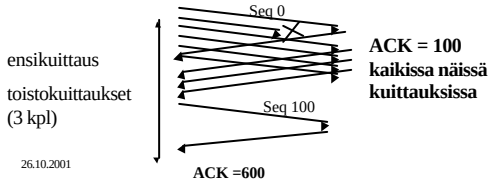
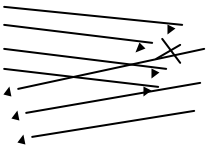
Nopea uudelleenlähetys (Fast retransmit)

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



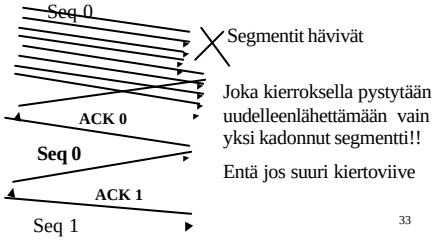
Ongelma 1: Ei saada kolmea toistokuittausta

- 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’



Joka kierroksella pystytään uudelleenlähettämään vain yksi kadonnut segmentti!!

Entä jos suuri kiertoviive

Limited Transmit

- RFC 3042: **Enhansing TCP’s Loss Recovery Using Limited Transmit**.

  M. Allman, H. Balakrishnan, S. Floyd. January 2001

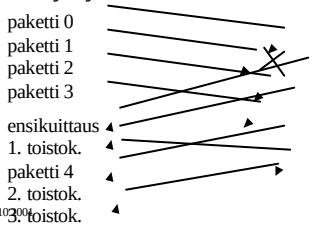
  (Status: PROPOSED STANDARD)

- 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 uudeelleenlähetys 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 uudelleenlähetyksiä

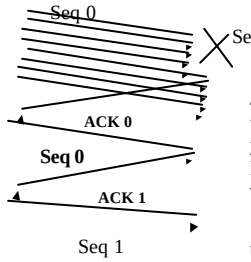
SACK (Selective Acknowledgement)

• 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 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.**

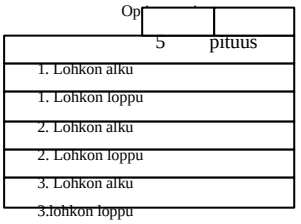
*** jos lähetetään monta uudelleen => voi aiheutua turhia uudelleenlähetyksiä**

- 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

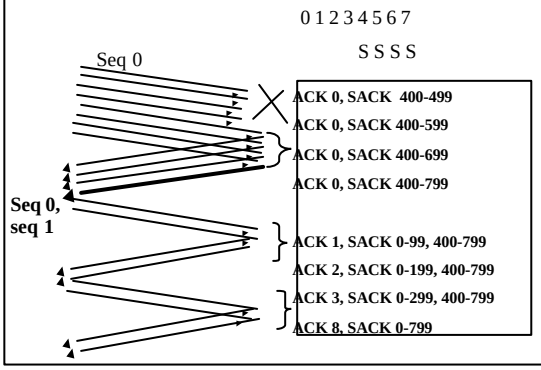


4 lohkoa mahtuu yhteen TCP-segmenttiin, jossa optiolle on varattu 40 tavua, jos ei käytetä muita optioita kuten aikaleimaa (timestamp).

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 uudelleenlähetystä 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

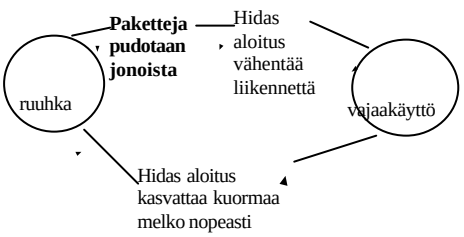
RED (Random Early Detection)

- Aktiivinen, ennaltaehkäisevä puskurijonon hallinta parantaa TCP:n suorituskykyä
  - proaktiivinen <=> 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 uudelleenlähetysajastin 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'.

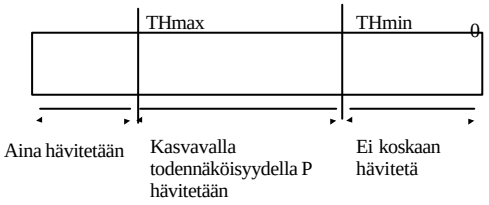
Tehokkaampi puskurijonojen hallinta

- 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 >= 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

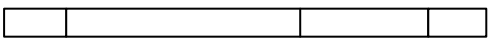
- Pyritään pitämään jonon pituus koko ajan annetuissa rajoissa hävittämällä paketteja kun jonon pituus kasvaa
  - eli voi tulla ruuhka
- Hävittämistodennäköisyys kasvaa, kun jono kasvaa.
- Hävittämistodennäköisyys kasvaa, kun paketteja ei ole hävitetty

Kun paketti saapuu FIFO-tyyppiseen ulostulojonoon:

- Lasketaan keskimääräinen jononpituus painotettuna keskiarvona aikaisemmista jononpituuksista
- avg <- (1-wq)*avg + wq* q
- jos jonon pituus q = 0
  - m<- f(time - q_time)
    avg<- (1-wq)**m * avg
    arvioidaan, kuinka monta pikkupakettia (= m) olisi voitu siirtää sinä aikana, jonka jono on ollut tyhjänä
- wq ~ 0.002 => avq reagoi hitaasti jonon pituuden muutoksiin

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=> kun agv=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 Thminin ja Thmaxin välissä muuttuja **count** laskee peräkkäisiä hylkäämättömiä paketteja
  - Pa <- Pb /(1-count*Pb)
  - Pa = 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!

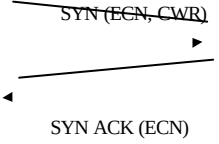
- Käyttöön 2 bittiä, joita käytetään ruuhkasta ilmoittamiseen
  - CE-bitti (Congestion Experienced)
    - » reititin asettaa, kun on havainnut ruuhkaa
    - » esim. kun RED-algoritmin jononpituus indikoi ruuhkaa
  - ECT-bitti (ECN Capable Transport)
    - » kertoo, että kuljetuskerros kykenee käsittelemään ruuhkailmoituksia (Explixit 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 ilmoittava 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 kynnysarvoa
- 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)

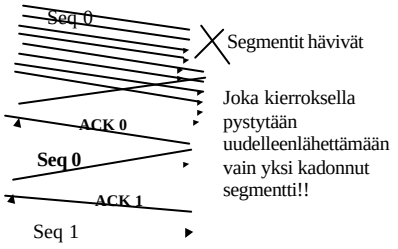
- RFC 2582: **The NewReno Modification to TCP's Fast Recovery Algorithm**

  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 (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 tänän 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!

Uudelleenlähetysajastin

- RFC 2988: **Computing TCP's Retransmission Timer.**
  V. Paxson, M. Allman.
  November 2000
  (Status: PROPOSED STANDARD)