

# Limited Transmit

---

- ◆ RFC 3042: Enhancing TCP's Loss Recovery Using Limited Transmit.

M. Allman, H. Balakrishnan, S. Floyd. January 2001  
(Status: PROPOSED STANDARD)

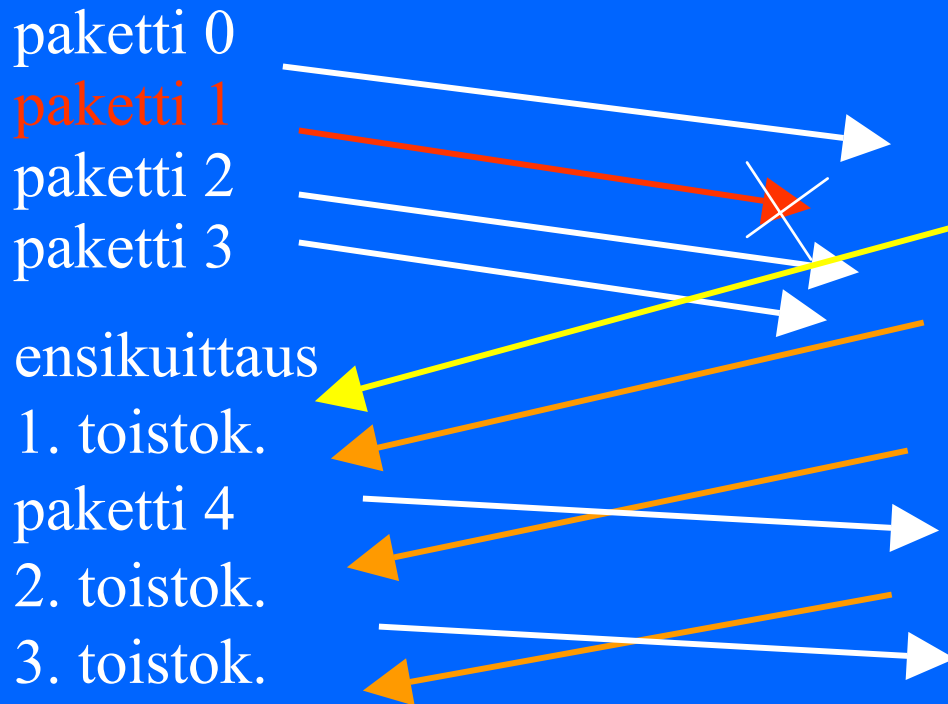
- ◆ Lähettäjä ei saa kolmea toistokuittausta =>
  - odotettava aina ajastimen laukeamista ja
  - suoritettava hidas aloitus
  - => hidastaa usein turhaan lähettämistä

# Ratkaisu:

---

- ◆ Lähettäjä saa lähettää yhden **uuden paketin** verkkoon vastaanotettuaan **1. ja 2. toistokuittauksen**.
  - » kuittaus kertoo, että verkosta poistettu paketti, joten verkkoon siis mahtuu!
  - » Tilapäisesti ruuhkaikkunan koko ylitetään kahdella MSS:llä
- ◆ Kun saman paketin toistokuittauksia tulee kolme, niin suoritetaan nopea uudelleenlähetyks ja nopea toipuminen (fast recovery)

- ◆ Vaikka ruuhkaikkuna on pieni, niin rajoitetulla lähetyksellä saadaan tarvittaessa syntymään kolme toistokuittausta



# Miksi lähetetään uusi paketti?

---

- ◆ Miksi ei heti ensimmäisen toistokuittauksen jälkeen lähetä uudestaan sitä jo lähetettyä kuittaamatonta pakettia?
- ◆ Koska ei vielä olla varmoja siitä, että paketti on todella kadonnut.
  - Se voi olla vain viivästynyt
  - tai paketit ovat matkalla joutuneet väärään järjestykseen
- ◆ => näin vältetään turhia uudelleenlähetystyksiä

# SACK (Selective Acknowledgement)

---

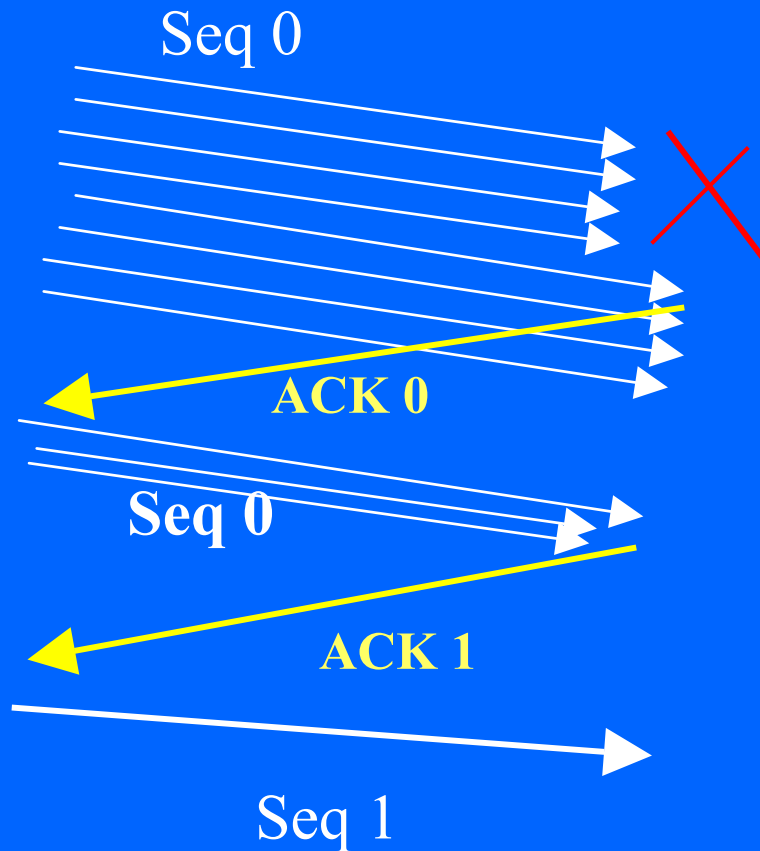
- RFC 2018  
TCP Selective Acknowledgement Options.  
M. Mathis, J. Mahdavi, S.Floyd, A. Romanow. October 1996.  
(Status: **PROPOSED STANDARD**)

