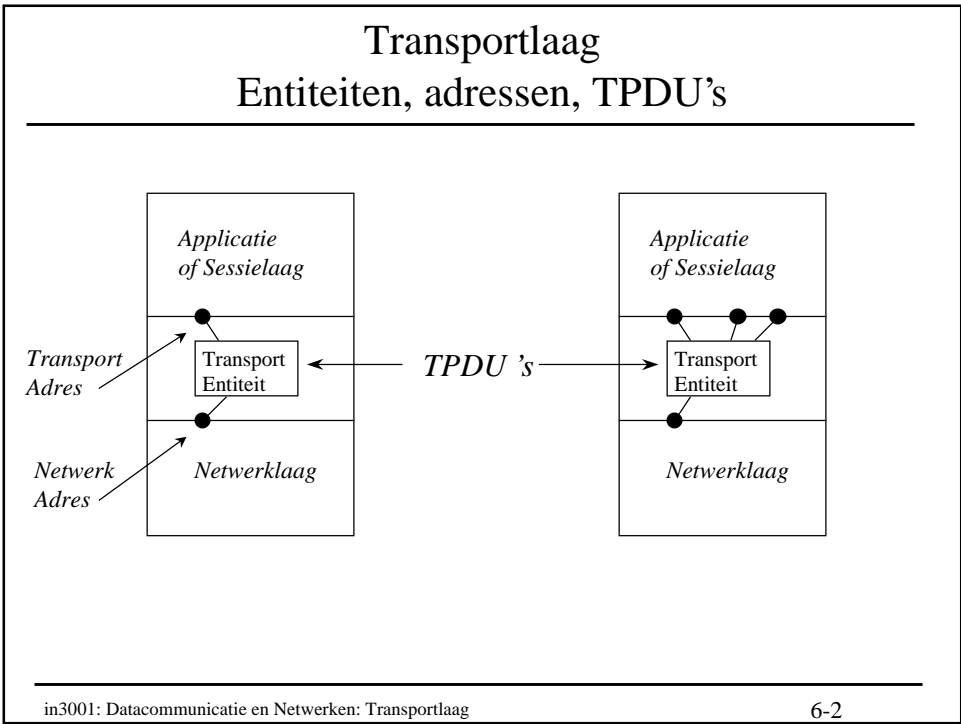


---

## 6. De Transportlaag

---

in3001: Datacommunicatie en Netwerken: Transportlaag 6-1



## Transportlaag vergelijking met netwerklaag

---

- Transportdiensten lijken op netwerkdiensten, b.v.
    - verbindingsgericht/verbindingsloze diensten
    - adressering
    - stroombeheersing
- maar:
- netwerklaag is onderdeel van het (communicatie)subnet ;
  - transportlaag kan fouten van de netwerklaag herstellen.

## Transportlaag Transportdienst primitieven (1)

---

- Transportdienst primitieven vormen het interface tussen de applicatie en de transportdienst.
- Verschil met netwerkdienst interface:
  - Transport interface is eenvoudiger, want
    - Netwerk interface bestemd voor de transportentiteit, maar Transport interface ook bestemd voor gebruikers (programmeurs)
    - Transportlaag levert betrouwbare service, transportentiteit zorgt ervoor dat problemen onzichtbaar zijn voor de gebruiker.

## Transportlaag Transportprimitieven (2)

---

voorbeeld: eenvoudige verbindingsgerichte transportdienst

primitieven:

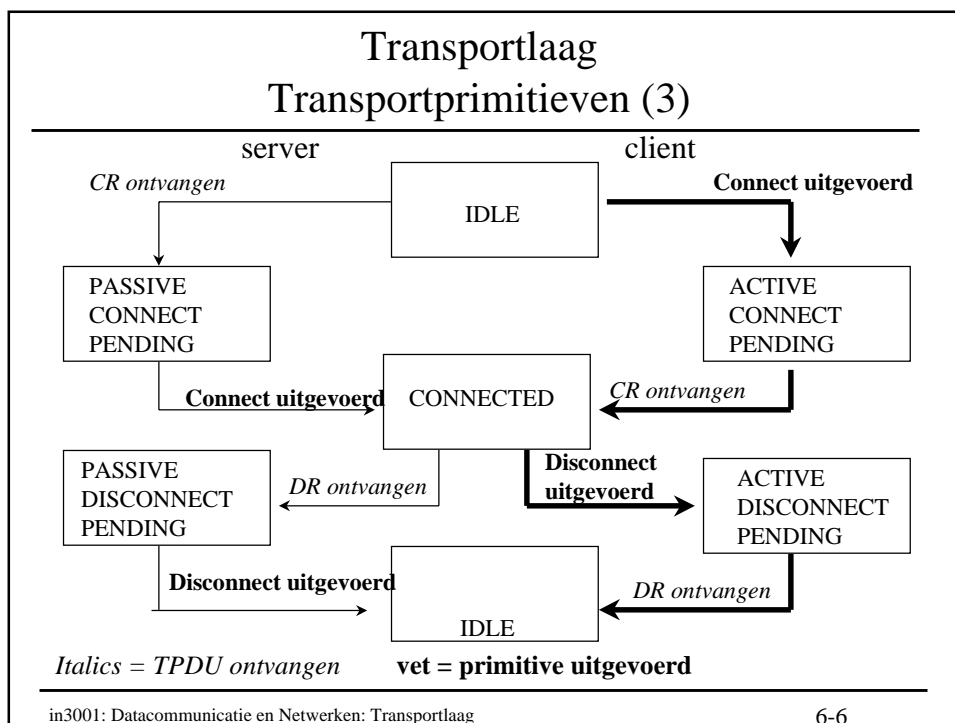
- LISTEN
- CONNECT
- SEND
- RECEIVE
- DISCONNECT

bij één server en meerdere clients

- maken van verbinding door:
  - server: LISTEN; client: CONNECT
- voor data-transport:
  - client en server : SEND en RECEIVE
- voor beëindigen van de verbinding:
  - client en server gebruiken: DISCONNECT

---

in3001: Datacommunicatie en Netwerken: Transportlaag 6-5



## Transportlaag

### Transportprimitieven (4)

---

Voorbeeld: UNIX en TCP

- Gaat uit van een client en een server

#### **Server:**

- creëert een socket m.b.v. **socket** system call
- koppelt socket aan TCP/IP adres d.m.v **bind**
- kondigt bereidheid tot verbinding aan d.m.v. **listen**
- accepteert verbinding d.m.v **accept** , krijgt hierbij een nieuwe socket
- ontvangen en zenden van buffers d.m.v. **read** en **write** (via socket)

