



# Tietoliikenne II (2 ov)

---

**Syksy 2003**

**Liisa Marttinen**

- Kurssikirja: Kurose & Ross, Computer Networking (2. edition)
  - ★ (kyllä 1. painoskin kelpaa, mutta siitä puuttuu mm. mobiiliverkot kokonaan)
- Lisämateriaalia: Aiheeseen liittyviä RFC:itä



# Tietoliikenne II

---

## Täydennystä Tietoliikenne I -kurssin asioihin

- perusteellisemmin
- laajemmin
- 'teoreettisemmin'
- TCP: suorituskyky ja uudet piirteet
- reititys, IPv6, IPSec, MobileIP, monilähetys
- WLAN, atm, fddi, SONET, fyysinen kerros
- Internetin QoS



# Alustava sisällysluettelo

---

- 1. TCP:n suorituskyky
  - optiot
  - uudet piirteet ruuhkanvalvonnassa
- 2. IPv6, Ipvsec, ICMP, Reititys
  - OSPF, BGP, monilähetysreititys, mobiilireititys
- 3. Gigabit Ethernet, WLAN, atm, FDDI
- 4. Linkkikerroksen ja peruskerroksen asioita
  - SONET, .modeemi
  - Tiedonsiirron teoreettinen perusta (Shannon, Nyquist, ..)

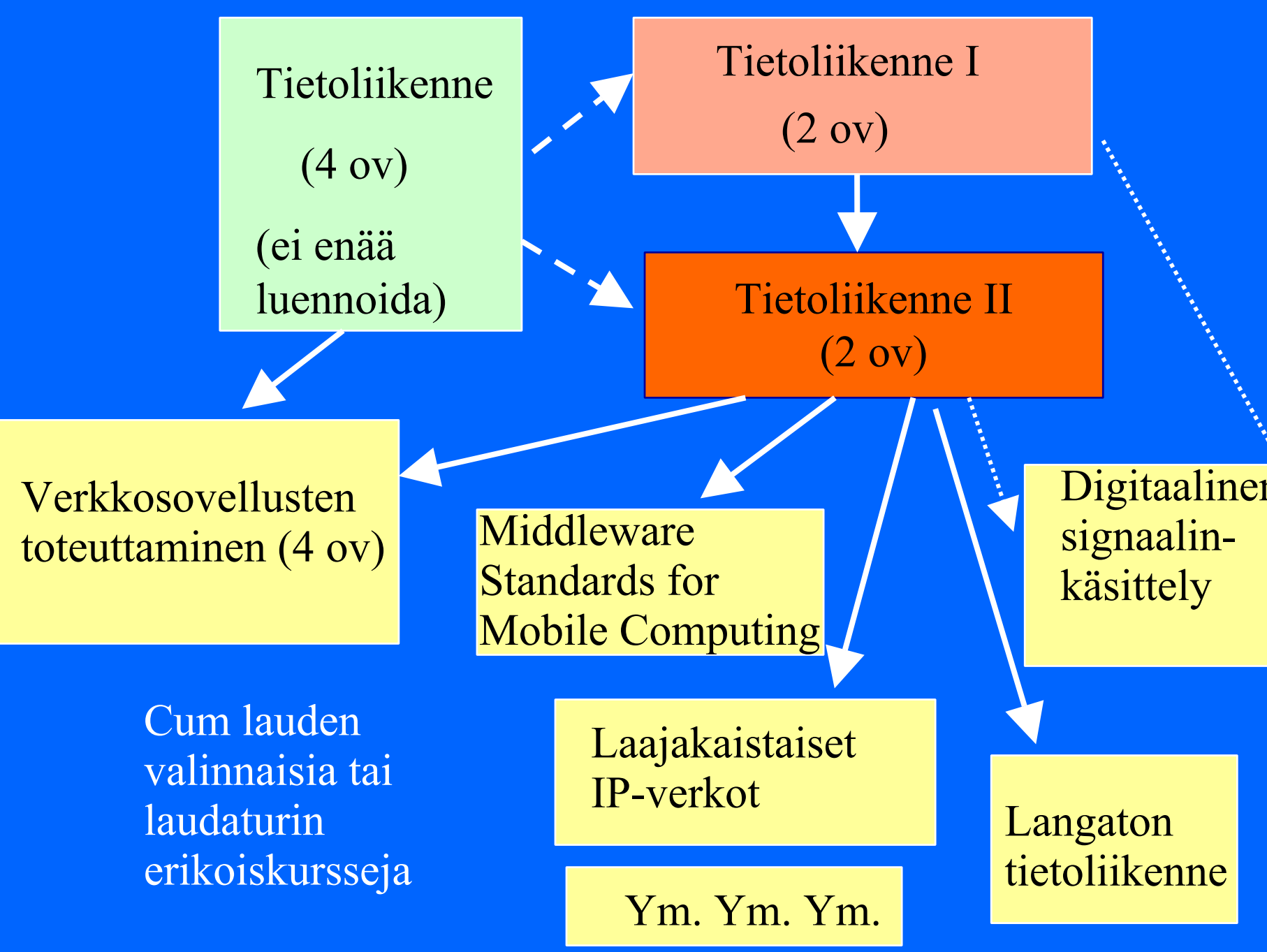


# Sisällysluettelo jatkuu

---

- 5. Internetin palvelun laatu (QoS)
  - integroidut palvelut
  - eriytyneet palvelut
- 6. Turvallisuus
  - Palomuri (?), PGP (?)

Tietoturva-asiat keväällä luennoitavalla  
Tietoturva-kurssilla





# Suoritus

---

- kurssikoe maks. 50 p, min 25 p
  - 6.11. klo 16-20 Porthania
- kurssiaktiivisuus maks. 20 p
  - traditionaaliset harjoitukset => maks. 10 p
  - miniesseet (1-10 sivua) ja -esitelmät (10-15 min), keskusteluaktiivisuus yms. => maks. 10 p
- 30 p => 1-, 51 => 3
- loppukokeella maks. 60 p; tammi - helmikuussa 2004

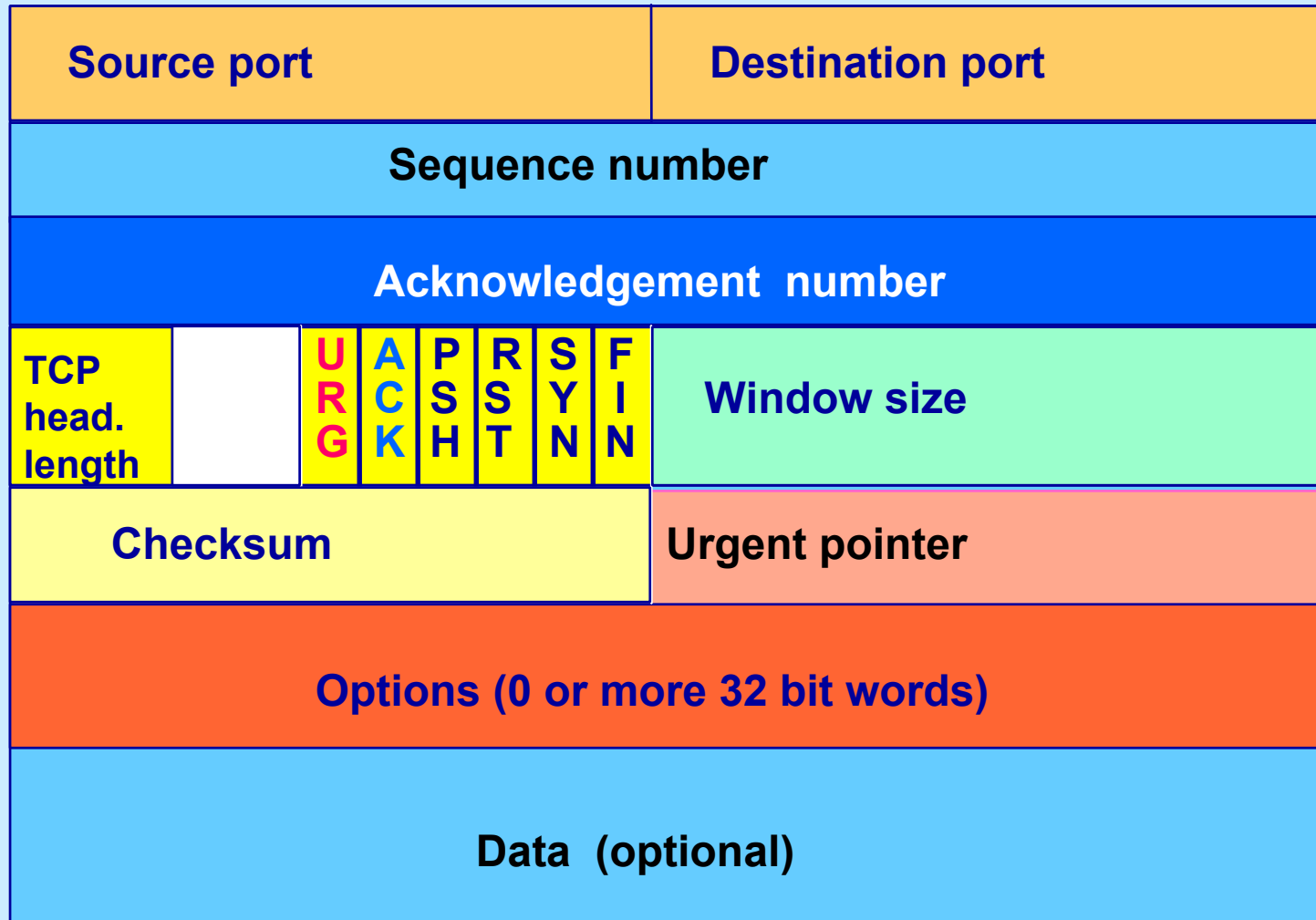


# 1. TCP ja suorituskyky

---

- TCP:n peruspiirteiden toiminta jo tunnettu
  - Tietoliikenne I
- TCP:n lisäpiirteitä
  - Ikkunaskaalaus (Window scaling)
  - Aikaleimaus (time stamping)
  - SACK (Selective Acknowledgement)
  - RED (Random Early Detection)
  - ECN (Explicit Congestion Notification)