## INTERNET DRAFT

Mark Allman, Ethan Blanton. "A Conservative SACK-based Loss Recovery Algorithm for TCP".  
(draft-allman-tcp-sack-02.txt), January, 2001

- **Valikoivien kuittauksien lisääminen TCP:hen**
  - TCP käyttää “Go Back N”-tyyppistä algoritmia ja kumulatiivista ACK-kuittausta
  - väärässä järjestyksessä saapuneet yleensä talletetaan

# Nopea toipuminen ei onnistu!



Segmentit häviävät

Joka kierroksella pystytään uudelleenlähettämään vain yksi kadonnut segmentti, koska **kuittaus paljastaa vain seuraavan puuttuvan.**

**\* jos lähetetään monta uudelleen => voi aiheutua turhia uudelleenlähetyksiä**

- ◆ Kumulatiivinen kuittaus paljastaa aina vain yhden puuttuvan kerrallaan

---

- ◆ SACK paljastaa kaikki puuttuvat

  - » ilmoittamalla, mitkä segmenttivalit on jo vastaanotettu

- ◆ Esim. Segmentin koko 1000 tavua

  - 1. segmentti katoaa ja muut tulevat perille

    - ◆ segmentin 2 kuittaus: ACK 0 , 1000: 1999

    - ◆ segmentin 10 kuittaus: ACK 0, 1000: 9999

  - 1. ja 3. segmentti katoavat

    - ◆ segmentin 10 kuittaus:

    - ◆ ACK 0, 1000:1999 3000:9999

# SACK-optiot

---

- ◆ **SACK-permitted** yhteyden muodostuksessa eli vain SYN-segmentissä ilmoittamaan, että yhteydellä voidaan käyttää SACK-kuittauksia
  - ◆ (type = 4, length = 2)
- ◆ **SACK-optio**
  - kuljettaa lisäinformaatiota saapuneista segmenteistä eli kertoo, mitkä ‘tavupätkät’ ovat jo valmiina vastaanottajan puskurissa
  - kuljetetaan TCP-segmentin optio-osassa



# TCP:n SACK-optio

Optiotyyppi

5	pituus
1. Lohkon alku	
1. Lohkon loppu	
2. Lohkon alku	
2. Lohkon loppu	
3. Lohkon alku	
3.lohkon loppu	

4 lohkoa mahtuu yhteen TCP-segmenttiin, jossa optiolle on varattu 40 tavua, jos ei käytetä muita optioita kuten aikaleimaa (timestamp).

# Vain neuvoa-antava!

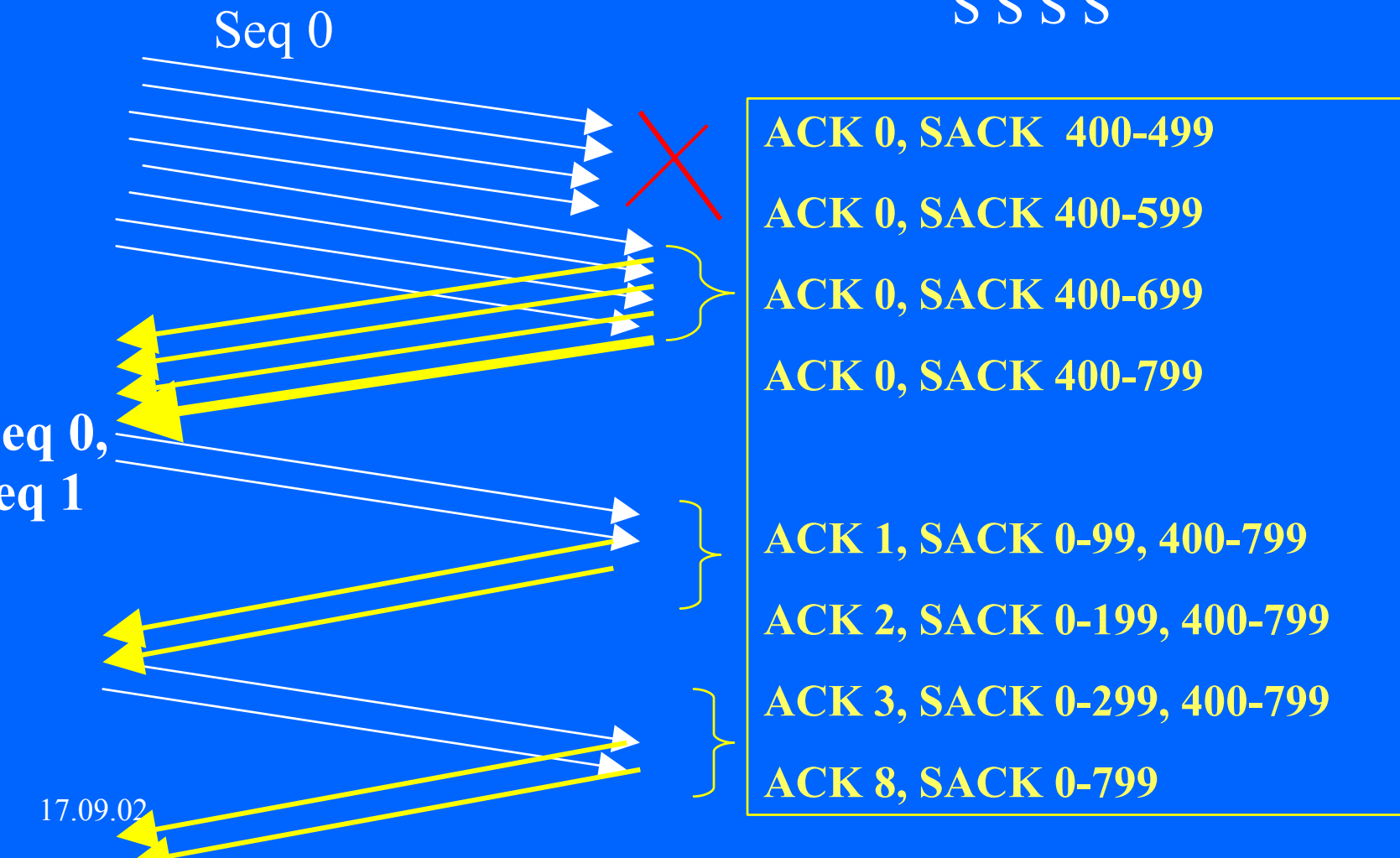
---

- ◆ Ohjeellista tietoa lähettäjälle
  - vastaanottaja voi tarvittaessa poistaa SACK-optiossa ilmoittamiaan tavuja puskureistaan
- ◆ Jos vastaanottaja käyttää SACK-optiota, niin sitä on käytettävä aina kun vastaanottajalla on puskureissaan epäjärjestyksessä olevaa dataa
  - tällöin kaikissa ACK:ssa on oltava ajantasalla oleva tieto siitä, mitkä tavut on jo puskureissa

