



Luento 6: Kuljetuskerros

UDP & TCP

TCP:n ruuhkanhallinta

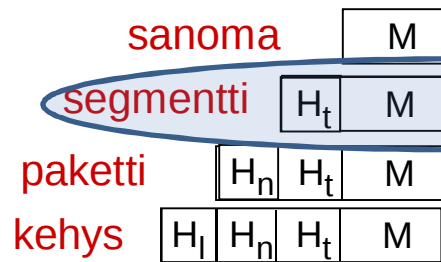
To 14.11.2013
Tiina Niklander

Kurose&Ross
Ch3

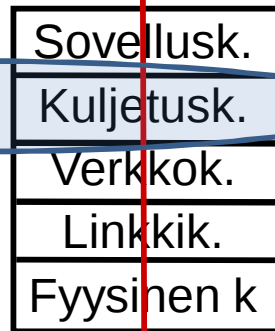
Pääasiallisesti kuvien
© J.F Kurose and K.W. Ross,
All Rights Reserved

Luennon sisältöä

Lähettäjä (sender)



message,
segment
datagram
frame



kytkin

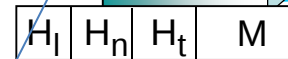
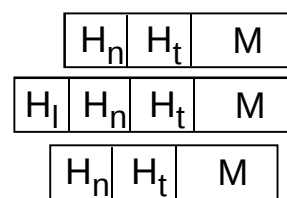
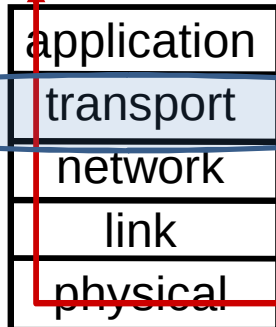
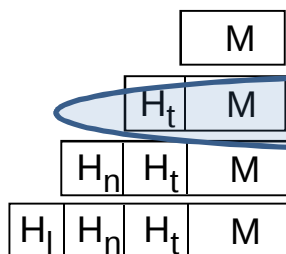


Fig 1.24 [KR12]

Vastaanottaja (recipient)



reititin



Sisältöä

Kuljetuspalvelut

Luotettavan kuljetuspalvelun periaatteet

Yhteydellinen kuljetuspalvelu, TCP

Ruuhkanhallinta TCP:ssä

Yhteydetön kuljetuspalvelu, UDP

Oppimistavoitteet:

- Tuntea Internetin kuljetusprotokollien (UDP/TCP) toiminnallisuus ja periaatteet
- Osata luotettavan kuljetuspalvelun ja vuonvalvonnan periaatteet ja toteutukset
- Osata TCP-ruuhkanhallinnan





UDP (User Datagram Protocol) (RFC 768)

Yhteydetön

KJ ei pidä tallessa mitään sovellusten väliseen keskusteluun liittyvää.
Sovellus antaa aina sanoman lisäksi kohdeosoitteen ja kohdeportin

Ei takaa luotettavuutta (~'epäluotettava'?)

vain minimipalvelu: mille koneelle, mihin porttiin, voi kadottaa sanoman

Ei säilytä sanomien järjestystä

Sovellus saa sanomat siinä järjestyksessä kuin ne tulevat perille

Vähän yleisrasitetta

Aikaa ei kulu yhteyden muodostukseen eikä purkuun

Ei resursseja yhteyden tilatietojen ylläpitoon

Pieni otsake (eli vähän itse protokollaan liittyviä tietoja)

Ruuhkanhallinta ei säännöstele liikennettä



TCP (Transmission Control Protocol) [RFC 793]

Yhteydellinen palvelu (connection-oriented)

Yhteyden muodostus ennen datan siirtoa (handshaking)

Kaksisuuntainen TCP-yhteys (full-duplex)

Yhteyden purku (shutdown)

Luotettava kuljetuspalvelu

Järjestyksen säilyttävä tavuvirta sovellukselle

järjestysnumerot, kuittaukset, uudelleenlähetykset

Vuonvalvonta (flow control)

Lähettäjä hiljentää vauhtia, jos vastaanottaja ei ehdi käsitellä

Ruuhkanhallinta (congestion control)

Lähettäjä hiljentää vauhtia, jos reitittimet eivät ehdi käsitellä

Verkkosovelluksia

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Figure 3.6 ♦ Popular Internet applications and their underlying transport protocols

K&R12

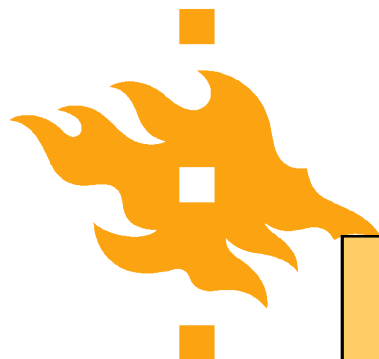
Miksi toiset sovellukset suosivat UDP:tä ja toiset TCP:tä?



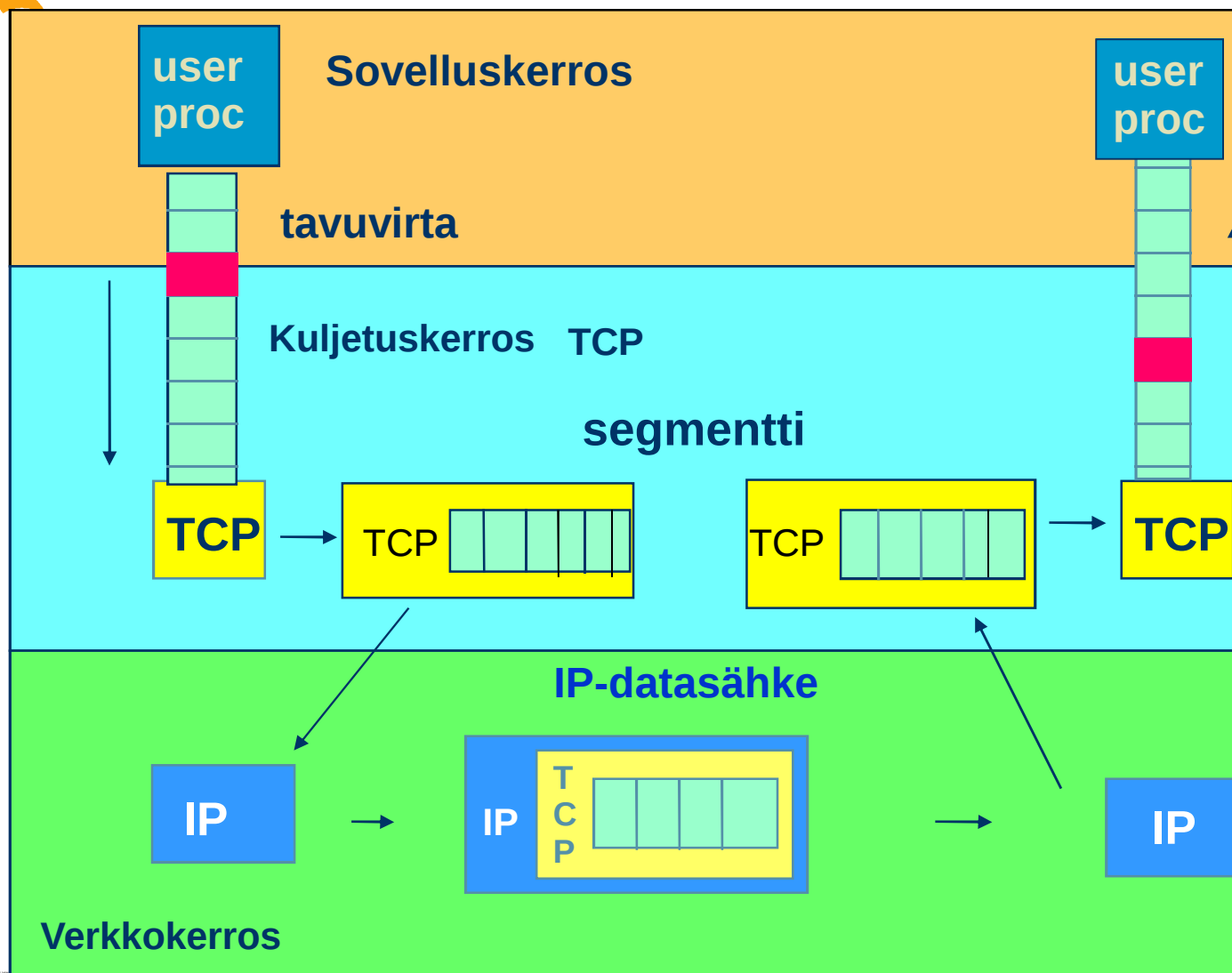
Kuljetuskerros

Yhteydellinen kuljetuspalvelu **TCP**

RFC 793,
RFC 1122,
RFC 1323,
RFC 2018,
RFC 2581



TCP: prosessilta prosessille -tavuvirta





TCP-protokolla

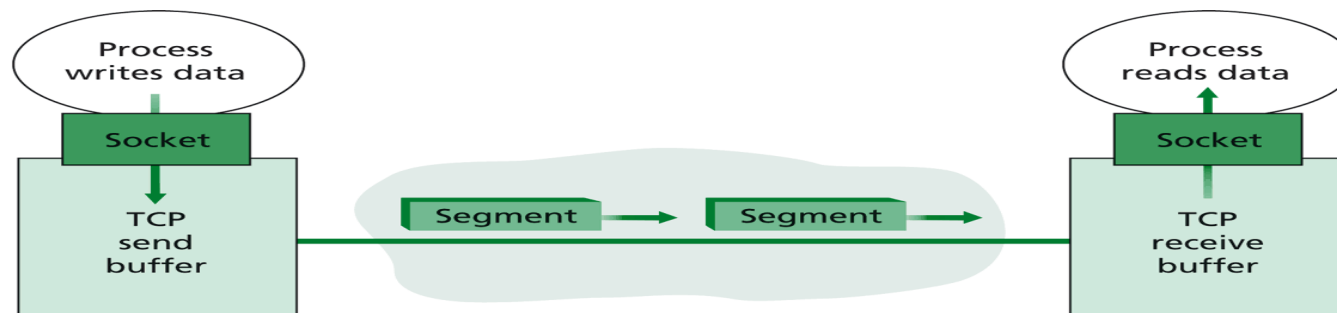
- **Päästä-päähän kuljetuspalvelu**
 - Yksi lähettäjä, yksi vastaanottaja
 - Reitittimet eivät ole kiinnostuneita kuljetustason protokollasta
- **Yhteydellinen** (connection-oriented)
 - Yhteydenmuodostus
 - Isäntäkoneissa: puskuritila, ikkunakoko, tavunumerointi
 - Yhteyden purku
- **Kaksisuuntainen** (full duplex)
 - Yksi yhteys, jossa yhtä aikaa liikennettä molempiin suuntiin
- **Luotettava, järjestyksen säilyttävä tavuvirta**
 - Ei sanomarajoja
 - Tavunumerointi
 - Kumulatiiviset kuittaukset



TCP-protokolla

Fig 3.28 [KR12]