# TCP-otsakkeen kentät





# TCP-optiot

- Optio-kenttä valinnaisia piirteitä varten
  - Option pituus on 40 tavua
    - TCP header length -kenttä = 4 bittiä kertoo otsakkeen pituuden 32 bitin sanoina =>  $15 \cdot 4$  tavua = 60 tavua
    - 20 tavua vakio-otsaketta => enintään 40 tavua optioita varten

Option tyyppi	Option pituus	Option merkitys
1 tavu	1 tavu	pituus - 2 tavua



# TCP:n suorituskykyongelmia

---

- TCP-protokolla on jo melko vanha (1983) ja viritetty tiettyyn ympäristöön
- nykyisin käytössä hyvin erilaisissa ympäristöissä
  - pitkän viipeen satelliittiyhteyksillä
  - erittäin nopeilla yhteyksillä
  - langattomilla yhteyksillä
- => suorituskykyongelmia



# Ongelmia:

---

- otsakkeen kentät liian pieniä:
  - **järjestysnumero 32 bittiä => 4 294 967 296 tavua**
    - rajoittaa siirtonopeutta erittäin nopeilla yhteyksillä
    - suurin segmentin elinikä (MSL, maximum segment lifetime) alunperin 2 minuuttia = 120 sekuntia
  - **ikkunankoko 16 bittiä => 65536 tavua**
    - rajoittaa siirtonopeutta mm. satelliittiyhteyksillä

# MSL (Maximum Segment Lifetime)

- MSL = 2 minuuttia
  - IP-kerroksen elinaikakenttä (time-to-live) rajoittaa
- **Järjetysnumerokenttään mahtuu 4 294 967 296 tavua**
- Esimerkkejä eri verkkojen numeroiden kiertoajasta
  - ARPANET    56 kbps            7 KBps            ~ 3.6 päivää
  - Ethernet        10 Mbps        1.25 MBps        ~ 3 0 min
  - FDDI            100 Mbps      12.5 MBps        ~ 3 min
  - **Gigabit            1 Gbps    125 MBps        17 sec**
- **Nopeissa verkoissa aiheuttaa ongelmia!**

# Järjestysnumeroiden uudelleenkäyttö



- Samannumeroinen segmentti voi yhä olla verkossa => se hyväksytään 'uutena'
- Samannumeroinen ACK-kuittaus voi yhä olla verkossa => lukkiutunut tilanne, josta toivutaan vain RST:llä
  - Lähettäjä saa kuittauksen ja siirtyy seuraavaan segmenttiin
  - Vastaanottaja jää odottamaan puuttuvia tavuja ja hylkää muut
  - Ikkuna täyttyy ja lähettäjä alkaa lähettää uudelleen, mutta .....

# Kaistan ja kiertoviiveen tulo

( bandwidth \* delay product)

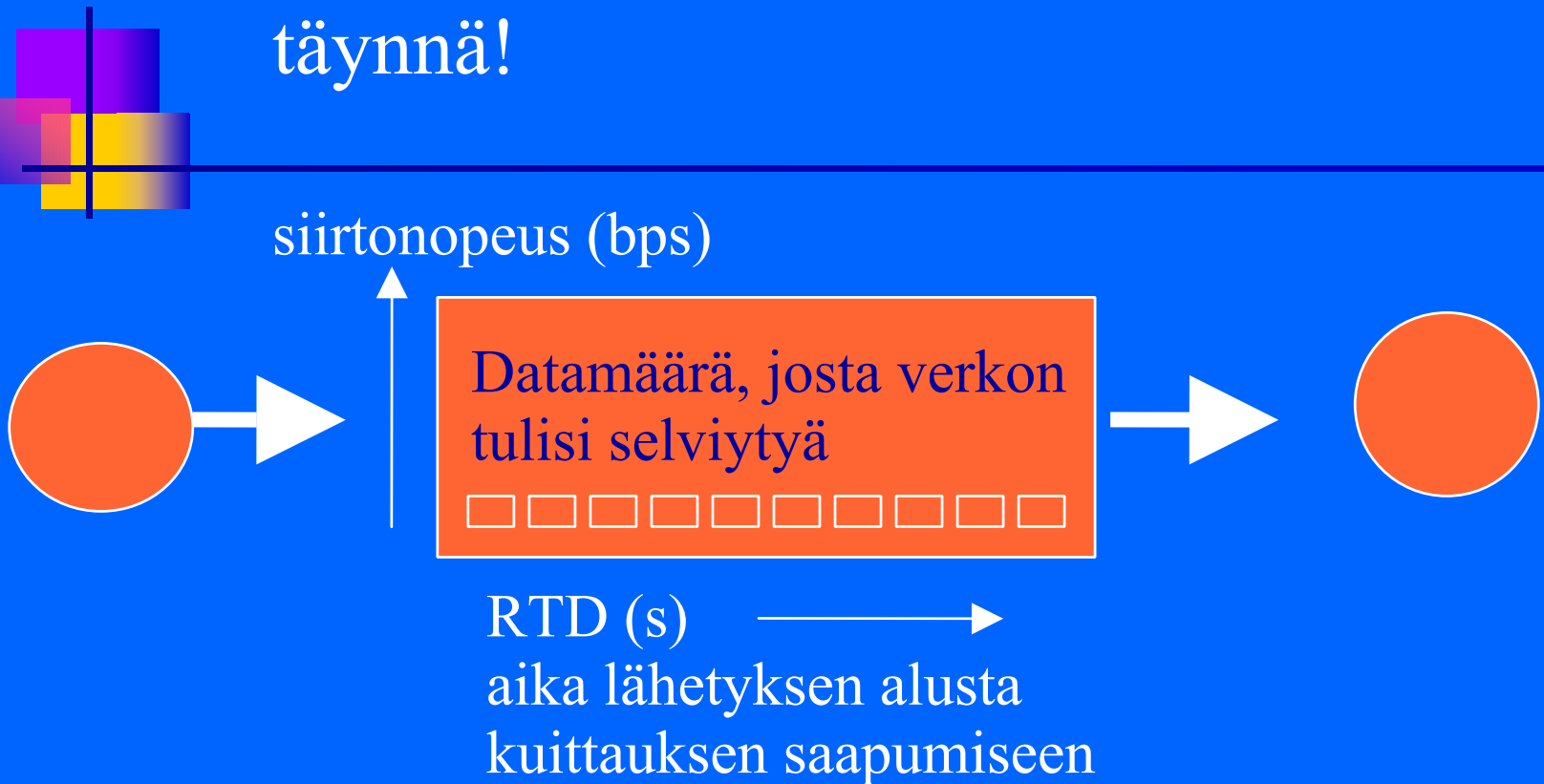
## TCP:n suorituskyky

- Riippuu siirtonopeudesta (kaista, bandwidth) ja
- kiertoviiveestä (RTD, round-trip delay, 1 ms - 100 s)

Tulo siirtonopeus \* kiertoviive kertoo sen datamäärän, joka TCP:n täytyy pystyä käsittelemään, jotta lähettäjä ja vastaanottaja voisivat toimia täydellä vauhdilla

- Eli paljonko kuittaamatonta dataa verkossa täytyy kulkea
- Ongelmia syntyy, jos tulo on hyvin suuri!

# Ideaalitilanteessa ‘siirtoputki’ on koko ajan täynnä!



Ongelmia aiheuttavat LFN-verkot (long, fat pipe network): pitkä etenemisviive ja suuri siirtonopeus, esim. satelliittiyhteydet ja nopeat runkolinjat eli joiden tulo  $> 1 \text{ Mb} \sim 100$  segmenttiä a' 1200 tavua.



