TCP:n vuonohjaus (flow control)

W. Stallings, High-Speed Networks, TCP/IP and ATM Design Principles, Prentice-Hall, 1998, Sections 10.1-10.2

- Ikkunointipohjainen vuonohjausmekanismi
- Pyrkii jakamaan kaistan tasan eri lähteiden kesken
- Timeout ja uudelleenlähetys
  - kiertoajan (RTT) mittaus
  - RTT:n varianssin arviointi (Jacobsonin algoritmi)
  - eksponentiaalinen RTO:n vetäytyminen
  - Karnin algoritmi
- Hidas aloitus (slow start)
- Dynaaminen ikkunan säätö
- Nopea uudelleenlähetys (fast retransmit)
- Nopea toipuminen (fast recovery)
- Selektiivinen kuittaus (selective acknowledgement), SACK (optionaalinen lisäys)

Mekanismien toteutus TCP:n eri versioissa

Mekanismi	RFC 1122	TCP Tahoe	TCP Reno	NewReno
RTT varianssin estimointi	√	√	√	√
Eksponentiaalinen RTO:n vetäytyminen	√	√	√	√
Karnin algoritmi	√	√	√	√
Hidas aloitus	√	√	√	√
Dynaaminen ikkunan säätö	√	√	√	√
Nopea uudelleenlähetys		√	√	√
Nopea toipuminen			√	√

- NewReno (RFC 2582, April 1999) on nykyisin yleisimmin käytetty TCP:n versio
- Sisältää parannetun nopean uudelleenlähetyksen useiden paketinmenetysten tapauksessa

Timeout ja uudelleenlähetys

- Jokaista lähetettyä segmenttiä kohti pidetään yllä uudelleenlähetysajastinta RTO (retransmission timeout)
- Ajastimen pyrkimyksenä on erotella tapaukset, joissa
  - kuittaus viipyy satunnaisten kulkuaikaviiveiden vuoksi
  - verkossa on ruuhkaa ja lähetetty segmentti on kadonnut

$$\mathrm{RTO} = \mathrm{SRTT} + f \times \mathrm{SDEV}$$

$$\begin{cases} \mathrm{RTO} &= \text{retransmission timeout} \\ \mathrm{SRTT} &= \text{smoothed roundtrip estimate} \\ \mathrm{SDEV} &= \text{smoothed roundtrip standard deviation estimate} \end{cases}$$

  - kertoimelle f käytetään tavallisesti arvoa $f = 4$ (alkuperäinen suositus $f = 2$)

Kiertoajan ja sen vaihtelun arviointi (Jacobsonin algoritmi)

- Jokaista lähetettyä segmenttiä kohti mitataan aika RTT kuittauksen saapumiseen
- Lasketaan mitatun kiertoajan erotus nykyiseen tasoitettuun estimaattiin SRTT

$$\text{SERR} = \text{RTT} - \text{SRTT}$$

- Päivitetään kiertoajan estimaattia SRTT liukuvasti (eksponentiaalinen tasoitus)

$$\text{SRTT} \leftarrow (1-g) \times \text{SRTT} + g \times \text{RTT}$$

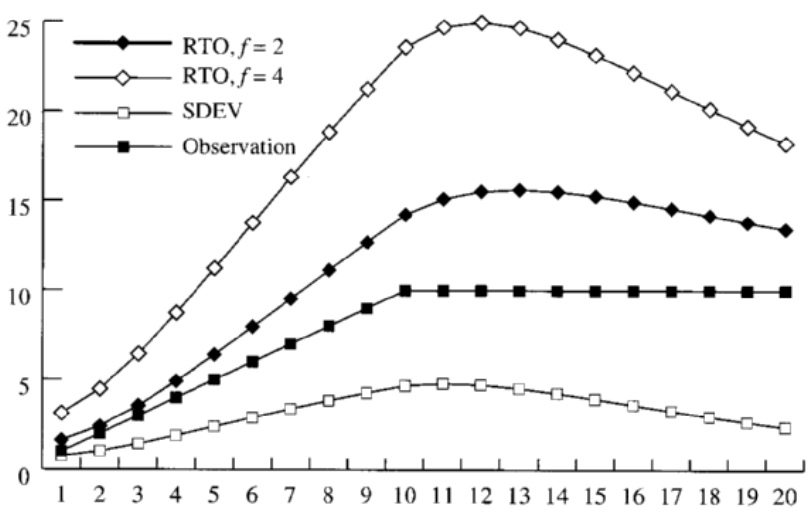
- Samoin päivitetään kiertoajan vaihtelun SDEV estimaattia liukuvasti

