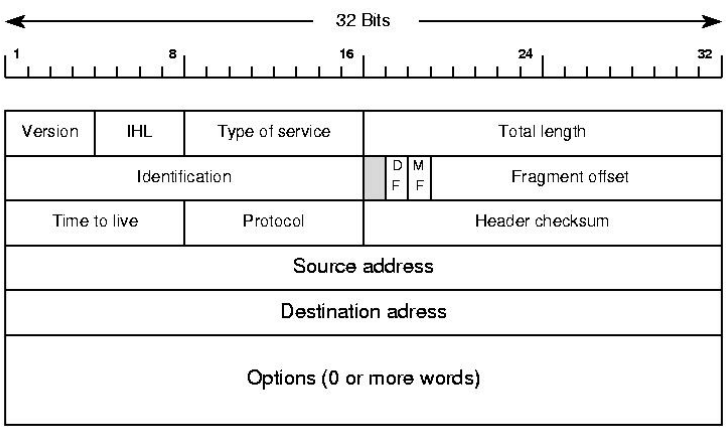
Hvorfor må data bufres når den ankommer/skal sendes ut?

- Det må bufres når det er kø ved inngangen.
- Fordi det kommer flere pakker inn samtidig, kun en kan behandles av gangen gjennom svitsjebussen. Hvis bufferet blir fullt kan data bli kastet

4.4 IP protokollen: Forwarding and Addressing in the Internet

Nå skal vi ta for oss hvordan adressering og ruting skjer i nettet. Hovedkomponentene innenfor nettverkslaget er: (1) Ruting protokoller (veivalg, RIP, OSPF, BGP) som er koblet til ruting tabeller. (2) IP protokoll (adressering, datagram format, behandling av pakker) og (3) IMCP protokoll (error reporting i datagram)

4.4.1 Datagram Format og fragmentering



Version: IPv4 eller IPv6. **Header length:** Normalt 5 bit. Evt. Ekstra headerelement som er med angis her. **Type of service:** Gir mulighet for prioritet. **Datagram (total) length:** 16 bit + header. **Identifier/flags/fragment offset:** Oppdeling av datagram I mindre deler (fragmenter).

Time-to-live: Tall som telles ned 1 for hver ruter den møter. Dette for å hindre datagram i å vandre i nettet i evighet.

Upper layer protokoll: Kode for hvilken høyere lags protokoll lasten skal leveres til. **Header checksum:** Som for UDP/TCP, men bare for header. For å sjekke biterror i et mottatt IP datagram.

Data (payload, under options): Transporterer UDP/TCP segment (lag 3). Kan også transportere andre typer data som ICMP meldinger.

IP datagram fragmentering

Det maksimale mengde data en ethernet-frame kan frakte er 1500 bytes. Det maksimale mengde data en link-lag frame kan bære blir kalt for MTU, Maximum transmission unit. Så hvis et IP datagram pakkes ned til en Ethernet frame og skal sendes ut til nettet, men er for stor. Må dataen deles opp i fragmenter. IP datagrammet blir delt opp i mindre deler før den pakkes inn som en Ethernet frame.

Har vi for eksempel et IP datagram på 4000 bytes og skal sende ut på en link som maks klarer å håndtere 1500 byte, må vi dele opp datagrammet i mindre deler. Vi deler opp datagrammet i tre mindre deler: Vi må huske å trekke fra payloaden med 20 da dette er header data.

20	1480 (1)	20	1480 (2)	20	1020 (3)

Flag (bit)	Fragment 1, 2, 3	Offset
1	ID = 777	0 (start posisjon)
1	ID = 777	1480/8 = 185 (startposisjon)
1	ID = 777	3980-1480/8 = 370

Flag sier noe om det kommer flere fragmenter eller ikke. Er den satt til 1 betyr det at det kommer flere, er den 0 betyr det at det er siste fragment. **Dettet flag bit finner vi i IP datagrammet.**

Vi deler på 8 fordi vi skal ha enheter på 8 byte. **Fragmenter** som hører sammen får en ID som er identisk.

Når alle fragmentene har kommet destinasjon, settes fragmentene sammen, sendes opp til transportlaget. Hvis en eller flere av fragmentene ikke kommer, blir resten av datagrammet kastet. Men er det en TCP tilkobling, vil det kun sendes den savnede fragmentet.

4.4.2 Ipv4 Addressing

Interface: The boundary between the host and the physical link.

En ruters jobb er å motta datagram fra en link og videre sende den tile n annen link, så en router har som oftest to eller flere linker som er koblet til, grensen mellom en ruter og en link er også et interface. IP krever at hver host og ruter interface har en egen IP adresse. En IP adresse er egentlig assosiert med et interface enn en ruter. Hver IP adresse er 32 bit lang (4 byte). Hvert interface på hver host eller ruter i det globale internettet må ha en IP adresse som er globalt unik.

Et nettverk som kobler sammen tre host interface og en ruter former et subnet. IP adresserer en adresse til subnettet.

223.1.1.4 = 1100 1111.0000 0001.0000 0001. 0000 0100 .0000 0000

Rød – blå –grønn = nettdel av adressen. Svart er host del av adressen.

200.23.16.0/23 = 1100 1000. 0001 0111. 0001 0000. 00000000

200.23.16.0/23 = 1100 1000. 0001 0111. 0001 0000. 00000000

----------R1-LINK 1----------

200.23.18.0/23 = 1100 1000. 0001 0111. 0001 0010. 00000000

200.23.18.0/23 = 1100 1000. 0001 0111. 0001 0010. 00000000

----------R1-LINK 2----------

200.23.20.0/23 = 1100 1000. 0001 0111. 0001 0100. 00000000

200.23.20.0/23 = 1100 1000. 0001 0111. 0001 0100. 00000000

----------R1-LINK 3----------

200.23.22.0/23 = 1100 1000. 0001 0111. 0001 0110. 00000000

200.23.22.0/23 = 1100 1000. 0001 0111. 0001 0110. 00000000

----------R2-LINK 4----------

/23 Sier hvor mange binære det skal tas hensyn til, kalles for en nettmaske.

Eksempel: Sende alle datagram med nettadresse: 200.23.16.0/21 til R1, 200.23.18.0/23 til R2

Vi har en pakke med IP adresse: 200.23.0001 0011

200.23.0001 0011 – sendes til R1

200.23.0001 0011 – sendes til R2

Her blir longest match brukt

Spesielle IP – adresser:

0.0.0/8	«Current network», bare en source
10.0.0.0/8	«Private network», kun gyldig i en bestemt ruter (NAT)
172.16.9.9/12 192.168.00/16	«Private network», NAT
224.0.0.0/4	«Multicast»
255.255.255.255.255	«Broadcast», sender til alle i subnettet.