- Vuonvalvonta, ruuhkanhallinta (-valvonta)
 - Lähettäjä ei voi tukahduttaa vastaanottajaa eikä reitittimiä
- Liukuvan ikkunan protokolla
 - Vuonvalvonta ja ruuhkanhallinta vaikuttavat lähetysikkunan kokoon
- Puskurointia molemmissa päissä
 - Uudelleenlähetystä varten
 - Jotta saadaan annettua sovellukselle järjestyksessä





TCP-protokolla

Fig 3.30 [KR12]

- Segmentillä maksimikoko
 - MSS (maximum segment size) = paljonko dataa segmentissä
- Varmistaa, että tässä koneessa ei tarvita lisäpilkkomista paketeiksi
- Linkkikerroksen fyysiset ominaisuudet vaikuttavat MSS:n arvoon
 - MTU (maximum transfer unit)
 - Ethernet MTU = 1500 => MSS = 1460, sillä TCP:n ja IP:n otsakkeet (20 +20 tavua) vievät oman tilansa

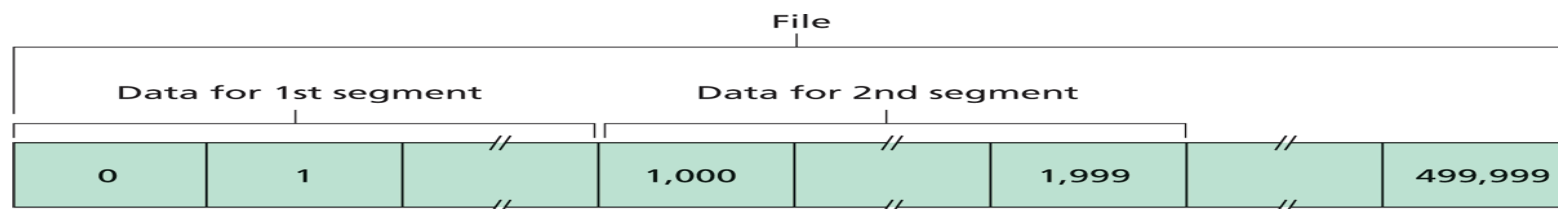


Figure 3.30 ♦ Dividing file data into TCP segments



TCP-segmentti

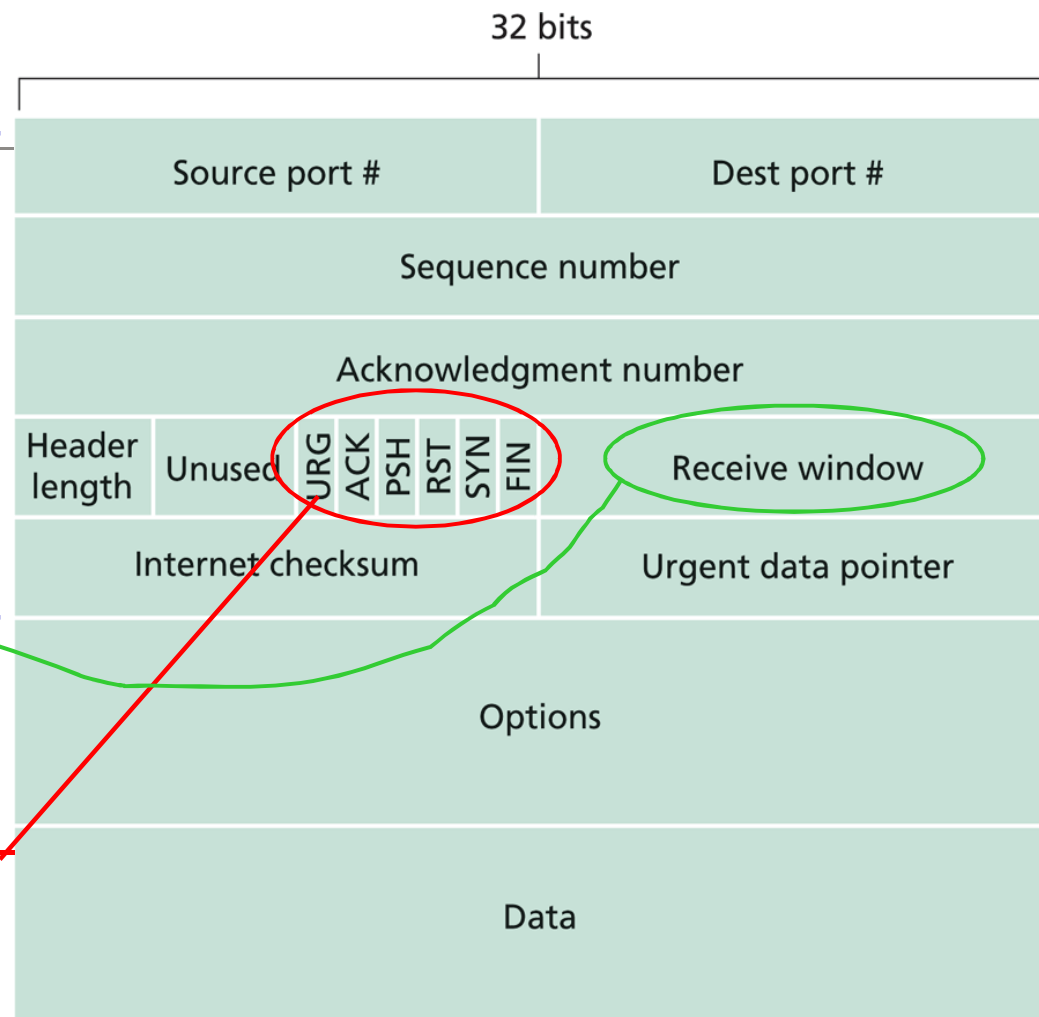
Fig 3.29 [KR12]

Otsake aina vähintään 20 B
Optio-osa tarvittaessa

Järjestys- ja kuittaus-
numerot **tavunumeroina**

Ikkunankoko: paljonko tilaa
vastaanottopuskurissa (tavua)

ACK= kuittausnumero validi,
RST (reset),
SYN yhteydenmuodostus
FIN yhteydenpurku
URG, PSH ei yleensä käytetä





Tavunumerointi

Kuittauksia ei kuitata ja ne eivät siirrä numerointia!

Niissä ei siirretä tavuvirtaa.

Tavuvirtaa ...

Segmentit voivat olla erikokoisia ($\leq$ MSS)

Järjestysnumero=

- ensimmäisen tavun numero
- alkuarvot sovitaan yhteyttä muodostettaessa

Kuittaus

- seuraavaksi odotetun tavun numero
- kumulatiivinen
- kylkiäisenä (piggybacked) mikäli mahdollista

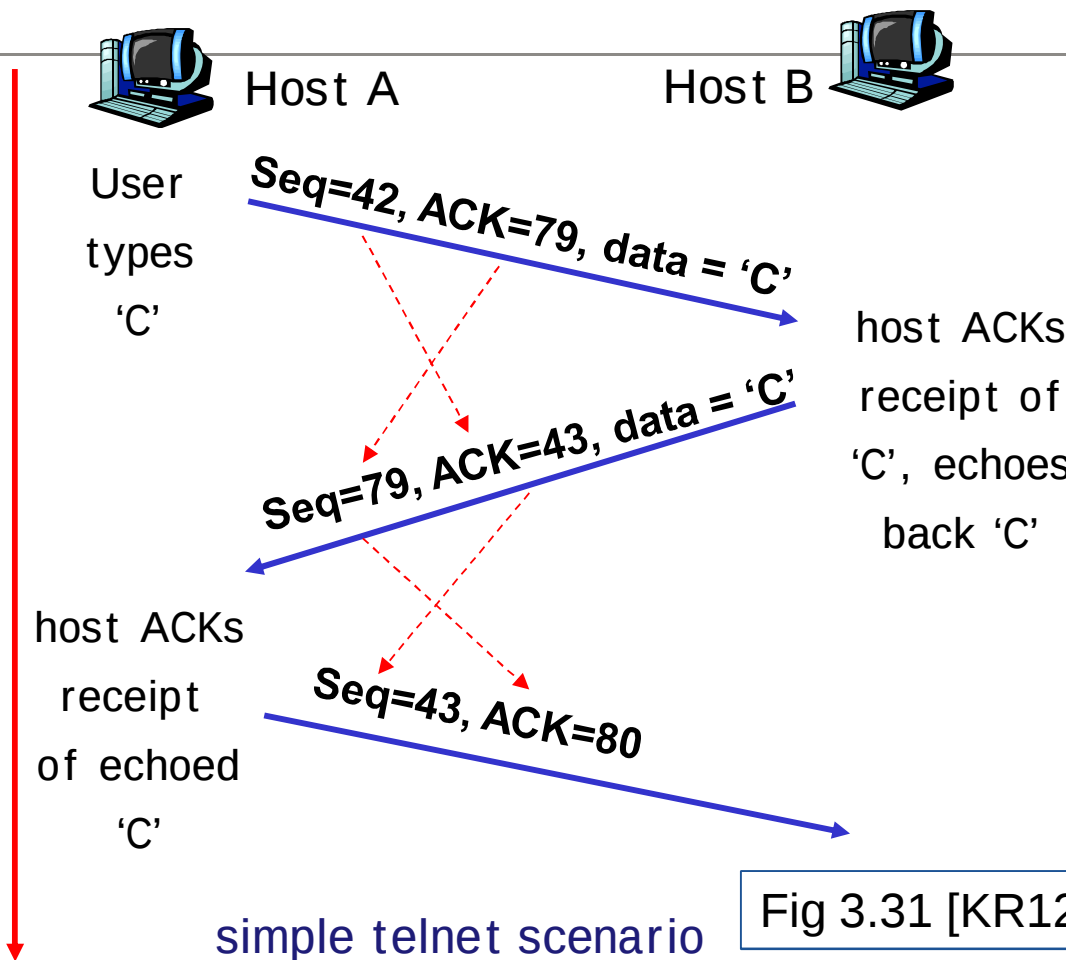


Fig 3.31 [KR12]

Oletus: ruuhkanhallinta tai vuonvalvonta ei rajoita, data jo valmiiksi sopivina paloina ($\leq$ MSS) eikä toinen osapuoli lähetä dataa.

NextSeqNum = InitialSeqNum; SendBase = InitialSeqNum

```
loop (forever) {  
  switch(event)
```

Vuonvalvonta ja/tai
ruuhkanhallinta voi
estää lähettämisen!

event: data received from application above

create TCP segment with sequence number NextSeqNum

if (timer currently not running) start timer

pass segment to IP

NextSeqNum = NextSeqNum + length(data)

Riittääkö 1
segmentti?

event: timer timeout

retransmit not-yet-acknowledged segment with
smallest sequence number

start timer

Ajastimen arvo?