## Transportlaag

### Transportprimitieven (5)

---

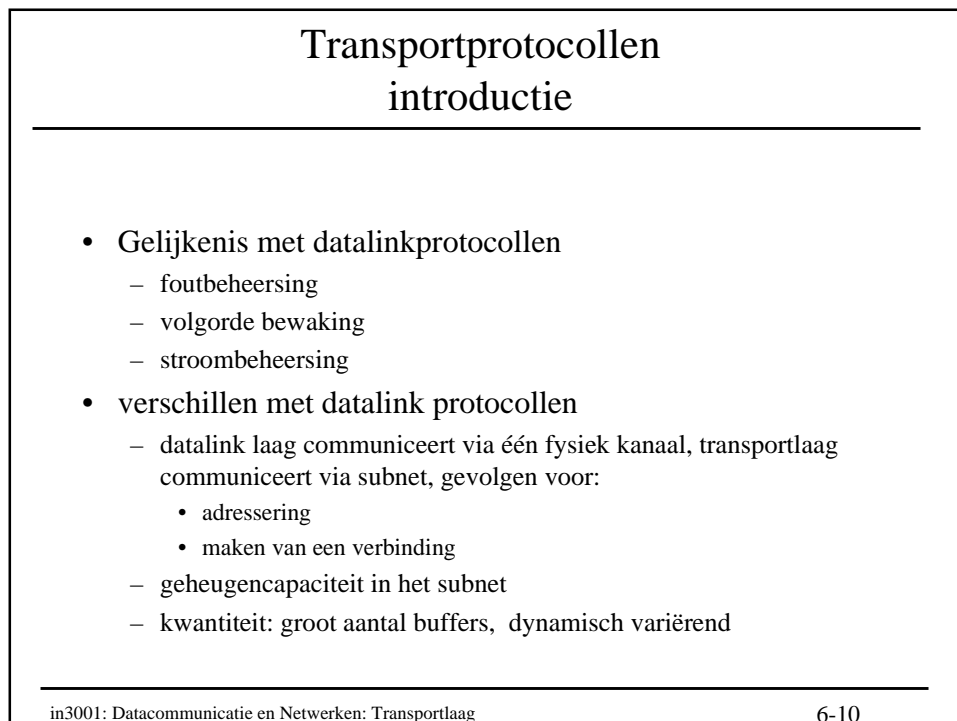
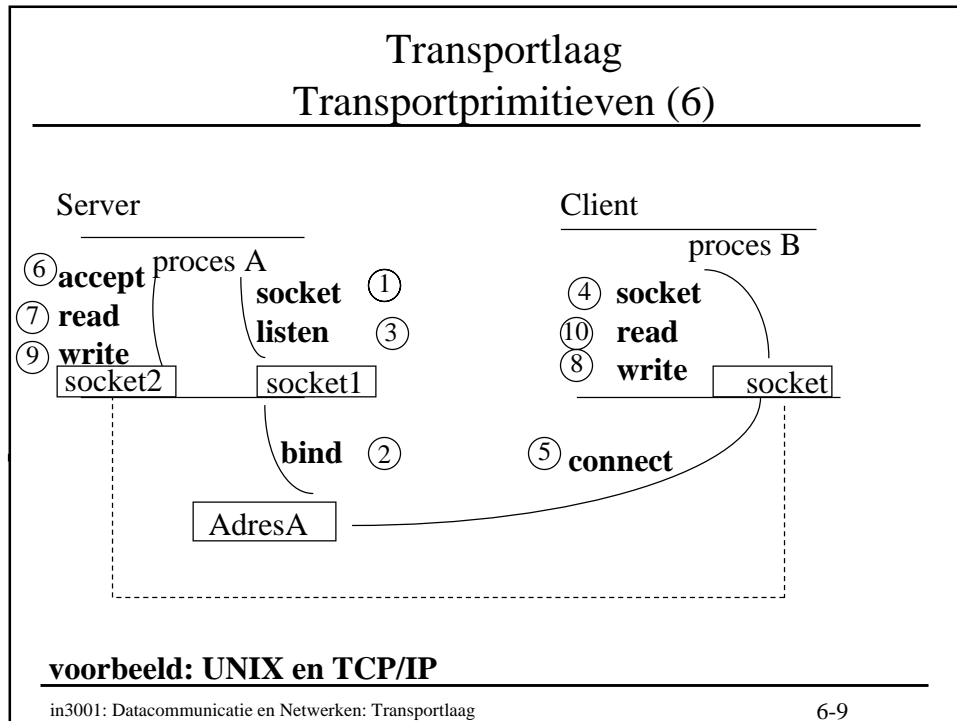
- Voorbeeld UNIX en TCP (vervolg)

#### **Client:**

- creëert socket d.m.v. **socket**
- vraagt verbinding d.m.v **connect**
- zenden en ontvangen van buffers d.m.v. **write** en **read** (m.b.v sockets)

N.B. Dit betekent dat servers vaak permanent aanwezig zijn, wachtend tot er een verzoek van een client binnenkomt.

Dit type processen wordt *daemons* genoemd.



## Transportprotocollen adressering (1)

---

- Transport adressen *TSAP's* (b.v. IP-adres + poortnummer)
- Netwerk adressen *NSAP's* (b.v. IP-adres)
- 1 transport entiteit support meerdere TSAP's
- sommige machines hebben ook meerdere NSAP's
- Mogelijkheden om verbinding tot stand te brengen:
  - bekende TSAP (vast)
  - *Initial Connection Protocol* (proces server creëert een server)  
voorbeeld: UNIX: Inetd
  - *name server of directory server*  
voorbeeld: Internet: Domain Name System (DNS)

## Transportprotocollen adressering(2)

---

- Inetd (internet daemon) maakt het mogelijk het aantal daemons te beperken
- inetd is a.h.w. een superdaemon
- luistert naar een groot aantal poorten (opgegeven in **/etc/inetd/conf** en **/etc/services**)
- Wanneer een request bij één van deze poorten binnenkomt, zorgt inetd ervoor dat de code van de bijbehorende server wordt uitgevoerd.
- voordeel van inetd:
  - minder daemons nodig (minder overhead)
  - eenvoudiger server programma's

## Transportprotocollen

### Het maken van een verbinding (1)

---

- In principe één transportentiteit geeft CR TPDU, en wacht op CA TPDU
- probleem: duplicaat TPDU's (gevolg van aflopen van timer)
- Oplossing:
  - maximum leeftijd voor pakketten vaststellen.
  - daarna te garanderen dat geen duplicaat-TPDU's niet mogelijk zijn.