# Toipuminen SACK:n avulla

0 1 2 3 4 5 6 7

S S S S



# Ruuhkanhallinta SACK:ia käytettäessä

---

- ◆ Noudatettava käytettyä ruuhkanhallinta-algoritmia
  - ei uudelleenlähetyistä heti 1. puuttuvan jälkeen
  - rajoitettu lähetykset puuttuvan jälkeen
    - » toimitaan ruuhkatilanteen mukaan
      - ◆ hidas aloitus: 1 tai 2 segmenttiä ensin, kun niihin kuittaus sitten kaksinkertaistetaan lähetykset jne
      - ◆ nopea toipuminen: yksi segmentti, jokaisesta kuittauksesta ja ruuhkaikkunan puolitus
  - jos ajastin laukeaa, niin kerätty SACK-tieto ei ole enää voimassa

# RED (Random Early Detection)

---

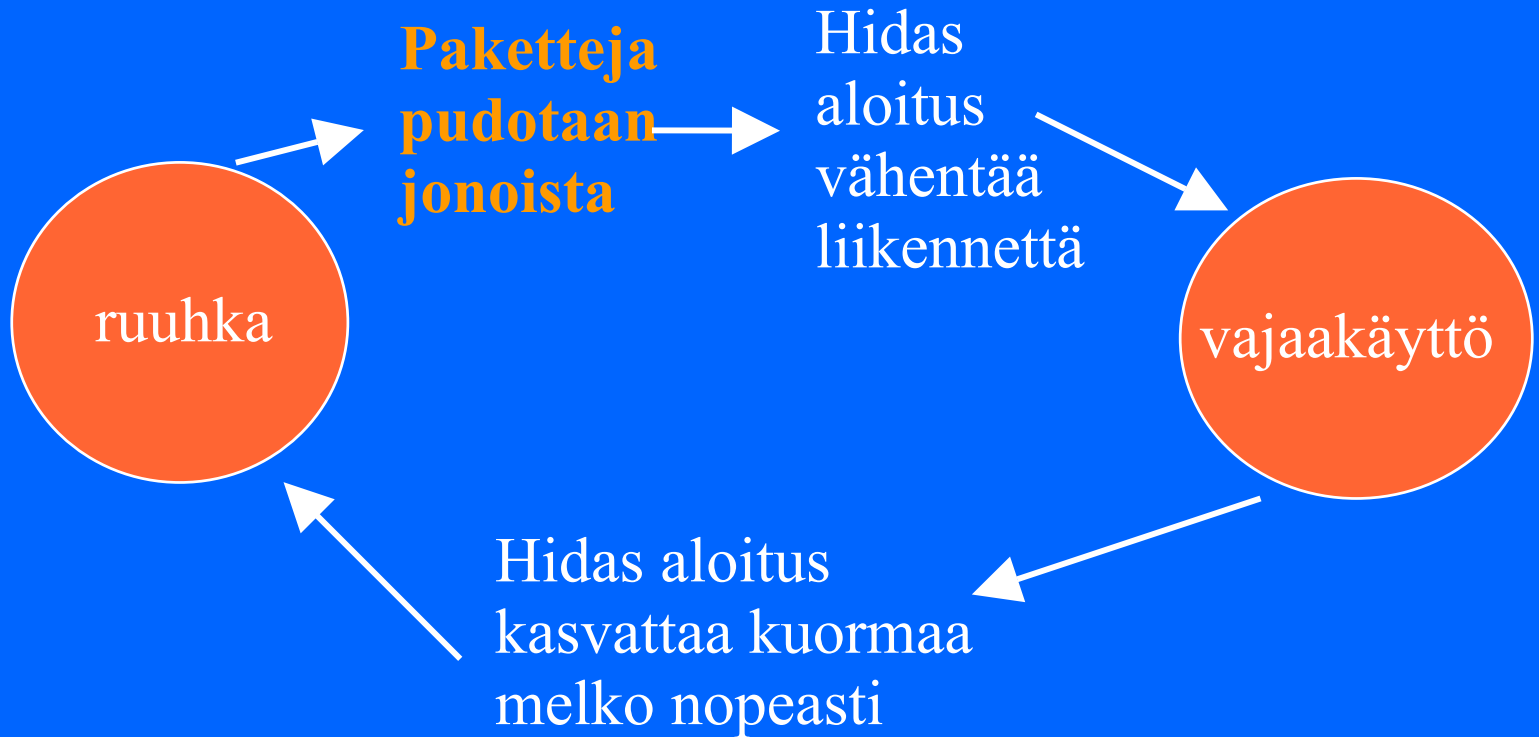
- ◆ Aktiivinen, ennaltaehkäisevä puskurijonon hallinta parantaa TCP:n suorituskykyä
  - proaktiivinen  $\Leftrightarrow$  reaktiivinen
- ◆ Ongelma:
  - Kun ruuhka on syntymässä, puskurien jonot kasvavat ja lopulta puskurit täyttyvät ja paketteja joudutaan hävittämään
    - » yleensä ‘pudotetaan’ viimeksi saapuvat paketit
      - ◆ tail-drop
  - tässä vaiheessa usein poistetaan paljon paketteja monelta lähettäjältä

## **Globaalin tahdistuksen ongelma** (Global Synchronization)

---

- ◆ Samanaikaisesti usean TCP-lähetyksen uudelleenlähetyksajastin laukeaa ja useat TCP:t vähentävät hyvin voimakkaasti lähettämistään hitaan aloituksen takia.
- ◆ => verkon vajaakäyttöisyys
- ◆ lähettäjät kasvattavat lähettämistään hitaassa ajoituksessa melko nopeasti ja jossain määrin samassa tahdissa
- ◆ => ruuhka verkossa

# Verkon suorituskyky on huono!



Oskilloidaan koko ajan ruuhkan ja vajaakäytön välillä eikä yhteyksillä saavuteta tasaista kuittauksien tahdistamaa 'lähetysputkea'.

