Transportlaa

AP HOGESCHOOL ANTWERPEN

AP.BE

Transportlaag

Doelstellingen van dit cursusonderdeel:

- Begrijpen van de basisprincipes van de transportlaag:
  - Multiplexing en demultiplexing
  - Reliable data transfer
  - Flow control
  - Congestion control

- Begrijpen van de twee voornaamste transportlaag protocols op het internet:
  - UDP: connectionless transport, unreliable
  - TCP: connection-oriented transport, reliable

Transportlaag

- Steunt op de diensten van de netwerklaag (communicatie tussen hosts)
- Voegt extra functionaliteiten toe aan de diensten van de netwerklaag
- Logische communicatie tussen processen op verschillende hosts die met mekaar communiceren
- Transport protocols zijn geïmplementeerd in de end-systemen die met mekaar communiceren
  - **Zender**: boodschappen opdelen in segmenten + doorgeven aan de netwerklaag
  - **Ontvanger**: segmenten samenvoegen en boodschap doorgeven aan applicatielaag

Transportlaag protocols

- Betrouwbare, in-volgorde aflevering: TCP (Transmission Control Protocol)
  - Connection setup

  → opzetten van een verbinding vooraleer data te verzenden
  - Error control

  → detectiemechanismen om fouten in de headers + payloads op te sporen
  - Retransmissie

  → Herzending van corrupte of niet-toegekomen segmenten
  - Flow control

  → mechanismen om de zendsnelheid te begrenzen conform de capaciteiten van de ontvanger
  - Congestion control

  →Mechanismen om de zendsnelheid te begrenzen conform de capaciteiten van het netwerk

- Onbetrouwbaar, niet-in-volgorde aflevering: UDP (User Datagram Protocol)
  - Error control

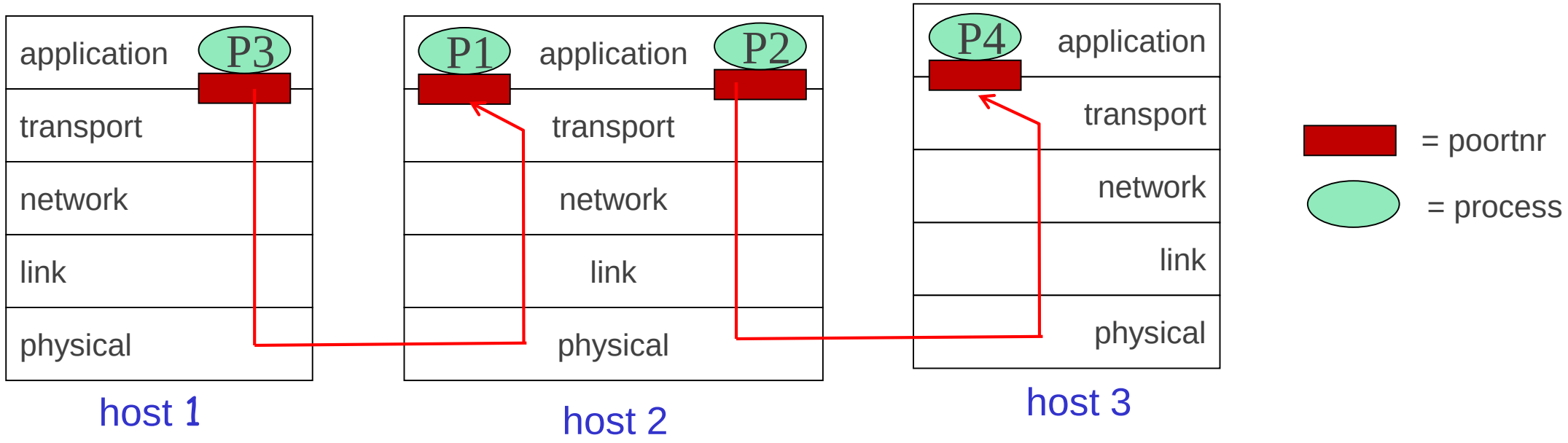
Multiplexing – Demultiplexing (1)

Multiplexing (zender):

