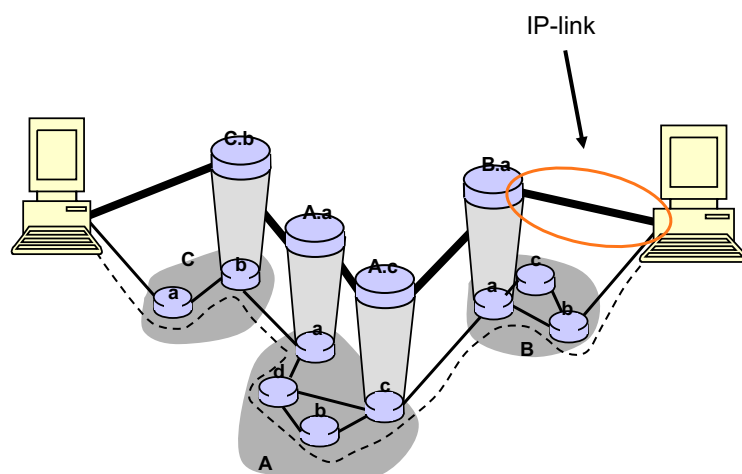


Linklaget

Olav Lysne

(med bidrag fra Stein Gjessing og Frank Eliassen)

Internettets Overlay Arkitektur



Link-typer

Tre typer av linker:

- (a) Punkt-til-punkt (enkel kabel)
- (b) Broadcast link (delt kabel eller annet medium; f. eks eldre Ethernet og trådløs)
- (c) Svitsjet (f.eks svitsjet Ethernet)

Linklagets tjenester

□ *Framing og linkaksess:*

- Pakke data inn i en "ramme" (frame), og legge til hode og hale.
- Implementere kanalaksess dersom det er delt medium.
- 'fysiske adresser' blir brukt i rammeheaderene til å identifisere kilde og destinasjon når det er delt medium.

□ *Pålitelig levering:*

- Sjelden brukt i fiberoptikk, co-axial kabel og noen varianter av twisted pair pga. lav feil-rate.
- Brukt på trådløslinker hvor målet er å redusere feil, og unngå ende til ende retransmisjoner.

Linklagets tjenester (mer)

Flytkontroll:

- Hastighetsavpassing mellom sender og mottaker.

Feildeteksjon:

- Feil kommer av støy og signalreduksjon
- Mottaker oppdager feil i mottatt ramme.
- Den signalerer for retransmisjon, eller den bare kaster rammen.

Feilkorreksjon:

- Mekanisme hvor mottakeren retter feilen uten å be om retransmisjon.

Feilfinning/feilretting

Oppgaver:

- 1. Finne feil
- 2. Rette feil
 - To alternativer til å rette feil:
 - A. Ha nok informasjon til å rette opp de mottatte dataene
 - B. Be om at dataene (rammen) blir sendt en gang til
 - [C. Gi blanke, det er ikke så farlig å miste litt data]
- Generelt prinsipp i informatikken:
Oppdag feilen så fort som mulig etter at den har oppstått !

Feil-deteksjon

- Bit-feil i rammer
 - behov for mekanismer som oppdager bit-feil
- Teknikker som ofte benyttes i datanett
 - Paritet - to-dimensjonal paritet
 - BISOYNC ved ASCII overføring
 - Sjekksum
 - flere Internett-protokoller
 - Cyclic Redundancy Check (CRC)
 - svært utbredt

Paritet (tversum)

- Ett paritetsbit:
 - F.eks. 7 bit data, sendes som 8 bit
 - Like paritet dvs. et like antall enere i resultatet
 - Odde paritet dvs. et odde antall enere i resultatet
- Like paritet: 0110001 sendes som 01100011
- Odde paritet: 0110001 sendes som 01100010
- Mulig med flere paritetsbit
 - Generelt: Jo mer data til redundanse, jo flere feil oppdages.

Internett sjekksum algoritme

- ❑ Se på en melding som en sekvens av 16-biters heltall
 - Senderen adderer disse heltallene sammen ved bruk av 16-biters aritmetikk
 - Dette 16-biters tallet er sjekksummen
 - Mottaker utfører samme beregning og sammenligner resultatet med den mottatte sjekksum
 - Får mottaker feil resultat er det bitfeil enten i dataene eller i sjekksummen
- ❑ Benyttes ende til ende i Internett

CRC: Cyclic Redundancy Check

Generalisering av paritet
Punkter i et n-dimensjonalt rom

Kodeord

Data (med hode)	Sjekk/CRC
-----------------	-----------

Like paritet: Kodeordet delt på 2 skal ikke gi rest
CRC: Kodeordet delt på et tall, G, skal ikke gi rest

Dette tallet vi deler på kaller vi Generatorpolynomet
Deling foregår med modulo-2 regning, dvs ikke mente eller låning.