Interface mellom ruter og hoster har en subnett adresse og dette former et subnet

DHCP (Dynamic Host Config Protocol)

DHCP tillater hoster å få IP adresse automatisk. Nettverksadministrator kan konfigurere DHCP slik at hoster får samme IP hver gang den kobler seg til samme nett eller få en midlertidig IP som er forskjellig hver gang. Når en hoster får en IP adresse, har adressen en bestemt levetid.

På grunn av DHCP's mulighet å koble hoster inn i nettet, blir ofte DHCP kalt for **plug-and-play protocol.** DHCP serveren finnes ofte i subnet. Når en host vil koble seg til nettet, og kontakter DHCP for å få IP adresse skjer følgende 4 steg:

Steg 1 DHCP server Discovery: Hosten prøver å finne en DHCP server å samhandle med. Dette gjøres ved å sende en DHCP Discover Message som blir sendt som en UDP pakket til port 67. UDP pakken blir pakket ned til et IP datagram. Hosten vet ikke hvem datagrammet skal sendes til, så han broadcaster ut i nettet. Så IP datagrammet har source IP: 0.0.0.0, destinasjon IP: 255.255.255.255 og destinasjon portnummer 67. Ved broadcasting blir IP datagrammet sendt til link-laget som broadcaster rammen til alle rutere som er koblet til subnettet.

Steg 2 DHCP server offer(s): Når en DHCP server mottar server discovery meldingen, så broadcaster den til alle inkludert hosten. Men nå har serveren foreslått en IP adresse + TTL til hosten og en transaction ID.

Steg 3 DHCP request: Hvis hosten er fornøyd med forslaget, så sender den en DHCP request og vil da få et svar om den får IP adressen eller ikke.

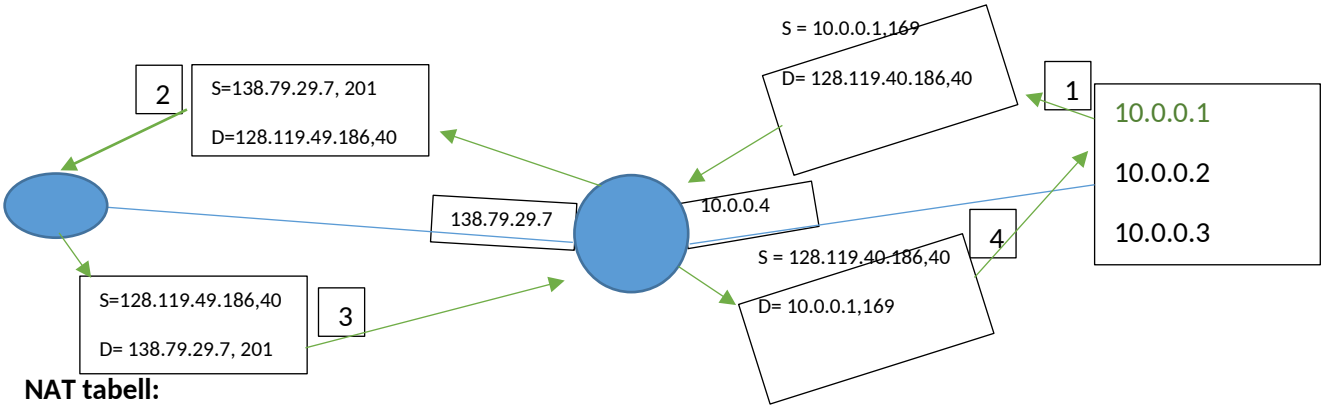
Steg 4 DHCP ACK: Serveren mottar request meldingen og godtar forslaget ved å sende en ACK tilbake til hosten.

NAT (Network Address Translation)

NAT er en metode for å gjenbruke IP adresser. Den benytter lokale IP adresser som f.eks. 10.0.0.0/8. NAT brukes blant annet i hjemmenettverk hvis en enhet i det private nettet vil ha tak i noe ute i den store verden. NAT kan generere hvilken som helst source port nummer som ikke er i tabellen.

En NAT ruter har en interface til hjemmenettverket og et interface ut til det globale nettverket. En host med en privat adresse som skal ha tak i noe i det globale nettet, kan ikke bruke den private adressen da den kun kan brukes i private nettverk som for eksempel hjemmenettet. For å kunne kommunisere med verden fra et hjemmenettverk, trengs en NAT ruter.

Det globale nettet ser på NAT ruteren som en enhet med en IP adresse, og ikke en ruter. Så alle enheter i det private nettverket som vil ha tak i noe i det globale, vil gå igjennom NAT ruteren. Alle disse enhetene vil da få samme source adresse. I NAT ruteren finnes det en NAT tabell som mapper det private IP adressen til en global IP adresse og omvendt. Det som spesifiserer hvem data skal til i det private nettverket er portnummeret. La oss ta for oss et eksempel



NAT tabell:

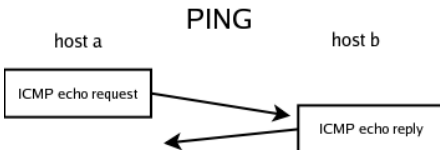
WAN side	LAN side
138.79.29.7,201 (portnummer)	10.0.0.1, 169 (portnummer).

1. Host med adresse 10.0.0.1 sender forespørsel om en webside på webserver med port 40, destinasjon IP adresse 128.119.40.186,49, source IP adresse 10.0.0.1 og source port 169.
2. Når datagrammet ankommer NAT ruteren, slår den opp i tabellen. Datagrammet får WAN adresse som ny source IP og et nytt portnummer. Sendes ut i det globale nettet.
3. Webserveren svarer, sender data tilbake. Destinasjons adressen er nå source og port nummer til NAT ruteren.
4. Når datagrammet ankommer NAT ruteren. Slår den opp i NAT tabellen, sjekker hva source adressen tilsvarer på LAN side (10.0.0.1, 169). Sender videre til riktig host i det private nettverket.