# Tehokkaampi puskurijonojen hallinta

---

- ◆ Puskureiden koon kasvattaminen ei ratkaise ongelmaa.
  - Miksi ei?
- ◆ Pitää ennakoida ruuhkatilanteen kehittyminen ja reagoida siihen ennenkuin tilanne tulee niin pahaksi, että joudutaan poistamaan paljon paketteja samalla kertaa.
- ◆ => Aktiivinen puskurijonon hallinta => RED

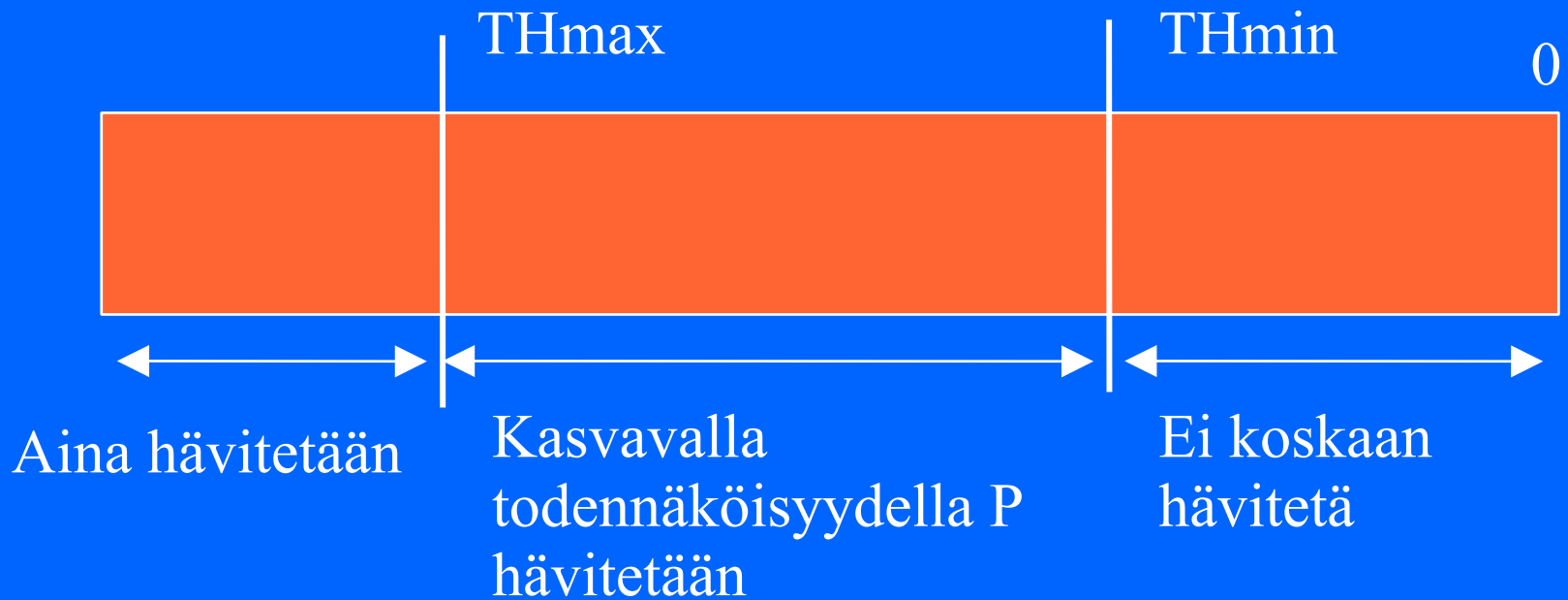


# Miten RED toimii?

---

- ◆ Jonon pituutta tarkkaillaan koko ajan
  - aina kun jonoon tulee paketti, niin
    - » jos jonon pituus  $<$  minimi kynnyspituus, paketti laitetaan jonoon
    - » jos jononpituus  $\geq$  maksimi kynnyspituus, paketti hävitetään
    - » jos jonon pituus minimi ja maksimi kynnysarvojen välissä, niin todennäköisyydellä  $P$  paketti hävitetään ja todennäköisyydellä  $1-P$  laitetaan jonoon
    - » hävittämistodennäköisyys kasvaa, kun jononpituus kasvaa

# Jonon pituuden vaikutus



## RED-puskuri

- 
- ◆ Pyritään pitämään jonon pituus koko ajan annetuissa rajoissa hävittämällä paketteja kun jonon pituus kasvaa
    - eli voi tulla ruuhka
  - ◆ Hävittämistodennäköisyys kasvaa, kun jono kasvaa.
  - ◆ Hävittämistodennäköisyys kasvaa, kun paketteja ei ole hävitetty

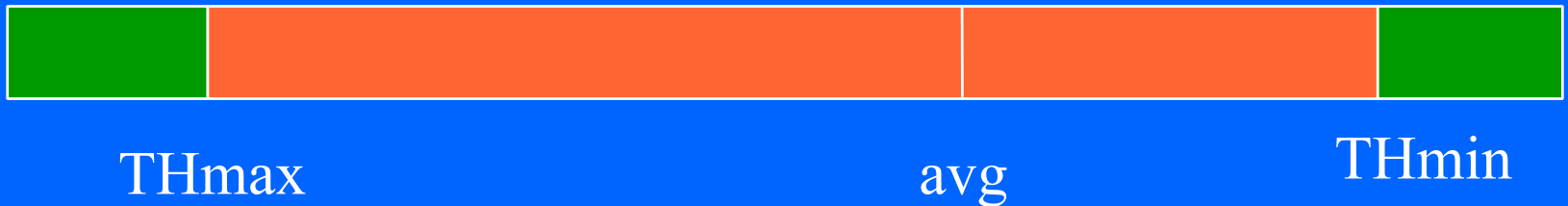
# Kun paketti saapuu FIFO-tyyppiseen ulostulojonoon:

---

- ◆ Lasketaan keskimääräinen jononpituus painotettuna keskiarvona aikaisemmista jononpituuksista
- ◆  $avg \leftarrow (1-wq) * avg + wq * q$
- ◆ jos jonon pituus  $q = 0$ 
  - $m \leftarrow f(\text{time} - q\_time)$
  - $avg \leftarrow (1-wq) ** m * avg$
  - arvioidaan, kuinka monta pikkupakettia (= m) olisi voitu siirtää sinä aikana, jonka jono on ollut tyhjänä
- ◆  $wq \sim 0.002 \Rightarrow avg$  reagoi hitaasti jonon pituuden muutokseen

# Hävittämistodennäköisyyden laskeminen

- ◆ Jos jononpituus  $avq$  on välillä  $Thmin$  ja  $Thmax$ , on laskettava hävittämistodennäköisyys  $P$
- ◆ mitä pitempi jono, sitä suurempi  $P$ 
  - $Pb = Pmax (avq - THmin) / (Thmax - Thmin)$



- ◆ Suositus  $Pmax = 0.02 \Rightarrow$  kun  $avg = 1/2(THmax - THmin)$ , niin 2 pakettia sadasta hävitetään.

## ◆ Lisäksi pyritään tekemään hylkäämiset tasavälein

- estämään hylkäysryöpyt



## ◆ mitä pitempään ei ole hylätty yhtään, sitä suurempi hylkäystodennäköisyys

- avg:n ollessa  $T_{\min}$ in ja  $T_{\max}$ in välissä muuttuja **count** laskee peräkkäisiä hylkäämättömiä paketteja
- $P_a \leftarrow P_b / (1 - \text{count} * P_b)$
- $P_a$  = hylkäämiskriteerinä käytetty todennäköisyys (P)

- 
- ◆ ruuhkan estämiseen
  - ◆ erityisesti globaalin tahdistumisen estämiseen
  - ◆ ryöppyisen liikenteen sorsiminen estämiseen
    - » liikenneryöpyt usein ruuhkan syy
    - » ryöppyisen liikenteen lähteet joutuvat kokemaan pakettien hylkäämistä tasaisen liikenteen lähteitä enemmän
  - ◆ rajoittaa keskimääräistä jononpituutta => rajoittaa keskimääräistä viivettä

# ECN (Explicit Congestion Notification)

---

- ◆ **The Addition of Explicit Congestion Notification (ECN) to IP,**

K. K. Ramakrishnan, Sally Floyd, D. Black.  
(draft-ietf-tsvwg-ecn-01.txt), January 2001.  
(STATUS: INTERNET DRAFT)

- ◆ **TCP and Explicit Congestion Notification.**

ACM Computer S. Floyd. , Communications Review, 24,  
October 1994



# Lisäys IP-arkkitehtuuriin!

---

- ◆ Käyttöön 2 bittiä, joita käytetään ruuhkasta ilmoittamiseen
  - CE-bitti (Congestion Experienced)
    - » reititin asettaa, kun on havainnut ruuhkaa
    - » esim. kun RED-algoritmin jononpituus indikoi ruuhkaa
  - ECT-bitti (ECN Capable Transport)
    - » kertoo, että kuljetuskerros kykenee käsittelemään ruuhkailmoituksia (Explicit Congestion Notification)
  - IPv4: TOS-kentän bitit 7 ja 6 ehdotettu
  - IPv6: Traffic Class -kentän bitit 7 ja 6 ehdotettu

# Muutoksia TCP-hen

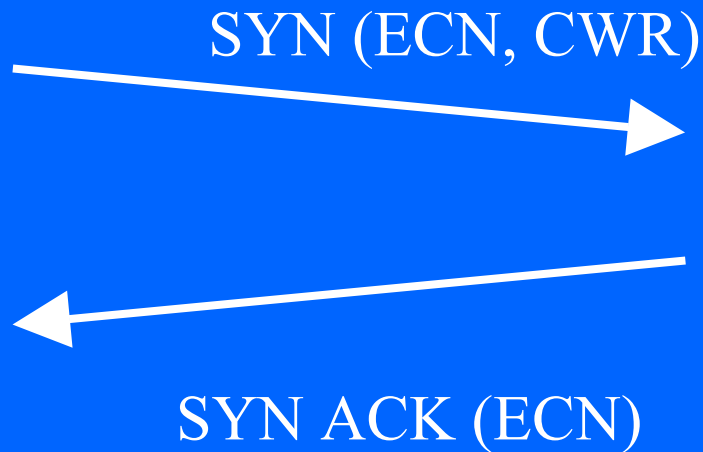
---

- ◆ Sovittava vastapuolen kanssa ECN:n käytöstä
- ◆ osattava reagoida CE-bitin asetukseen
  - vastaanottajan ilmoitettava lähettäjälle
    - » ECN Echo -lippu
  - lähettäjän vähennettävä lähetystään samalla tavalla kuin jos paketti olisi todella kadonnut
  - lähettäjän ilmoitettava vastaanottajalle, että on vähentynyt jo lähettämistään
    - » Congestion Window Reduced (CWR) -lippu

# Yhteydenmuodostus

---

- ◆ Yhteyden muodostusvaiheessa sovitaan ECN:n käytöstä käyttäen ECN Echo -lippua (bitti 9 TCP-otsakkeen varattukentässä)



- 
- ◆ Kun käytöstä on sovittu, lähetettyjen IP-pakettien ECT-kenttä on asetettu
    - » jos ei ole asetettu, ei vastaanottajan tule reagoida
  - ◆ kun vastaanottaja saa ruuhkasta ilmoittavan IP-paketin, sen tulee kuittauksessa asettaa ECN Echo -bitti
  - ◆ CE-bittejä asetetaan kuittauksiin niin kauan, kunnes saadaan paketti, jossa CWR-bitti on asetettu

- 
- ◆ Kun lähettäjä saa kuittauksen, jossa ECN Echo -bitti on asetettu, niin sen tulee
    - puolittaa ruuhkaikkuna
    - pienentää hitaan aloituksen lopettamisen kynnyksarvoa
  - ◆ toiminta sama kuin paketin kadotessa
  - ◆ vähennys korkeintaan kerran yhden kiertoviiveen aikana