Yksi vai
monta?

event: ACK received, with ACK field value of y

if (y > SendBase) {

SendBase = y

if (there are currently not-yet-acknowledged segments)
start timer

}

Toistokuittaukset?

```
} /* end of loop forever */
```

TCP: lähetys (simplified)

Comment:

- SendBase-1:
viimeisin kuitattu
tavu

Example:

- SendBase-1 = 71;
y = 73, vast.ottaja
odottaa tavuja 73+ ;
y > SendBase, joten
kuittaa uusia tavuja

Fig 3.33 [KR12]



TCP: uudelleenlähetyksiä

lost ACK scenario

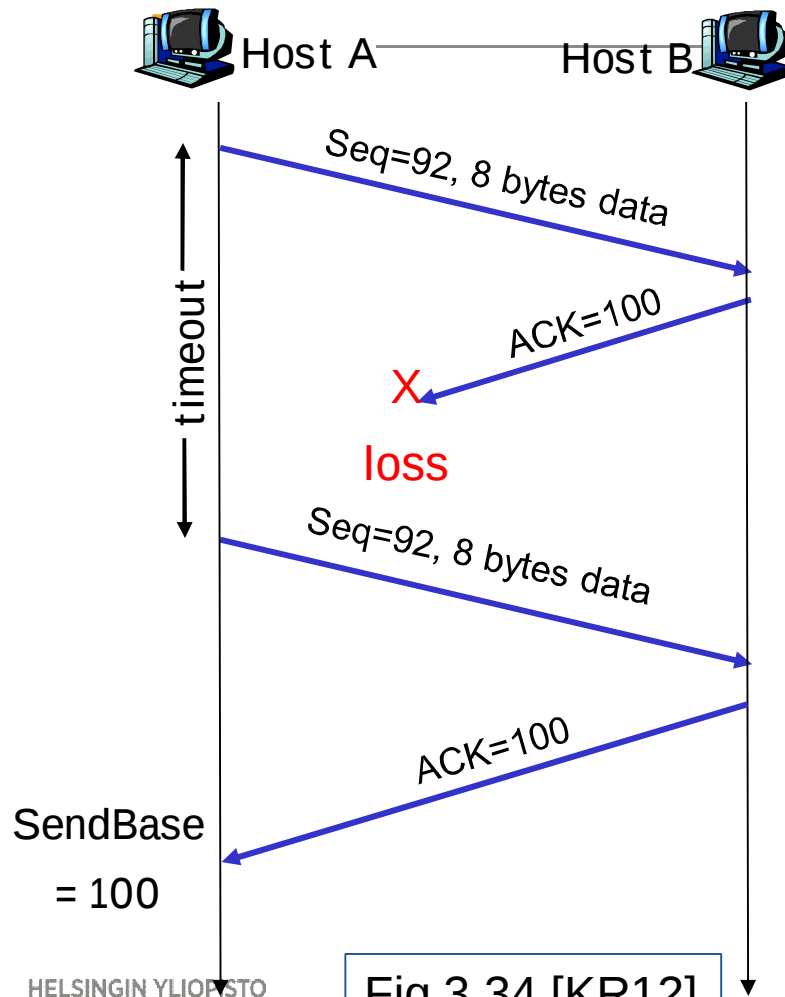


Fig 3.34 [KR12]

premature timeout

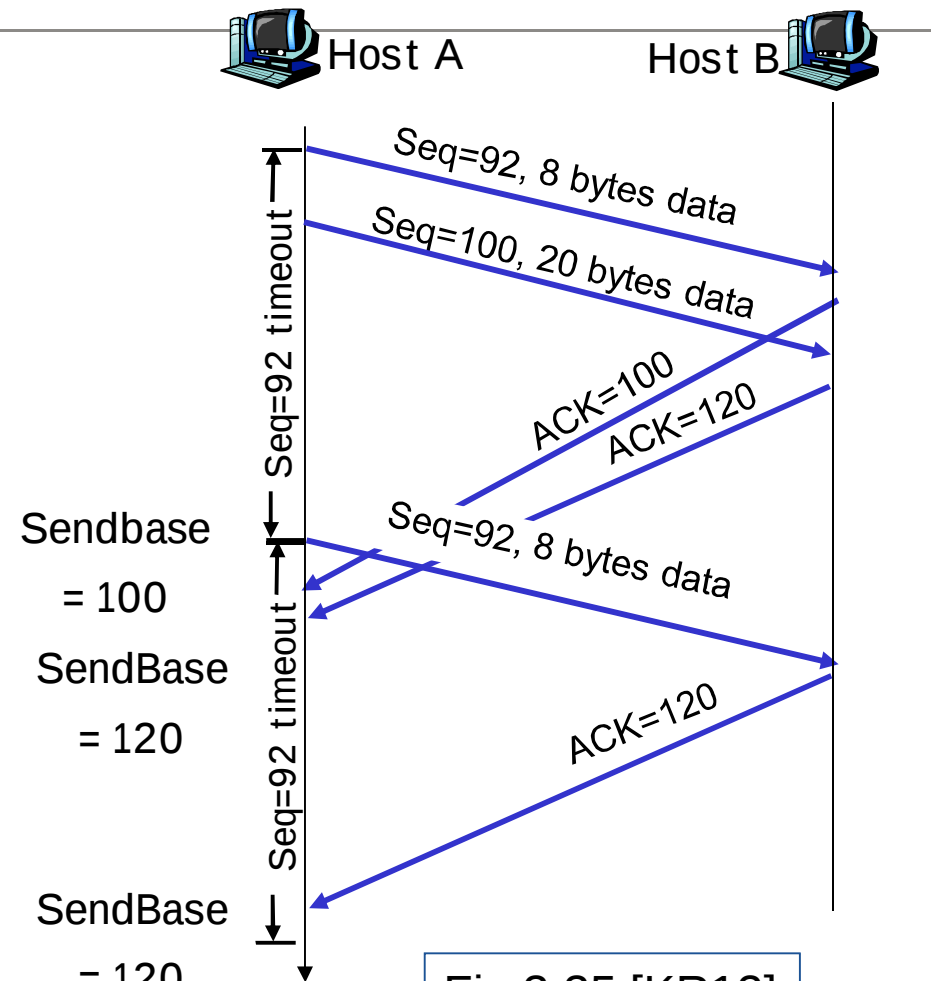


Fig 3.35 [KR12]



TCP: kumulatiivinen ack

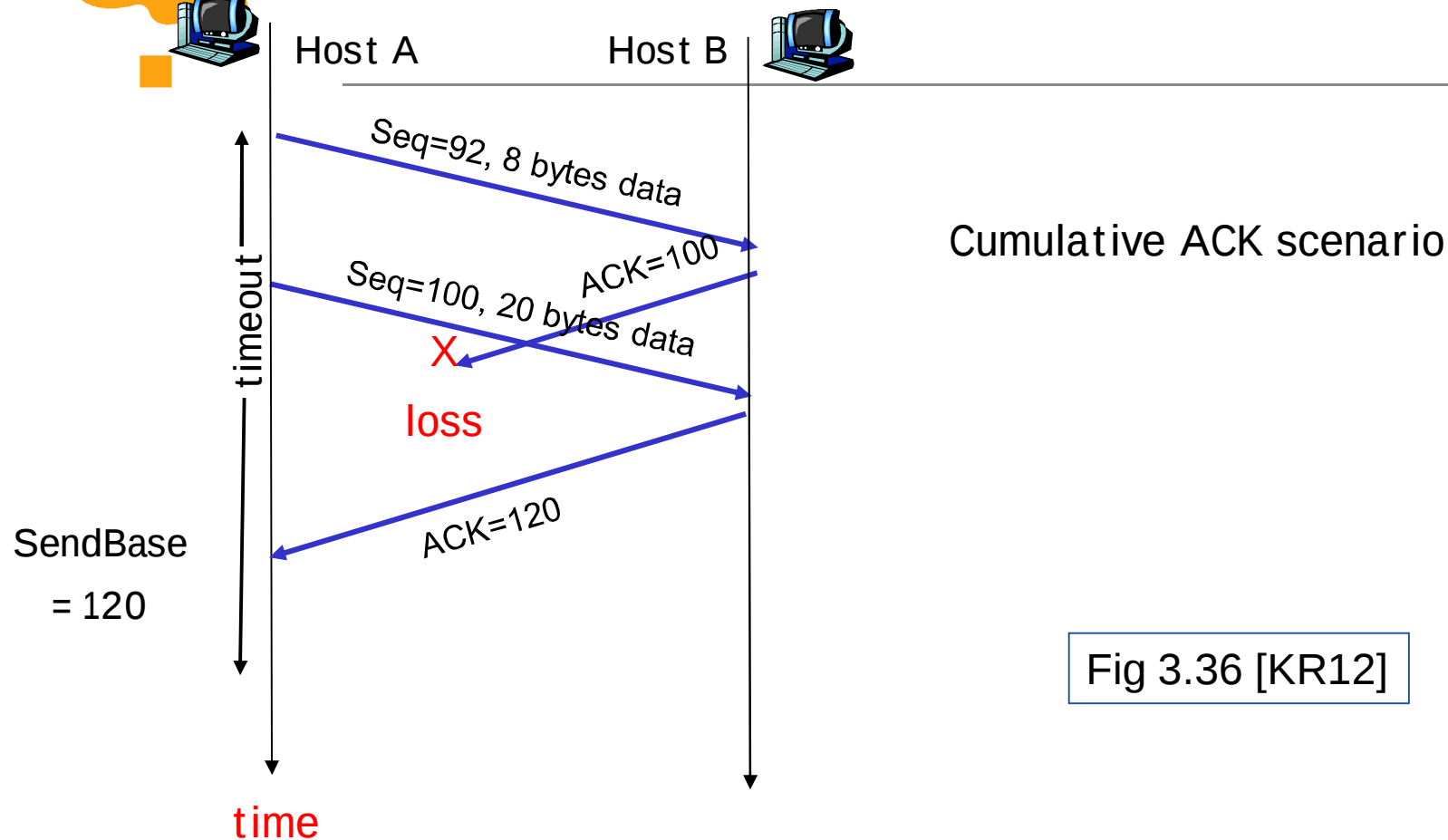


Fig 3.36 [KR12]



TCP ACK generation [RFC 1122, RFC 2581]

Table 3.2 [KR12]

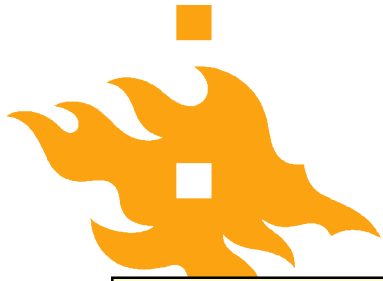
Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments Joka toinen kuitattava!
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	Immediately send <i>duplicate ACK</i> , indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap

TCP:n kuitattava vähintään joka toinen segmentti ja kuittausta saa viivyttaa korkeintaan 500 ms.



Nopea uudelleenlähetys (fast retransmit)

- Timeout suhteellisen pitkä
 - => aika iso viive ennen uudelleenlähetystä
- Vastaanottaja ilmoittaa puuttuvasta segmentistä toistokuittauksilla (duplicate ACK)
 - Liukuhihnoituksen vuoksi useita segmenttejä voi olla kuittaamatta
 - Jos välistä puuttuu segmentti, seurauksena on useita ACK-kuittauksia
- Jos lähettäjä saa 3 samaa segmenttiä kuittaavaa **toistokuittauksia**, se olettaa, että seuraava segmentti on kadonnut (siis kaikkiaan 4 samaa)
 - Ja lähettää puuttuvan segmentin heti
- **Nopea uudelleenlähetys** = **lähetä uudelleen** jo ennen kuin ajastin laukeaa eli **kolmen tuplakuittauksen jälkeen**.