Data van verschillende processen verpakken tot segmenten met juiste headerinfo

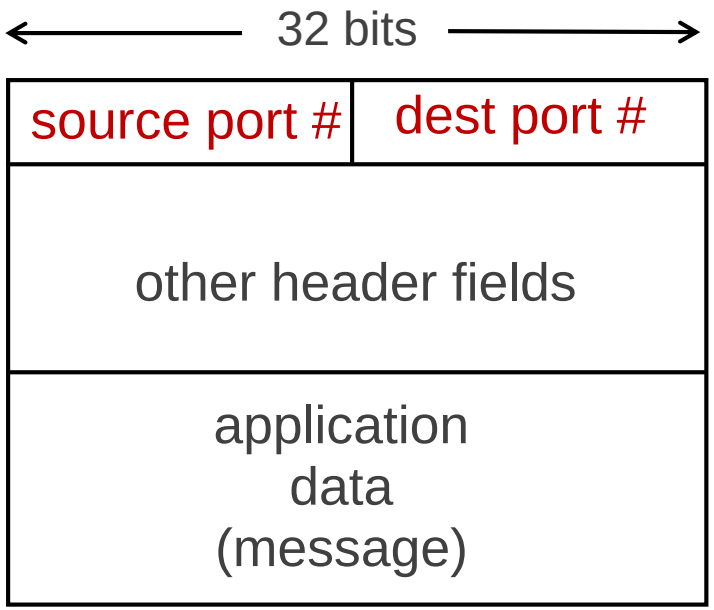
Demultiplexing (ontvanger):

Segmenten bij de juiste Processen afleveren



Multiplexing – Demultiplexing (2)

- Host stuurt/ ontvangt geen segmenten
  - Transportlaag-segmenten worden aan de netwerklaag doorgegeven om er adresinfo aan toe te voegen (IP adressen)
- Netwerklaag vormt transportlaag segmenten om tot datagrams door header info toe te voegen
  - Elk datagram bevat een bron IP adres en een bestemmings IP adres
  - Elk datagram bevat een transportlaag segment als payload
  - Elk transportlaag segment bevat een bron poort en een bestemmings poort
- Hosts gebruiken de gecombineerde info van IP adressen & poort nummers **(= sockets)** om aan te geven bij welk applicatielaag proces een datastroom hoort



TCP/UDP segment format

Multiplexing – Demultiplexing (3)

Connection oriented communicatie (via TCP als transportlaag protocol)

- TCP socket wordt bepaald door 4 parameters:
  - Bron IP
  - Bron TCP poort
  - Bestemmings IP
  - Bestemmings TCP poort
- Ontvanger gebruikt alle 4 de parameters om éénduidig 1 socket te identificeren, en dus de juiste processen (=applicatielaag) op zender en ontvanger met elkaar in contact te brengen.

Connection-less communicatie (via UDP als transportlaag protocol)

- UDP socket wordt bepaald door twee parameters:
  - Bestemmings IP
  - Bestemmings poort
- IP datagrammen met verschillende bron IP's en bron poorten kunnen bij dezelfde ontvangende socket terechtkomen.

UDP – User Datagram Protocol (1)

- Best effort: segement kunnen verloren gaan
- Connectieloos: Geen connectie setup tussen zender en ontvanger.

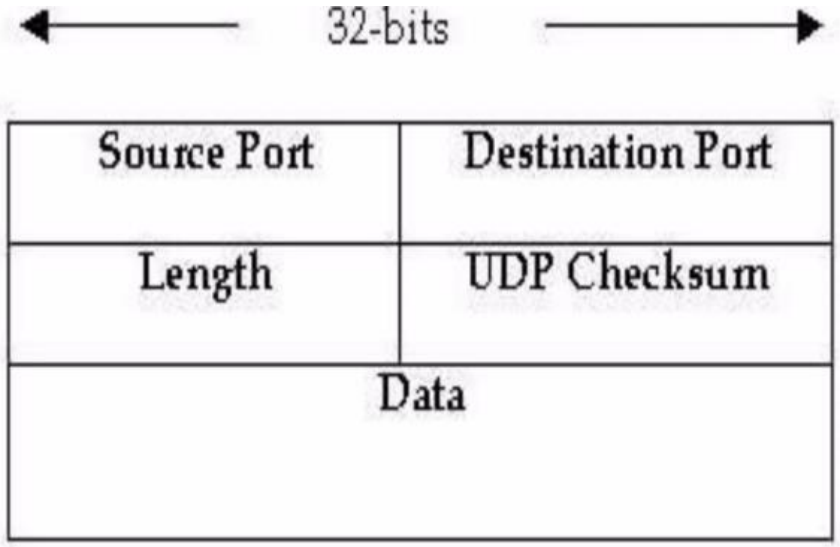
  → Elk UDP segment wordt onafhankelijk van de andere behandeld

Bestaansreden voor UDP?