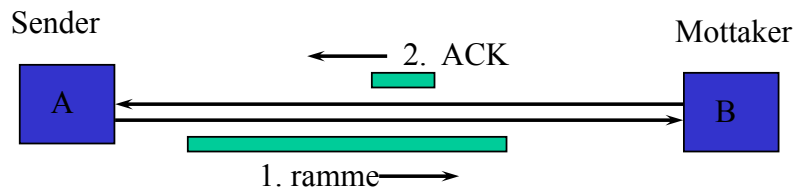
Pålitelig overføring

- Rammer med feil CRC kastes
- Fint om vi kan rette opp feilen
- Hvis feilen ikke kan rettes opp, og vi trenger rammen, da
må den sendes en gang til !
- Også her er det en avveining:
Ende-til ende eller mellom noder ?
(Problemkomplekset med doublet/triplet (mm.) funksjonalitet)

Pålitelig overføring

- Når omsending av rammer er nødvendig:
- To fundamentale mekanismer
 - kvitteringer (engelsk: acknowledgements, ack)
 - tidsfrister (timeouts) vha. vekkeklokke (timer)
- Husk at også kvitteringer kan bli borte
- Ønsker vi at rammene skal komme frem i riktig rekkefølge?

Stop-and-Wait (stopp og vent)



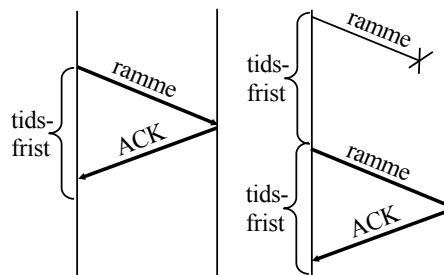
Mottaker sender ack tilbake når en ramme er mottatt, og først når sender mottar ack, sendes ny ramme. På denne måten blir ikke mottaker oversvømmet av rammer, og avsender vet at alle rammer er kommet trygt fram.

Men hva hvis rammer blir borte?

Stop-and-Wait (stopp og vent)

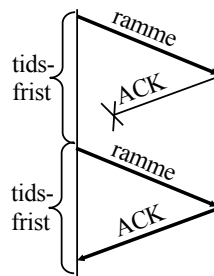
□ Grunnleggende algoritme:

- send én ramme og vent på kvittering (ACK ramme)
- dersom ACK ikke mottatt innen gitt tidsfrist, send rammen på nytt.



Stop-and-Wait

- Problem 1 med grunnleggende algoritme:
 - Men kanskje det var ACK som ble borte
 - Vi må kunne sende den samme rammen på nytt, selv om den allerede er kommet riktig frem

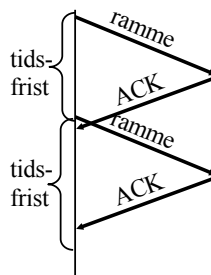


[[simula](#) . research laboratory]

Linklaget 15

Stop-and-Wait

- Problem 2 med grunnleggende algoritme:
 - Kanskje vi sendte rammen omigjen for tidlig
 - Vi må godta at ACK kommer for sent

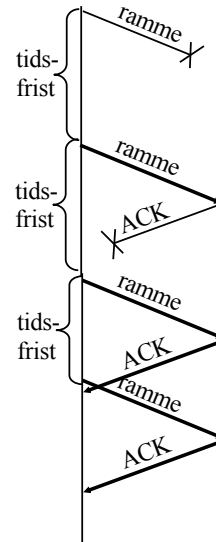


[[simula](#) . research laboratory]