# Ongelmia LFN-verkoissa

---

- **Ikkunan koko -kenttä liian pieni** => 'putki' ei täyty  
(suurin mahdollinen ikkuna  $2^{16} = 65$  KB)
  - ikkunan skaalaus -optio
- **pakettien katoaminen** => hidas aloitus eli 'siirtoputki' joudutaan tyhjentämään
  - parannukset ruuhkanhallintakäytäntöön
    - Fast Retransmit, Fast Recovery, Limited Transmit
    - SACK, ENC, RED
- **Uudelleenlähetysajastimen tarkempi ajastus**
  - Timestamp -optio





# Optioita:

---

- **MSS (Maximum Segment Size)**

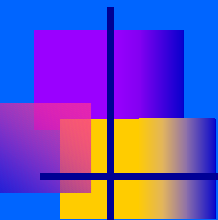
- käytetään ilmoittamaan vastaanottajan yhteydellä hyväksymä suurin segmentin koko
- eri suuntiin voi olla eri koko
- voi olla suurempi tai pienempi kuin oletus-MSS



# MSS ja MTU

---

- **MTU (Maximum Transfer Unit)** = suurin yhdessä verkon kehyksessä kulkeva datamäärä
  - Eri verkoissa eri kokoja, mutta minimi MTU= 576 B
- $MSS = MTU - IP\text{-otsake} - TCP\text{-otsake}$ 
  - oletus  $MSS = 576 - 20 - 20 = 536$
  - Mutta otsakkeet voivat olla suurempia!

- 
- MSS ilmoitetaan yhteyttä muodostettaessa eli SYN-segmenteissä
    - kumpikin osapuoli voi ilmoittaa oman MSS-arvonsa
      - jos ei ilmoita , niin suostuu vastaanottamaan minkä tahansa kokoisia segmenttejä.

2	4	maksimi segmentin koko
---	---	------------------------

# Muita optioita (RFC 1323):

## ■ **ikkunaskaalaus** (window scaling factor)

- kasvattaa TCP-otsakkeen 16-bitin ikkunan koon 32-bitin ikkunan kooksi



## ■ **aikaleimaus** (timestamp)

- segmentin aikaleima palautetaan kuittauksessa





# Ikkunan skaalaus (Window scale factor)

---

- ikkunakoko = 16 bittiä => 65536 tavua
  - kertoo vastaanottajan ikkunan = kuinka monta tavua voi lähettää ennenkuin täytyy jäädä odottamaan kuittausta
  - Jos RTT (Round-trip-time) on suuri, niin joudutaan odottelemaan
  - Efektiivinen nopeus  $B = 2^{**}16/RTT$
- jos käytössä ikkunan skaalaus -optio, ikkunakentän arvo kerrotaan  $2^{**}F$ :llä, jossa F on skaalausoption arvo.
  - Suurin F:n arvo on 14.
- käytetään vain yhteyden aloituspyynnössä

# Miksi uudelleenlähetyksajastimen arvo on tärkeä!

- Ruuhkan oikea havaitseminen riippuu uudelleenlähetyksajastimen 'oikeasta' arvosta.
  - Liian suuri arvo => alkavaa ruuhkaa ei huomata ajoissa => verkkoa ylikuormitetaan => syntyy ruuhkatilanne => resurssien hukkakäyttöä
  - Liian pieni arvo => luullaan ruuhkaksi, vaikka ei olekaan => hidastetaan turhaan lähetystä => resurssien hukkakäyttöä



# Uudelleenlähetyksajastimen arvo

---

- mitataan paketin kiertoviive  $M$  ja viiveen poikkeama odotetusta eli  $|RTT-M|$ 
  - $RTT = aRTT + (1-a)M$
  - $D = bD + (1-b)|RTT-M|$
  - ajastimen arvo =  $RTT + 4D$
- Ongelmia aiheuttavat uudelleenlähetykset
  - Mikä sanomista kuitataan?
  - Karn: uudelleenlähetyksiä ei oteta mukaan, vaan ajastimen arvo kaksinkertaistetaan aina uudelleenlähetyksessä, kunnes saadaan onnistuneesti kuittaus.
- Useat toteutukset mittaavat vain yhden paketin ikkunasta!
  - Ongelmia, jos ikkuna suuri.



# Aikaleima (timestamp)

---

- Kaksi eri optiota
  - Timestamp Value
    - lähteissä segmenteissä,
  - Timestamp Echo Reply
    - kuittauksessa
    - sama kuin kuitatun segmentin Timestamp-arvo
- Voidaan käyttää missä tahansa datasegmentissä
- => Voidaan laskea kiertoviive jokaiselle segmentille, myös uudelleenlähetyksille.





# RTTM (Round-Trip Time Measurement)

---

- lähetettävään sanomaan liitetään aikaleima-  
optioon aikaleima
  - aikaleimakello, joka tikittää riittävän nopeasti
- sama aikaleima palautetaan sanoman  
kuittauksessa
- ongelmatilanteita:
  - viivästetty kuittaus: aikaisin kuittaamaton
    - TCP:n ei tarvitse kuitata jokaista segmenttiä
  - puuttuva segmentti: viimeisin hyväksyty
  - puuttuvan segmentin saapuminen: viimeisin puuttuva



# Viivästetty ACK (Delayed ACK)

---

- TCP:n ei tarvitse välttämättä kuitata jokaista segmenttiä
  - kuitenkin kuitattava ainakin joka toinen ja viive saa olla korkeintaan 500 ms,
    - usein noin 200ms
- Hyöty: kuittaus kulkee datan mukana
  - samalla kertaa ikkunankoon muutos, kuittaus ja kaiutus
- Haitta: kiertoviiveen laskeminen, pakettien kellotus

# TCP-ruuhkanvalvonta



- toimii lähetyksen tahdistajana
- putkesta poistunut dataa, joten voidaan lähettää sama määrä lisää



# TCP:n itsetahdistus (self-clocking)

---

- TCP tahdistaa itse oman lähetyksensä ACK:ien avulla
  - nopeutta voi rajoittaa
    - verkko
      - ruuhkan takia syytä vielä pienentää lähetysnopeutta
    - vastaanottaja
      - lähetysnopeus ok
  - lähettäjä ei voi tietää kumpi



# Lähetettynä voi olla vain rajallinen määrä kuittaamatonta dataa ('Flight size')

---

- vastaanottoikkuna (receiver window, **rwnd**)
  - vastaanottaja ilmoittaa lähettämiensä segmenttien ikkunakentässä
  - vastaanottaja voi vapaasti kasvattaa tai pienentää
  - vuonvalvontaa varten
- ruuhkaikkuna (congestion window, **cwnd**)
  - lähettäjä saa korkeintaan lähettää verkkoon, jotta verkko ei tukkeutuisi
  - ruuhkanhallintaa varten
- **$\min(\text{rwnd}, \text{cwnd})$**  rajoittaa lähettämistä



# Ruuhkanvalvonta on hankalaa

---

- Sitä varten on koko ajan kehitetty yhä parempia menetelmiä
  - uudelleenlähetyssajastimen arvo
    - RTT:n varianssin arviointi
    - Karnin algoritmi
    - exponential retransmission timer backoff
  - lähetyssikkunan hallinta
    - slow start
    - congestion avoidance
    - fast retransmit
    - fast recovery



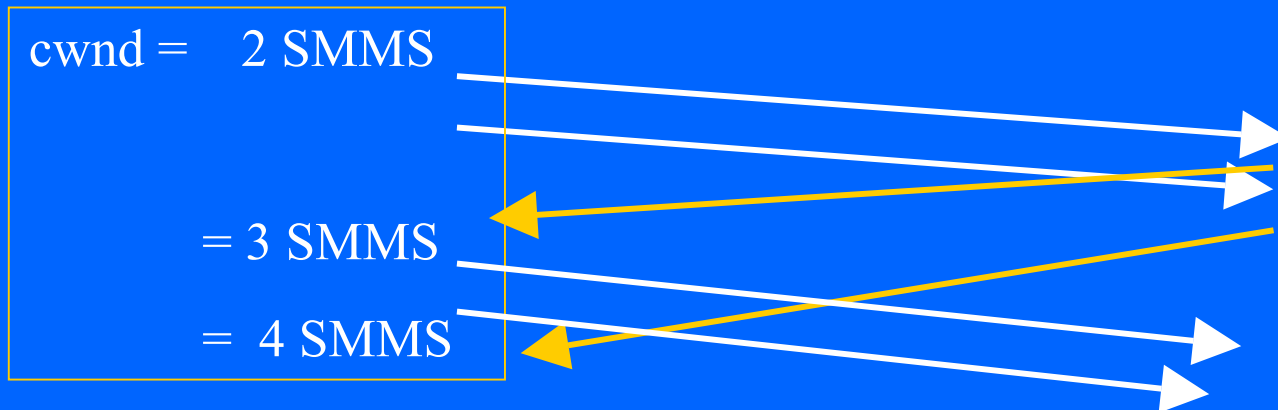
# Ruuhkaikkunan arvo eri tilanteissa

---

- initial window (IW)
  - ruuhkaikkunan arvo heti kolminkertaisen kättelyn jälkeen
    - korkeintaan kaksi segmenttiä tai  $2^*$  suurin määrä tavuja, jonka lähettäjä voi kerralla lähettää (SMSS)
- loss window (LW)
  - ikkunan arvo, kun TCP on havainnut ,uudelleenlähetyssajastimen lauettua segmentin kadonneeksi
- restart window (RW)
  - kun lähetys käynnistetään uudelleen joutilaana olon jälkeen

# Slow start

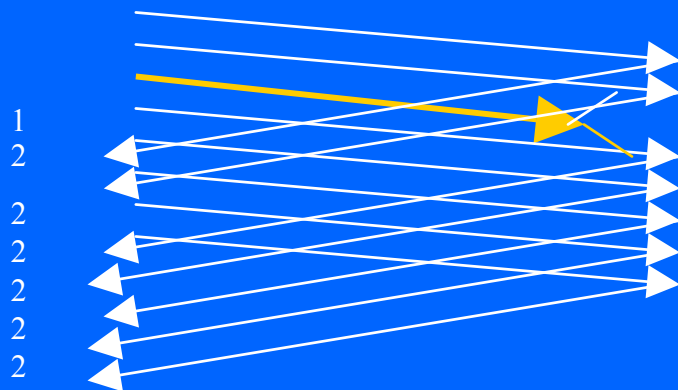
- Hitaan aloituksen aikana
  - Ruuhkaikkunaa `cwnd` kasvatetaan korkeintaan maksimilähetysmäärällä (SMSS) jokaista uutta dataa kuittaavaa ACKia kohden





# Hidas aloitus

- Aina yhteyden alussa
- kun kuittausta ei tule ajoissa (paketti kadonnut!)