$$\text{SDEV} \leftarrow (1-h) \times \text{SDEV} + h \times |\text{SERR}|$$

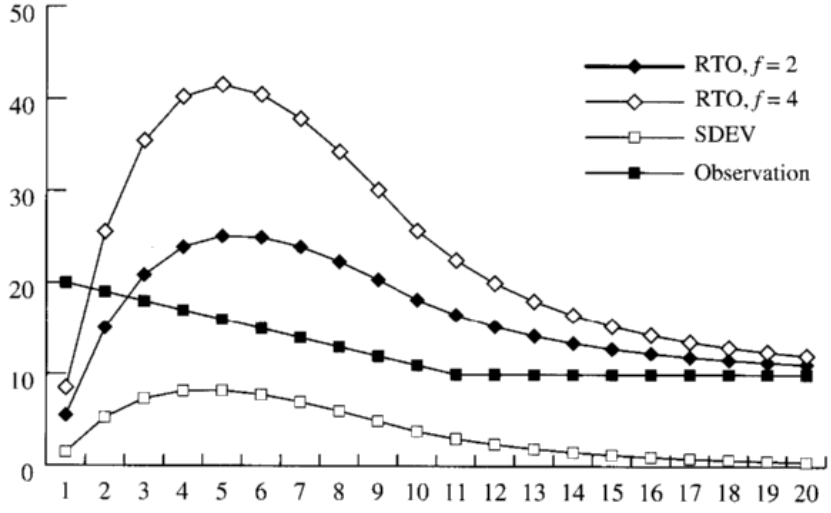
  – kertoimille g ja h suositellaan arvoja

$$\begin{cases} g = 1/8 = 0.125 & \text{‘keskiarvo 8 viimeisestä mittauksesta’} \\ h = 1/4 = 0.25 & \text{‘keskiarvo 4 viimeisestä mittauksesta’} \end{cases}$$

Esimerkki estimaattien kehittymisestä[1]



(a) Increasing function



(b) Decreasing function

[1] from W. Stallings *High Speed Networks: TCP/IP and ATM Design Principles*, Prentice Hall, 1997.

Eksponentiaalinen RTO:n vetäytyminen (exponential RTO backoff)

- Ajastimen RTO lauettua segmentti lähetetään uudelleen
  - merkki siitä, että verkossa on ruuhkaa
  - saman RTO:n käyttäminen uudelle segmentille ei olisi kovin järkevää, koska se saattasi helposti johtaa ajastimen laukeamiseen uudelleen
  - RTO-arvoa kasvatetaan tekijällä q

$$\text{RTO} \leftarrow q \times \text{RTO}$$

- Jos ajastin edelleen laukeaa, jatketaan samalla tavalla (jolloin RTO kasvaa eksponentiaalisesti johonkin ylärajaan (esim. 64 s) asti
- Kertoimelle q käytetään useimmiten arvoa $q = 2$
  - kyseessä on tällöin binäärinen eksponentiaalinen vetäytyminen
  - samanlainen kuin Ethernetin CSMA/CD-protokollassa
- Uudelleenlähetyksiä yritetään uudelleen asetettuun katkaisuaikaan (esim. 9 min) asti

Karnin algoritmi

- Kun uudelleenlähetyksen jälkeen saadaan segmentistä kuittaus, ei voida tietää liittyykö kuittaus alkuperäiseen lähetykseen vai uudelleenlähetykseen
- Kiertoajasta RTT ei saada tällöin luotettavaa mittausta
- Karnin algoritmi määrittelee menettelytavan tässä tapauksessa:
  1. SRTT- ja SDEV-arvoja ei päivitetä
  2. Uudellenlähetyksen tapauksessa RTO-arvoa kasvatetaan tekijällä q
  3. Seuraaville segmenteille käytetään samaa RTO-arvoa
  4. Vasta, kun on saatu kuittaus ei-uudelleenlähetettyyn pakettiin, siirrytään 'normaaliin päiväjärjestykseen'

Hidas käynnistys (slow start)