- Geen connectie setup (dus minder vertraging)
- Simpel: er wordt geen status bijgehouden, noch bij zender, noch bij ontvanger
- Kleine segment header (dus minder overhead)
- Geen flow en congestion control → UDP kan verzenden zo snel als het voor de zender kan en hoeft zich niets aan te trekken van de capaciteiten van de ontvanger of van het netwerk (dus hogere performantie kan behaald worden)
- Kan gebruikt worden bij broadcasting/multicasting (TCP niet)

UDP – User Datagram Protocol (2)

- Vaak gebruikt voor:
  - Verlies-tolerante toepassingen
  - Snelheidsgevoelige toepassingen
  - Bvb. streaming applicaties
- Andere gebruiken omwille van redenen van eenvoud en low-overhead:
  - DHCP (omwille van broadcasting noodzaak)
  - DNS
  - SNMP
- Betrouwbaarheid moet ingebouwd worden op applicatie-niveau
  - Betrouwbaarheid = zekerheid dat wat verstuurd werd ook daadwerkelijk aankomt bij de ontvanger
  - Heeft niets te maken met security ...



TCP – Transmission Control Protocol kenmerken

- Point-to-point / unicast:
  - 1 zender, 1 ontvanger
- Reliable, in-volgorde byte stream service:
  - Negeert grenzen van boodschappen van de applicatie en bekijkt data als een stroom bytes
- Pipelined:
  - TCP congestion and flow control window sizing
- Zend en ontvang buffers
  - Zendbuffers om reeds verzonden opnieuw te kunnen terughalen als herzending nodig is
  - Ontvangbuffers om ontvangen segmenten te stockeren en ze te processen alvorens ze door te geven aan de applicatielaag
- Connection-oriented:
  - Verbinding op voorhand opzetten
  - 3-way handshake setup
- Flow control:
  - Capaciteiten van ontvanger respecteren
- Congestion control
  - Capaciteiten van het netwerk respecteren
- Full duplex data-transfer

TCP – Segment formaat

U: urgent flag =1 → moet onmiddellijk beantwoord worden.