## Transportprotocollen

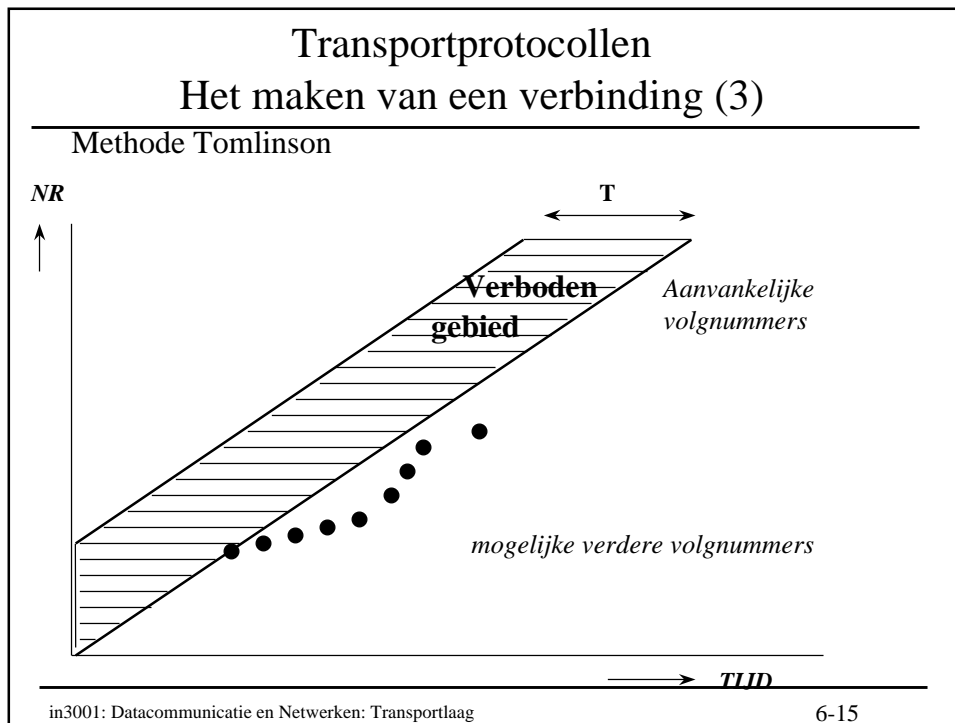
### Het maken van een verbinding (2)

---

Unieke nummering bij eindige levensduur (T) van TPDU's

*Methode van Tomlinson*, basisprincipes:

- ieder systeem heeft een TOD klok
- klok loopt door als systeem down
- TPDU's krijgen nummers
- nummer van eerste TPDU gebaseerd op laatste k-bits van waarde van TOD-klok
- nummer van volgende TPDU steeds 1 hoger
- *verboden gebied* zorgt ervoor dat nooit 2 TPDU's kunnen bestaan met hetzelfde nummer en van dezelfde afzender



### Transportprotocollen

#### Het maken van een verbinding (4)

---

Bij verzenden van een TPDU eerst te controleren of deze niet in het verboden gebied valt.

Dit kan als gevolg van:

- te snel aanbod van TPDU's
- te traag aanbod van TPDU's

---

in3001: Datacommunicatie en Netwerken: Transportlaag 6-16

## Transportprotocollen

### Het maken van een verbinding (5)

---

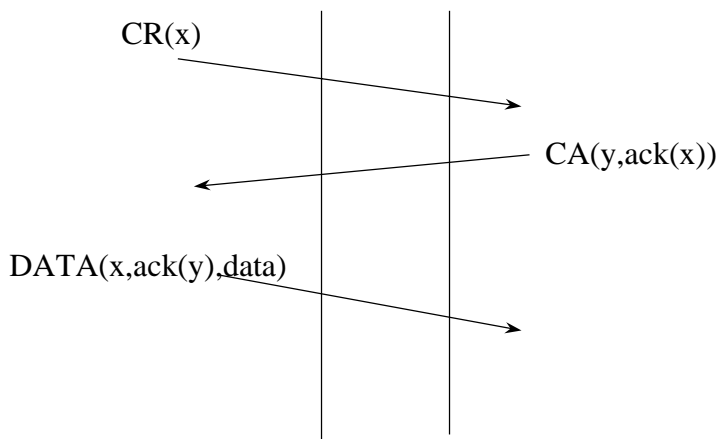
- **Probleem:** hoe worden beide partijen het eens over de te gebruiken nummers?
  - oplossing: *Drievoudige handdruk (three-way handshake)*
  - Een three-way handshake is een mechanisme waarbij 3 PDU's worden overgebracht.  
Stel A is de initiatiefnemer en B is de andere partij.
    1. Een verzoek van A naar B
    2. Een reactie van B op 1)
    3. Een reactie van A op 2)
  - Een vertraagde TPDU kan daardoor geen verbinding veroorzaken.