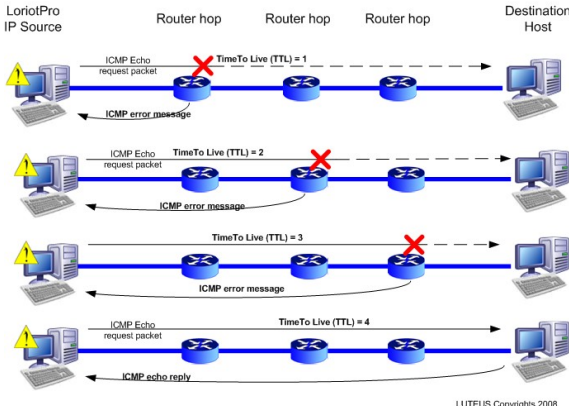
4.4.3 ICMP (Internet Control message Protocol)

ICMP er en protokoll som mottar og håndterer feilmeldinger. ICMP ligger retter over IP. ICMP finnes i IP payload, akkurat som TCP/UDP segment i IP payload. ICMP har en «type» og en «code» felt. Den fins i header og dekker de 8 første bytene av et IP datagram om ICMP skulle oppstå. Eksempler på feilmeldinger:

ICMP type	Code	Beskrivelse
0	0	Echo reply (to ping)
3	0	Destination Network unreachable
3	1	Destination host unreachable
3	2	----------‘’--- protocol----‘’---------
3	3	--------‘’-----port-------‘’---------
8	0	Echo request
11	0	TTL expired



Traceroute program: Mulighet fir å spore en rute fra host til en annen host i verden.



4.4.4 IPv6 (32 bit i lengden)

Version (6bit): Hvilken versjon det er (IPv6 nå)

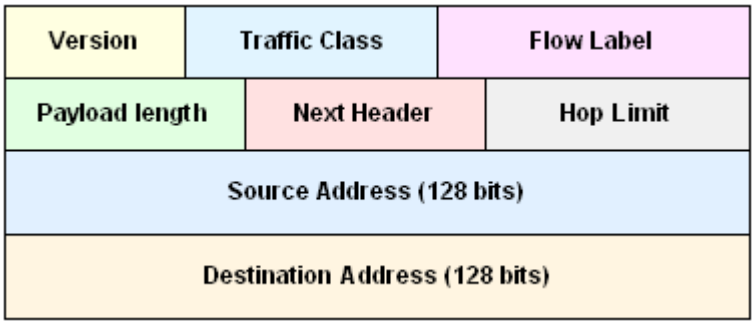
Traffic class (8bit): Tilsvarer TOS i IPv4

Flowlabel (20bit): Identifikator for datagram som hører sammen. Kan også være prioritet.

Payload length (16bit): Antall byte i datainnhold

Next header: Angir neste header enten i IPv6 header eller i nyttelasten.

Hop limit: Tilsvarer TTL i IPv4



Felt IPv4 har som ikke IPv6 har:

- Fragmentation/reassembly i rutere. Kun ved source/destinasjon. Er datagrammet mottatt fra ruter for stor til å bli videresendt til link, forkast og ICMP melding.
- Header checksum: Kostet mye da HC måtte beregnes ved hver ruter, ved f.eks. fragmentasjon.
- Options: gir 40 byte IP header.

4.5 Routing Algorithms

- Rutingalgoritme: Skal finne beste vei mellom to rutere (eller nett). Rutingtabellene skal inneholde neste ruter langs beste vei til alle andre rutere.
- Ruting algoritmene kan være globale (sentralisert) eller desentralisert
  - ➔ Globale baseres på totalt nettbilde (link - state algoritmen), altså at man kjenner hele nettverket. Deretter regner ut korteste vei mellom source og destinasjon.
  - ➔ Desentraliserte er iterative (tar tid) og trenger ikke totalt nettbilde. Hver ruter får informasjon om sin nabo og finner costen mellom de.
- Distance vector algoritmen er slik: (utveksle innhold i ruting tabeller med nabo). Desentralisert.
- Link – state algoritmen: En algoritme som baseres på kjennskap av hele nettbildet. Hver node broadcaster link – state pakke til alle noder i nettet, hver pakke inneholder ID og costen. Sentralisert.

4.5.3 Hierarkisk Ruting

Rutere kan organiseres i **autonome systemer (AS)**, der hver AS består av en gruppe rutere som er under samme administrative kontroll. Rutere innen samme AS kjører samme ruting algoritme og har informasjon om hverandre. Ruting algoritmene som brukes innad et AS blir kalt for **intra-autonomous system routing protocol** som finner korteste vei til Gateway ruter**.** For at et AS skal kunne koble seg til andre AS, trengs det en ruter som kan videresende pakker til destinasjoner utenfor AS, disse blir kalt for **Gateway rutere.** Hvert AS har enn ruting tabell over området sitt, så hver ruter vet den kjappeste veien å videre sende en pakke lokalt.

Dette prinsippet går som smurt hvis et AS kun har en Gateway ruter å sende ut fra. Men hva hvis et AS har to eller flere slike rutere? Dette kan løses ved **inter-AS routing protocol,** som samler inn info fra rutere fra andre AS og sender deretter infoen til alle rutere innad AS. Da vil rutere innad et AS vite hvilket Gateway ruter den skal sende til.

4.6 Ruting i Internett + RIP, OSPF + BGP

Avstandsvektor protokoll: Utveksle ruting tabeller med naboen

Intra-AS ruting protocol som blir brukt innen AS:

RIP (Routing Information protocol): Protokollen tar bare med rutere med mindre enn 15 hopp (cost), det vil si at alle rutere cos = 1 ved bruk av denne protokollen. En annen begrensning ved denne protokollen er at den sender bare en liste over maks 25 beste oppføringene. Dette oppdateres hvert 30. sekund (ruting). **Link state-protocol:** Utveksle egen direkte ruter med alle, beregner rute ved hjelp av Dijkstras algoritme. **OSPF (Open shortest path first):** Den sender sin «konnektivitetstabell» (sine direkte ruter) med «cost». Sendes hvert 30 min til alle, det vil si at ved sending så vil alle få et bilde av hvordan nettet ser ut. Hvis en ruter oppdager en endring i nettet, sendes det en beskjed og nettbilde oppdateres med en gang. Cost kan for eks. være hastighet på link.

Oppdeling i hierarkisk nivå:

AS består av mange rutere, men UIA sitt nett er for lite til å være et AS. Så UIA er bare et område i AS. Et såkalt «**backbone» nettverk** kobler sammen alle AS. Den samler og sender til adresser, knytter nett sammen. Samler oppføring av IP adresser, og sender pakker videre til AS' er.

BGP (border gate protocol): BGP formidler eksistensen av subnett til alle AS i internett. En av oppgavene den har er å finne «beste ruter», men også ta hensyn til for eks. økonomisk og politiske restriksjoner og preferanser. BGP går over semi permanente TCP forbindelser mellom alle rutere i en hierarkisk struktur. Hovedhensikten med BGP er å formidle, samle info om nettverket og finne ut hvem de går igjennom. Et eksempel.

Vi har tre AS der hver består av tre rutere. AS1 vil formidle til AS2, da sender ruteren fra AS1 en eBGP melding til en ruter i AS2 og formidler AS1 sin adresse. En iBGP melding blir formidlet videre til alle rutere internt i AS2. AS2 vil sende til AS3 og sender en eBGP melding. AS3 mottar eBGP og sender iBGP internt i A3. En eBGP forteller alle AS den har vært innom.