Linklaget 16

Stop-and-Wait

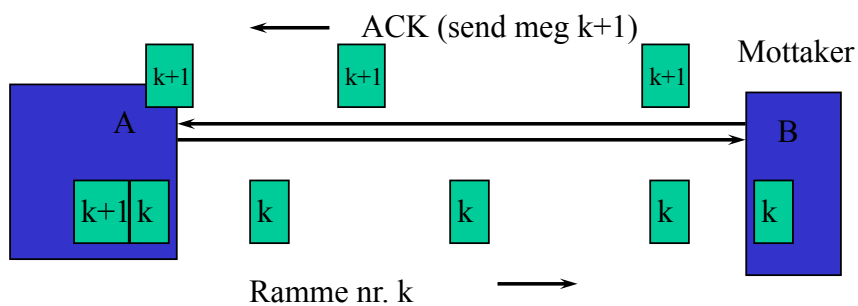
- Må sende rammen på nytt og på nytt helt til ACK kommer tilbake



[**simula** . research laboratory]

Linklaget 17

Løsning: sekvensnummer



Neste bilde:
Det er nok med en én-bit teller (0 og 1)
 $k, k+1$ regnes da ut modulo 2.

(Én buffers "Sliding window" protokoll)

[**simula** . research laboratory]

Linklaget 18

Sekvensnummer som 0 og 1

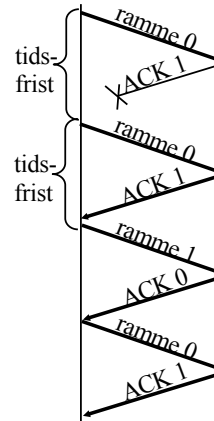
- 0 og 1 som sekvensnummere

- En bit er nok når vi har én ramme ad gangen

ACK 1: Send ramme med
odde sekvensnummer

ACK 0: Send ramme med
like sekvensnummer

- Altså:
ramme 0, 1, 2, 3, 4, 5, ... sendes som
ramme 0, 1, 0, 1, 0, 1, ...



0 - 1 sekvensnummer

- Går dette bra?

- Ja, fordi:

- like ramme (0) sendes ut
- ignorerer (gamle) ACK 0
- resender like ramme (0)

I det ACK 1 kommer:

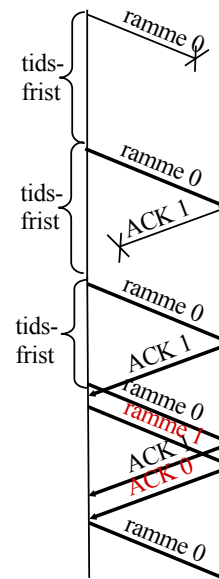
- odde ramme (1) sendes ut
- ignorerer (gamle) ACK 1
- resender odde ramme (1)

I det ACK 0 kommer:

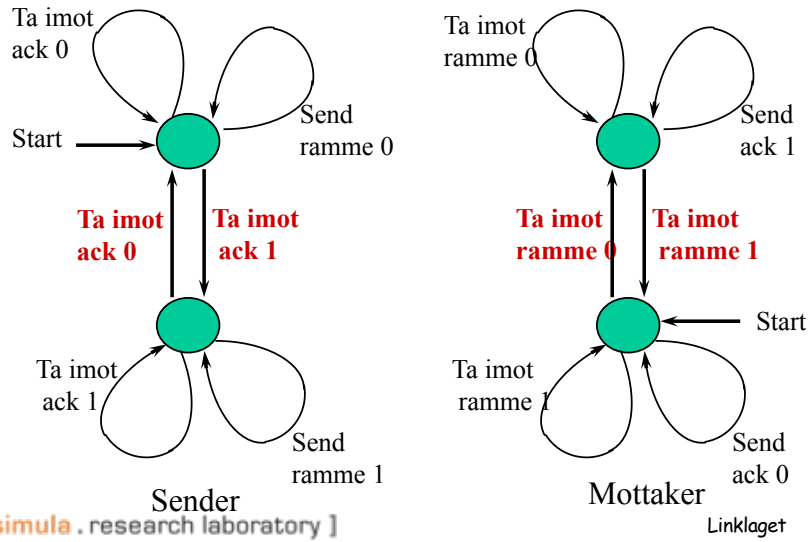
- like ramme (0) sendes ut
- ignorerer (gamle) ACK 0
- resender like ramme (0)

I det ACK 1 kommer:

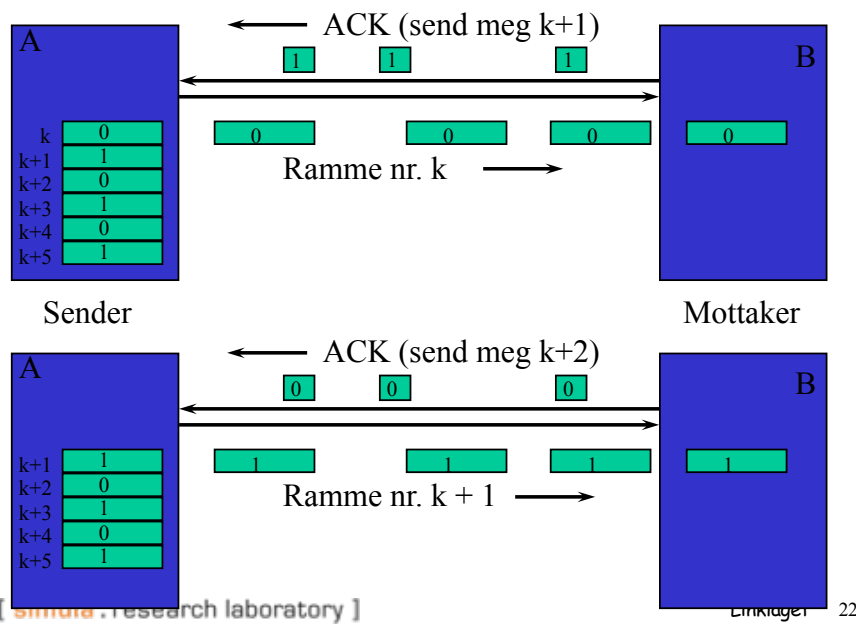
osv.



Tilstandsmaskin for ett-bit protokoll

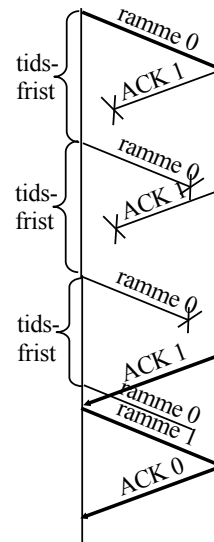


Et-bit protokoll



Stop-and-Wait

- Det gjør ikke noe om mottaker gjentar en ACK, men det er ikke vanlig, dvs. at det er bare vanlig å sende ACK når en ramme ankommer. Ved mye rammetap kan det være gunstig å gjenta ACK selv om det ikke kommer en ny ramme



Stop-and-Wait

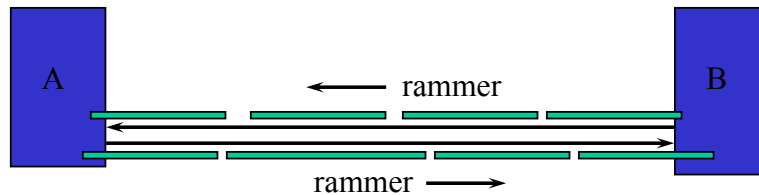
- Grunnleggende svakhet:
 - utnytter linjekapasiteten dårlig
 - senderen kan bare ha én utestående ramme til enhver tid
- Eksempel:
 - 1.5Mbps link x 45ms RTT = 67.5Kb (8KB)
 - dvs. 8KB data kan sendes før første ack kan ventes tilbake
 - anta rammestørrelse 1KB
 - stop-and-wait bruker ca. 1/8 av linjekapasiteten (om ingen feil)
 - Mål: senderen må kunne sende opp til 8 rammer før den må vente på en ACK
 - moralen er: "fyll opp røret"



Fyll opp røret

Utnytte linjen bedre.

Sender flere rammer rett etter hverandre:



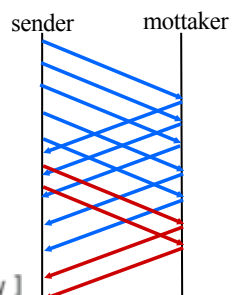
Putte ACK/NAK på ryggen til meldinger som går den andre veien ("piggyback")

Glidende vindu

□ Idé:

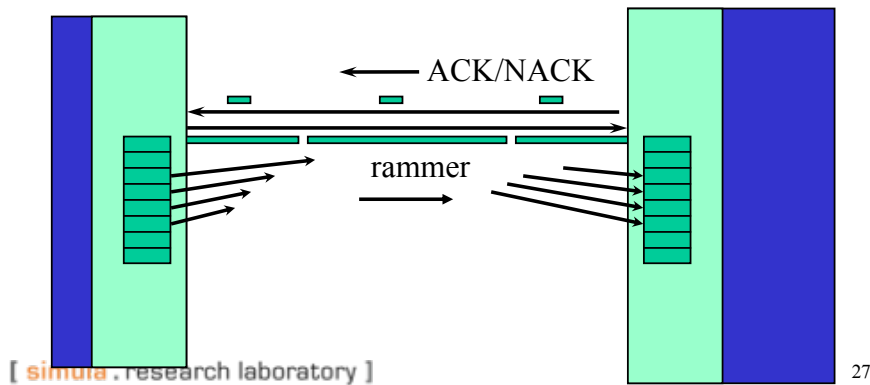
- Tillat senderen å sende flere rammer før den mottar ACK for derved å holde "røret fullt".
- Det må være en øvre grense på antall rammer som kan være utestående (som det ikke er mottatt ACK for).