## Transportprotocollen

### Het maken van een verbinding (6)

---

Drievoudige handdruk, scenario zonder fouten



Tijd

## Transportprotocollen het verbreken van verbindingen (1)

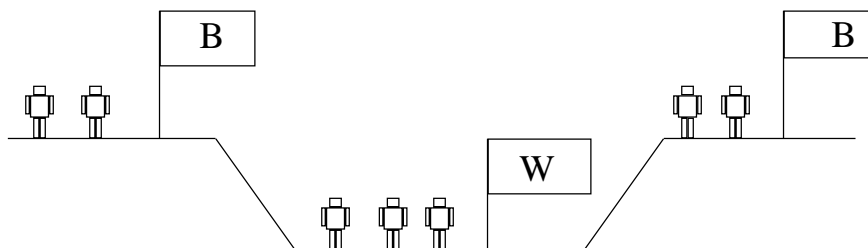
---

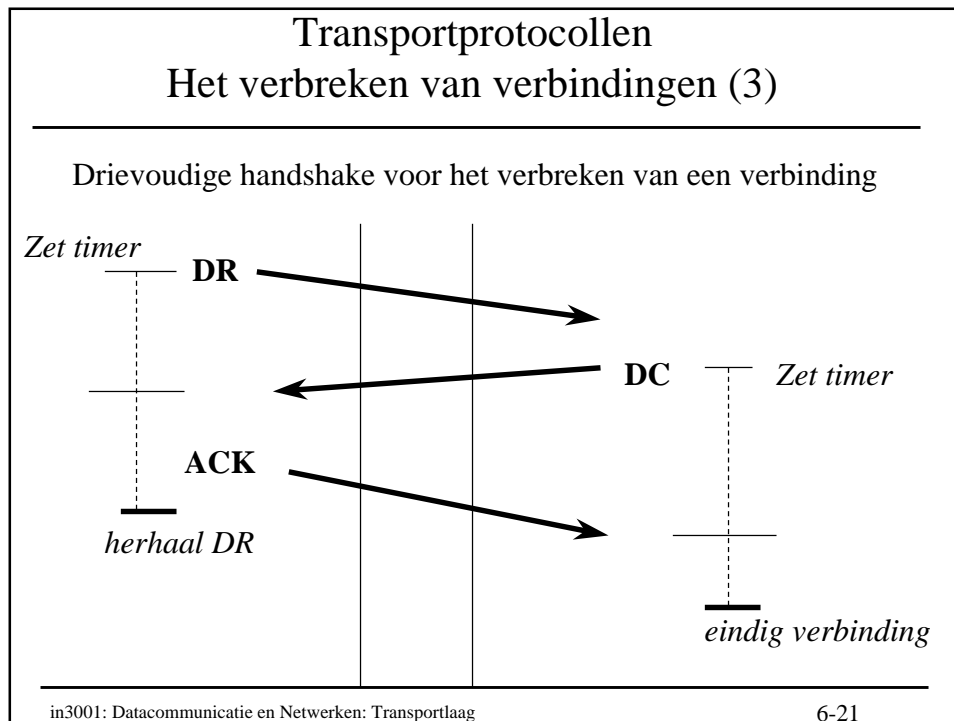
- Asymmetrisch: één partij geeft DISCONNECT .
  - probleem: mogelijk verlies van data
- Symmetrisch: beide partijen moeten het eens zijn.
  - probleem: Het twee-leger probleem
  - redelijke oplossing: three-way handshake met time-out.

## Transportprotocollen Het verbreken van verbindingen (2)

---

Het twee-leger probleem





### Buffering en Flow control

---

- vergelijkbaar met datalinklaag
- overeenkomst: glijdend venster
- verschil router heeft weinig lijnen, een host heeft veel verbindingen
- voor de transportentiteiten geldt:
  - Bij onbetrouwbare netwerkdienst: zender moet TPDU's altijd bewaren, totdat zij bevestigd zijn
  - bij betrouwbare netwerkdienst:
    - als de ontvanger garandeert dat TPDU's worden geaccepteerd : zender hoeft **niet** TPDU's te bewaren
    - anders **wel**

---

in3001: Datacommunicatie en Netwerken: Transportlaag 6-22

## Buffers en Flow Control netwerkcapaciteit

---

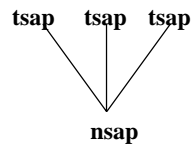
- Flow control primair bedoeld om ontvanger te beschermen tegen te snelle zender
- Zender moet ook afgeremd worden wanneer hij meer dreigt te zenden dan capaciteit van het subnet toelaat
  - zender moet maximaal uitstaande TPDU's (window size) aanpassen aan capaciteit van het subnetwerk
  - capaciteit van subnet is  $c$  TPDU's per seconde
  - 'cyclustijd' van subnet is  $r$  seconde
  - window size  $w = cr$
  - $c$  en  $r$  variëren en moeten regelmatig (door de zender) bepaald worden.

## Multiplexing

---

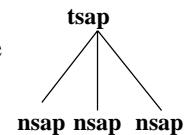
- Opwaartse multiplexing:  
meerdere transportverbindingen over 1 virtueel circuit

- aspecten
  - kosten
  - respons
  - id nodig in transportheader



- Neerwaartse multiplexing:  
één transportverbinding gebruikt meerdere virtuele circuits

- aspecten
  - grotere bandbreedte (mits onderliggende lagen voldoende)



## Herstel na storingen (1)

---

- storing in netwerklaag (verbinding weg, of datagrammen zoek), eenvoudig op te lossen door transportlaag
- host down is moeilijk (transportlaag in host)
  - mogelijke toestanden van andere partij op het moment van down gaan:
    - T1: wel uitstaande TPDU's
    - T0: geen uitstaande TPDU's
  - mogelijke acties bij het weer opkomen hangen samen met de manier waarop berichten vanuit de transportentiteit worden doorgegeven aan de transportgebruiker:
    - eerst bericht doorgeven, daarna ACK sturen, of
    - eerst ACK sturen en daarna bericht doorgeven

## Herstel na Storingen (2)

---

Mogelijke acties van zender, na down en herstart van de ontvanger:

Ontvangende host AW (eerst ACK, daarna Write naar appl.

of WA (eerst naar appl., daarna ACK)

C = Crash, geeft aan waar Crash optreedt

Zender \ Ontvanger	C(AW)	AC(W)	AWC	C(WA)	WC(A)	WAC
Altijd R	O.K.	O.K.	dubbel	O.K.	dubbel	dubbel
nooit R	kwijt	kwijt	O.K.	kwijt	O.K.	O.K.
R als in T0	kwijt	O.K.	dubbel	kwijt	O.K.	dubbel
R als in T1	O.K.	kwijt	O.K.	O.K.	dubbel	O.K.

R betekent: retransmissie van laatst verzonden TPDU

## Herstel na storingen (3)

---

- Bij host down is er geen strategie die onder alle omstandigheden goed werkt.
- Algemeen: Echte end-to-end bevestiging is niet te bereiken. (d.w.z. bevestiging betekent: het werk is gedaan, uitblijven betekent: het werk is niet gedaan)

## Representatie van transportprotocollen (1)

---

Een eenvoudig voorbeeld

aanname:

- netwerkdiensten verbindingsgericht, foutloos
- Transportentiteit gerealiseerd als library functies in de user address space.