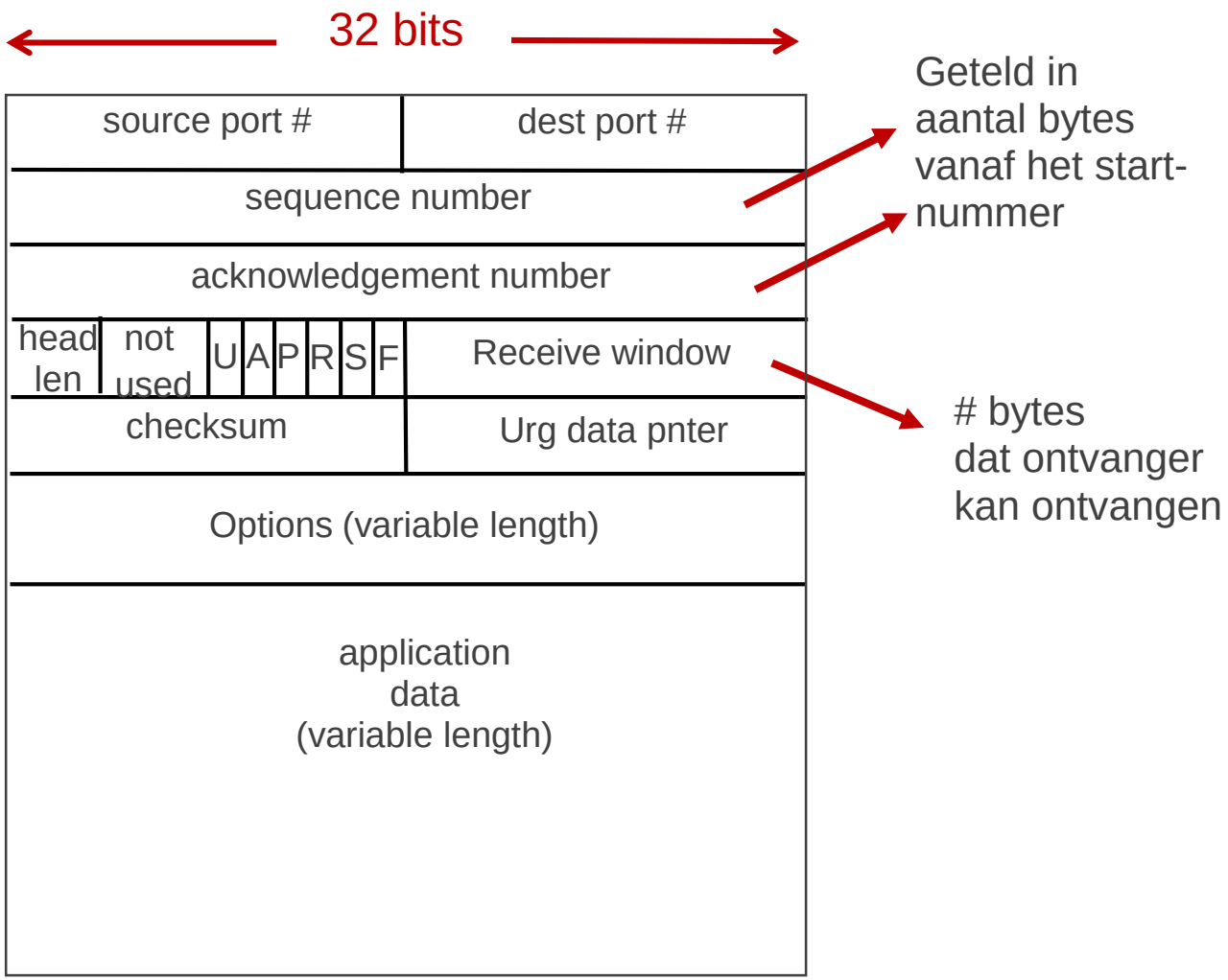
A: ACK = 1→ ACK nr geeft weer tot hoever datastroom correct ontvangen.

P: push = 1 → segment onmiddellijk doorgeven aan hogergelegen applicatie.

R: Reset = 1 → reset de verbinding.

S: SYN = 1→ opzetten en synchroniseren van een verbinding

F: FIN = 1 → Verbinding afbreken

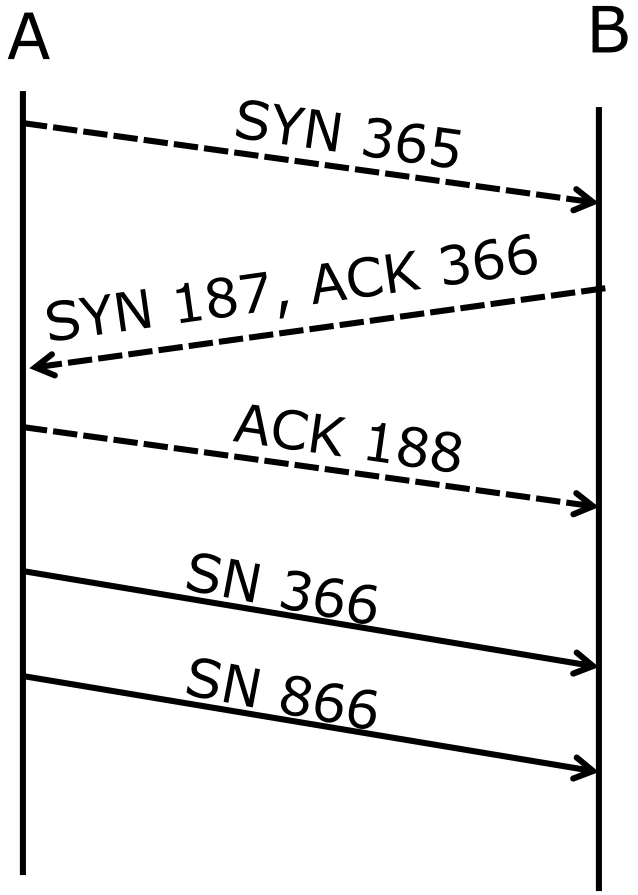


TCP – Opzetten van de verbinding

- Geïnitieerd door client
- Kijken of server kan/wil communiceren
- Afspreken van volgnummers om segmenten aan te duiden
  - Volgnummering vanaf random startnummer laten beginnen
  - Volgende volgnummers = startnummer + aantal reeds verzonden bytes
  - Zowel zender als ontvanger kiezen een startnummer aangezien communicatie full duplex kan zijn (beiden zijn zowel zender als ontvanger)
- Bestaat uit 3 stappen → 3-way handshake