Eksempel:
maks. 5
utestående
rammer



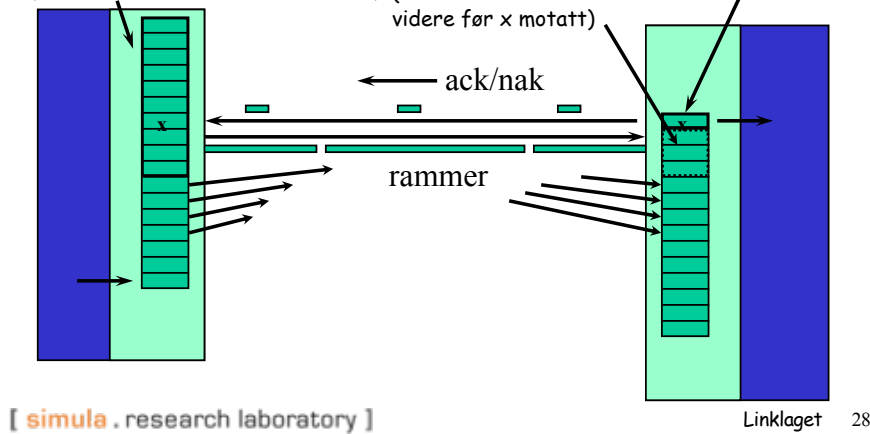
Glidende vindu

Flere bufre hos sender og flere bufre hos mottaker, mange rammer med forskjellige nummer og mange ack/nack med forskjellige nummer underveis hele tiden.



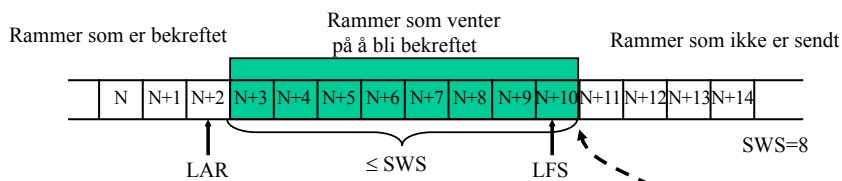
Glidende vindu

Sendt men ikke fått ack, må kanskje sendes på nytt (vil bli resendt om ack aldri mottas) Disse tre er motatt og kvittert (ack sendt) Ikke motatt (men kan ikke sendes videre før x motatt)



Glidende vindu: sender

- Tilordner sekvensnummer til hver ramme (SeqNum)
- Vedlikeholder tre tilstandsvariable
 - send window size (SWS)
 - last acknowledgment received (LAR)
 - last frame sent (LFS)
- Vedlikeholder invariant: $LFS - LAR \leq SWS$



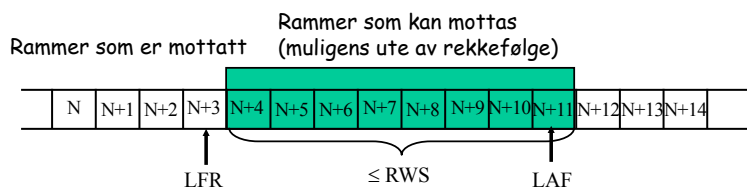
Når ACK mottas, økes LAR, og derved kan ny ramme sendes slik at LFS økes.