Nopea uudelleenlähetys

Sender

```
event: ACK received, with ACK field value of y
  if (y > SendBase) { /* uuden segmentin kuittaus */
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
      start timer
  }
  else { /* toistokuittaus */
    increment count of dup ACKs received for y
    if (count of dup ACKs received for y = 3)
      resend segment with sequence number y
  }
```

a duplicate ACK for
already ACKed segment

Nopea uudelleenlähetys
(fast retransmit)



Paluu N:ään vai valikoiva toisto?

Kumpaa TCP käyttää?

- Liukuvan ikkunan protokolla
- Hybridi, tavallaan 'best-of-both'
- Go-Back-N-tyyppinen
 - Virheellisiä tai väärässä järjestyksessä tulleita ei hyväksytä, mutta ne yleensä talletetaan puskuriin
 - Kumulatiivinen ACK
 - Kaikkia virheellisestä lähtien ei tarvitse lähettää uudestaan
- Valikoiva toisto
 - Kumulatiivinen ACK ei kelpaa
 - SACK-kuittaus (selective ACK), joka kertoo, mitkä segmentit vastaanotettu (RFC 2018)



Vuonvalvonta

Jotta lähettäjä ei tukahduta vastaanottoa

Siirtonopeus sovitettava vastaanottavan sovelluksen mukaan

Kuittaus on irroitettu vuonvalvonnasta

Liukuva ikkuna, koko vaihtelee

Kuittaus siirtää ikkunaa

Vuonvalvonta määrää ikkunankoon

Kun ikkunankoko = 0, ei saa lähettää!

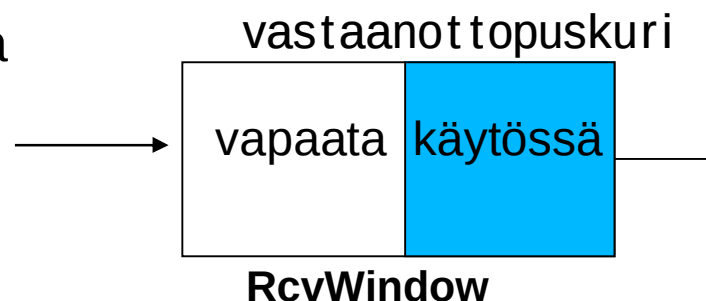
Vastaanottaja kertoo, montako tavua puskkureihin vielä mahtuu

TCP-segmentin otsakkeen **kenttä Receive window**

Sovellus lukee tavut silloin kun haluaa

Koko on mukana jokaisessa TCP-segmentissä (molempiin suuntiin)

Myös ruuhkanhallinta rajoittaa lähettämistä





Vuonvalvonta

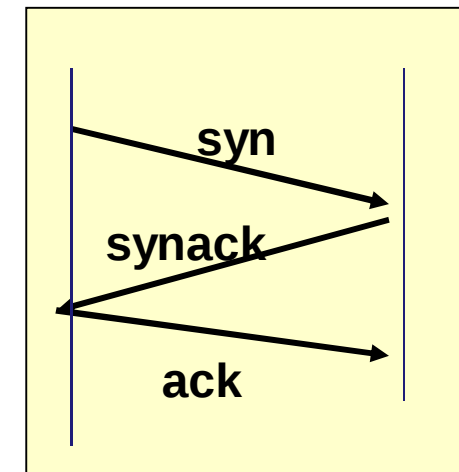
Kun ikkunankoko ==0, milloin voi taas lähettää?

- Jatka **lähettämällä tavun kokoisia segmenttejä**
 - Kysely
- Kuittaus antaa ajantasalla olevan tiedon vastaanottajan puskuritilasta
 - Edellisen ACK:n toistokuittaus => ei tilaa
 - Normaali ACK, kun vapaata vähintään täydelle TCP-segmentille
- Miksi lähettäjä ei vain odota, että vastaanottaja kertoo, kun tilaa jälleen tulee?
 - Entä, jos tämä kuittaus katoaa!
 - Lähettäjä odottaa turhaan ja vastaanottaja luulee, ettei ole lähetettävää => lukkiutuminen!



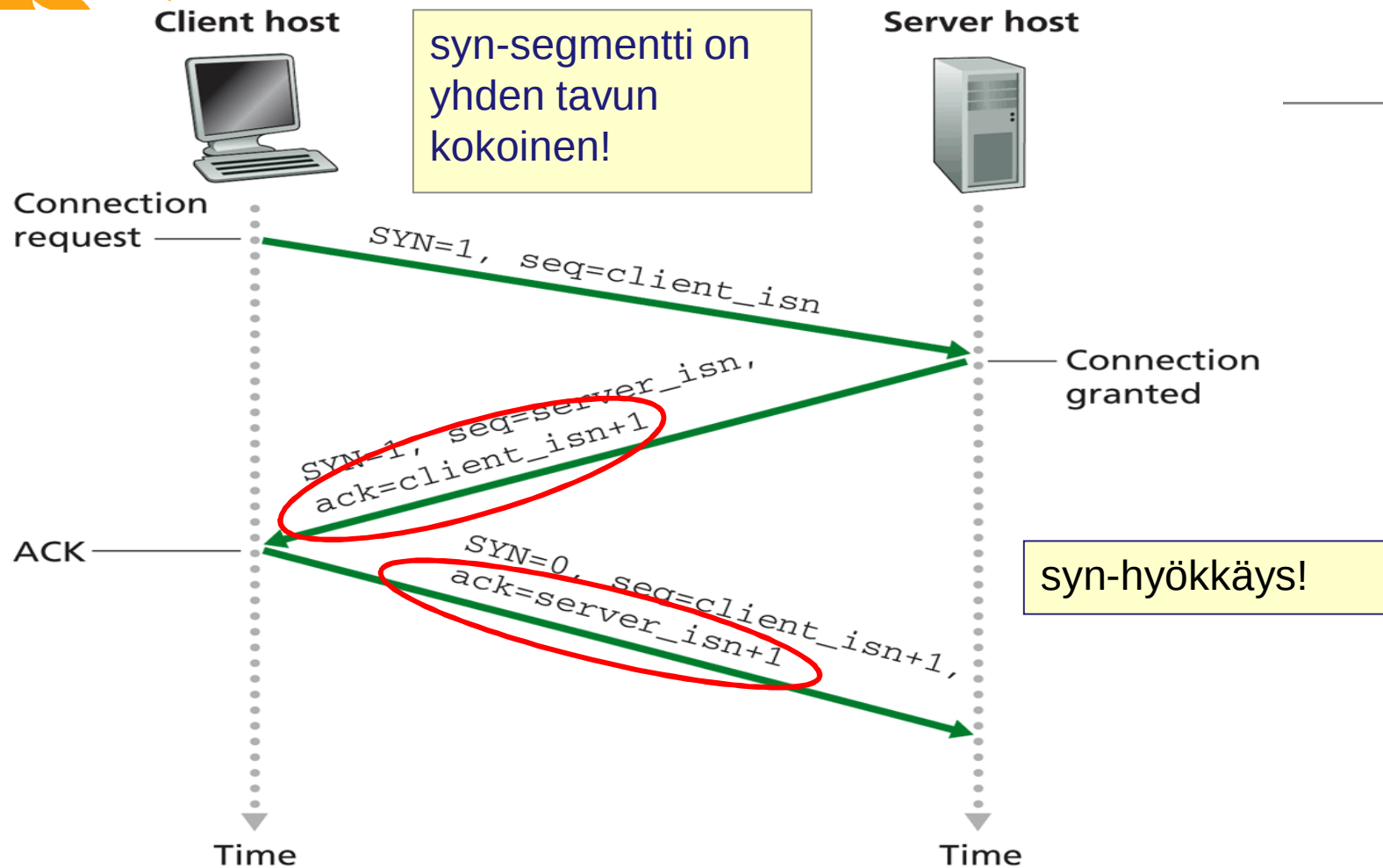
Yhteyden muodostus

- **Kolmivaiheinen kättely (three-way handshake)**
 - 3 segmenttiä: SYN – SYNACK – ACK
 - Otsakkeen bittikentät
 - Viimeinen voi sisältää dataa (piggybacked)
 - Jos porttiin ei liity prosessia, vastaukseksi RST-segmentti eli yhteyttä ei voida muodostaa
- **Varaa puskuritilaa**
 - Lähettäjä puskuroid uudelleenlähetystä varten
 - Vastaanottaja saatujen pakettien järjestämistä varten
- **Sovitetaan tavunumeroinnin alkuarvoista**
 - Kuittauksia varten
- **Ilmoita oma vastaanottoikkunan koko**
 - Vuonvalvontaa varten