Ajastin laukeaa noin 400 ms kuluttua, jonka jälkeen aloitetaan hidas aloitus!



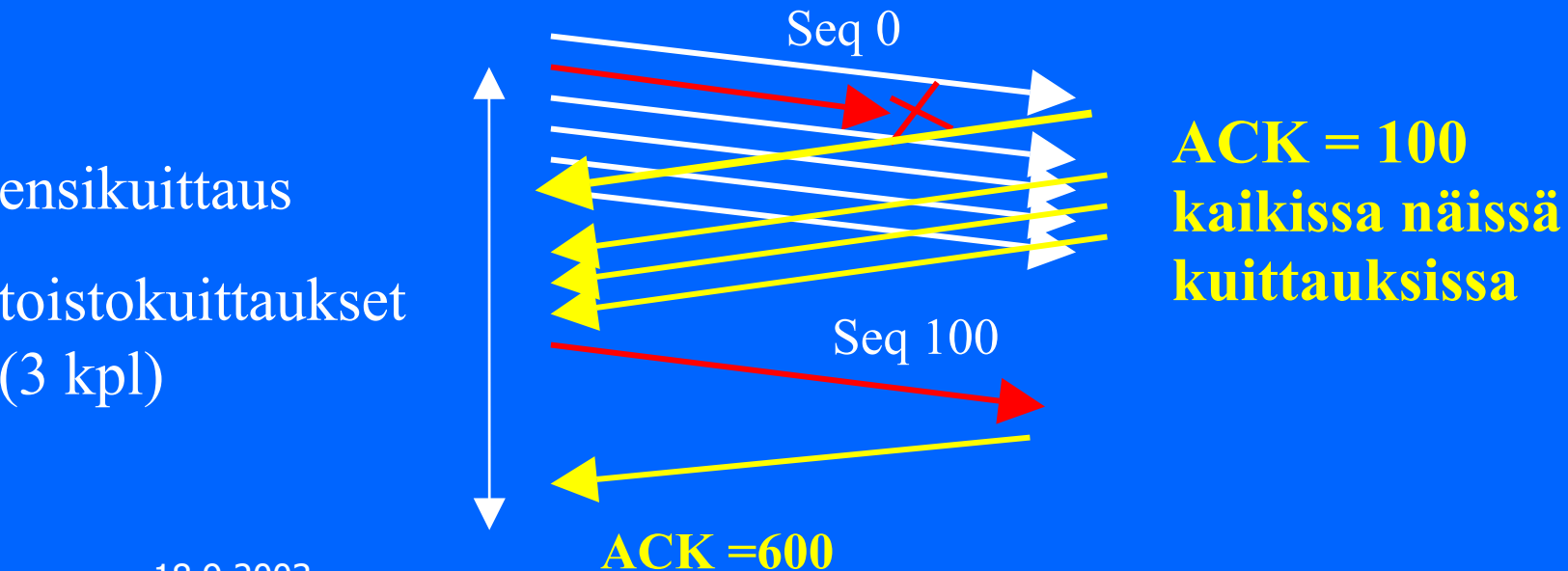
# Toistokuittaus (Duplicate Ack)

---

- ensikuittaus (first-time ACK)
  - segmentin ensimmäinen kuittaus
  - tähän saakka kaikki on kunnossa
- toistokuittaus (duplicate ACK)
  - vastaanottaja kuittaa viimeksi saatua hyväksytyä segmenttiä aina kun saa virheellisen tai väärässä järjestyksessä tulevan segmentin
  - NAKin korvike, jolla ilmoitetaan ongelmista lähettäjälle

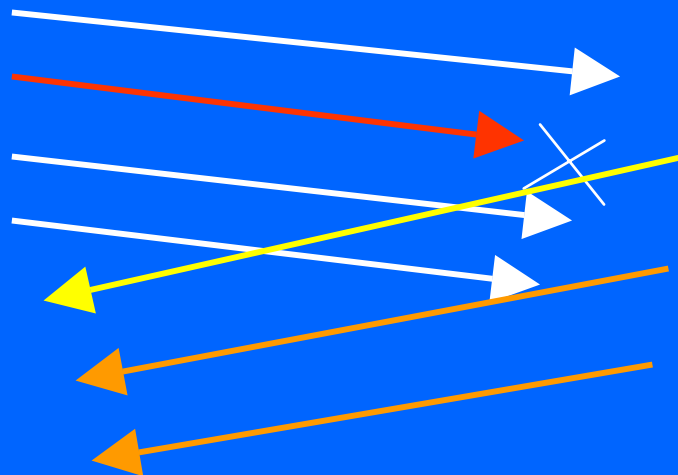
# Nopea uudelleenlähetytys (Fast retransmit)

- Kun lähettäjä vastaanottaa 3 toistokuittausta samalle segmentille, se lähettää heti puuttuvan segmentin uudestaan
  - eikä odota segmentin ajastimen laukeamista



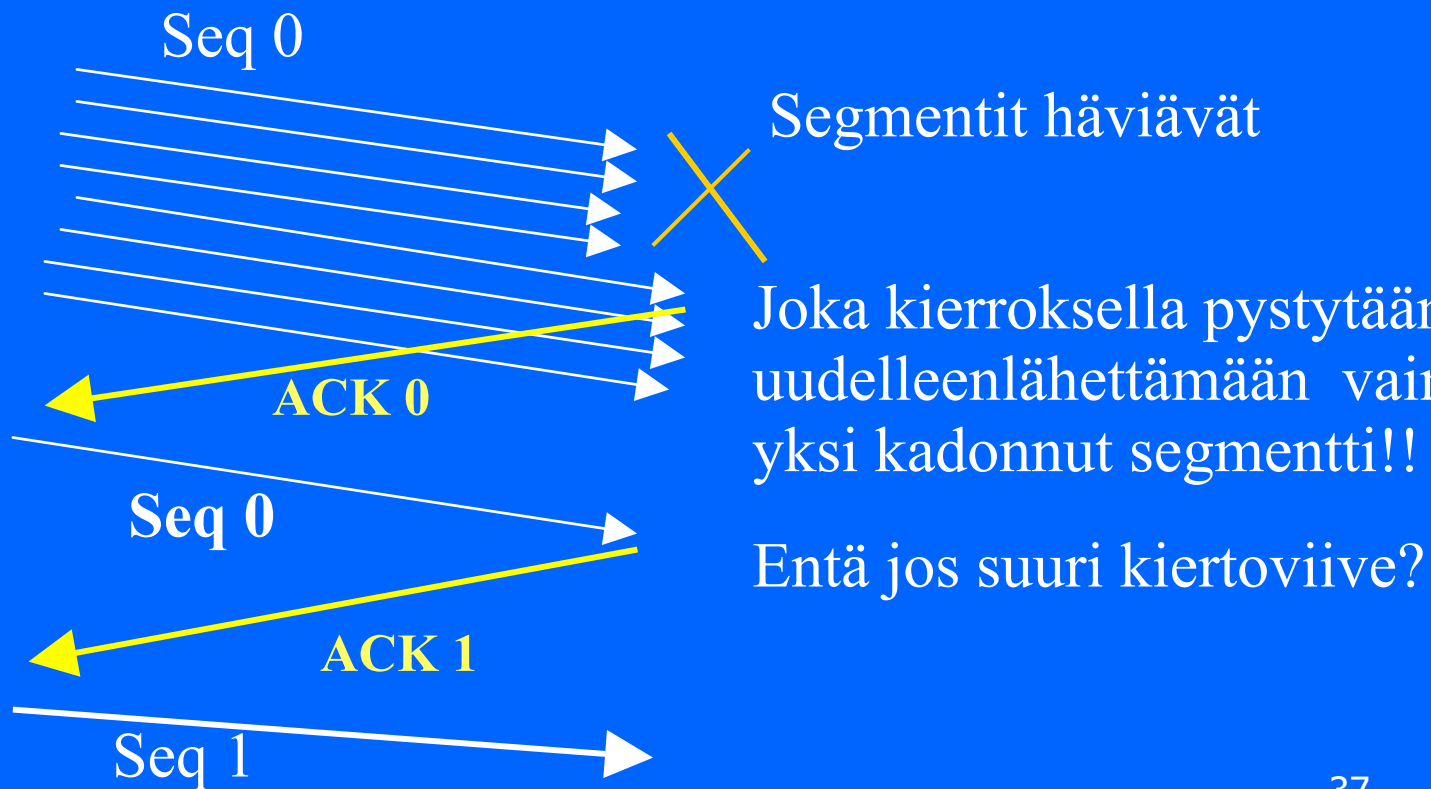
# Ongelma 1: Ei saada kolmea toistokuittausta