4.7 Broadcast and Multicast routing

Broadcast: Nettverks-laget tilbyr en tjeneste ved at den sender pakke fra source node til alle andre noder I nettet. **Multicast:** Source node sender pakkekopi til en gruppe noder i nettet.

4.7.1 Broadcast ruting:

Uncontrolled flooding: Når en node mottar en broadcast, lager den kopi og sender til sine naboer unntatt noden som sendte pakken. Naboene igjen vil sende til sine naboer igjen. Ulempen er at flere kan få samme pakke flere ganger, eller at det går i en evigløkke. **Controlled flooding:** Pakken inneholder nå en broadcast sekvensnummer fra source node. Hver node har en liste over source adresse og sekvensnummer over alle broadcast pakker den allerede har mottatt, duplisert og videresendt. Løser litt av problemet ved duplisering, men det er fortsatt duplisering når noder sender tvers over og ikke duplisering nedover i «treet». **Spanning-tree broadcast:** Denne unngår redundans fullstendig. Videresender kun i sitt eget tre (holder seg helt til venstre og høyre i treet).

4.7.2 Multicasting

IPv4 har eget adresseområdet til multicast: 224/4 => 1110 xxxx.a.b.c.

Fra: 224 = 1110 0000 til 239 = 1110 1111

Multicast adressene betegner grupper av hoster som ikke behøver å være i samme geografiske område. Mange enheter som vil ha samme data, kan befinne seg i en multicast group. Hvis et datagram sendes til multicast group adressen, vil alle i gruppen motta samme data.

I et tilfelle der mange er med i en gruppe brukes følgende prinsipp:

En source multicast enhet multicaster til alle rutere og enheter som er koblet til nettet. Etter hvert som de mottar meldingen kan de sende en melding tilbake til kilden om en vil være medlem eller ikke. Hvis en ruter ikke er koblet til en host sender den en «ikke send til meg, jeg har ingen medlemmer». En protokoll kalt IGMP fungerer slik at hoster sier ifra til ruteren den er koblet til om at den vil koble seg til en bestemt multicast group.

Ved spredte mottakere sendes aktiv «join» fra alle som ønsker medlemskap. Forutsetter at potensielle medlemmer kjenner source adresse. Medlemmer trenger bare gå til ruter som har medlem, får den all info uansett.

Kapittel 5 Link – laget

Link-laget danner en forbindelse mellom rutere og mellom ruter og hoster. Punkt til punkt eller broadcast. Link-laget sender sine pakker som en ramme (frame). Dette kan f. eks være Ethernet eller trådløst LAN. Så i et IP datagram kan vi kan en Ethernet header.

Link-laget tilbyr tjenester som: (1) Innkapsling av data i rammestruktur (alltid med) som vil si at link-laget pakker IP datagram til en frame før den sendes ut på en link. (2) Aksessmekanisme som vil at host kan sende en ramme så lenge linken er ledig, (3) Pålitelig overføring, (3) feildeteksjon og feilkorreksjon (oppdager feil, finner hvor feilen er og retter opp).

Ofte blir link-laget implementert i nettverksadaptere, og i adapteren finner vi link-lags kontrolleren som har implementert mange av link-lagets tjenester.

5.2 Error Detection and Error Correction Techniques

Prinsippet fungerer slik at senderen sender en link-lag ramme med link-lag header – nyttelast – EDC. Når mottakeren mottar rammen så beregner den den mottatte EDC' med den originale EDC og sammenligner de med hverandre. Hvis de er identiske, så har ingen feil oppstått. Den sammenligner også nyttelast D med nyttelast D'. Vi har tre teknikker for å oppdage feil. (1) paritetskontroll, (2) checksum metoder og (3) cycle redudancy checks.

5.2.1 Paritetskontroll

En enkelt paritetskontroll fungerer at du har informasjon D som har d antall bit. I en partalls paritet, så tar sendere å legger til en ekstra bit P og velger verdien slik at d+1 gir partall antall 1. Og på samme måte for oddetalls paritet. Mottakeren teller bare antall 1 og hvis han finner et odde-antall i en partalls paritet, vet han at et oddetall nummer bit error har oppstått. Men hva når det er partalls nummer bit error? To dimensjonal.

Vi har også to dimensjonal paritetskontroll for bedre feilkontroll. Nå blir d bits delt inn i rader og kolonner slik at paritetskontrollen blir j+i+1. Denne oppdager feil og retter den. La oss si du har kolonner og rader med d bits, også blir en av bitene endret til 0 underveis. Da er det bare å telle antall 1' ere langs kolonnen og raden der feilen oppstå og sjekke at du får et odde-antall 1' ere. Det at mottakeren kan oppdage og rette feil kalles for **forward error correction (FEC).**

5.2.2 Checksum (transportlaget).

Mottakeren sjekker summen ved å 1st kompliments av summen av mottatt data. Blir alle bit 1, gir det ingen feil og motsatt for error.

5.2.3 Cyclic Redudancy Check (CRC) dafuq?

5.3 Multiple Access Control

Kommunikasjon i grupper: Slipp alle til (alltid aktuell) – vent til du blir bedt om å si noe – ikke overta samtalen – Si ifra hvis du vil si noe – Ikke avbryt andre (alltid aktuell) – ikke sov når noen snakker til deg (alltid aktuell). Alle punktene er relevant i MAC protokoller.
Funksjoner i MAC: Alle skal få rettferdig tilgang til mediet (ikke nødvendigvis lik) – unngå at flere sender samtidig (alt ødelegges ved kollisjon) – Alle enheter må kunne nås tilnærmet alltid når de er tilkoblet

5.3.2 Random Access Protocol

Slotted Aloha: Alle rammer er lik lange – Tiden deles i enheter lik rammelengden (tidsluker) – noder starter alltid ved start av tidsluke – alle noder er synkronisert – ved kollisjon oppdages denne før hele rammen er sendt – ved kollisjon, send om igjen med sannsynlighet p i neste luke, eller senere.

Carrier sende multiple Access (CSMA): Hvis en annen sender er aktiv når en enhet vil sende, venter den til mediet blir ledig – *Carrier sensing (CSMA)*. En node lytter til kanalen om det er ledig.
Hvis en node som sender lytter til kanalen og oppdager en annen node også sender data på samme kanal. Stopper den å sende og venter en tilfeldig tid før den lytter på nytt – *Collision detection (CSMA/CD)*. **Kollisjonsdeteksjon:** Alle sendinger må være så lange at de oppdager alle kollisjoner, dette gjøres ved at alle må være minst like lange som rundturforsinkelsen. Hvis en sending oppdager kollisjon på veien, stopper den overføringen og vente en tilfeldig tid før den lytter på nytt om det er ledig. Lite kollisjoner gir kort ventetid og omvendt. Det fins opptil 0-1023 tilfeldige ventetider.