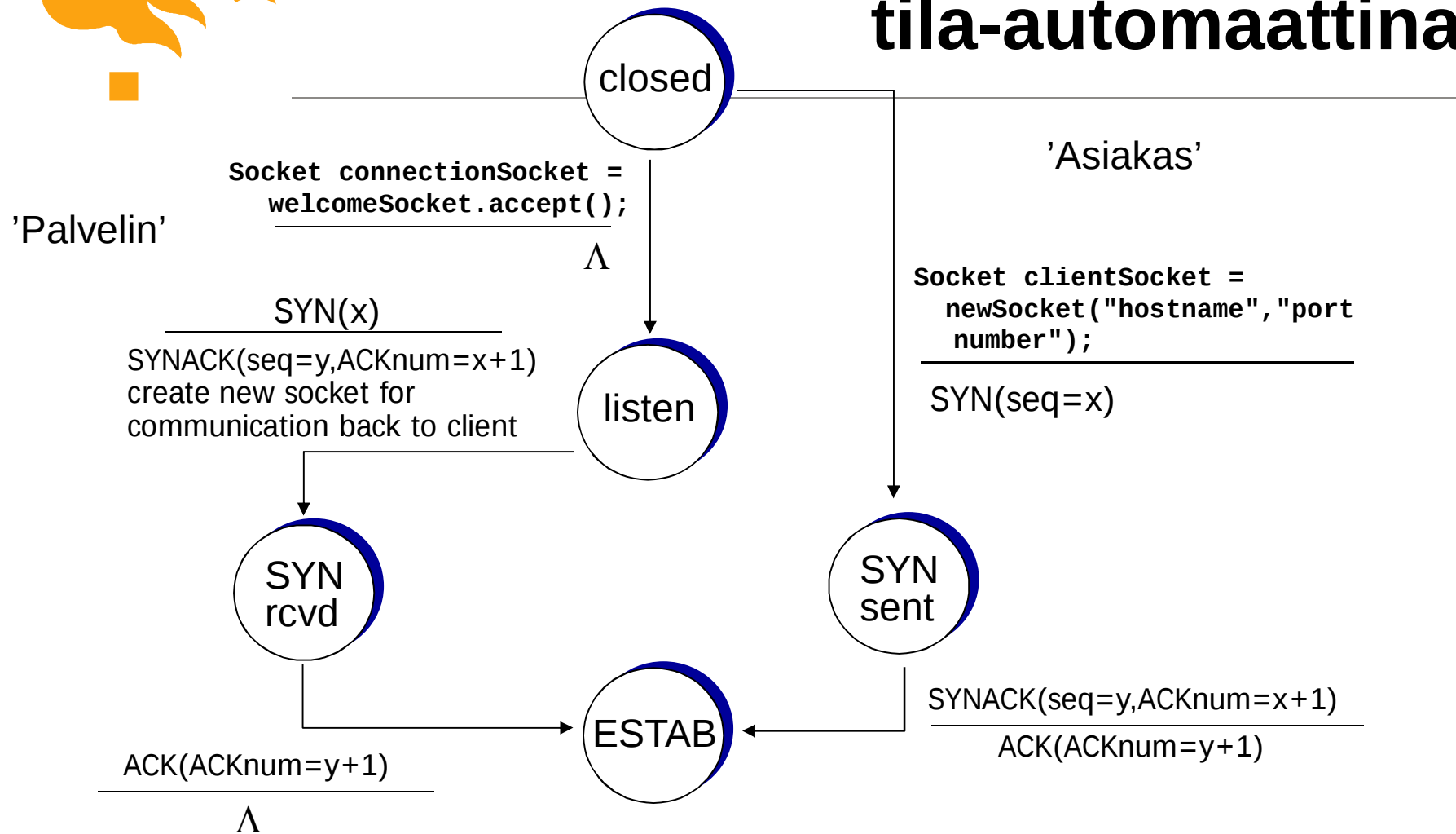


Yhteyden muodostus





Kolmivaiheinen kättely tila-automaattina





Yhteyden purku

Molemmat suunnat puretaan erikseen

4 segmenttiä: FIN – ACK, FIN – ACK

Yhteys on kokonaan purettu, kun molemmat suunnat purettu

Purku käyttää ajastimia erikoistilanteiden hallintaan

Noin 2 * paketin maksimaalinen elinikä

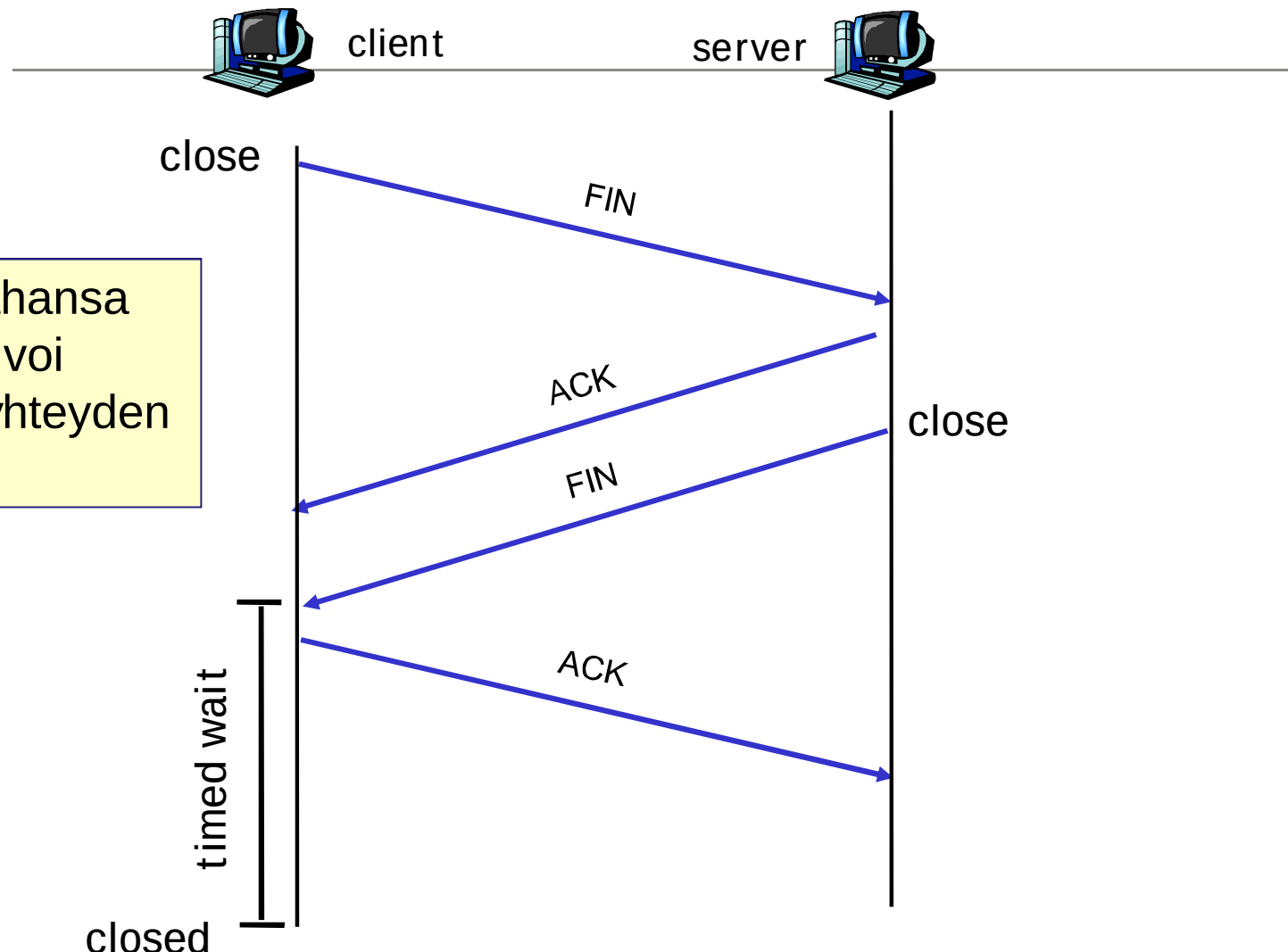
Tyypillisesti 30, 60 tai 120 s



Yhteyden purku

Fig 3.40 [KR12]

Kumpi tahansa osapuoli voi aloittaa yhteyden purun.





Kuljetuskerros

Ruuhkanhallinta TCP:ssä



Ruuhkanhallinta: Miksi?

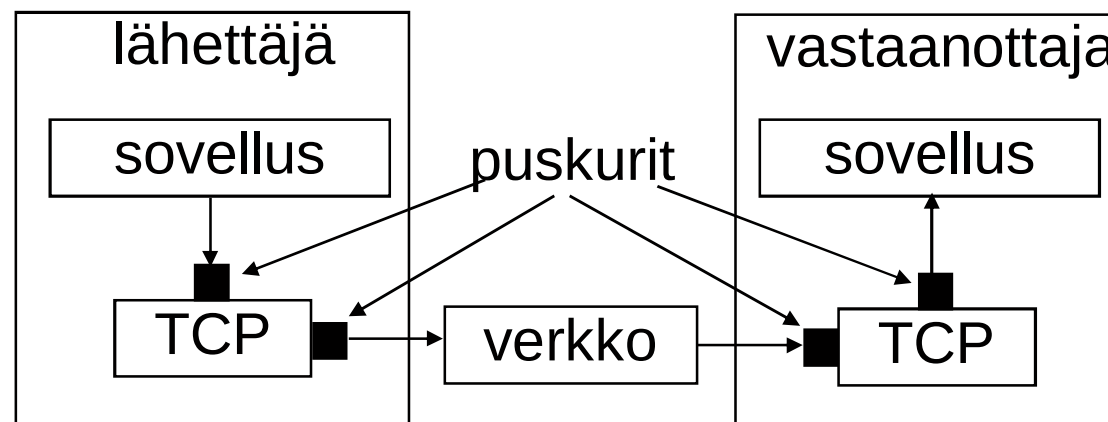
Verkon hetkellinen jäljellä oleva vapaa kaista vaihtelee

Voi olla vähemmän kuin lähettävän ja vastaanottavan sovelluksen kapasiteetti -> pelkkä vuonhallinta ei riitä

Monta lähettäjää jakaa samoja verkkoresursseja

Verkko voi siis olla pullonkaula

Ruuhkanhallinta varmistaa että verkkoa ei ylikuormiteta





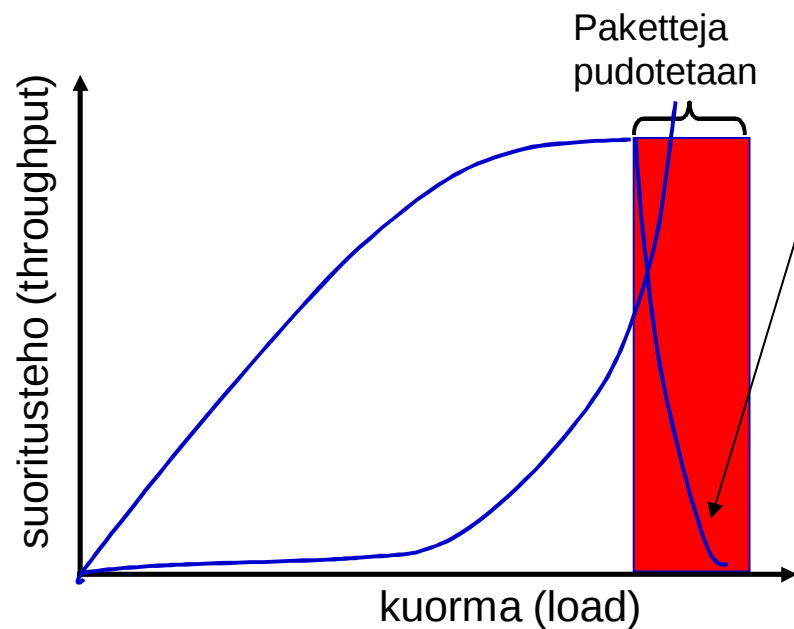
Ruuhkanhallinta: Miksi?

Verkkoelementeissä (reitittimet) on puskurit

FIFO+drop tail

Puskurin täyttyessä paketit joutuvat jonottamaan -> viive kasvaa

Puskurien ollessa täynnä uudet paketit pudotetaan



“Congestion collapse”:

- Pudotettujen pakettien uudelleenlähetys
 - Lisää kuormaa
- Uudelleenlähetetään tarpeettomasti vielä matkalla olevia paketteja
 - Viive kasvaa nopeasti -> järj. ei pysy perässä
 - Viive kasvaa yli maksimitoipumisajan (RTO)
 - Lisää edelleen kuormaa!
 - Vrt. tulen sammuttaminen bensalla...
- Reitittimet tekevät enenevästi turhaa työtä
 - Esim. paketin pudottaa loppusuoralla oleva reititin



Miten ratkaista ongelma?