- Jos ruuhkaikkuna on hyvin pieni,
  - ei voi tulla kolmea toistokuittausta, jos ruuhkaikkuna sallii vain kolme kuittaamatonta lähetystä



# Ongelma 2: Virheryöppy tuhoaa monta segmenttiä

- Kun useita segmenttejä katoaa 'samasta ikkunasta'





# 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:

---

- Kun lähettäjä saa toistokuittauksen, se saa aina lähettää yhden **uuden paketin** verkkoon
  - kuittaus kertoo, että verkosta poistettu paketti, joten verkkoon siis mahtuu!
- Kun saman paketin toistokuittauksia tulee kolme, niin suoritetaan nopea uudelleenlähetys ja nopea toipuminen (fast recovery)

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

paketti 0

paketti 1

paketti 2

paketti 3

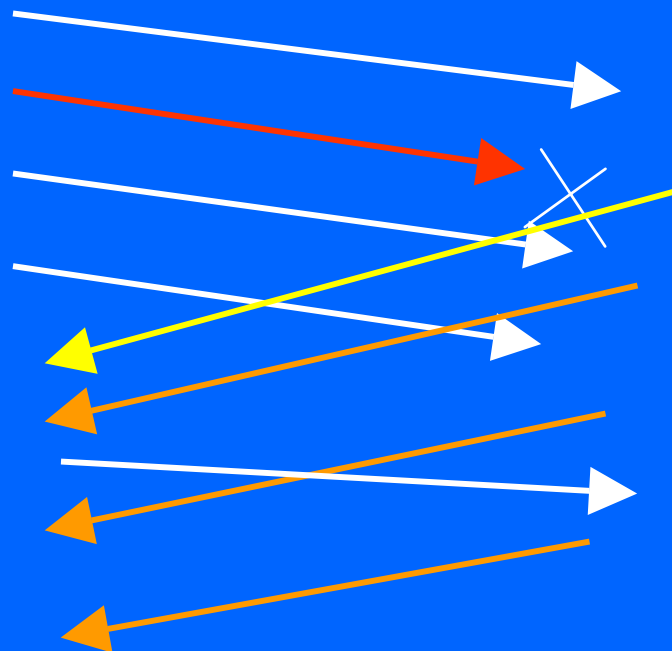
ensikuittaus

1. toistok.

paketti 4

2. toistok.

3. toistok.







# 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 uudelleenlähetyksiä



# SACK (Selective Acknowledgement)

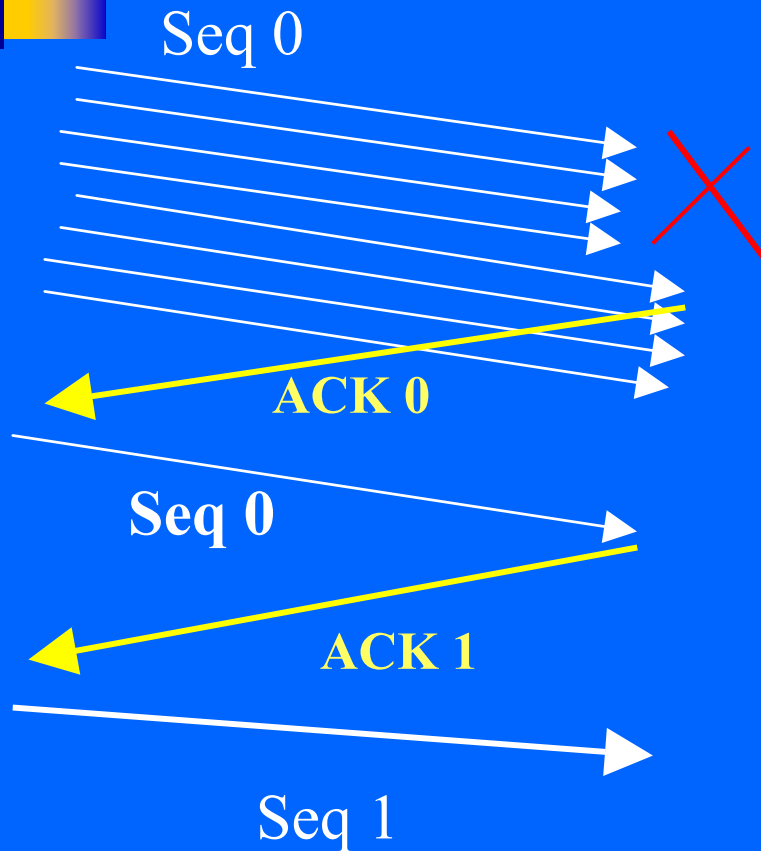
---

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

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

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

# Nopea toipuminen ei onnistu!



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 segmenttivälit on jo vastaanotettu
- Esim. segmentin koko 1000 tavua
  - 1. segmentti katoaa ja muut tulevat perille
    - segmentin 2 kuittaus: ACK 0 , 1000: 2000
    - segmentin 10 kuittaus:ACK 0, 1000: 10000
  - 1. ja 3. segmentti katoavat
    - segmentin 10 kuittaus:
    - ACK 0, 1000:2000 3000:10000



# 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

5	pituus	Optiotyyppi
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).

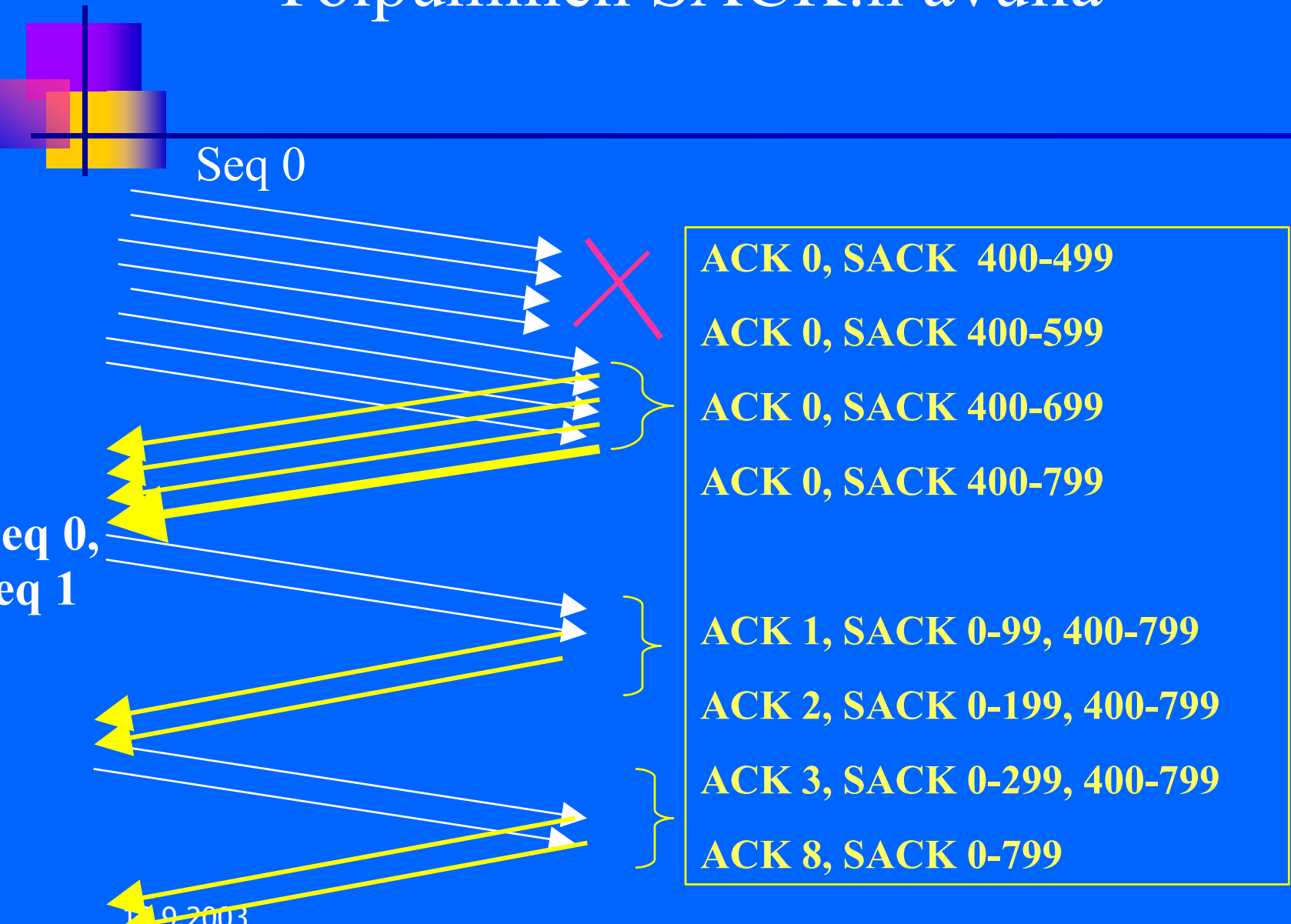


# Vain neuvoa-antava!

---

- 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







# Ruuhkanhallinta SACK:ia käytettäessä

---

- Noudatettava käytettyä ruuhkanhallinta-algoritmia
  - ei uudelleenlähetyistä heti 1. puuttuvan jälkeen
  - rajoitettu lähetys puuttuvan jälkeen
    - toimitaan ruuhkatilanteen mukaan
      - hidas aloitus: 1 tai 2 segmenttiä ensin, kun niihin kuittaus sitten kaksinkertaistetaan lähetys 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