Buffer for opptil SWS rammer, dvs SWS rammer i "røret" samtidig
 [[simula . research laboratory](http://simula.research.laboratory)]

Linklaget 29

Glidende vindu: mottaker

- Vedlikeholder tre tilstandsvariable
 - receive window size (RWS)
 - largest acceptable frame (LAF)
 - last frame recived (LFR) (med alle "mindre" rammer også motatt)
- Vedlikeholder invariant: $LAF - LFR \leq RWS$

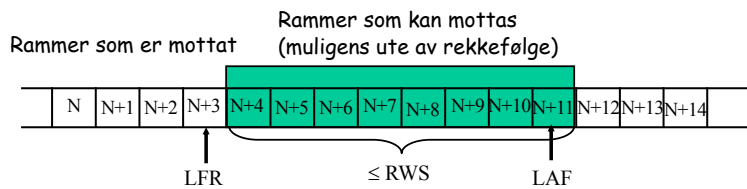


[[simula . research laboratory](http://simula.research.laboratory)]

Linklaget 30

Glidende vindu: mottaker

- Kumulativ kvittering ("go back n" protokoll), dvs. vi ack-er ikke nye rammer hvis det er hull i sekvensen av mottatte rammer
 - **if** LFR < MottatRamme.SeqNum < LAF **then**
ta-imot-rammen-og-legg-den-på-plass;
bereg-n-nye-grenser (MottatRamme.SeqNum); // se neste lysark
 - else**
kast rammen;
send ACK (LFR + 1)



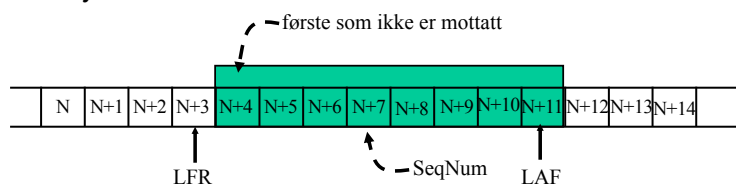
[simula . research laboratory]

Linklaget 31

Glidende vindu: mottaker

- Invariant: LFR + 1 er første ramme som ikke er mottatt
- beregn-nye grenser(seqNo)

```
• if seqNo = LFR + 1  
  { beregn ny LFR og LAF:  
    for (i= LFR + 1; ramme i er mottatt; i++) { } ;  
    LFR = i-1;  
    LAF = LFR + RWS;  
  }
```



[simula . research laboratory]

Linklaget 32

Glidende vindu: mottaker

- Varianter til "go back n" protokoll
 - Negativ kvittering (NAK)
 - mottaker sender NAK på rammer som savnes
 - Selektiv kvittering (SAK)
 - mottaker sender ACK på nøyaktig de rammer som mottas
 - disse behøver da ikke re-sendes
 - Både SAK og NAK øker kompleksiteten til implementasjonen, men kan potensielt bedre utnyttelsen av linjen

Glidende vindu: endelige sekvensnummer

- I praksis representeres sekvensnummer med et endelig antall biter.
- n biters sekvensnummer \Rightarrow intervall sekv. nr. = $(0..2^n-1)$
- I HDLC: $n = 3$, dvs. sekvensnummerintervall $(0..7)$
 - \Rightarrow sekvensnummer må gjenbrukes

0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

Problem

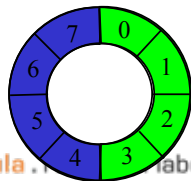
skille mellom ulike inkarnasjoner av samme numre

Minimum krav:

sekvensnummerintervallet må være større enn maks. antall utestående rammer

Glidende vindu: endelige sekvensnummer

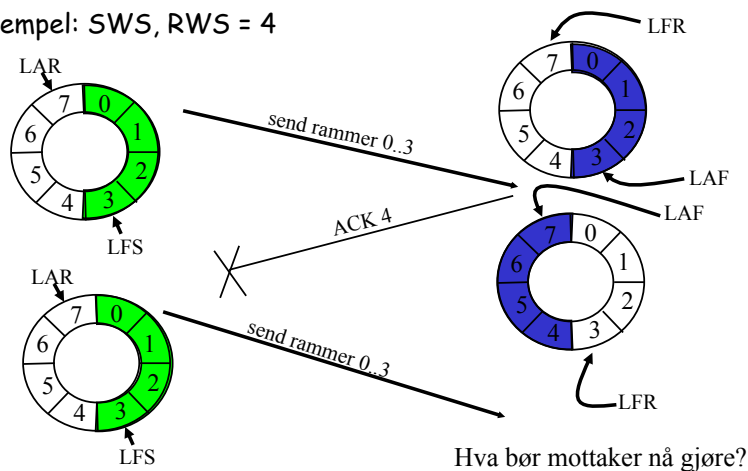
- $SWS \leq \text{MaxSeqNum}+1$ er ikke tilstrekkelig
 - anta 3 biters SeqNum felt (0..7)
 - $SWS=RWS=8$
 - senderen transmitterer rammene 0..6
 - mottas uten feil, men ACK går tapt
 - senderens tidsfrist utløper, rammene 0..6 sendes på nytt
 - mottaker forventer 7, 0..5, men mottar andre inkarnasjon av 0..5
- $SWS, RWS \leq (\text{MaxSeqNum}+1)/2$ er riktig regel



Hindrer overlapp mellom
nedre kant av sendervinduet og
øvre kant av mottakervinduet.