# NewReno

---

- ◆ **RFC 2581: TCP Congestion Control**

M. Allman, V. Paxson, W. Stevens

April 1999 (Obsoletes RFC 2001)

(Status: PROPOSED STANDARD)

- ◆ **RFC 2582: The NewReno Modification to TCP's Fast Recovery Algorithm**

S. Floyd, T. Henderson, April 1999

(Status: EXPERIMENTAL)

# Nopea toipuminen (Fast Recovery)

---

- ◆ Toteutettiin ensimmäisen kerran 1990 Reno-versioon
- ◆ Ei toimi hyvin, jos useita paketteja katoaa
  - tarvitaan kiertoaika jokaista kadonnutta pakettia kohden
- ◆ eräs ratkaisu on SACK-optio
  - kuittauksessa ilmoitetaan, mitkä saatu kunnolla ja mitkä vielä puuttuvat

- ◆ Kun useita segmenttejä katoaa ‘samasta ikkunasta’





# NewReno -ratkaisu

---

- ◆ Kun saadaan kolmas toistokuittaus, niin
  - asetetaan kynnysarvo  $ssthresh = \max(\text{FlightSize} / 2, 2 * \text{MSS})$
  - ja talletetaan viimeksi lähetetty järjestysnumero muuttujaan “recover”
  - Lähetetään puuttuva segmentti ja asetetaan ruuhkaikkunan arvoksi
$$cwnd\ to = ssthresh + 3 * \text{MSS}$$
  - kaikki vielä tulevat toistokuittaukset kasvattavat ruuhkaikkunaa yhdellä MSS:lla

---

## ◆ Lähetetään segmentti

- mikäli ruuhkaikkuna ja vastaanottajan ikkuna tämän sallii

## ◆ Kun segmenttiin saadaan kuittaus,

- se joko kuittaa kaikki `recover` -muuttujan ilmoittamaan arvoon asti  $\Rightarrow$  toipuminen suoritettu loppuun
- tai kuittaus on johonkin aikaisempaan järjestysnumeroon

» osittainen kuittaus (partial ACK)

# Kun tulee osittainen kuittaus

---

- ◆ Lähetetään uudelleen ensimmäinen kuittaamaton segmentti,
- ◆ kasvatetaan ruuhkaikkunaa kuitatuilla ja vähennetään siitä juuri lähetetty
- ◆ lähetetään uusia segmentteja, jos ruuhkaikkuna ja vastaanottajan ikkuna tämän sallii
- ◆ ja jatketaan toipumisvaihetta

# Eri versioita

---

- ◆ Pitäisikö kuitenkin uudelleenlähettää kerralla useampi kuin yksi?
- ◆ Miten ajastin tulisi parhaiten asettaa kun uudelleenlähetetään segmentti?
- ◆ Yms.
- ◆ Nyt ollaan jo melko kaukana itse standardista. Nämä ovat yhä avoimia tutkimuskysymyksiä!

# Uudelleenlähetyssajastin

---

## ◆ RFC 2988: Computing TCP's Retransmission Timer.

V. Paxson, M. Allman.

November 2000

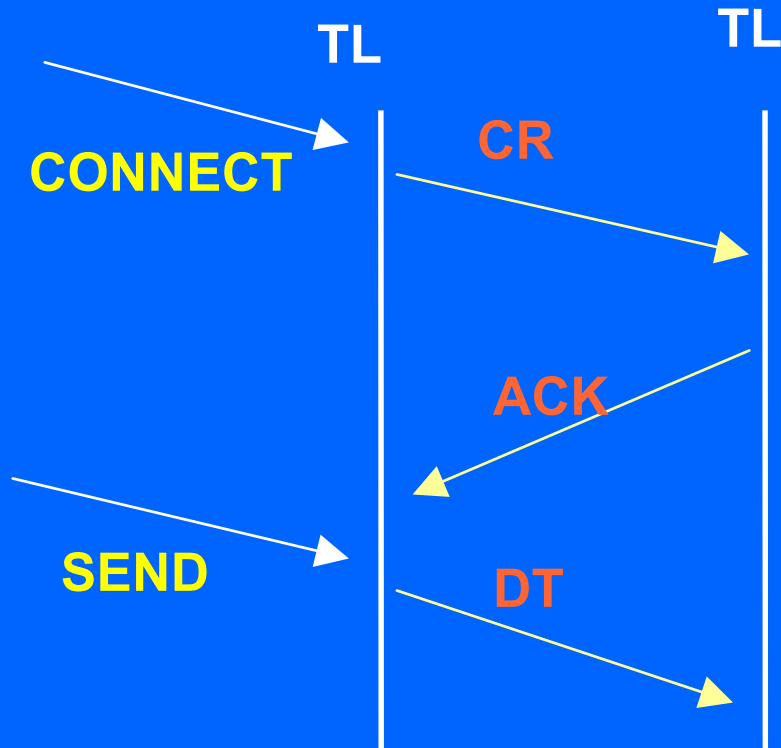
(Status: PROPOSED STANDARD)

# Yhteyden muodostus

---

- ◆ perusmalli hyvin yksinkertainen
- ◆ ongelmana viivästetyneet kaksoiskappaleet
  - » esim. yhteys pankkiin laskun maksamiseksi
  - » lasku maksetaan useaan kertaan
- ◆ => yksikäsitteisen yhteyden muodostaminen vaikeaa