- Lähettäjän on hidastettava vauhtia
- Verkkoavusteinen ruuhkanvalvonta (network assisted congestion control)
 - Reitittimet antavat tietoa ruuhkasta
 - Lisäbitit kertomassa ruuhkasta tai kenttä, joka ilmoittaa yhteydelle sallitun lähetysnopeuden (explicit rate)
- Päästä-päähän ruuhkanvalvonta (end-to-end congestion control)
 - Reitittimet eivät kerro ruuhkaantumisestaan isäntäkoneille
 - Isäntäkoneet huomaavat itse ruuhkan lisääntyneestä pakettien katoamisesta ja uudelleenlähetyksistä
 - TCP käyttää tätä
- Tällä kurssilla käsitellään **vain TCP:n ruuhkanhallintaa**
 - Lähinnä **TCP Reno -algoritmi**
 - Ruuhkanhallintaan kehitetty runsaasti erilaisia algoritmeja



Ruuhkaikkuna (congestion window)

- Internetin hetkellinen kuormitus vaihtelee
- Ruuhkaikkuna
 - Paljonko lähettäjä saa tietyllä hetkellä kuormittaa verkkoa
 - Paljonko lähettäjällä saa olla kuittaamattomia segmenttejä
- Lähettäjän pääteltävä itse sopiva ikkunankoko
 - Jos uudelleenlähetysajastin laukeaa, on ruuhkaa
 - Jos kuittaukset tulevat tasaisesti, ei ole ruuhkaa
- Dynaaminen ruuhkaikkunan koko
 - Kasvata ikkunaa ensin nopeasti, kunnes törmätään ruuhkaan
 - Pienennä sitten ikkunaa reilusti ja kasvata varovasti
- Lähetysikkunan raja voi tulla vastaan ensin
 - Kuittamatta saa olla $\min(\text{lähetysikkuna}, \text{ruuhkaikkuna})$



TCP Reno: Hidas aloitus (slow start) ja ruuhkanvälttely (congestion avoidance)



Aluksi ruuhkaikkuna = yksi segmentti

Alussa hidas siirtonopeus = MSS/RTT

Kukin kuittaus kasvattaa

ruuhkaikkunan kokoa yhdellä

Ekspontiaalinen kasvu

Ikkuna kaksinkertaistuu yhden RTT:n aikana

Sen jälkeen (=kynnysarvo saavutettu)

kasvata ikkunankokoa vain yksi

segmentti/RTT

Lineaarinen kasvu (Additive increase)

Ruuhkan välttely (congestion avoidance)

Siirtonopeus = $CongnWin / RTT$ tavua/sek

RTT- kiertoviive (Round Trip Time)

hidas
aloitus

ruuhkan-
välttely

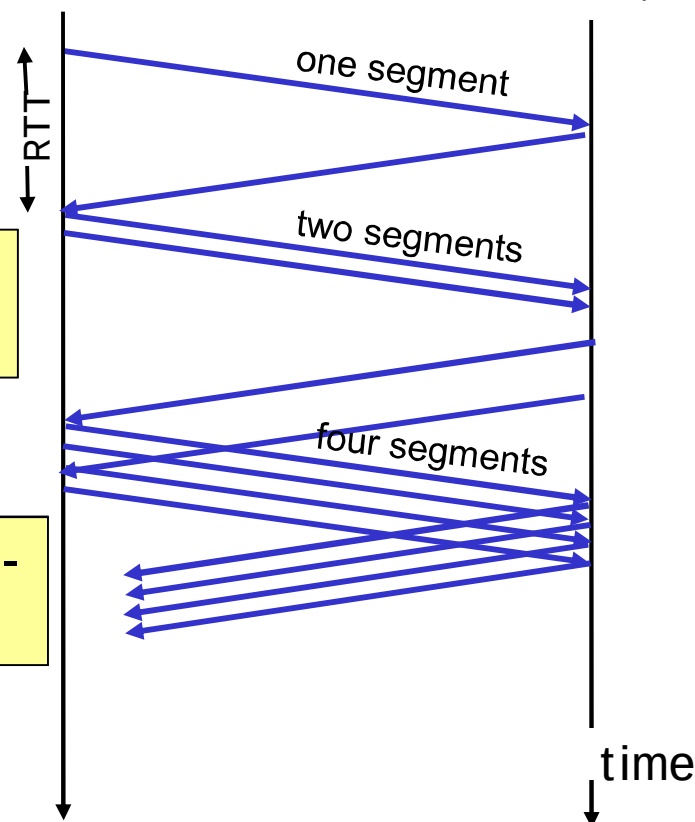


Fig 3.51 [KR12]



Kynnysarvo (threshold)

= ~ **varoitus**: tästä lähtien syytä varoa ruuhkaa

Aluksi kynnysarvo (threshold) on 64 KB

Ajastimen laukeamisen (timeout) jälkeen

Kynnysarvo = ruuhkaikkunan koko (CongnWin) / 2

Myös Renon toipuesssa kolmen tuplakuittauksen jälkeen

Kynnysarvoon saakka ikkunan kasvatus eksponentiaalista

Yhtä kuitattua segmenttiä kohden saa lähettää kaksi uutta

Eli kaksinkertaistuu yhden RTT:n aikana

Sen jälkeen ikkuna kasvaa lineaarisesti

Kasvaa yhdellä yhden RTT:n aikana

ruuhkanvälttely

Miksi näin?



TCP Reno: Reagointi kadonneisiin viesteihin

Saatu 3 ACK-kaksoiskuittausta (double ACK) (4 samaa kuittausta!)

Vanha TCP
Tahoe Ei
osaa!

Verkko pystyy välittämään dataa!

Ei siis (pahaa) ruuhkaa, ehkä paketissa bittivirhe tai paketti kadonnut jostain muusta syystä

Nopea uudelleenlähetys (fast retransmit)

Nopea toipuminen (fast recovery)

kynnysarvo?

Puolita ruuhkaikkunan arvo ja kasvata sitten lineaarisesti

Vanha TCP
Tahoe
pudotti aina
kokoon 1.

Ajastin laukeaa (timeout) (AINA: uusi hidas aloitus)

Verkko pahasti ruuhkautunut!

Pudota ruuhkaikkunan arvoksi 1

hidas aloitus

Kasvata eksponentiaalisesti kynnysarvoon asti

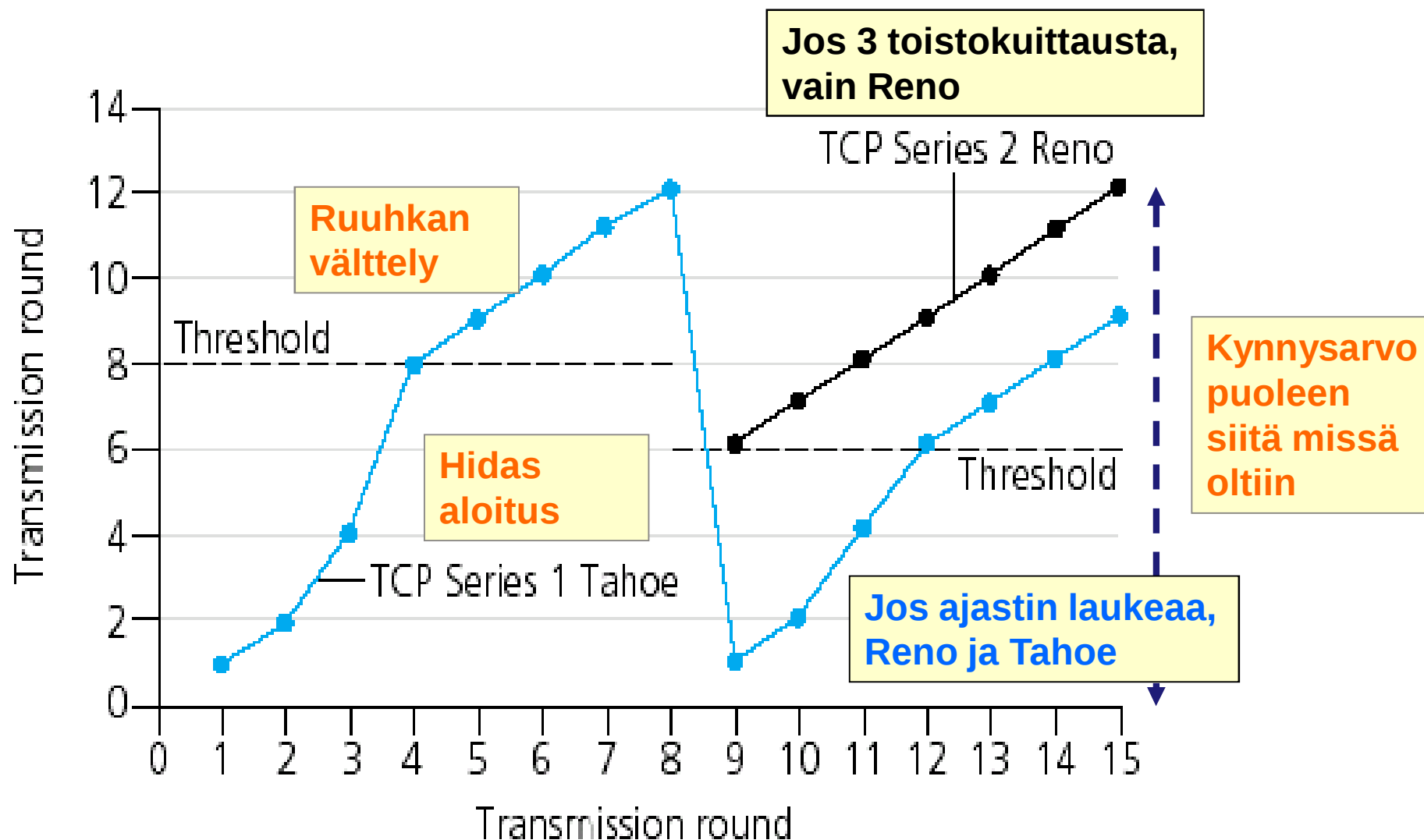
Kasvata sitten lineaarisesti

ruuhkanvälttely



TCP Reno ja Tahoe

Fig 3.53 [KR12]





Ajastimen arvo?

- Aseta ajastin, kun segmentti on lähetetty
- Liian lyhyt aika
 - Ennenaikainen timeout, turha uudelleenlähetys
 - Turhat ruuhkatoiminnot
- Liian pitkä aika
 - Turhan hidas reagointi segmentin katoamiseen
 - Ei huomata ruuhkaa ajoissa
- Alkujaan: $Timeoutinterval = 2 * RTT$
- Kuittausaika vaihtelee suuresti ja nopeasti => käytössä dynaaminen arvo
 - Saadaan jatkuvien mittausten perusteella
- Jos ajastin laukeaa, kaksinkertaista aikaraja
 - Exponential backoff



Ajastimen arvo?

$$\text{Timeoutinterval} = \text{EstimatedRTT} + 4 \text{ DevRTT}$$

Arvioi kiertoviive eli EstimatedRTT

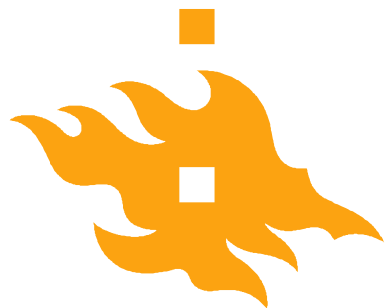
Mittaa jokaisen lähetetyn segmentin kiertoviive
tai noin RTT:n välein normaalien lähetysten aikana (SampleRTT).
Laske painotettu arvo, tyypillisesti $\alpha = 1/8 = 0.125$

$$\text{EstimatedRTT} = (1-\alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