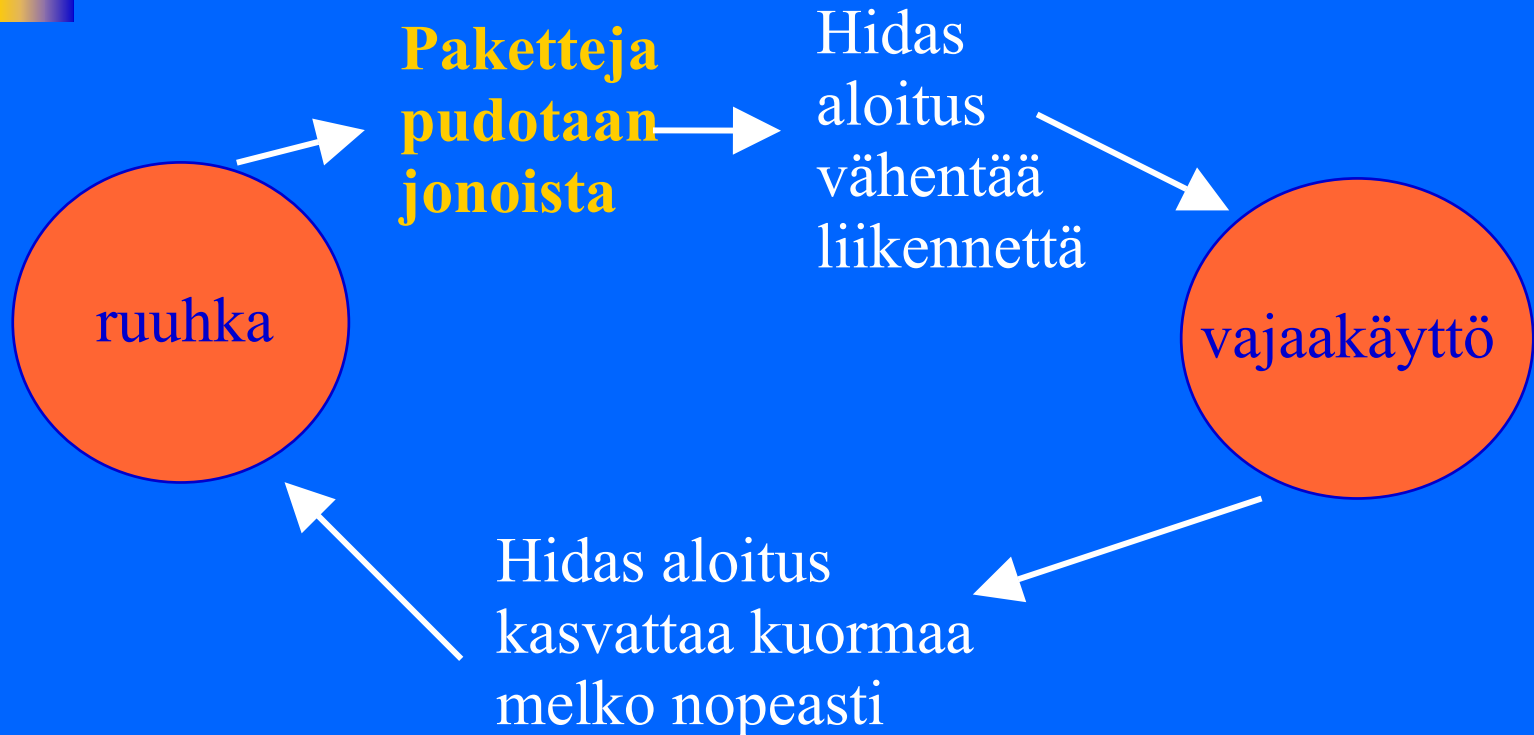


# Globaalin tahdistuksen ongelma (Global Synchronization)

---

- 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 ehtii niin pahaksi, että joudutaan poistamaan paljon paketteja samalla kertaa.
- => Aktiivinen puskurijonon hallinta => RED

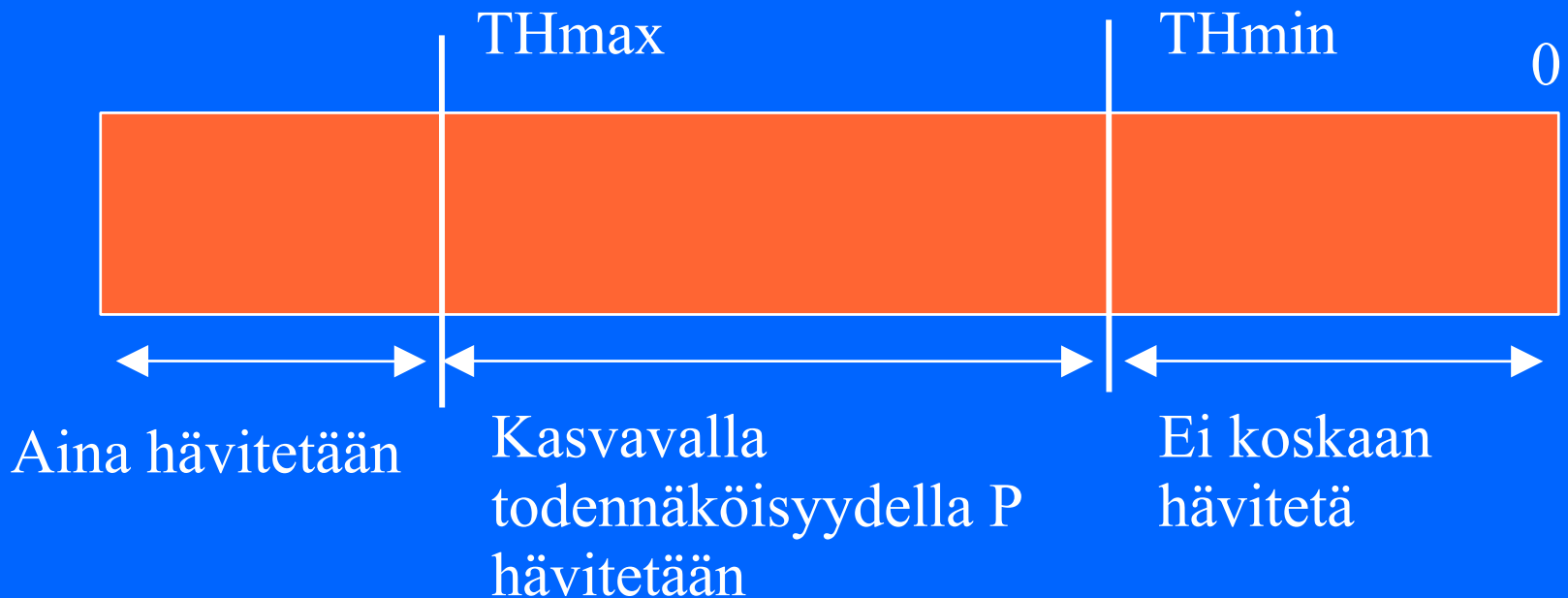


# Miten RED toimii?

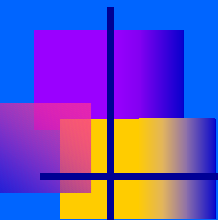
---

- 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

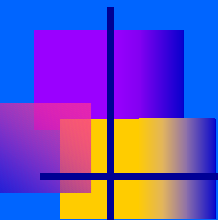


# tasavälein

- estämään hylkäysryöpyt



- mitä pitempään ei ole hylätty yhtään, sitä suurempi hylkäystodennäköisyys
  - avg:n ollessa  $T_{hmin}$  ja  $T_{hmax}$  välissä muuttuj count laskee peräkkäisiä hylkäämättömiä paketteja
  - $P_a <- P_b / (1 - \text{count} * P_b)$
  - $P_a =$  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