Glidende vindu: endelige sekvensnummer

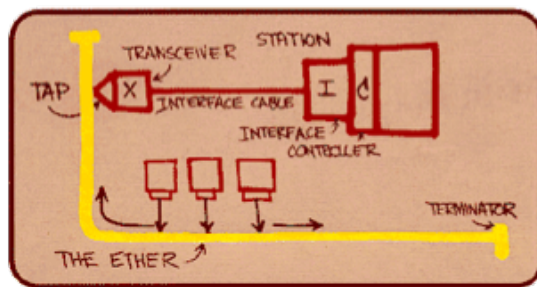
- Eksempel: $SWS, RWS = 4$



Ethernet

“Dominerende teknologi for kablet LAN:

- Billig!
- Første LAN-teknologi i vanlig bruk
- I utgagnspunktet enklere enn konkurrentene
- Stadig nye standarder 10 Mbps – 10 Gbps

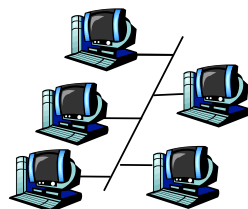


Metcalfe sin opprinnelige Ethernet-tegning

Linklaget

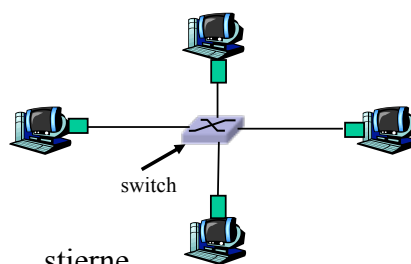
Stjerne-topology

- Opprinnelig var Ethernet bus-basert
 - Alle noder i samme kollisjonsdomene
- I dag: stjerne-topologier
 - aktiv *switsj* i sentrum



bus: coaxial kabel

[simula . research laboratory]

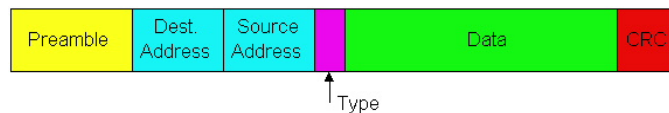


stjerne

Linklaget

Ethernet Rammestruktur

- **Adresser:** 6 bytes
 - Om adaptere mottar ramme med matchende destinasjonsadresse eller med broadcastadresse tas pakken imot.
 - I motsatt fall tas den ikke imot
- **Type:** indikerer høyere lags protokoll (ofte IP men andre er mulige, f.eks Novell IPX, AppleTalk)
- **CRC:** sjekkes hos mottaker. Om den ikke stemmer kastes rammen.



[[simula](#) . research laboratory]

[Linklaget](#)

Ethernet: upålitelig, forbindelsesløs

- **forbindelsesløs:** ingen handshake mellom sender og mottaker før sending
- **upålitelig:** mottagende NIC sender ingen bekreftelser til sendende NIC
 - Strømmen av rammer som leveres videre opp til nettlaget kan inneholde "hull" (manglende rammer).
 - Hullene kan fylles igjen av nettlaget (TCP)
 - I motsatt fall vil hullet bli synlig for applikasjonen

[[simula](#) . research laboratory]

[Linklaget](#)

Ethernet CSMA/CD algoritme

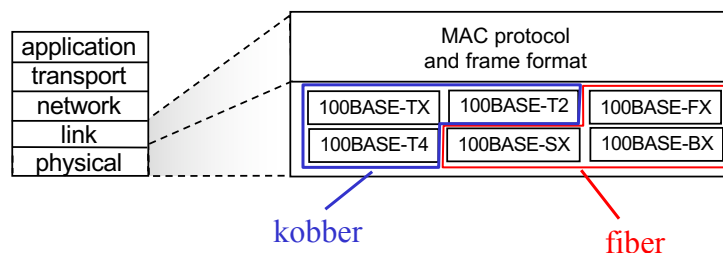
1. NIC receives datagram from network layer, creates frame
2. If NIC senses channel idle, starts frame transmission. If NIC senses channel busy, waits until channel idle, then transmits
3. If NIC transmits entire frame without detecting another transmission, NIC is done with frame !
4. If NIC detects another transmission while transmitting, aborts and sends jam signal
5. After aborting, NIC enters **exponential backoff**: after m th collision, NIC chooses K at random from $\{0,1,2,\dots,2^m-1\}$. NIC waits $K \cdot 512$ bit times, returns to Step 2