- Lähetysikkunan $awnd$ koko määritellään segmentteinä (ei oktetteina)
- Sitä säätelee kaksi tekijää: $awnd = \min(credit, cwnd)$

  $awnd$ = sallittu lähetysikkunan koko segmenteissä
  $cwnd$ = TCP:n vuonohjauksen ylläpitämä ruuhkaikkuna (congestion window)
  $credit$ = vastaanottajan ilmoittama sallittu lähetysikkuna;
  lasketaan vastaanottajalta saadun TCP-segmentin $window$-kentän arvosta
  muuntamalla segmenteiksi: $credit = window \,/\, segment\ size$
- TCP-yhteyden käynnistyessä asetetaan $cwnd = 1$
- Jokaista saatua kuittausta kohti $cwnd$-arvoa kasvatetaan yhdellä aina johonkin maksimiin asti

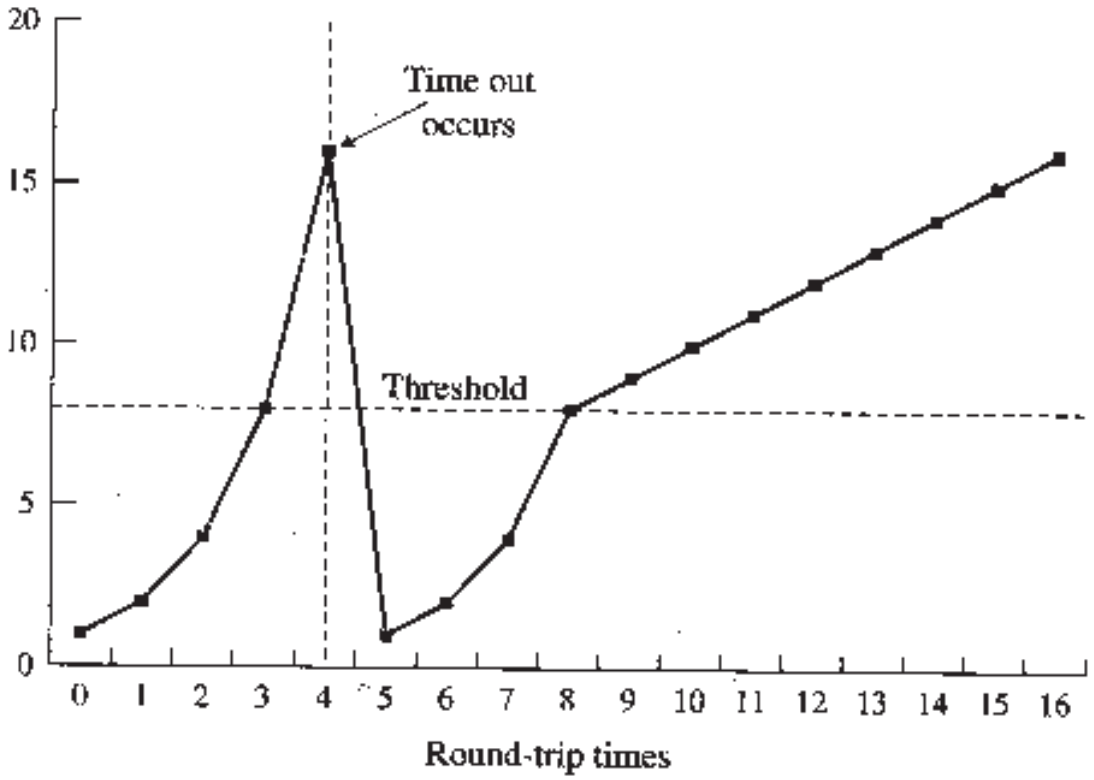
$$cwnd \leftarrow cwnd + 1$$

- Itse asiassa käynnistyminen ei ole lainkaan hidas vaan eksponentiaalinen
  - jokaista kuitattua täyttä ikkunallista kohti ikkunan koko kaksinkertaistuu
  - 'ahneen' lähteen tapauksessa kaksinkertaistuminen tapahtuu RTT:n välein