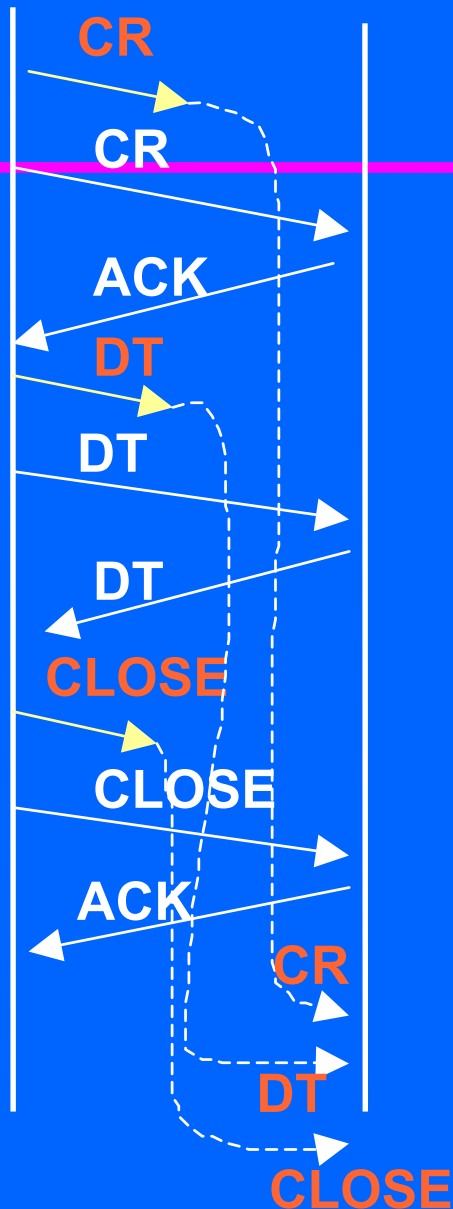
# Yhteydenmuodostus: perusmalli



Hyvin yksinkertaista!

# Yhteyden muodostus ruuhkaisessa verkossa

Jokainen paketti lähetetään kahteen kertaan



Kun yhteys on purettu, viivästyneet kaksoiskappaleet saapuvat

Ne tulkitaan uudeksi yhteydeksi, ja data otetaan vastaan kahteen kertaan!



# Ongelman ratkaisuehdotuksia:

---

- ◆ kertakäyttöiset kuljetusosoitteet
  - » nimipalvelu?
- ◆ yhteystunnus jokaiselle yhteydelle
  - yhteyden purkamisen jälkeen sen TPDU:t epäkelvoja
    - » lista epäkelvoista yhteystunnuksista
  - **kuinka kauan historiatietoja säilytettävä?**
  - **entä jos kone kaatuu ja unohtaa tietonsa?**
- ◆ **rajallinen elinikä paketeille**
  - » elinaikalaskuri, hyppylaskuri

# Tomlinsonin menetelmä

---

- ◆ koneessa vikasietoinen kello

  - » etenevä laskuri

  - » vaikka kone kaatuu, laskuri toimii

- ◆ yhteyttä muodostettaessa

  - » kellon bitit ilmoittavat numeroinnin aloituskohdan

  - » bittejä riittävästi (32 bittiä)

    - ◆ jotta uudelleen käyttöön vasta riittävän pitkän ajan päästä

    - ◆ vanhoilla numeroilla varustetut segmentit ehtivät hävitä

- ◆ yhteydellä ei koskaan kahta saman numeroista segmenttiä

k bittiä

Kello:

järjestysnumero:



Eri yhteyksillä numerointi aloitetaan eri kohdasta

## ◆ Kun kone kaatuu

- » se kadottaa tiedon viimeksi käytetystä järjestysnumerosta
- » uusi numero ei saa olla sama kuin jonkun vielä elossa olevan TPDU:n

## ◆ Miten selvittää tilanteesta?

- odotetaan T aikayksikköä
  - » kaikki aikaisemmat TPDU:t varmasti kadonneet ja voidaan aloittaa, mistä numerosta tahansa
  - » entä, jos T on pitkä
- ‘kielletyn alueen’ (forbidden region) käyttö
  - » ei oteta käyttöön numeroita, joiden duplikaatit voivat olla vielä elossa