TCP – 3 way handshake

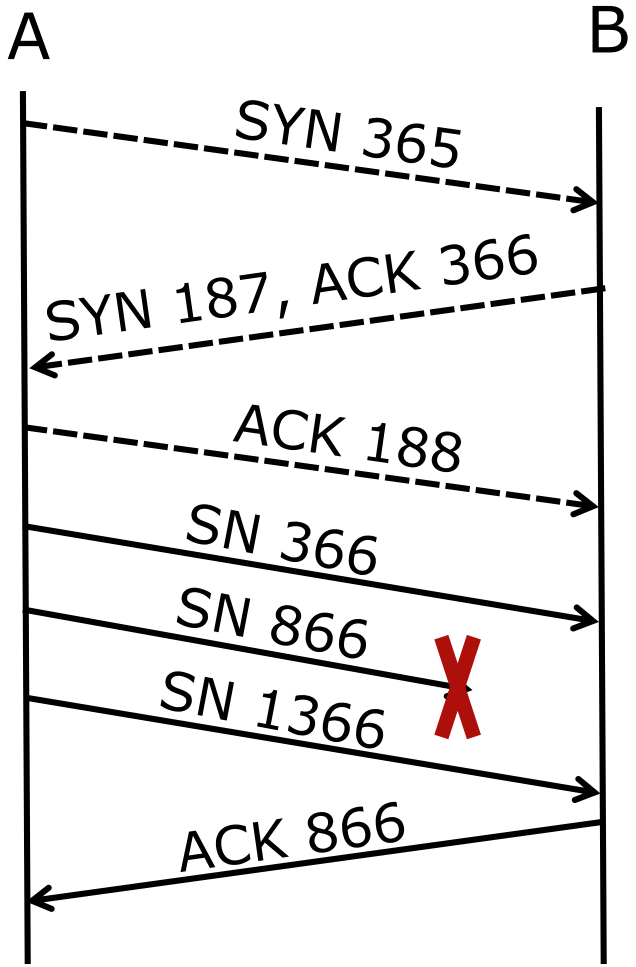
- Client A contacteert server B met een SYN boodschap
  - A maakt zijn keuze van startnummer bekend
  - A kiest om te beginnen tellen vanaf startnummer 365
  - Deze SYN is dus segment nummer 365
- B is bereid om te communiceren en stuurt via SYN ook zijn keuze van startnummer door.
  - B kiest ervoor om te beginnen tellen vanaf 184
  - Via ACK piggy-backing bevestigd B ook ineens dat hij nu het segment met nummer 366 verwacht
    - → Dit wil zeggen dat 365 goed is ontvangen!
- Via ACK bevestigt A dat hij nu segment 188 verwacht (en dat dus 187 goed ontvangen werd)
- A zendt nu segment 366, en dat bestaat uit 500 bytes
  - Het volgende segment dat bij A vertrekt heeft dan volgnummer 866
  - 366 + 500 = 866



TCP – betrouwbaarheid verzekeren (1)

- TCP is reliable (betrouwbaar)
  - Segmenten die verloren gaan: herzending nodig
  - Segmenten die corrupt toekomen: herzending nodig
- Zender dient reeds verzonden segmenten te onthouden in geval ze moeten herzonden worden
  - Buffers nodig als opslagruimte
- Hoe weet zender of er iets moet herzonden worden en wat er moet herzonden worden ?
  - Segmenten die verloren gaan: Ontvanger zal deze niet ACK-en. Als een timer aan zendkant verloopt → impliciete herzending
  - Segmenten die corrupt toekomen: Ontvanger zal enkel voorgaande segmenten ACK-en→ expliciete herzending
- Hoe weet zender wat er moet herzonden worden?
  - Aan de hand van ge-ACK-te volgnummers!