Dynaaminen ikkunan säätö ruuhkatilanteessa

- Kun segmenttiin liittyvä timeout RTO laukeaa, tämä on merkki verkossa olevasta ruuhkasta
  - segmentti on joko kadonnut kokonaan tai ainakin pahasti viivästynyt matkallaan
- Tällöin siirrytään hitaan käynnistyksen mukaiseen alkutilaan
  - asetetaan $cwnd = 1$
  - kasvatetaan ikkunaa yhdellä jokaista kuittausta kohti
- Ikkunan eksponentiaalinen kasvu on ruuhkatilanteessa liian agressiivista
- Varoivasempi menettely (congestion avoidance) on seuraava:
  1. Aseta kynnysraja $ssthresh$ puoleen nykyisestä ikkunasta $ssthresh = cwnd/2$
  2. Aseta $cwnd = 1$ ja jatka hitaan käynnistyksen mukaisesti kunnes $cwnd = ssthresh$
  3. Tästä eteenpäin (kun $cwnd \geq ssthresh$) $cwnd$-arvoa kasvatetaan yhdellä jokaista kiertoaikaa kohti (täyttä ikkunallista kohti). Tämä voidaan toteuttaa esim. päivittämällä $cwnd$-arvoa jokaista kuittausta kohti seuraavasti: $cwnd \leftarrow cwnd + 1/cwnd$.

'Hidas käynnistyminen' ja dynaaminen ikkunan säätö[2]



[2]from W. Stallings *High Speed Networks: TCP/IP and ATM Design Principles*, Prentice Hall, 1997.

Nopea uudelleenlähetys (fast retransmission)

- Uudelleenlähetysajastimen RTO arvo on käytännössä selvästi suurempi kuin todellinen kiertoaika RTT
- Tämä on hyvin perusteltua, koska kiertoajan luotettava mittaaminen on useista eri syistä vaikeaa
- Seuraus tästä kuitenkin on, että pelkästään ajastimen laukeamiseen perustuvat uudelleenlähetykset voivat tapahtua hitaasti
- Nopeassa uudelleenlähetyksessä käytetään hyväksi sitä, että TCP lähettää aina kuittauksen viimeisestä oikeassa järjestyksessä saadusta segmentistä
- Toistuvat saman segmentin kuittaukset ovat merkkinä puuttuvasta, mahdollisesti kokonaan kadonneesta segmentistä
- Sen poissulkemiseksi, että kyseessä olisi vain jonkin verran viivästynyt segmentti, odotetaan että saadaan kolme kuittausta samasta segmentistä
- Nopeassa uudellenlähetyksessä tämä tulkitaan merkiksi paketin todellisesta katoamisesta ja sitä käytetään laukaisemaan uudelleenlähetys

Nopea toipuminen (fast recovery)

- Myös nopean uudelleenlähetyksen tapauksessa ruuhkaikkunan kokoa tulee pienentää ruuhkan poistamiseksi
- Saadut kuittaukset ovat kuitenkin merkkinä siitä, että jotain menee verkosta läpi, eikä ruuhka ole ehkä niin paha kuin ajastimen lauetessa
- Tämän vuoksi ikkunan kokoa ei pienenennetäkään 1:een, vaan 'nopean toipumisen' menetelmän mukaisesti toimitaan seuraavasti (kolmannen kuittauksen saapuessa samasta segmentistä):
  1. Aseta $ssthresh = cwnd/2$
  2. Lähetä puuttuva segmentti uudelleen
  3. Aseta $cwnd = ssthresh + 3$ (koska kolme segmenttiä on kuitenkin mennyt perille)
  4. Aina, kun samasta segmentistä tulee vielä lisäkuittauksia, kasvata *cwnd*-arvoa yhdellä
  5. Kun lopulta tulee kuittaus uudelleenlähettämättömästä segmentistä (joka kuittaa kerralla kaikki puuttuvat ja myöhemmät segmentit), asetetaan $cwnd = ssthresh$