Huomioi poikkeama

tyypillisesti $\beta = 0.25$

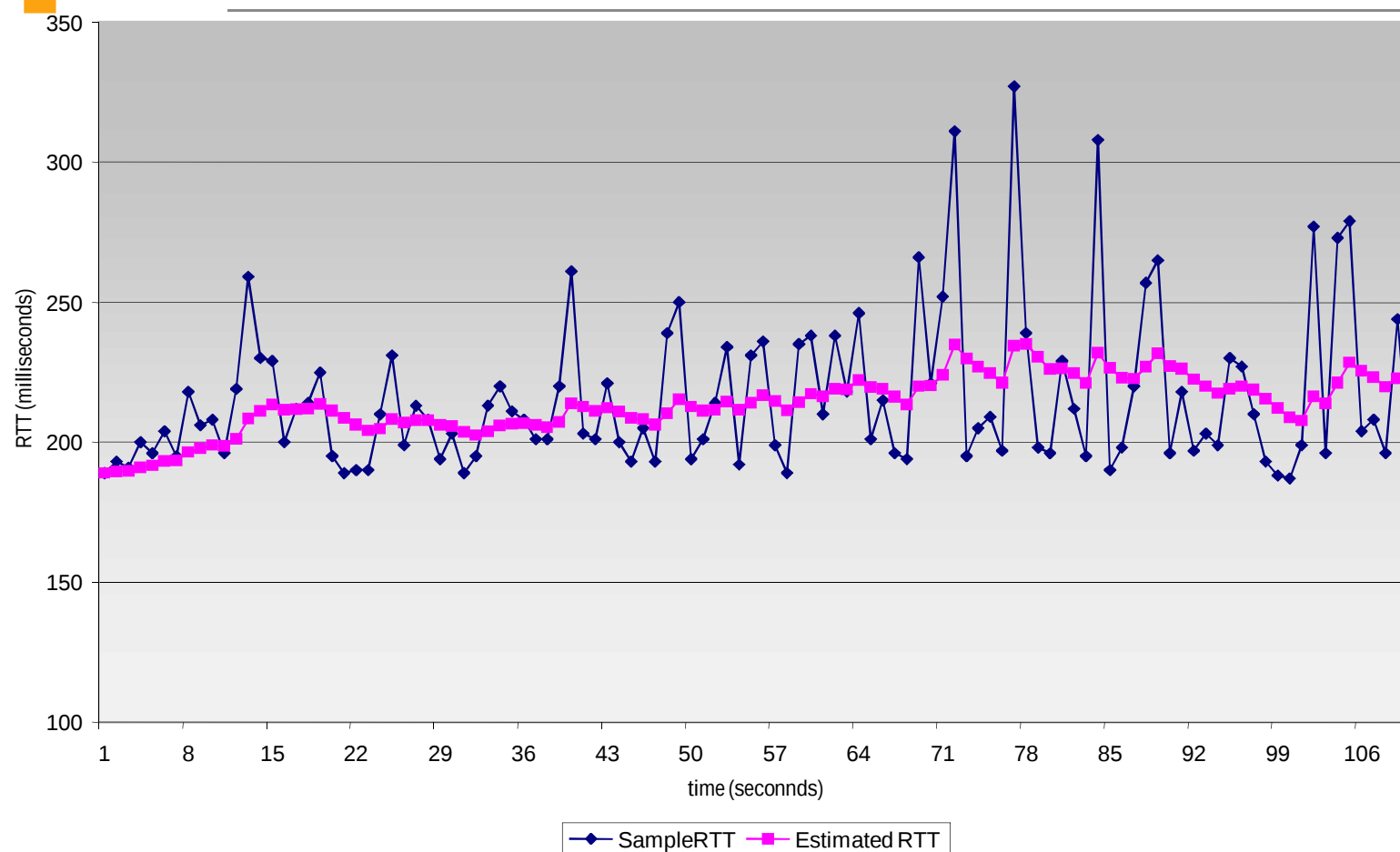
$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$



Esimerkki

Fig 3.32 [KR12]

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr





Ajastinajan kaksinkertaistaminen

- Aina kun saadaan lähetettäväksi sovellukselta dataa tai saadaan kuittaus jo lähetettyyn segmenttiin, otetaan käyttöön tuorein estimoitu ajastimen arvo.
- Ajastimen laukeaminen =>
 - Kun segmentti lähetetään uudelleen, kaksinkertaistetaan ajastimen arvo.
 - Jos sama segmentti joudutaan lähettämään uudestaan, niin ajastimen arvo kaksinkertaistetaan joka kerta.



Ajastinajan kaksinkertaistaminen

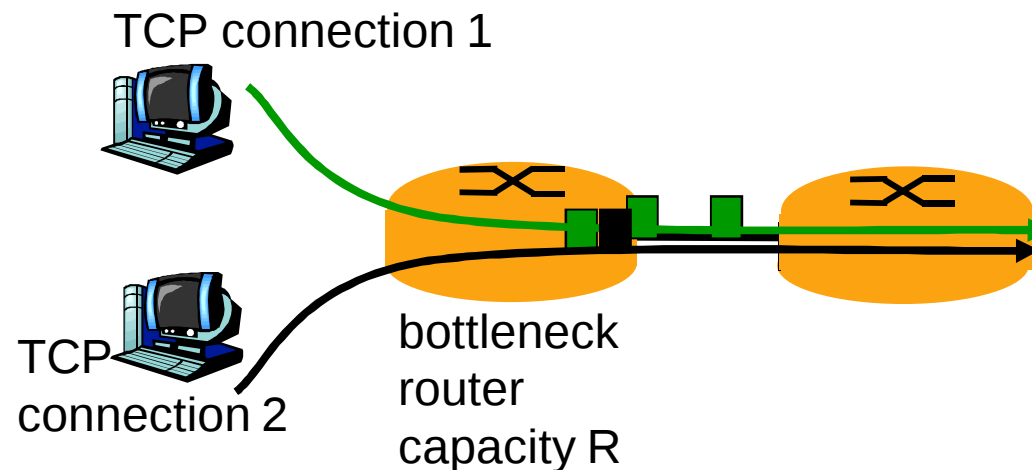
Esimerkki: Lähetetään segmentti 100 ja ajastimen arvo 2.5 sekuntia.

- Ajastin laukeaa ja lähetetään segmentti 100 **uudestaan** ja nyt **ajastimen arvo on 5** sekuntia.
- Kun kuittausta ei tule 5 sekunnin sisällä, niin ajastin taas laukea ja segmentti 100 **lähetetään vielä kerran**. Nyt **ajastimen arvoksi asetetaan 10** sekuntia.
- Tähän saadaan kuittaus ajoissa ja siirrytään lähettämään segmenttiä 1100. **Ajastimen arvoksi asetetaan tuorein estimoitu arvo 3.2** sekuntia.



Onko TCP reilu? (Fairness)

tavoite: jos K kpl TCP yhteyttä jakaa saman pullonkaulalinkin, jonka kapasiteetti on R , jokaisen tulisi saavuttaa keskimäärin R/K suoritusteho





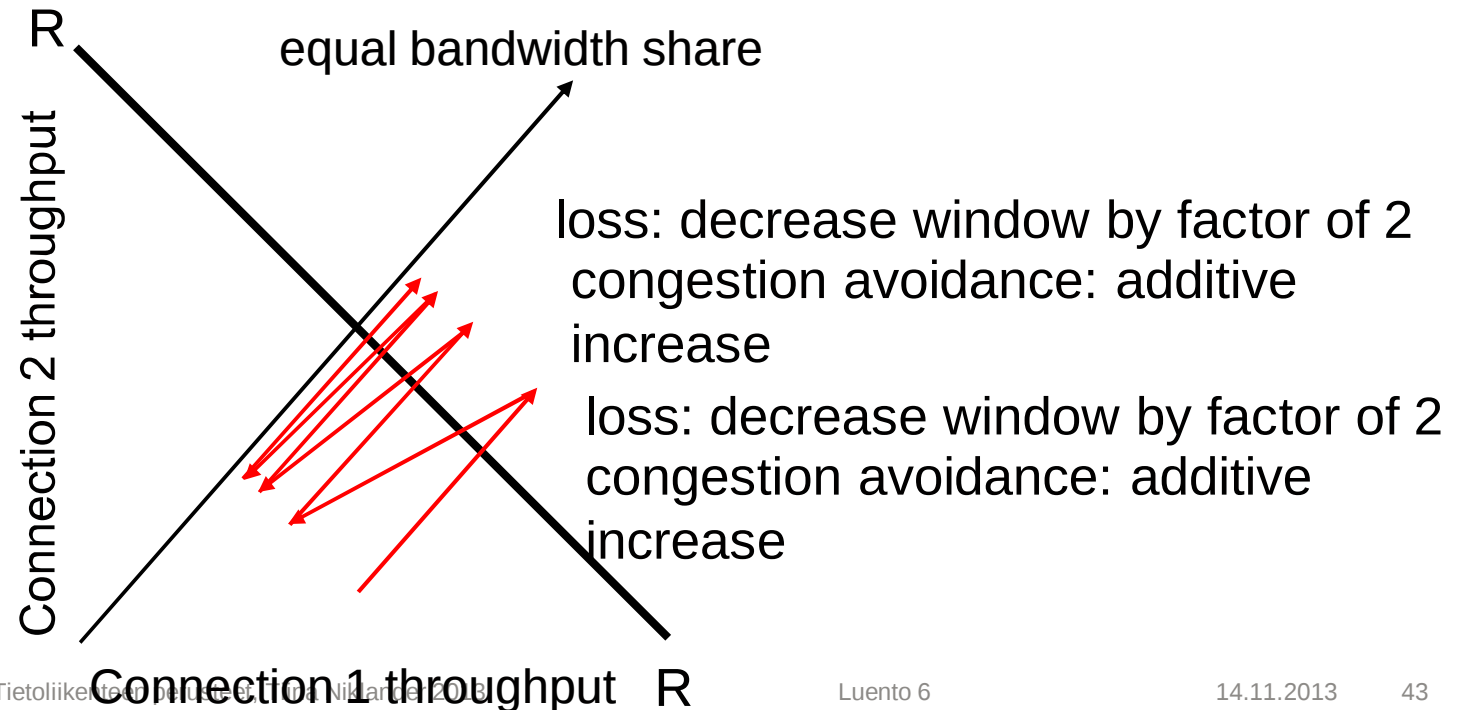
Onko TCP reilu?

Kaksi kilpailevaa yhteyttä:

Lineaarinen kasvu -> kulmakerroin 1 kun suoritusteho kasvaa

Multiplicative decrease -> suoritusteho pienenee suhteessa sen hetkiseen

Nopeammat yhteydet perääntyvät enemmän





Onko TCP reilu?

Kohdellaanko TCP-yhteyksiä reilusti?

- Jokainen reitittimessä kulkeva yhteys kärsii ruuhkasta
- Vain TCP-yhteydet kiltisti vähentävät lähetystään
 - AIMD: additive increase, multiplicative decrease
- Sovellus voi avata monta rinnakkaista yhteyttä
 - Onko tämä reilua muita kohtaan?
- UDP?
 - Ei ruuhkanhallintaa, ei välitä ruuhkasta eikä vähennä lähetystä
 - Multimediasovellukset käyttävät UDP:tä, työntävät dataa vakionopeudella ja sietävät katoamisia
- TCP-ystävällinen reititin?