5.4 Switched Local Area Networks

Alle nettkort har en MAC adresse. Fysisk adresse om er knyttet til kortet av produsenten, normalt 6 byte. Link-laget gjenkjenner ikke nettverks-lag adresser (IP), i stedet bruker den link-lag adresser for å videresende frames gjennom svitsjer.

5.4.1 Link-Layer Adressing and ARP

Ruteren og svitsjens Interface' er MAC adresser. Det vil si at man kan komme inn med en adresse og bli sendt ut med en annen adresse. MAC adressene skrives på heksadesimal form.

En **ARP (Address Resolution Protocol)** er en protokoll som oversetter mellom link- og nettverks-lag adresser. Et lite eksempel er hvis A vil sende til B, men A kjenner bare B sin IP adresse og ikke MAC adressen. Så sender A ut en broadcast «Hvem har IP adresse x?», den som har IP adressen svarer med sin MAC adresse.

Host A med IP: a vil sende et IP datagram til host C med IP: c. For å sende et datagram må source og destinasjons IP- og MAC- adresse oppgis. Senderen vil lage en link-lag ramme som inneholder destinasjon MAC adresse og broadcaster rammen i nettet. ARP ser på IP adressen og returnerer MAC adressen. ARP løser kun IP adresser for host og rutere i samme subnett.

Hver ruter har en ARP tabell som inneholder «mappingene» av IP adresser til MAC adresser. I tillegg inneholder den også TTL. Men hva skjer hvis senderen ikke destinasjons adressen i tabellen? Jo, ved en ARP pakke. En ARP query blir opprettet og sendt ut til nentet til alle i subnettet via en broadcast og spør «hvem har IP adresse x?». Alle hoster tar imot rammen med meldingen og sjekker den med sin ARP om IP adressen matcher en MAC adresse. Den som har MAC adressen sender en respons tilbake med MAC adressen. Sendere oppdaterer tabellen og kan endelig begynne å sende.

Hvis en host skal sende data til et annet nett. Så må den første sende til ruterens MAC adresse og fra ruteren til endelig destinasjon. Ruteren vet hvor den egentlig skal ved å se på pakken.

5.4.2 Ethernet

Første versjon bestod av et felles medium der det opprinnelig var en kabel til alle, dette skapte krasj. Neste versjon av «Hub» som gikk ut på at det var et felles medium der alt som kom inn på en inngang, ble sendt ut mot alle. CSMA/CD er relevant for begge versjonene for å hindre krasj. Versjonen Vi benytter oss av i dag er svitsj. Enhetene er koblet til en felles buss i små nettverk. Sendes bare ut til de som skal ha rammen når mottaker er oppført i svitsjetabellen, ellers til alle. Hver linje har en tabell med hver sin unike MAC adresse. Tabellen er selvlærende, den vet ingenting i utgangspunktet. Når en sender, vil rammen inneholde fra/til MAC adressen.

Ethernet ramme struktur:

Preamble	Dest. Adr	Soruce adr	typ e	Data	.....		CRC

Preamble (8 byte) viser taktinfo ved lesing, «vekker» mottakeren. **Dest.Adr (6 byte)** inneholder MAC adressen til mottaker. **Source address (6byte)** MAC adressen til den som sender rammen. **Type (2 byte)** definerer pakketype som IPv4, IPv6 og ARP. **Data (46-1500 byte)** frakter IP datagrammet. **CRC (4 byte)** sjekke bit error i rammen.

5.5.1 MPLS (Multiprotocol Label Switching)

MPLS utvidet header:

Label (20 bit)	Class of service (3bit)	Stackbit (1bit)	TTL (8 bit)

MPLS er en pakke svitsjet VC nettverk og har sine egne pakkeformater og videresendingsfunksjoner. MPLS befinner seg i linklaget og sammenkobler IP enheter. Fordelen med MPLS er at videresendingen skjer raskere ved bruk av en **labelverdi** med en bestemt lengde. I stedet for å videresende ved hjelp av å se på IP datagrammet. MPLS header ligger mellom link og IP, og en MPLS kompatibel ruter vil kunne derfor lese MPLS header i stedet for å IP headeren. Før MPLS header har vi en Ethernet header som kan si noe om type/versjon data som sendes. En MPLS kompatibel ruter blir kalt for **label-switched ruter.** Ruteren ser på MPLS sin label i dens rutingtabell og sender ut på riktig output Interface. Ruterne trenger dermed ikke se på IP dest.adr og bruke longest prefix match i rutingtabellen. MPLS rutere i nettet fungerer slik at en pakke som kommer inn med en label verdi vil bli sende ut på en ny link med en ny label verdi bestemt ut ifra rutingtabellen i hver MPLS ruter. En bestemt in label gir en bestemt out label. MPLS rutere funger slik at rutere forteller hverandre om hvilke label verdier de må inneholde for å bli sendt til en bestemt destinasjon.

Fordeler med MPLS er at den utfører svitsjing basert på label uten å tenke på IP adressen til pakken. Dette fører til at rutingen er raskere da den ikke trenger å se på IP header. **Traffic engineering** er en metode som går ut på at MPLS kan videresende pakker på tilpassede linjer ut ifra pakkens krav og hva som lønner seg. IP ruting fungere slik at den velger ruten som koster minst, men ikke nødvendigvis sikker f.eks. Siste tingen er ved etablering av VPN der en kan bygge egne forbindelser gjennom et OP nett ved hjelp av MPLS.

AR-ABR-BR rutere og MPLS

AR (Access Router) eller Access Gateway befinner seg i AS (Area Network) og brukes for å koble til hoster. AR rutere har en forbindelse inn til **ABR** rutere som befinner seg i et **backbone area network** som skaper forbindelse mellom to forskjellige area network. AR må gjennom backbone for å komme til en AR i et annet nett. AR klassifiserer alle pakker som sendes til ABR. Klassifisering er TOS feltet i IP header, IP fra/til, TCP/UDP port nummer + at de har et merke på seg som sier noe om det er UDP eller TCC. AR klassifiserer alle pakker som sendes til ABR avhengig av TOS.
Backbone area Network består av **ABR** og **BR** rutere. ABR (area border router) tar imot forespørsler fra AR og sender videre til BR. ABR lager køer fro de ulike typene av trafikk (ser på klassifisering), og sender data til utgangs ABR ved hjelp av MPLS. **BR** (backbone router) kan lese både MPLS og IP noe ABR også kan. BR må ha forskjellige tabeller for å håndtere merkelapper. Når en ethernet ramme ankommer ABC, blir CRC og type fjernet. Deretter går den inn i MPLS sin header og tabellen tas i bruk. MPLS tas i bruk i backbone nettverk.