Selektiivinen kuittaus, SACK

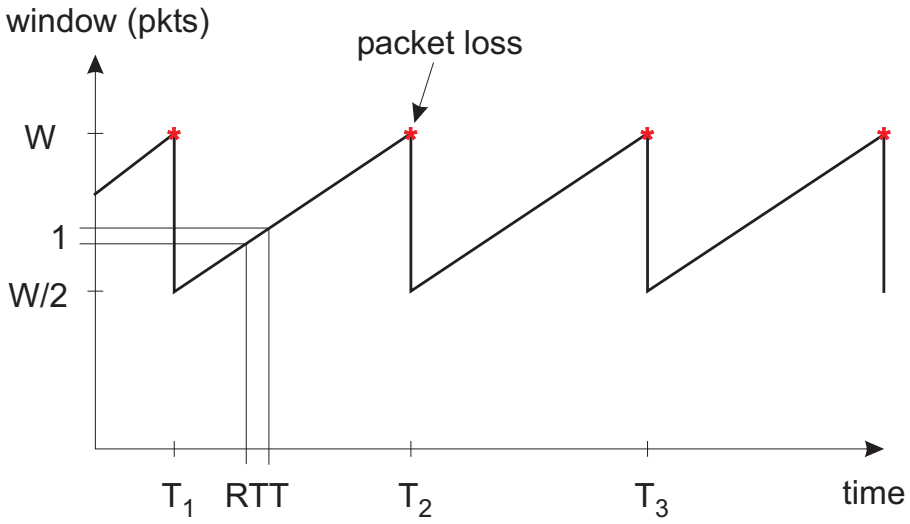
- Valinnainen mekanismi (RFC 2018, lokakuu 1996)
- Vastaanottaja voi kuitata myös väärässä järjestyksessä saapuneita paketteja
- Vaatii, että TCP-yhteyden molemmat päät tukevat sitä
  - yhteys toimii ilman SACK:ia, jos vain toinen pää tukee sitä

TCP Vegas

- Uusi kokeellinen TCP:n versio
- Käyttää kolmea eri tekniikkaa läpäisyn lisäämiseksi ja häviöiden pienentämiseksi
  - ensimmäinen tekniikka johtaa hävinneiden segmenttien nopeampaan uudelleenlähetykseen
  - toinen tekniikka antaa TCP:lle kyvyn ennakoida ruuhkaa ja säätää lähetysnopeutta vastaavasti
  - kolmas tekniikka modifioi hitaan käynnistyksen mekanismia paketinmenetysten välttämiseksi etsittäessä käytettävissä olevaa kaistaa

TCP:n läpäisy

- Yksinketainen Floydin ja Fallin[3] malli
  - stationaarinen malli (pitkät vuot) TCP:lle, joka toimii ruuhkanvälttämismoodissa
  - keskimääräinen paketinmenetystodennäköisyys p ja RTT määräävät läpäisyn
  - naiivi jaksollisuusoletus: joka $1/p$:s paketti menetetään



[3] S. Floyd and K. Fall, *Promoting the use of end-to-end congestion control in the Internet*, IEEE/ACM Trans. on Networking, 1999.

TCP:n läpäisy (jatkoa)

- Hävinneiden pakettien välillä (esim. aikaväli T_1 and T_2) siirrettyjen pakettien määrä

$$\frac{W}{2} + (\frac{W}{2} + 1) + \cdots + W \approx \frac{3}{8}W^2$$

- Yksi hävinnyt paketti $(3/8)W^2$ lähetettyä pakettia kohti. Siis

$$p = \frac{8}{3W^2} \qquad \Rightarrow \qquad W = \sqrt{\frac{8}{3p}}$$

- Pakettihäviöiden väliaika on $W/2$ kiertoaikaa. Saamme läpäisyn T

$$T = \frac{(3/8)W^2 B}{(W/2)RTT} = \frac{3}{4}\frac{W \cdot B}{RTT} = \frac{\sqrt{3/2}\, B}{RTT\sqrt{p}} \approx \frac{1.22\, B}{RTT\sqrt{p}}$$

  missä B on yhden segmentin koko

TCP:n läpäisy (jatkoa)

- Tarkemman mallin ovat esittäneet Padhye et al.[4]
- Ottaa huomioon uudellenlähetyksen ajastimen (RTO) laukeamisen sekä viivästetyt kuittaukset

$$T \approx \min\left\{\frac{W_m B}{RTT}, \frac{B}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\{1, 3\frac{3bp}{8}\}\, p\,(1+32p^2)}\right\},$$

  missä
  - W_m on vastaanottajan asettama ruuhkaikkunan suurin arvo
  - b on yhdellä vastaanotetulla ACK:illa kuitattujen pakettien lukumäärä
  - T_0 on RTO:n ensimmäinen arvo

[4] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, *Modeling TCP throughput: a simple model and its empirical validation*, in Proc. of ACM SIGCOMM, 1998.