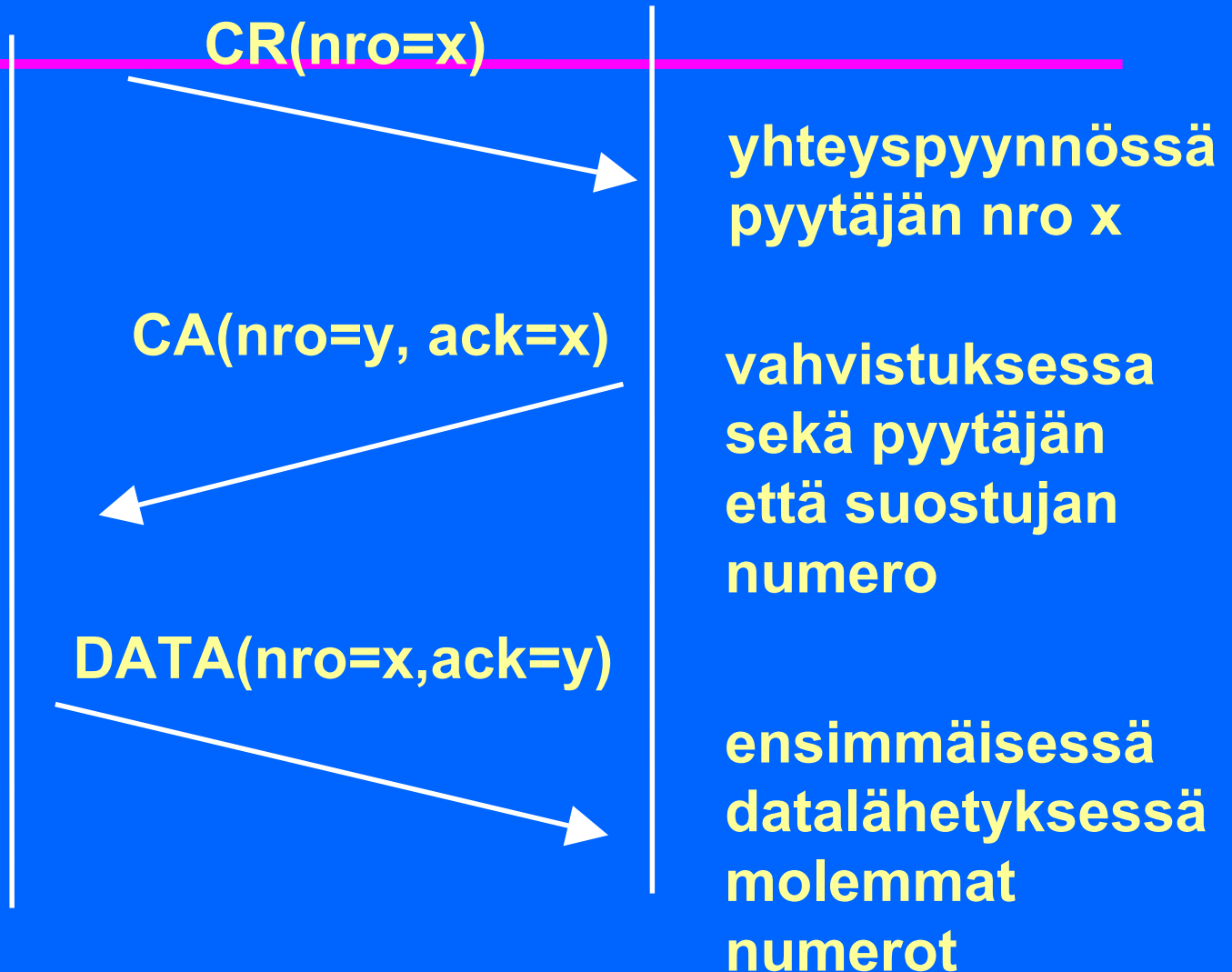
järjes-  
tysnu-  
merot

**Kielletty  
alue**

aina ennen uuden  
paketin lähettämistä  
tarkistetaan, ettei  
järjestysnumero ole  
kielletyllä alueella

aika

# Kolminkertainen kättely



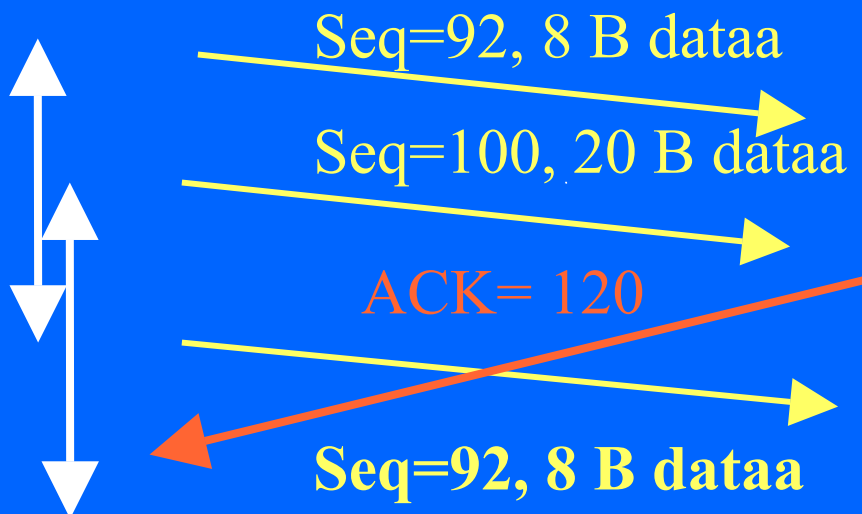
# Kuittaukset TCP:ssä

---

- TCP käyttää kumulatiivista kuittausta
  - kuittaus varmistaa lähettäjälle, että kaikki segmentit kuitattuun segmenttiin saakka ovat saapuneet kunnolla perille
  - väärässä järjestyksessä saapuneita segmenttejä ei kuitata
  - ei käytetä NAK-kuittausta
  - “duplicate ACK” = virhetilanteissa lähetetään uudelleen kuittaus samasta jo kuitatusta segmentistä

# Uudelleenlähettäminen

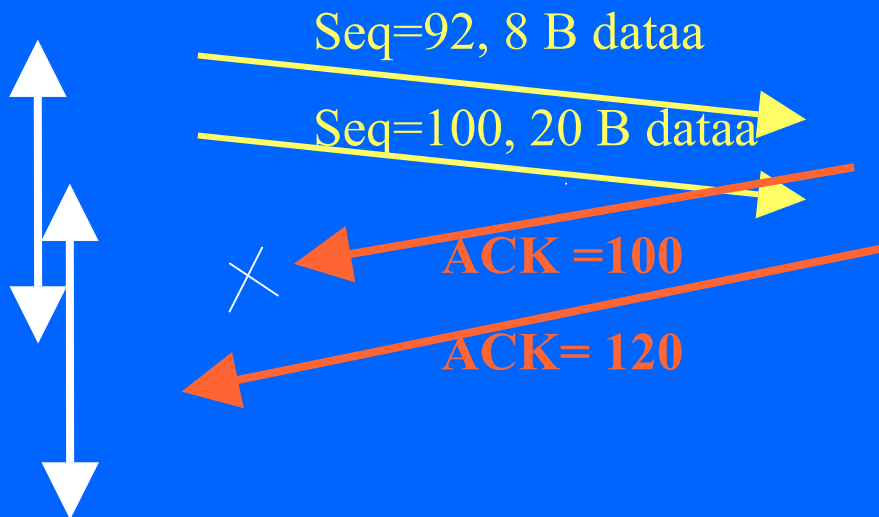
- TCP lähettää segmentin uudestaan, kun ajastin laukeaa
  - TCP ei automaattisesti lähetä kaikkia puuttuvan segmentin jälkeisiä segmenttejä uudelleen





# Vain osin “Go-Back -N -tyyppinen”

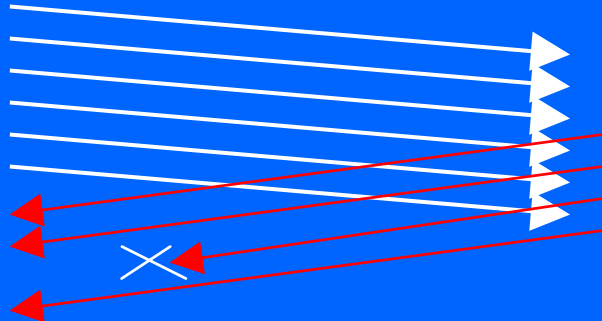
- Oletetaan, että segmentit 1-N tulevat oikein perille ja kuittaus esim. segmenttiin 1 katoaa. Jos muut kuittaukset tulevat perille, enintään yksi segmentti 1 uudelleenlähetetään.
  - Eikä sitäkään tarvitse lähettää, jos seuraava kuittaus tulee ennen ajastimen laukeamista



# Kuittaukset voivat kadota

---

- ◆ Erilliset kuittaukset eli pelkät ACK:it eivät sisällä yhtään tavua dataa, joten niitä ei numeroida eikä kuitata.
  - Kuittauksia voi helposti hävitä



Kumulatiivisissa kuittauksissa seuraava kuittaus paikkaa hävinneen informaation