Kapittel 6 Wireless and mobile Networks

Vi har følgende elementer i et trådløst nettverk: **Wireless hosts** som kan være mobil, pc osv. **Wireless links**, hoster kobler seg til en base stasjon eller til en trådløs host gjennom en trådløs kommunikasjons link. **Base stasjon** hoveddelen i et trådløst nettverk sitt infrastruktur. BS har ansvaret for å sende og motta data til og fra trådløse hoster som er assosiert med base stasjonen.

6.3.1 802.11 Architecture

802.11 infrastrukturen er bygd opp av en **BSS (Basic service Set)** som inneholder en eller flere trådløse stasjoner og en sentral base stasjon bedre kjent som aksess punkt (AP). De trådløse stasjonen er koblet til sin AP i sitt BSS. En svitsj eller ruter er koblet til AP i hvert BSS og til internett. Hver AP har en MAC adresse for dens trådløse Interface. Enheter kan også grupperes i et felles BSS og danne et AD-HOC nettverk der enhetene kommuniserer direkte mellom seg og ikke via en trådløs ruter. En AP består av en SSID som er nettverksnavnet til en ruter som er det navnet som dukker opp når du skal koble deg til et trådløst nettverk.

6.3.2 802.11 MAC Protocol

Når en trådløs stasjon har blitt assosiert med en AP, kan den begynne å sende og motta data rammer til og fra AP. Men det kan hende at flere stasjoner sender rammer på samme over samme kanal. Løsningen på dette er **CSMA/CA** som er en random aksess protokoll. Denne kan minne om ethernet sin CSMA/CA der enhetene lytter før de sender og venter når kanalen er opptatt. Det er en vesentlig forskjell mellom ethernet og MAC 802.11 sin CSMA. I stedet for å bruke kollisjonsdeteksjon, bruker 802.11 kollisjons unngåelse. Den sjekker om kanalen er ledig før overføring og holder igjen hvis kanalen er opptatt og venter en random ventetid før den prøver på nytt.
Grunnen til at 802.11 ikke kan implementere kollisjonsdeteksjon er at signalet fra mottaker sin ruter er sterkere enn senderens signal, noe som gjør det vanskelig å lytte om det kan oppstå kollisjon. MAC protokollen bruker link-lags kvittering, for å kvittere for pakker som kommer frem i riktig tilstand. Når en ramma ankommer og kommer seg forbi CRC, må den vente en kort periode **SIFS (Short Inter-frame Spacing)** før ACK kan sendes tilbake. Hvis mottakeren ikke mottar ACK etter mange forsøk på retransmisjon, vil CSMA/CA fjerne rammen. CSMA/CA fungerer som følgende: Hvis en lytter og kanalen er ledig, send data etter en kort ventetid **DIFS + random.** Hvis ikke ledig, DIFS + random tid som var igjen før det ble opptatt. Når det er ledig, DIFS + random.
Så igjen, forskjellen mellom CSMA/CD og CSMA/CA: Hvis to stasjoner lytter og kanalen er ledig, begynner de å sende samtidig og en kollisjon vil oppstå. CD sørger for at stasjonen venter en tilfeldig

venter tid før de prøver igjen. I CA så vil de to stasjonene lytte og se at det er opptatt. Da vil stasjonen begge vente en tilfeldig ventetid. På denne måten unngår vi at kollisjon oppstår.

Hidden terminal

Hidden terminal oppstår når to eller flere trådløse enheter ønsker å sende til samme AP. AP sitt signal rekkevidde er stor nok til at de trådløse enhetene kan høre den. Men signalene til de trådløse enhetene har ikke stor nok rekkevidde til at enhetene kan høre hverandre. Hvis en H1 lytter, ledig, også begynner å sende til AP. H2 lytter, men kan ikke høre H1 og begynner også å sende data. Dette fører til kollisjon da AP mottar data fra begge på samme tid.
Dette kan løses ved å bruke kontroll rammene **RTS** og **CTS** for å reservere tilgang til kanelen. Hvis H1 vil sende til AP, så sender den en RTS som inneholder tid som trengs og en kvittering. Når AP mottar RTS, så sender den ut en CTS til H1 om at forespørsel er godkjent. I tillegg sender AP også ut CTS til alle andre trådløse enheter som vil sende til samme AP om at kanalen er opptatt.

6.3.3 The IEE 802.11 Frame

Frame Controll (2 byte)	Varighet (2byte)	Adr1 6 byte	Adr2 6 byte	Adr3 6 byte	Sekvens kontroll 2 byte	Adr4 6 byte	Payload 0-2312 byte	CRC 4 byte

Frame control inneholder disse feltene:

Protocol Version	Type	Subtype	To AP	From AP	More frag	Retry	Power mgt	More data	WEP	Rsvd

Bruk av 4 adresser:

Til AP	**Fra AP**	**Adresse 1**	**Adresse 2**	**Adresse 3**	**Adresse 4**
0	0	Destinasjon	Source	SSID	------
0	1	Destinasjon	SSID	Source	-------
1	0	SSID	Source	Destinasjon	-------
0	1	Recieve	Transmit	Destinasjon	Source
		Til	**Fra**	**Egt. Fra**	**Egt. Til**

Adresse 1 er adressen til den som sender rammen. Adresse 2 er adressen til den som skal motta rammen. Adresse 3 inneholder adressen til ruterens Interface.

La oss ta for oss et eksempel er en ruter **R1 skal sende data til et BSS der H1 befinner seg**. (1) Ruteren kjenner IP adressen til H1 (Fra dest. Adresse til datagram) og bruker ARP for å få MAC adressen til H1. (2) R1 pakker IP datagram til et Ethernet ramme og legger inn Source (R1 MAC) og destinasjon (H1 MAC). (3) Når rammen ankommer AP, fyller den inn H1 MAC i adresse 1, AP MAC i adresse 2 og adresse 3 fylles med R1 MAC som da er hvor den egentlig er fra. Når H1 mottar dataen, vil den kunne vite hvor pakken egentlig kom fra og derfor også kjenne til destinasjons adressen (R1) når den skal svare.
Nå skal H1 svare og sende til R1. (1) H1 lager først en 802.11 ramme og fyller adresse 1 med AP sin

MAC, adresse 2 fylles med H1 sin MAC og adresse 3 fylles med R1 sin MAC. (2) AP mottar rammen og konverterer til en Ethernet ramme. Source fylles med H1 MAC og destinasjon fylles med R1 sin MAC