    → ACK 866 = “Ik heb de 500 bytes in 366 goed ontvangen, ik verwacht nu segment 866”



TCP – betrouwbaarheid verzekeren (2)

- Wat als response ACK's verloren gaan? Impliciete herzending
  - Timer aan zendzijde om te beslissen om tot herzending over te gaan wanneer er binnen de tijd geen bevestiging (=ACK response) van ontvangst komt
- Hoe lang moet timer lopen (time-out interval) ?
  - Afhankelijk van Round Trip Time (RTT)
  - Tijd tussen het verzenden van een segment en het ontvangen van de ACK dat het goed aangekomen is
- Mogelijk probleem als timer te klein staat afgesteld?
  - Onnodig dubbel gezonden segmenten als timer te snel afloopt = extra netwerkbelasting!
  - Onnodig dubbel ontvangen segmenten = onnodig werk voor de ontvanger
  - Ontvanger dient dit af te handelen via ontvangstbuffer om dubbels weg te filteren
  - Ontvangstbuffer kan ook gebruikt worden voor sortering van out-of-order segmenten (niet vaak gebruikt! Meestal out-of-order = weggooien)

TCP – betrouwbaarheid verzekeren (3)

- Zowel aan de kant van de zender als aan de kant van de ontvangers zijn buffers nodig om tijdelijk segmenten in te bewaren.
- Buffers hebben een beperkte opslagcapaciteit!
- Buffers kunnen dus vol geraken = buffer overflow = crash!
- Er moeten dus regelmatig segmenten verwijderd worden om overflow te vermijden
- Wanneer data verwijderen uit de buffers?
  - Aan zendzijde: segmenten verwijderen zodra hun goede ontvangst ge-ACK't werd
  - Aan ontvangzijde: regelmatig buffers leegmaken door segmenten door te geven aan de applicatielaag en door dubbels snel weg te filteren.

TCP – Verbindingen afbreken (1)

- Stap 1: A stuurt FIN segment naar B.

  → De connectie wordt in toestand “close” gezet aan de kant van A.

- Stap 2: B ontvangt FIN, antwoordt met FIN/ACK en sluit zijn sockets

  → De connectie wordt in toestand “close” gezet aan de kant van B.

- Stap 3: A ontvangt FIN/ACK

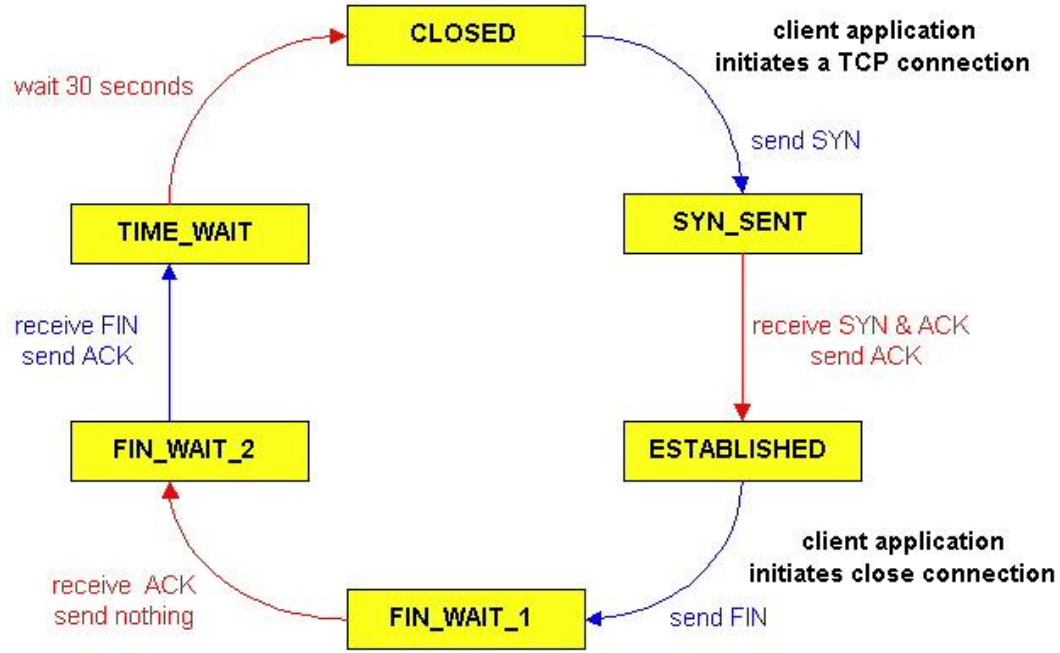
  → De connectie wordt in toestand “time-wait” gezet aan de kant van A.