Kuljetuskerros

Yhteydetön
kuljetuspalvelu
UDP



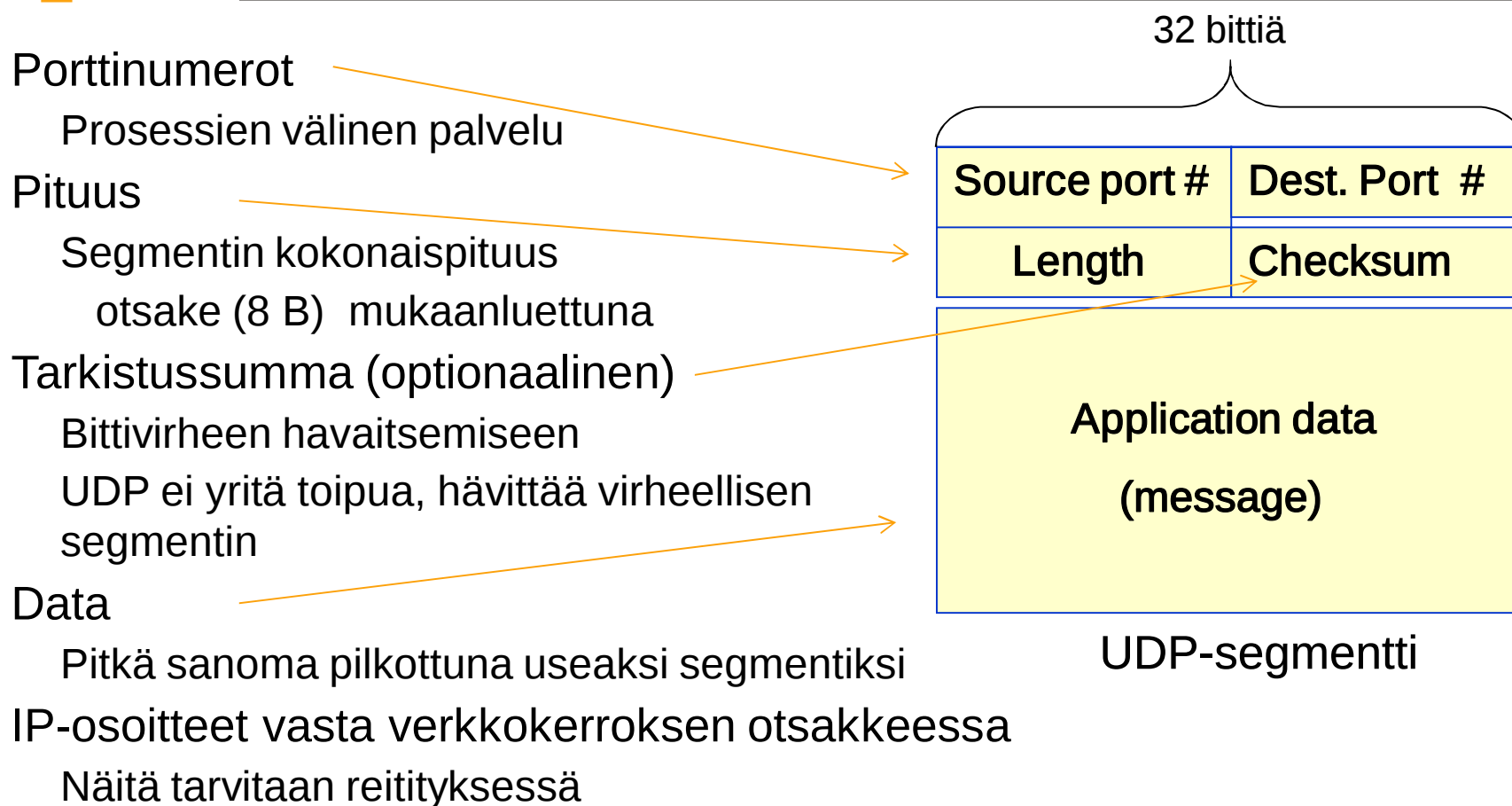
Miksi UDP?

- Pieni yleisrasite, koska pieni otsake
- Ei tilatietojen tallettamista eikä lähetys- ja vastaanottopuskureita
- Sovellus sietää virheitä
- Reaaliaikavaatimuksia: data lähtee heti verkkoon, koska ei yhteydenmuodostusta eikä vuon- tai ruuhkanvalvontaa
- Tarvittava luotettavuus on räätälöitävissä sovellukseen
 - Sovellusprotokolla: oma sanomanumerointi, oma uudelleenlähetys



UDP-segmentin rakenne

Fig 3.7 [KR12]





Kuljetuskerroksen pseudo-otsake

Käyttö vain isäntäkoneen sisäisesti. Ei lähetetä verkkoon.

UDP (ja TCP) laskee tarkistussumman otsakkeelle, datalle ja

ns. **pseudo-otsakkeelle**, joka sisältää IP-otsakkeen tietoja

Varmistus, että segmentti on tullut oikeaan koneeseen ja oikeaan porttiin

Source IP-address		
Destination IP-address		
00000000	Protocol	TCP/UDP segment size



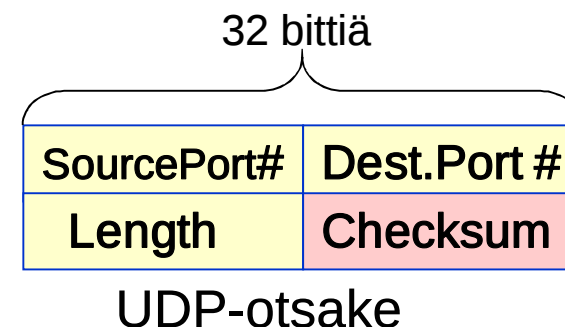
UDP&TCP-tarkistussumma

Lähetys

Summaa 16 bitin kokonaisuudet (otsake + pseudo-otsake mukana), ylivuotobitit lasketaan mukaan, talleta **yhden komplementtina**

Vastaanotto

Summaa 16 b kokonaisuudet (myös tarkistussumma). Jos tuloksena on 16 ykköstä, niin OK!

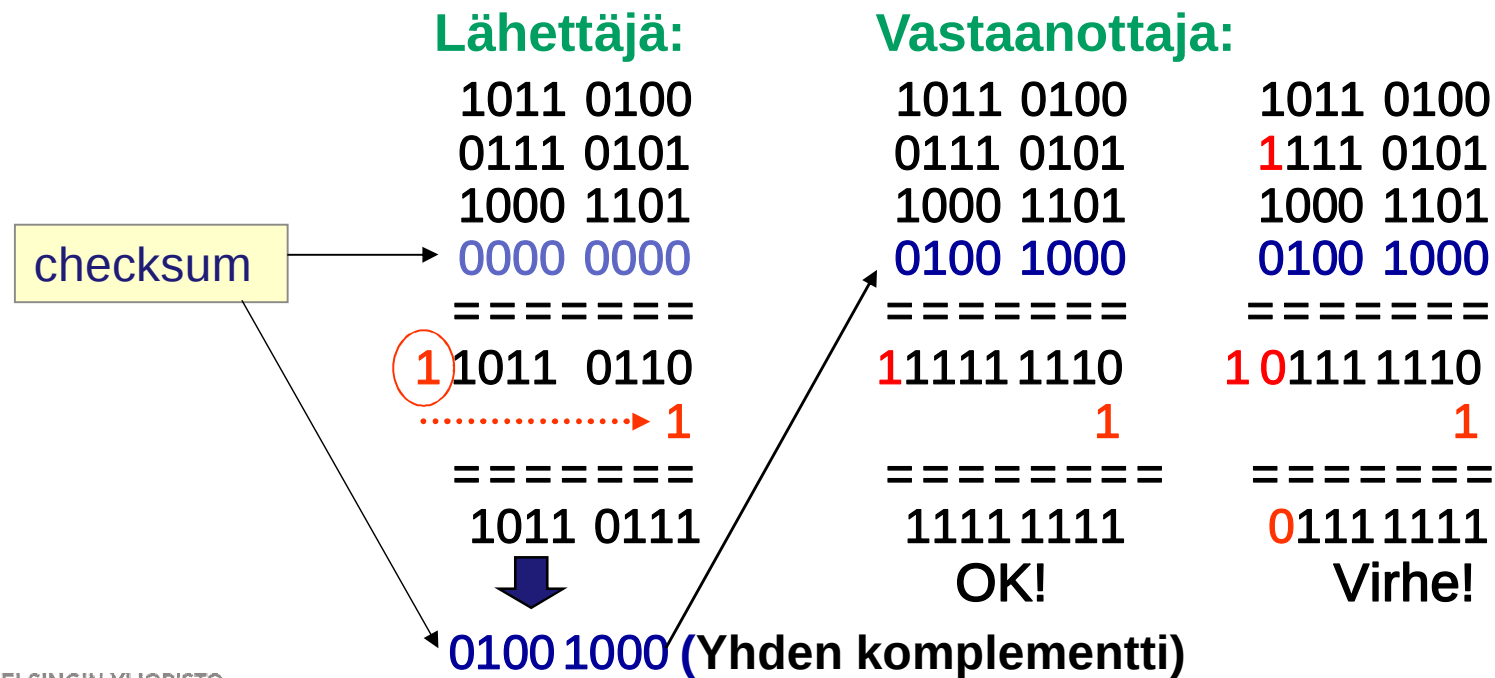




Esimerkki

0+0 = 0 no carry
1+0 = 1 no carry
0+1 = 1 no carry
1+1 = 0 with carry

Lasketaan tarkistussumma kolmen tavun mittaiselle sanomalle (tässä vain 8 bitin mittaisena):





Miksi UDP-tarkistussumma

- **Kaikki linkkikerrokset eivät suorita tarkistuksia!**
 - Ethernet huolehtii kyllä
 - Huom. IP checksum vain otsakkeelle (ei datalle!)
- UDP-tarkistussumma ei ole kovin tehokas havaitsemaan virheitä!
- UDP ei yritä toipua virheistä!
 - Jotkut toteutukset voivat tuhota virheellisen segmentin
 - Jotkut antavat sen sovellukselle varoituksen kera
- Lisärasite?
 - Ei tarvitse käyttää, jos ei halua. Tällöin lähettäjä laittaa tarkistussummaksi pelkkiä nollia.



Laskutehtävä: Lähetä 10 tavun viesti UDP:llä

Miten kauan kestää lähettäminen, jos lähetysnopeus on 56 kbps?

- $10 \text{ tavua} + 8 \text{ tavua} = 18 * 8 \text{ b} = 144 \text{ bittiä}$
- $144 \text{ b} / 56\,000 \text{ b/s} = 2.57 \text{ ms}$

Miten suuri on etenemisviive, jos etäisyys lähettäjältä vastaanottajalle on 1000 km?

- $1000 \text{ km} / 200\,000 \text{ km/s} = 5 \text{ ms}$ (huom! valonnopeus $> 200\,000 \text{ km/s}$)

Miten suuri on UDP-otsakkeen aiheuttama yleisrasite (overhead)?

- $8/18 = 0.44$ eli 44 %



Kertauskysymyksiä

TCP vs. UDP?

Miten tehdä kuljetuspalvelusta luotettava?

Keskeisimmät TCP:n otsakkeessa olevat tiedot?

Mitä tapahtuu TCP:n yhteydenmuodostuksessa?

Vuonvalvonta?

Ruuhkanhallinta?



ks. Kurssikirja s. 311-314