6.5 Mobility Management: Principles

Ingen mobilitet vil si at brukeren kun beveger seg innen samme trådløse aksess nettverk. Middels mobilitet vil si at bruker beveger seg mellom akses nettverk, men er «slått av» ved bevegelse til mellom nettverk. Full mobilitet betyr at bruker beveger seg mellom aksess nettverk og ha kontinuerlig forbindelse.
Hva er problemet ved mobilitet når en applikasjon ønsker å flytte på seg? I TCP blir en forbindelse opprettet før overføring og IP fra/til og port fra/til må være konstant, i tillegg må socket være uendret til enhver tid. Ved UDP trenger man ikke å opprette en forbindelse før sending ved sending av UDP segment, den bare sender og får svar tilbake og har derfor ikke problemer med mobilitet. Mobil IPv4 gjør det mulig å opprettholde TCP forbindelsen ved mobilitet. Et mobilt nettverk består av følgende komponenter: **Mobile node** er enheten som forflytter på seg. **Home Network** er IP adressen til nodens «hjemsted» før flytting. **Home Agent** er ruteren som tar seg var mobilitets funksjonene. **Foreign Network (Visited)** er nettet som enheten har flyttet til. **Foreign Agent** er ruteren som tar se av mobilitets funksjonene i FN. Han skal også informere HA at den nye noden nå befinner seg i dette nettet og har en **Foreign address**. **Correspondent node** er han som ønsker å kommunisere med den mobile noden som flytter på seg. **Foreign adresses eller care-of-address** adressen noden får i Foreign Network av Foreign agent. Vi har to typer ruting, indirekte og direkte ruting til en Mobile node.

Indirekte ruting vil si at correspondent node sender datagram til mobile node sin permanent address i nettet uten å vite om den har flyttet på seg eller ikke. (1) Correspondent ønsker å sende data til Mobile node, destinasjon adresse er adressen i HA. Datagrammet blir sendt til Home agent. (2) HA har fått beskjed av FA at enheten har forflyttet seg og videresender til det nye nettet. Dette gjøres ved at HA pakker den permanente adressen inni et nytt datagram som har en ny destinasjons adresse, nemlig Mobile node sin COA og sendt til FA. (4) FA mottar datagrammet, pakker den ut og «fjerner» COA og sender til Mobile node med permanent adressen. Hvis den mobile noden vil sende til Correspondent, trenger den bare å oppgi destinasjon til Correspondent og source er permanent adresse. Trenger ikke gå igjennom Home Network. Direkte ruting vil si at Correspondent får vite COA og sender direkte til Foreign network.

La oss ta i utgangspunkt at den Mobile noden skal registrere seg med HA i stedet for FA. (1) Den mobile noden sender en mobil IP registering til FA. Meldingen blir fraktet inni et UDP datagram og inneholder COA (bestemt av FA), adressen til HA og permanent address. (2) FA mottar meldingen med registreringen og lagrer den permanente adressen. FA sender mobil IP registerings melding til HA som inneholder COA, HA og permanent adresse inni et UDP datagram. (3) HA mottar registeringsforespørselen og binder mobil nodens permanente IP med COA, sånn at HA nå vet at om den får en pakke med permanent IP som destinasjon, vil den bli pakket inn og sendt til COA. (4) HA sender en mobil IP registering svar som inneholder HA, permanent adresse. (5) FA mottar melingen to videresender til den mobile noden.

Kapittel 7.3 Overføring av tale over IP nett (Voice-over-IP)

IP nettet bruker best effort som vil si at den finne raskeste vei gjennom nettet, men problemet er at det ofte kan oppstå kø. I telefonnettet er det fast forbindelse og det er isokront som vil si at de ulike nettkomponentene jobber synkront. Ved tale over IP nett så blir det analoge signalet gjort om til digital signal før talen blir overført gjennom nettet. Denne konverteringen skjer med sampling der hvert punkt får en bitkode.

End-to-End Delay er forsinkelse ved overføring, prosessering og kø. Akseptabel IP nett forsinkelse er mellom 0-150ms, dårlig forbindelse er mellom 150-500 ms og elendig forsinkelse er 500ms->.

Det er mulig å sende tale uten komprimering (G.711) og sende tale med komprimering (G.729). G.711 samler tale for 20 ms og overfører dette i en IP pakke. Den bearbeider ikke samplene. Bit hastigheten er 6400bits/sek og i løpet av 20ms tilsvarer dette 160 byte overføring. G.729 samler ta for 10 ms og overfører dette i en IP pakke. Den bearbeider samplene. Bit hastigheten er 8000 bits/sek og i løpet av 10ms tilsvarer dette 10 byte overføring.
Man kan oppleve variasjon i rutere i nettet, dette kalles for **jitter**. I utgangspunktet skal neste pakke sendes 20ms etter første pakke er sendt. Men i nettet kan dette endres, ved at første pakke ikke opplever kø i rutere mens andre pakke kan oppleve kø. Eller at første opplever stor kø, mens andre opplever liten kø, da vil avstanden være mindre enn 20 ms. Jitter kan fjernes ved å bruke timetstamp eller playout delay. Andre problemer over IP nett kan være tap og feil rekkefølge. En protokoll må til, RTP.

Protocols for Real-Time Conversational Applications

Vi har RTP som brukes for å overføre tale (video) og SIP som brukes for å sette opp forbindelse. En samtale mellom to enheter etableres ved bruk av SIP og transport av tale skjer ved hjelp av RTP. RTP befinner seg i transportlaget inni UDP datagram.

RTP (Real-Time Conversational Applications)

RPT brukes for å overføre tale (video). Den hjelper oss med transport av sanntidsdata gjennom nettet og for å unngå jitter. Problemer som kan oppstå ved tale er forsinkelse (kø i ruter) der det største problemet er varighet + variasjon, mottaker må ha et buffer som håndterer jitter. Et annet problem er tap, pakker mottas i feil rekkefølge, og dublering. RTP kjører på toppen av UDP. Media pakker blir pakket inni en RTP og deretter pakket inn i et UDP segment, og videre til IP. Mottakeren pakker ut RTP pakken og henter ut media filene og sender det videre til media player for dekoding. RTP headeren består av **payload type** for å spesifisere type codec som brukes (eks. G.711). **Sekvensnummer** økes for hver RTP pakke sendt, og brukes for å oppdage pakketap. **Timestamp** (angir når pakken ble laget), osv. Hvis du har mange RTP forbindelser blir disse skillet ved hjelp av portnummer.