  : A antwoordt met ACK en sluit zijn sockets

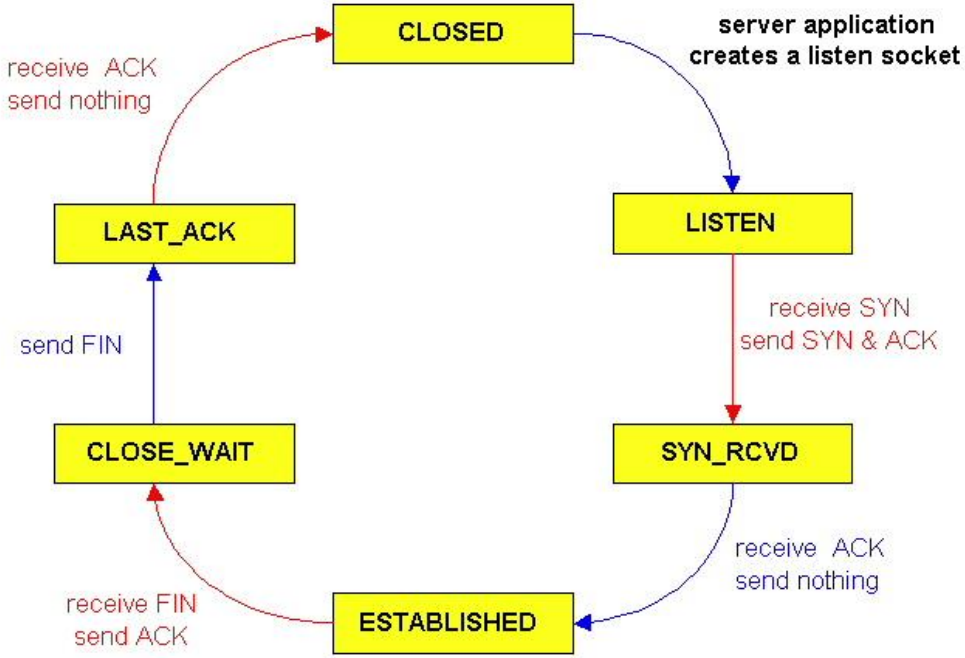
  → Na het verstrijken van een wachttijd wordt de connectie in toestand “closed” gezet aan de kant van A. De connectie is nu volledig weg.

TCP – Verbindingen afbreken (2)

- TCP connecties kunnen zich in verschillende toestanden bevinden.
- Na te kijken via tool netstat en in onderstaande schema's



TCP client connection lifecycle



TCP server connection lifecycle

TCP – Flow versus Congestion control (1)

- Congestion = Teveel zenders / teveel verzonden data / te snel verzonden data waardoor het netwerk de trafiek niet aankan.
- Symptomen van congestion:
  - Verloren paketten door buffer overflow bij routers
  - Grotere vertragingen (queueing in router buffers)
- Congestion control = verzonden hoeveelheid beperken zodat het netwerk niet overbelast geraakt
- Flow control = Verzonden hoeveelheid beperken zodat de ontvanger niet overbelast geraakt
  - Communicerende partijen maken hun ontvangstcapaciteit kenbaar via het window veld in de TCP header
  - Bij elk uitgewisseld segment kan de verzender van dat segment een andere window size meegeven om wisselende omstandigheden op te vangen (wisselende belasting)
  - Hosts moeten hun zendsnelheid afstemmen op de window size van de tegenpartij

TCP – Congestion control

Hoe komen communicerende partijen te weten dat het netwerk overbelast is?

1. End-end congestion control:
   - Geen expliciete feedback van het netwerk
   - Congestion wordt verondersteld door hosts wanneer ze veel verliezen en vertragingen opmerken in hun communicatie.
   - Ze vertragen hun zendsnelheid en vergroten hun resend timers als ze dat opmerken.
2. Network-assisted congestion control:
   - Routers geven feedback aan de end systems omtrent de netwerktoestand
   - Hosts moeten niets veronderstellen maar krijgen informatie waarop ze hun zendsnelheid en time-out intervallen kunnen afstemmen.

Vragen?