[simula . research laboratory]

Linklaget

802.3 Ethernet Standarder: Link & Fysiske Lag

- **Mange** forskjellige Ethernet standarder:
 - Felles rammeformat
 - Forskjellige hastigheter: 2 Mbps, 10 Mbps, 100 Mbps, 1 Gbps, 10 Gbps
 - Forskjellige fysiske lag: fiber, cable



[simula . research laboratory]

Linklaget

Ethernet Switch

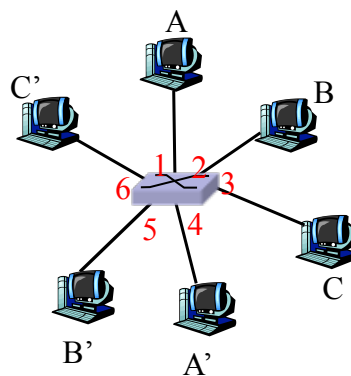
- **link-lag device:**
 - Videre sender Ethernet rammer
 - Leser MAC-adressen til innkommende ramme, og sender rammen ut til en eller flere utgående linker basert på denne adressen.
- **transparent**
 - Maskinene som er tilkoblet vet ingen ting om nettstrukturen
- **plug-and-play, self-learning**
 - Switchene trenger ikke konfigureres

[**simula** . research laboratory]

Linklaget

Svitsj-tabell

- **Q:** hvordan vet svitsjen at A' kan nås via interface 4, og B' via interface 5?
- **A:** hver svitsj har en **svitsj-tabell**, hvor hver tabellentry inneholder :
 - (MAC address til maskin, hvilket interface maskinen nås på, time stamp)
- **Q:** hvordan blir tabellen fylt opp med informasjon?



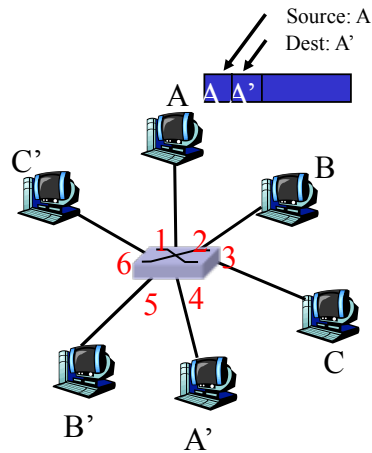
Svitsj med seks interface:
(1,2,3,4,5,6)

[**simula** . research laboratory]

Linklaget

Switsj: self-learning

- svitsjen *lærer* hvilke maskiner som kan nå gjennom hvilke interface
 - Når en ramme mottas, kan svitsjen lære lokasjonen til senderen.
 - tar vare på sender/lokasjonspår i en tabell



MAC addr	interface	TTL
A	1	60

Svitsjetabell
(tom ved oppstart)

[**simula** . research laboratory]

Linklaget

Switsj: rammefiltrering/videresending

When frame received:

1. record link associated with sending host
2. index switch table using MAC dest address
3. if entry found for destination
 - then {
 - if dest on segment from which frame arrived
 - then drop the frame
 - else forward the frame on interface indicated

else flood

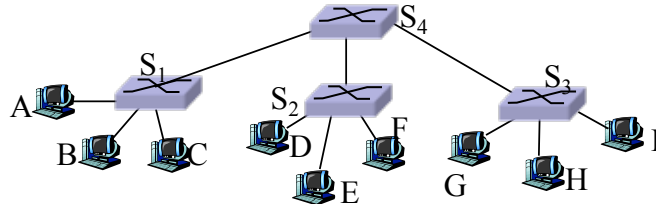
forward on all but the interface
on which the frame arrived

[**simula** . research laboratory]

Linklaget

Flere switsjer

- Switcher kan kobles sammen i nettverk



- **Q:** sende fra A til G – hvordan kan S₁ vite hva den skal gjøre?
- **A:** self learning! (virker på akkurat same måte som for en enkelt svitsj)

[**simula** . research laboratory]

Linklaget