SIP (Session Initiation Protocol)

SIP gir mekanismer muligheten til å etablerer anrop til motparten over IP nettverk. Tillater ringeren å varsle mottakeren om at han vil starte et anrop. Gir ringeren muligheten til å bestemme IP adresse til mottakeren. De involverte har dynamiske UP adresser (DHCP).

Eksempel på session initiation: Vi har to hoster som vil kommunisere med hverandre. (1) Host A sender en ID registrar melding til SIP registrar for å registrere sin ID (IP adresse). (2) SIP registrar svarer med en OK melding tilbake til host A. (1) og (2) skjer også hos Host B. (3) Host A sender nå en invite til SIP proxy som skal hjelpe til med å kontakte B. SIP proxy videresender INVITE meldingen til host B sin proxy som videresender til registrar som videresender INVITE til host B sin enhet. (4) Så sender host B et SIP svar melding tilbake til host A med sin IP adresse. De kan snakke.
Et mer kompleks SIP oppkobling der Host sin IP er privat. SIP og RTP tilkoblingene skjer gjennom et NAT ruter. RTP og SIP kontakter en STUN server som gjør det mulig å sette opp anrop utenfor det lokale nettverket, når STUN serveren skal svarer er destinasjon adresse NAT sin. SIP går også opp til SIP server som registrerer ID og en proxy. I tillegg finnes det ogå en media gateway controller som hjelper til ved transport av tale og en media gateway som der selve talen skjer.

QoS – Quality of Service

Vi har 4 faktorer som påvirker QoS parameterne: **Reability/Error rate** der den er høy for mail, filoverføring og lav for telefoni. **Delay** som er speech sensitive. **Jitter** som er variasjon i forsinkelse. **Båndbredde** som kan være høy på video, lav på voice og mail.

Andre kan være scheduling som FIFO/prioritet. Traffic shaping som vil si jevn sending i stedet for mellomrom mellom hver sending. Leaky bucket (brukt for å kontrollere raten i nettet) og Tocken bucket er eksempler på traffic shaping. Leaky bucket går ut på at når «bucket» (buffer) er overfylt, blir pakkene kastet. Resource reservation blir brukt for å reservere ressurser gjennom nettet, hindre pakketap, og har adgangskontroll (avvis hvis host sender for mye inn i nettet). Til slutt admission control.

Vi har også QoS garantier parametere som skal oppnå best kvalitet uten best effort. Vi har **best effort. Integrated Serivce** som har RSVP som er en protokoll for å reservere ressurser. Dette skjer i rutere og krever mye arbeid da rutere må identifisere hver pakke og gi denne ønsket QoS. Også har vi COPS. **Differentiated services** som bruker et felt i IP header til å klassifisere trafikken (TOS) – DSCP. I nettet kan det være forskjellige linjer med forskjellige egenskaper TOS feltet spesifiserer hvilken linje pakken skal velge.

Kapittel 9 Network Management

Network management inkluderer installering, integrere og koordinere hardware og software. Kort fortalt hvordan IP nett skal drives. Infrastrukturen er slik: Rutere og hoster etc. er managed devices som inneholder mange managed objects som er samlet inni et MIB som inneholder management Information som counter som for eksempel teller antall error. Vi har to Network management standarder for å drive et nett, OSI CMIP med lite suksess og SNMP som benyttes i dag er og er en internasjonal standard. SNMP har fire hoveddeler: MIB som er distribuert lagring av NMD, SMI som er datadefinisjonsspråk for MIB objekter, SNMP protokoll og Security, administration capabilites. Kort sagt kan vi si at SNMP er en svært mye brukt protokoll for Network management for å samle informasjon, og konfigurere nettverks enheter som printer, svitsjer, rutere.

9.3.1 Structure of Management Information: SMI

SMI blir regnet som en data definisjonsspråk. OBJECT TYPE brukes for spesifisere data type, status og

semantikken. Vi har basis data typer som int, int32, IP address, osv. MODULE IDENTITY tillater relaterte objekter til å bli gruppert sammen innen samme MIB modul.

9.3.2 Management Information Base: MIB

MIB modilen blir spesifisert via SMI MODULE IDENTITY. Inni modulen kan det befinne seg mange OBJECT TYPE. MIB kan ses på som et virtuel informasjonsbutikk. Eksempel på MIB ved bruk av UDP modul: Object ID: 1.3.6.1.2.1.7.**1** har et navn og type, det siste tallet sier noe om hvordan UDP det er. Alle tallene sier noe hvilket type objekt det er i OSI Object Identifier Tree.

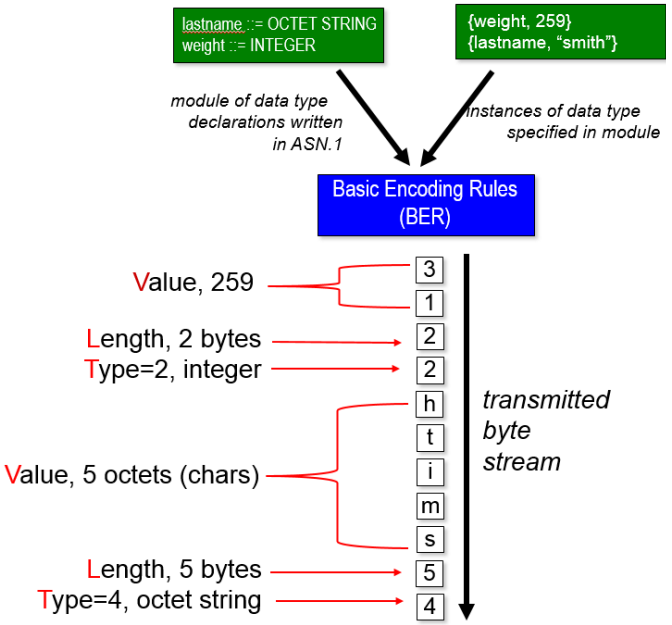
9.3.3 SNMP Protocol

Det finnes to måter å få tak i info på. Den ene måten er å enten gå ut å spørre om å få noe (request/respone) eller at ytterenhetene sender info til oss (trap msg). SNMP har også forskjellige message types som GetRequest, InformRequest, Response, Trap

9.4 ASN.I Abstract Syntax Notation I

ASN er en standard måte å beskrive en melding som kan bli sendt eller mottatt i et nettverk. Tilbyr Basic Encoding Rules (BER) for transmisjon av data. Hvert transmitterte objekt har type, lengde og verdu (TLV). T står for data type som ASN har definert. L står for lengden av data i bytes og V står for verdien av dataen når den er enkodet. ANSI har bestemte data typer:

Tag	Type
1	Boolean
2	Integer
3	Bitstring
4	Octet String
5	Null
6	Object Identifier
9	Real



TLV enkoding eksempel - >