5 primitieven (LISTEN, CONNECT, etc)

6 types pakketten (Call-req, DataPkt, etc)

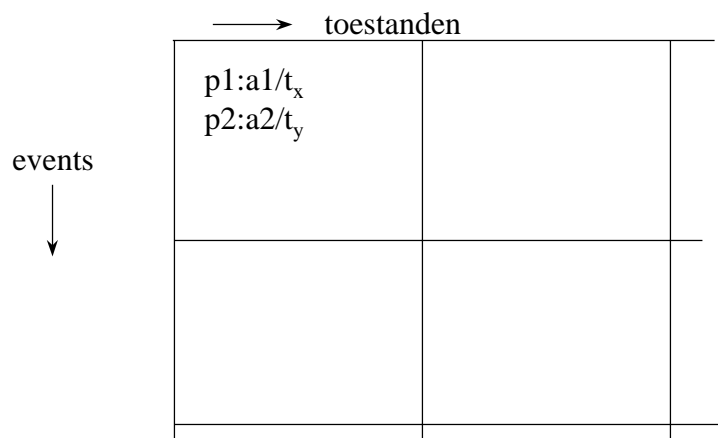
1 Timeout

geeft in totaal 12 mogelijke events

7 mogelijke toestanden (IDLE, WAITING, etc)

*Het protocol is op verschillende manieren te representeren*

## Representatie van transportprotocollen (2) Eindige toestandsmachine in matrixvorm

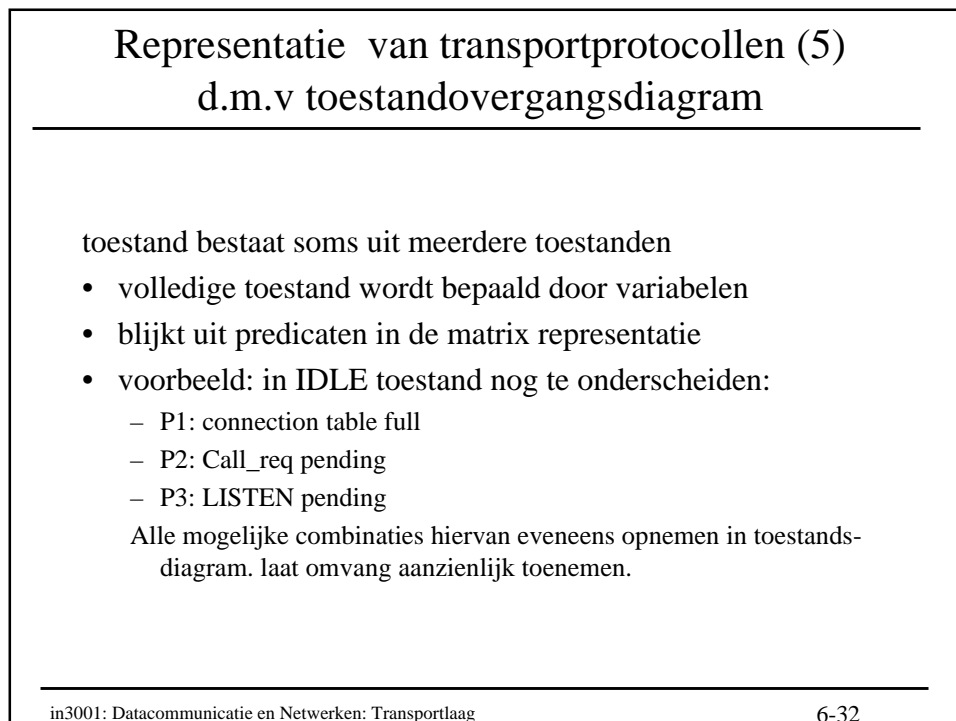
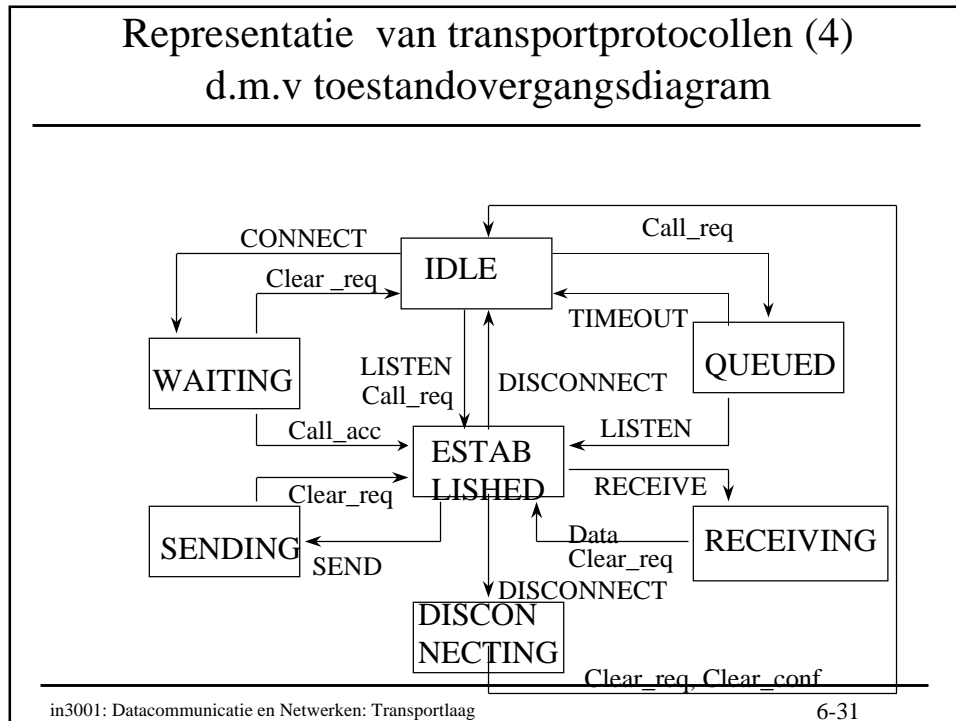


$p_i$  predikaat; geeft voorwaarde,  $a_i$  uit te voeren actie  
 $t_i$  toestand, de nieuwe toestand

## Representatie van transportprotocollen (3) Eindige toestandsmachine in matrixvorm

voordelen van de matrixrepresentatie:

- volledig
- eenvoudige implementatie
- (soms) aansluiting op documentatie van de standaard



## Representatie van transportprotocollen (6) d.m.v toestandovergangsdigram

---

Voordelen van representatie d.m.v. diagram:

- mogelijke gang van zaken beter te volgen

Nadelen:

- moeilijker beoordelen van de volledigheid
- moeilijker implementatie

## UDP (User Data Protocol)

---

- UDP is het verbindingsloze transportprotocol in Internet
- nuttig als er maar één bericht verzonden hoeft te worden.
- applicatie moet zelf nagaan of bericht is aangekomen.
  - (goed mogelijk als het om een request-respons toepassing gaat)
- UDP header (8 bytes)
  - bron poort
  - doel poort
  - lengte
  - checksum