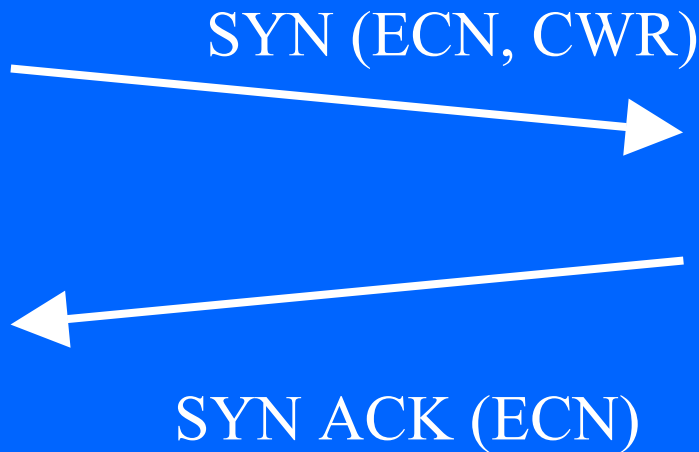
# Muutoksia TCP-hen

---

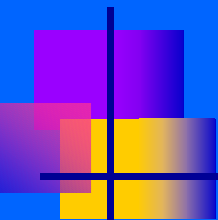
- Sovittava vastapuolen kanssa ECN:n käytöstä
- osattava reagoida CE-bitin asetukseen
  - vastaanottajan ilmoitettava lähettäjälle
    - ECN Echo -lippu
  - lähettäjän vähennettävä lähetystään samalla tavalla kuin jos paketti olisi todella kadonnut
  - lähettäjän ilmoitettava vastaanottajalle, että on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

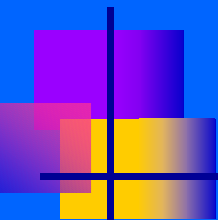
# Yhteydenmuodostus

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





- 
- 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 kynnyisarvo  $ssthresh = \max(\text{FlightSize} / 2, 2 * MSS)$
  - ja talletetaan viimeksi lähetetty järjestysnumero muuttujaan "recover"
  - Lähetetään puuttuva segmentti ja asetetaan ruuhkaikkunan arvoksi
$$cwnd\ to = ssthresh + 3 * MSS$$
  - 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 => 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 tänään sallii
- ja jatketaan toipumisvaihetta





# Eri versioita

---

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



# Uudelleenlähetyksajastin

---

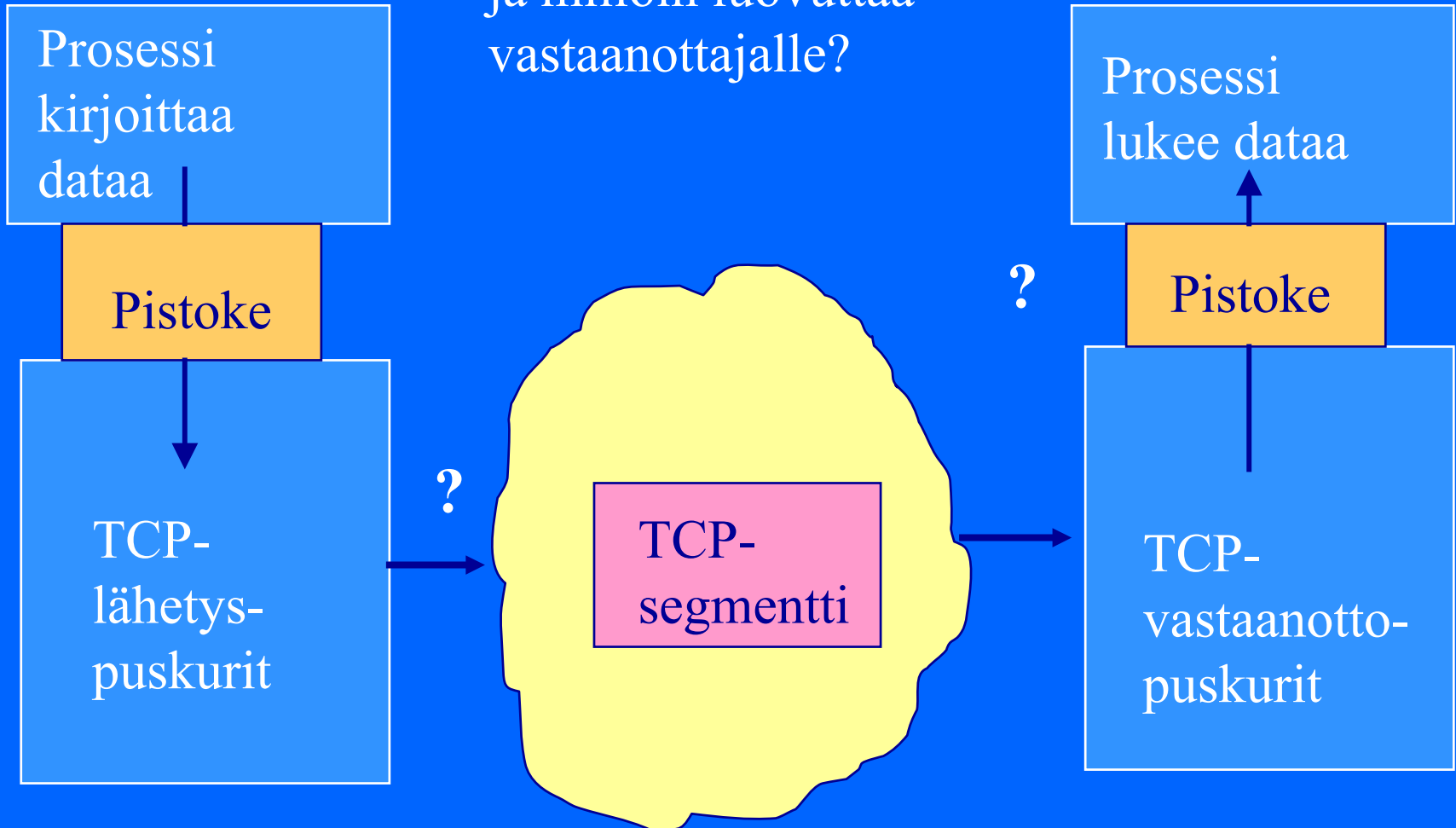
- **RFC 2988: Computing TCP's Retransmission Timer.**

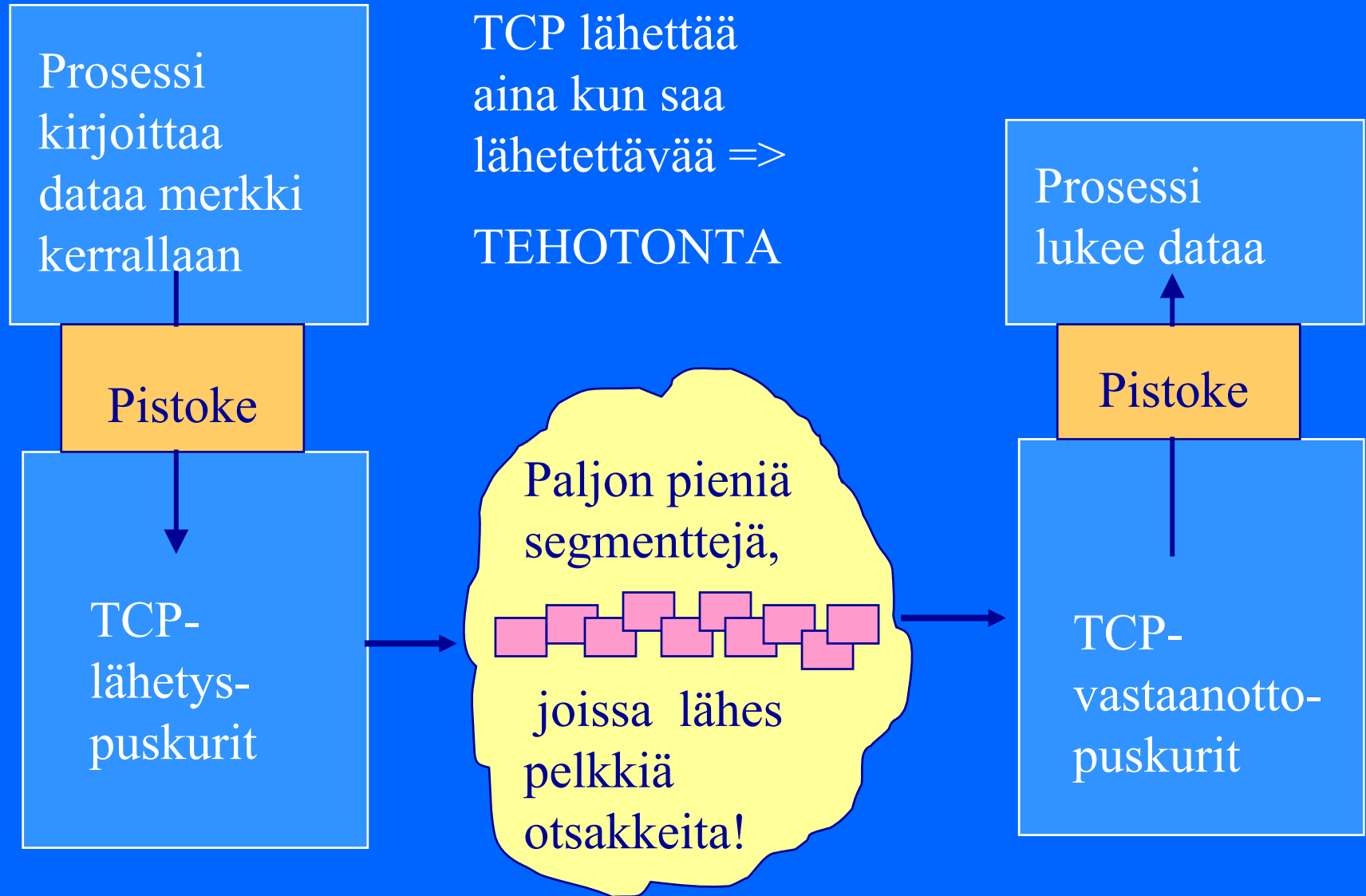
V. Paxson, M. Allman.