## RPC (Remote Procedure Call)

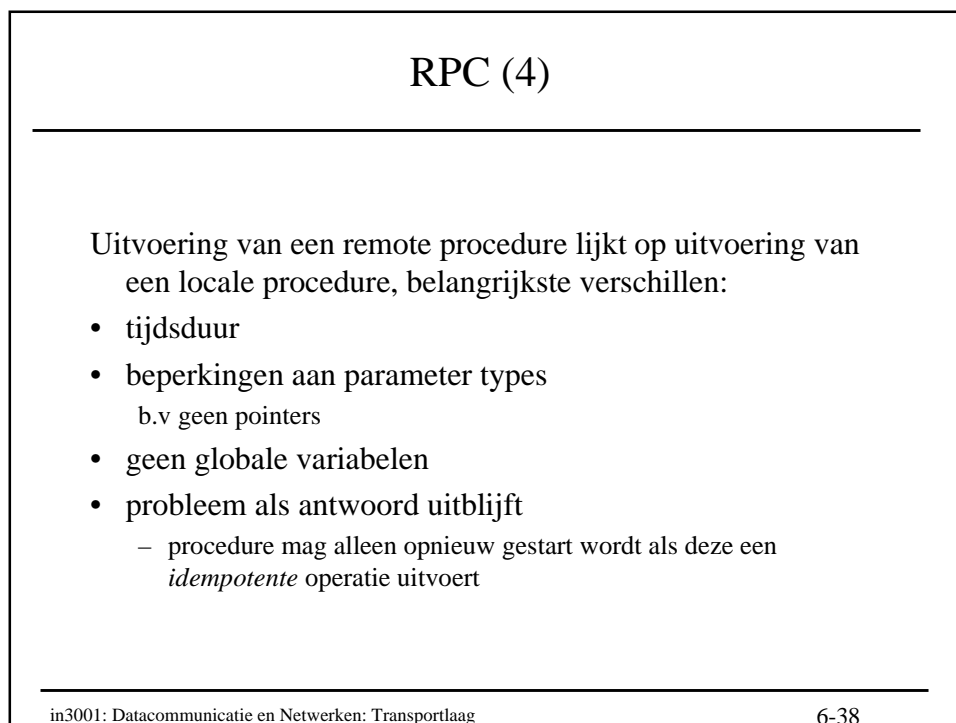
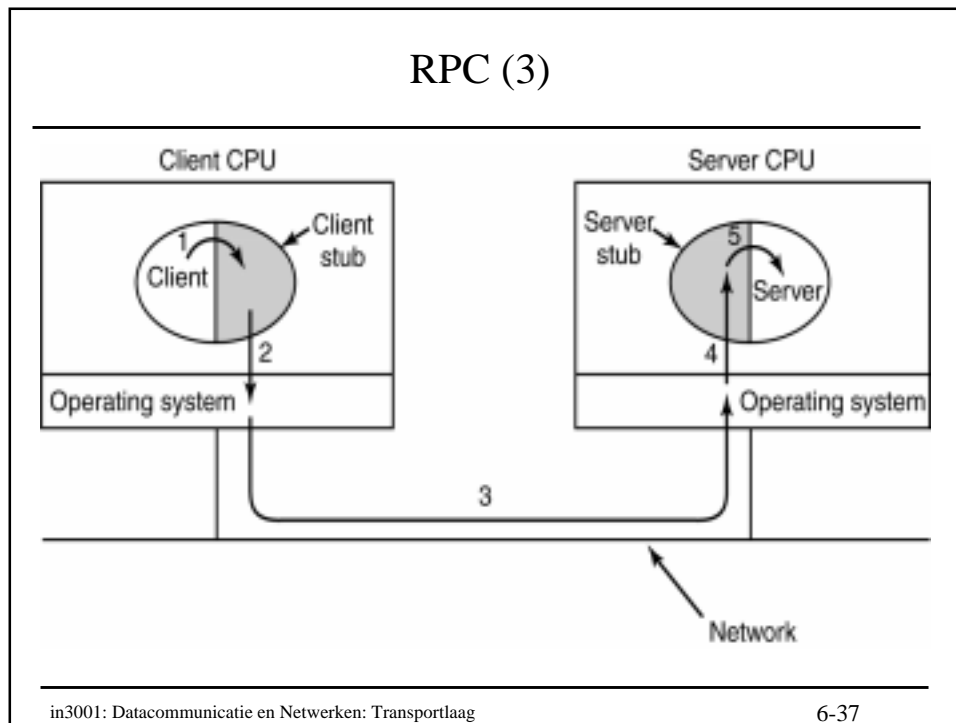
---

- maakt het mogelijk functies in andere machines aan te roepen alsof het in de eigen address space is.
- past in client/server model
- gebruikt UDP of TCP
- het aanroepende programma (de client) bevat:
  - een normale functie aanroep
  - een stub (de client-stub) in plaats van de functie  
de stub zorgt voor
    - *marshalling* (het “verpakken van de parameters”)
    - creatie van een bericht (waarin de parameters) voor de server
    - het verzenden naar de server
    - het uitpakken en doorgeven van het resultaat van de functie, de aanroeper is geblokkeerd tot het resultaat er is

## RPC (2)

---

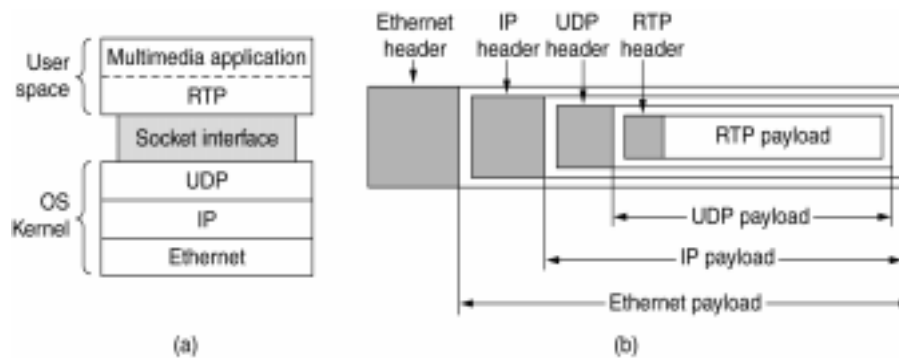
- De server bevat
  - de aangeropen functie
  - stub (server stub), draagt zorg voor
    - het uitpakken van de parameters
    - het aanroepen van de functie
    - het verpakken van het resultaat
    - het versturen van het resultaat naar de client



## RTP (Real-time Transport Protocol)

- Voor Real time multimedia toepassingen
- gebruikt UDP
- werkwijze:
  - verschillende streams worden gemultiplexed en gecodeerd in RTP pakketten
  - RTP pakketten worden verpakt in UDP pakketten
  - UDP stroom unicasting of multicasting
  - RTP pakketten genummerd
  - inhoud van ontbrekende pakketten “schatten” door interpolatie
  - verschillende profielen mogelijk, bv GSM encoding, MP3 etc.
  - time stamping relatief t.o.v. begin, bevordert correct afspelen
- RTCP (Real-time Transport Control Protocol)
  - voor besturing (feedback)

## RTP (2)



## TCP introdactie

---

- TCP (Transmission Control Protocol) ontstaan als het verbindingsgerichte transportprotocol van ARPA (DoD)
- TCP/IP protocolsuite standaard binnen Internet
- TCP beschreven in RFC's: 793 (1981), 1122 (1989), 1323 (1992)
- Verbindingsloze transportprotocol heet UDP (User Datagram Protocol)

## TCP service

---

- verbinding tussen sockets
  - socket komt overeen met TSAP , is:  
IP adres + poort ; (32 bits + 16 bits)
  - 'well-known' ports voor standaard services (RFC 1700)
    - b.v. FTP = poort 21; TELNET = poort 23
- full duplex
- point-to-point (geen multicast, geen broadcast)
- byte stream (geen message stream)
- PUSH flag, d.w.z. verzend nu
- *urgent data* flag t.b.v. interrupts (b.v. ^C)

## TCP protocol (1)

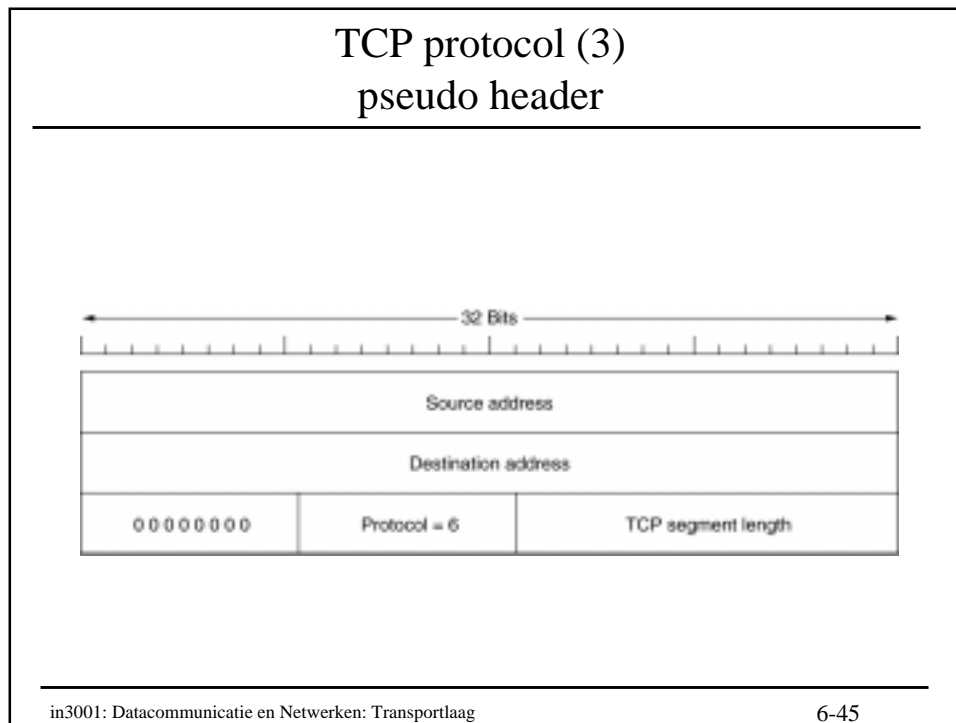
---

- bytes zijn genummerd (32 bits)
- data wordt verzonden in *segmenten*.
  - (segment bevat header + data)
  - segmentgrootte beperkt door
    - IP-payload (64K)
    - maximum transfer unit (MTU) per netwerk (b.v. 2K)
- flow control d.m.v. sliding window protocol met variabele grootte
  - complicaties:
    - segmenten kunnen gefragmenteerd zijn
    - segmenten kunnen in de verkeerde volgorde aankomen
    - opnieuw verzonden segmenten kunnen anders gefragmenteerd zijn

## TCP protocol (2)

---

- segment header
- vast deel, bevat o.a.
  - bron- en doelpoort
  - bytenummer (van eerste byte in dit segment)
  - ACK-nummer (bevestigt alle voorgaande bytenummers)
  - URG pointer
  - PSH bit
  - SYN bit, FIN bit, ACK bit
  - Checksum (houdt rekening met pseudoheader)
  - options:
    - max TCP payload (default 536)
    - window size scale factor
    - *selective repeat* i.p.v. *go-back-n* bij hertransmissie



### TCP verbinding maken/opheffen

---

- Verbindingen maken d.m.v. 3-way handshake  
(Bij call collision één verbinding)
  - nummering starten m.b.v. klok
  - maximale packet lifetime 120 seconden
- full duplex verbinding verder te beschouwen als 2 simplex verbindingen.
- Elke simplex verbinding te beëindigen door FIN-bit
- Verbinding opgeheven als beide simplex verbindingen beëindigd.

---

in3001: Datacommunicatie en Netwerken: Transportlaag 6-46

## TCP

### Transmissie policy

---

- Basis: sliding window protocol met variabele window size
- probleem bij interactieve toepassingen, b.v. bij editor,
  - verbetering: Nagles algoritme
    - Eerste byte meteen verzenden, volgende bytes bufferen tot ACK ontvangen is. (niet geschikt voor muis)
- probleem: Silly window syndroom
  - d.w.z. ontvanger neemt iedere keer 1 byte van buffer en past window size met 1 aan.
  - oplossing van Clark: ontvanger stuurt alleen nieuw window size, wanneer hij voldoende vrije ruimte heeft.