November 2000

(Status: PROPOSED STANDARD)

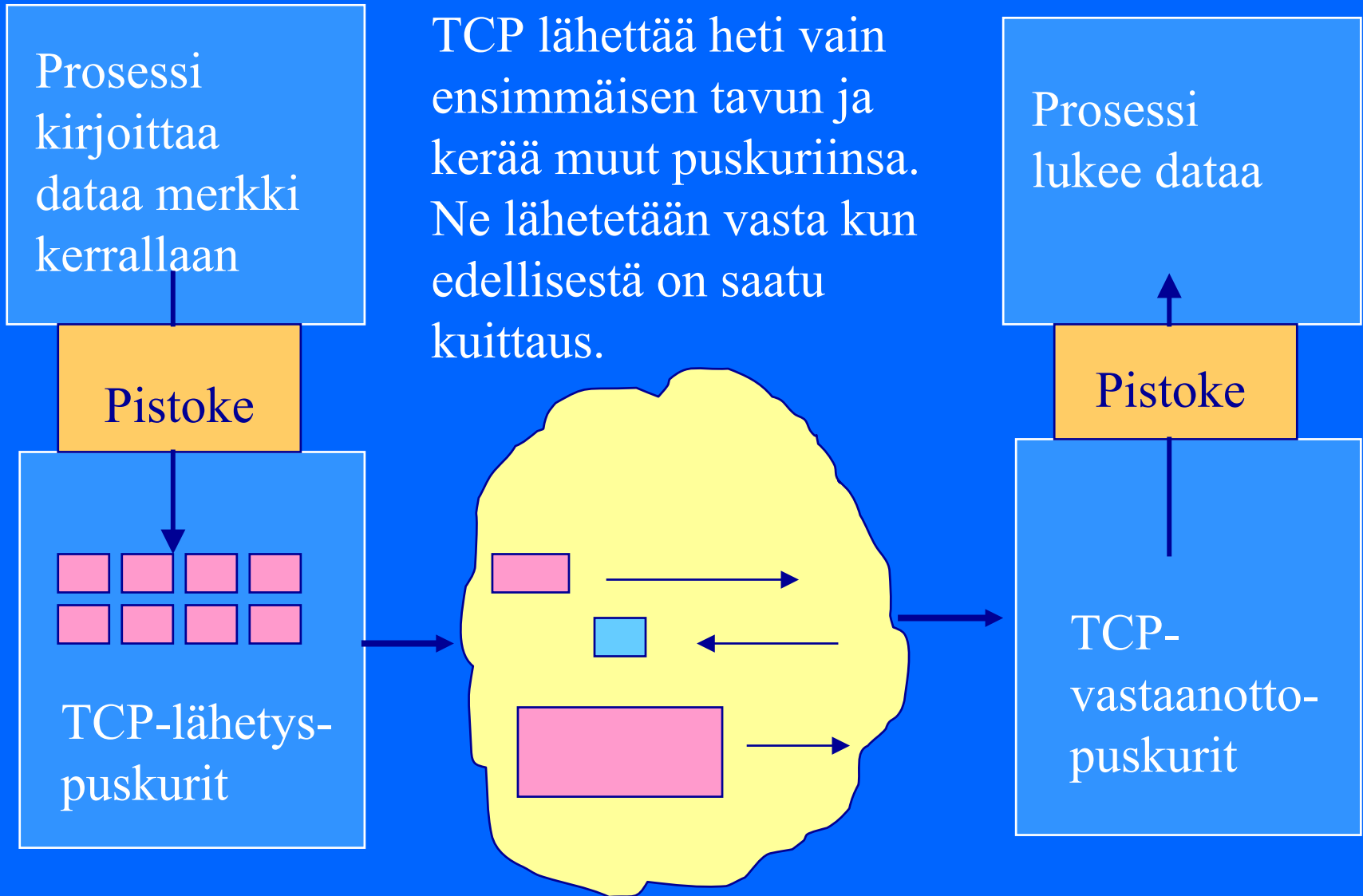
Milloin TCP todella lähettää puskurista ja milloin luovuttaa vastaanottajalle?





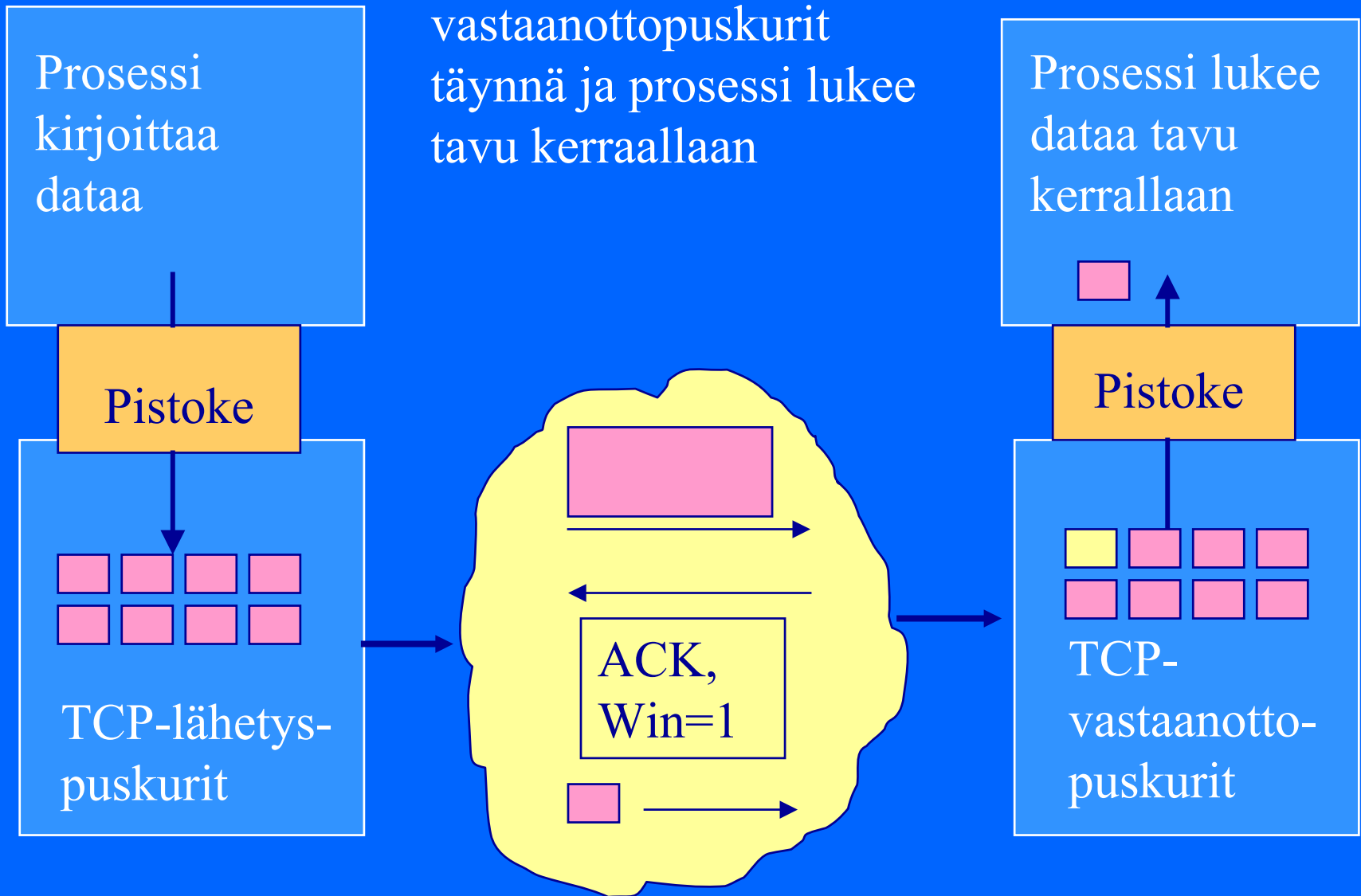
## Naglen algoritmi:

TCP lähettää heti vain ensimmäisen tavun ja kerää muut puskuriinsa. Ne lähetetään vasta kun edellisestä on saatu kuittaus.



# Silly window syndrome:

vastaanottopuskurit  
täynnä ja prosessi lukee  
tavu kerrallaan



## Silly window syndrome:

Clarkin ratkaisu: ei lähetyslupaa lähettäjälle ennenkuin puskureissa on reilusti tilaa. Vähintäänkin maksimisegmentin koko.