## TCP

### congestion control (1)

---

- aanname:  
transmissie time-outs veroorzaakt door congestie
- congestie control :  
door afremmen van de zenders die te maken hebben met transmissie time-outs
- methode:
  - naast window t.b.v. ontvanger, ook window t.b.v. congestion control
  - kleinste van beide windows wordt gebruikt

## TCP congestion control (2)

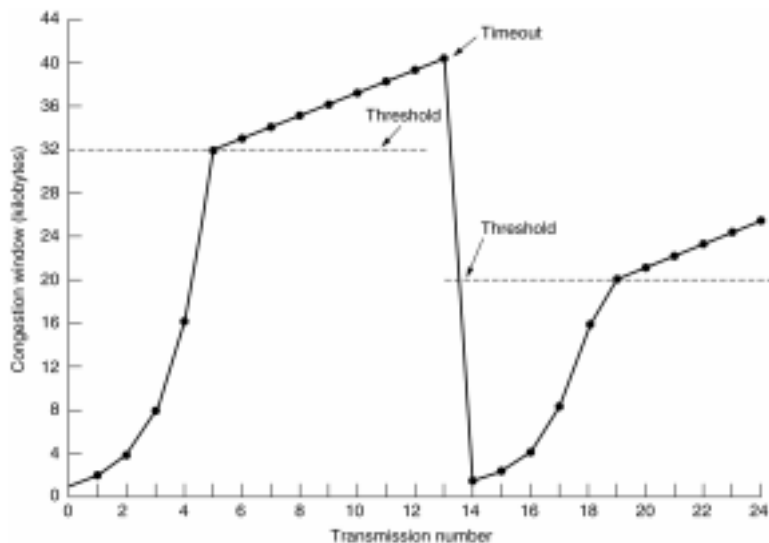
---

Bepaling van window t.b.v congestion control:

- start met window size = eenmaal max segment size
- als ACK binnen vóór timeout, verdubbel window size
- verdubbel steeds als er geen timeout is,  
(dit heet het *slow start algoritme*)
- als er een timeout optreedt of een *threshold* waarde wordt bereikt:
  - bij timeout: halveer threshold en begin opnieuw met slow start
  - bij threshold: verhoog window size steeds met vast bedrag tot window t.g.v ontvanger is bereikt

## TCP congestion control (3)

---



## TCP Timer management

---

### A. Retransmissie timer ( $T_R$ )

- wordt gezet bij verzenden van een pakket. Als ACK niet binnen voor aflopen van timer, dan retransmissie
- Baseren op Round Trip Time (RTT)
- dynamisch aanpassen ( *exponentieel gemiddelde* )  
*Jacobson's algoritme*
- $RTT := \alpha RTT + (1 - \alpha) M$   
M is laatst gemeten RTT  
D is gemiddelde afwijking, ook dynamisch aanpassen:  
 $D := \alpha D + (1 - \alpha) | RTT - M |$
- timeout waarde =  $RTT + 4 * D$
- Als timeout optreedt RTT verdubbelen (*Karn's Algoritme*)

## TCP timer management

---

### B. persistentie timer ( $T_p$ )

Als zender bij window size 0, gedurende tijd  $T_p$  niets van de ontvanger heeft gehoord, zendt zender een verzoek om de window size mee te delen (anders misschien deadlock)

### C. Keep-alive timer ( $T_A$ )

Als een verbinding gedurende tijd  $T_A$  inactief is geweest, wordt door één zijde aan de andere kant gevraagd of deze er nog is.

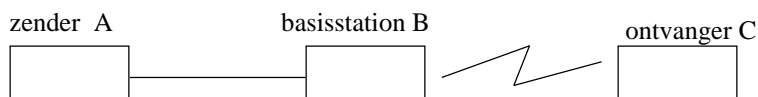
### D. timed wait state ( $T_C$ )

Bij een close wordt 2 maal de maximum packet lifetime gewacht of er nog een TPDU onderweg is.

## Draadloos TCP

---

- TCP protocol functioneert correct in bij draadloze verbinding
- probleem: performance
  - oorzaak: congestie control werkt averechts
  - Pakket verlies nu meestal **niet** door congestie
  - bij pakketverlies meer pakketten zenden I.p.v. vertragen
- probleem: inhomogene paden: half kabel, half draadloos



## Draadloos TCP

---

### Oplossingen voor inhomogeen pad:

- indirect TCP: splits verbinding in 2 TCP-verbindingen
  - bij timeouts over AB, moet A vertragen
  - bij timeouts over BC, moet B snel opnieuw zenden
  - nadeel: bevestiging door B (naar A) betekent niet dat C het segment ontvangen heeft.
- Uitbreiden netwerklaag in B met *snooping agent*
  - snooping agent bewaakt de overdracht van B naar C, zorgt eventueel voor retransmissie
  - nadeel: kans op timeout van A en ten onrechte aanname van congestie

## Draadloos UDP

---

- UDP functioneert correct bij draadloze verbindingen
- probleem: UDP wordt minder betrouwbaar
- gevolg: applicatie heeft vaker te maken met verloren berichten. Recovery kan duur zijn.

## Performance problemen

---

- congestie
- onbalans
  - systeem te traag voor net
  - systeem slecht getuned
- synchrone overload
  - b.v. *broadcast storm*
  - massale reboot na stroomstoring
- window size
  - (ideaal window is produkt van bandbreedte en delay)
- *jitter* (is variatie in delay)

## Performance Meten (1)

---

interessante gegevens:

- Round Trip Time (RTT)
- aantal retransmissies (als percentage)
- effectieve bandbreedte (bits/s)

## Performance Meten(2)

---

Punten van aandacht

- sample size
- representativiteit
- klok resolutie
- controleerbare omgeving
- caching en buffering
- begrijpen wat je meet
- oppassen met extrapolatie

## Performance Design aspecten

---

- CPU snelheid vaak belangrijker dan netwerkcapaciteit
- beperk aantal TPDU's
- minimaliseer aantal contextswitches
- buffers zo weinig mogelijk kopiëren
- congestie voorkomen is beter dan genezen
- time-out waarden conservatief kiezen
- software van communicatiefuncties optimaliseren op de meest voorkomende situaties ( normale foutloze overdracht)

## Giga-bit netwerken (1)

---

### Problemen:

- sequencenummers te klein
- communicatiesnelheid is sneller toegenomen dan CPU snelheid
- go-back-n slecht bij verbindingen met groot bandbreedte-delay produkt
- bij grote bandbreedte is de delay de beperkende factor
- voor nieuwe toepassingen (multimedia ) is de variatie in de delay belangrijker dan een lage gemiddelde delay.

## Giga-bit netwerken (2)

---

### Aanbevelingen voor oplossingen

Uitgangspunt: bandbreedte is geen probleem meer

- protocol processing moet snel
  - speciale hardware (alleen voor eenvoudige protocollen)
  - software richten op performance
- terugkoppeling vermijden
  - sliding window protocol vervangen door gegarandeerde capaciteit
  - slow start algoritme (Jacobson) vervangen door het reserveren van resources door zender, ontvanger **en netwerk**

## Giga-bit netwerken (3)

---

### Aanbevelingen (vervolg)

- packet layout aanpassen
  - velden op gemakkelijke grenzen
  - velden groot genoeg voor nieuwe situatie (b.v. grotere sequence nummers)
  - aparte checksums voor header en data
  - maximale datasize groot
- eerste data meezenden met connectie request