

3. Kuljetuskerros

3.1. Kuljetuspalvelu

- **'End- to- end'**
 - prosessilta prosessille looginen yhteys
 - portti
 - verkkokerros koneelta koneelle
 - IP-osoite
- **peittää verkkokerroksen puutteet**
 - jos verkkopalvelu ei ole riittävän hyvä, sitä voidaan parantaa kuljetuskerroksella
 - kuljetuskerros huomaa verkkokerroksen kadottamat paketit ja pyytää niiden uudelleenlähetystä

Sovelluksen vaatimuksia kuljetuspalvelulle:

- Virheetön, luotettava
- järjestyksen säilyttävä
- kaksoiskappaleet karsiva
- mielivaltaisen pitkiä sanomia salliva
- vuonvalvonnan mahdollistava

Verkkokerros kuitenkin voi

- kadottaa sanomia
- toimittaa sanomat epäjärjestyksessä
- viivyttää sanomia satunnaisen pitkän ajan
- luovuttaa useita kopioita samasta sanomasta
- rajoittaa sanomien kokoa

kuljetuspalvelut parantavat verkkopalveluja

**Sovelluksen näkemä palvelun laatu
(Quality of Service, QoS)**

kuljetuskerroksen palvelut

verkkokerroksen palvelut

kuljetuskerroksen palvelut

Verkkokerroksen palvelut

Internetin kuljetuskerros

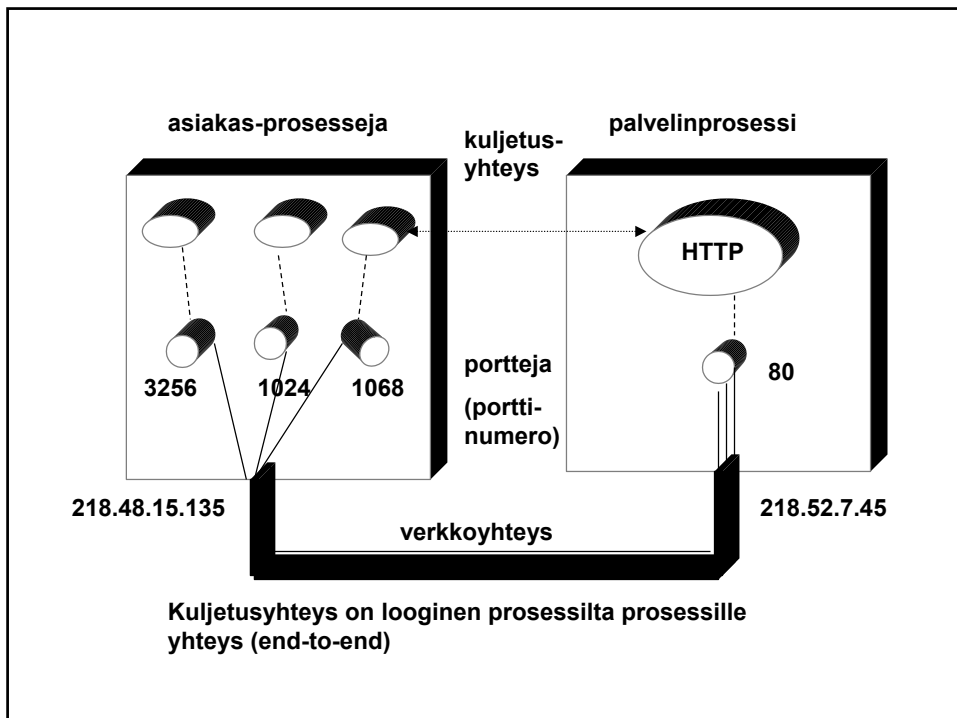
- **UDP (User Datagram Protocol)**
 - yhteydetön, epäluotettava palvelu
- **TCP (Transmission Control Protocol)**
 - yhteydellinen, luotettava palvelu
 - **virhevalvonta**
 - havaitsee ja korjaa siirrossa syntyneet virheet
 - **vuonvalvonta**
 - ei ylikuormita vastaanottajaa
 - **ruuhkanvalvonta**
 - huolehtii ettei verkko pääse ruuhkautumaan

3.2. Sovelluksien datavirtojen erottaminen

- **IP-osoite**
 - osoittaa koneen yksikäsitteisesti
- **Sovellusprosessi koneessa tunnustetaan porttinumerosta (16 bittiä =>0-65535)**
 - jokaisessa lähetetyssä segmentissä on
 - lähettäjän porttinumero
 - vastaanottajan porttinumero
- **Yleisillä palvelimilla omat varatut porttinumerot (0-1023)**
 - SMTP 25, HTTP 80, jne

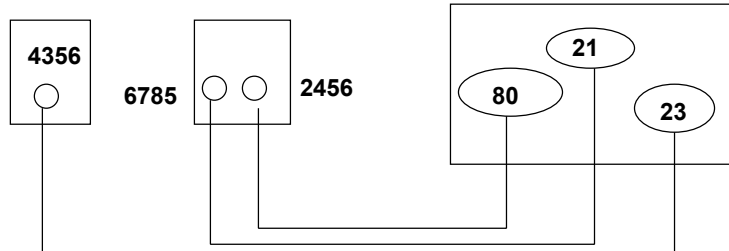
8/18/2003

5



Asiakkaalle kuljetuskerros usein
automaattisesti antaa käyttöön jonkin
vapaan porttinumeron yhteyden ajaksi

Palvelimilla kiinteät
numerot yhteydenottoa
varten (ohjelmallisesti
luodut pistokkeet)

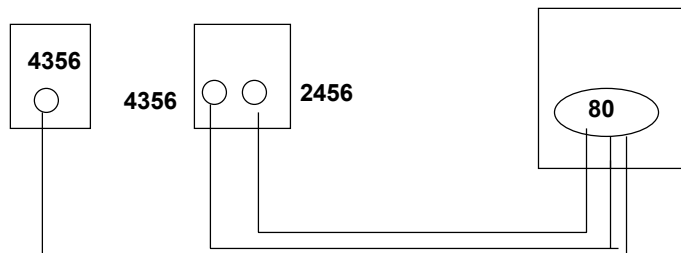


Kolme yhteyttä: 4356 \Leftrightarrow 23, 6785 \Leftrightarrow 21, 2456 \Leftrightarrow 80

8/18/2003

7

Eri koneissa voidaan ottaa sama numero!



Kolme yhteyttä: 4356 \Leftrightarrow 80, 4356 \Leftrightarrow 80, 2456 \Leftrightarrow 80!

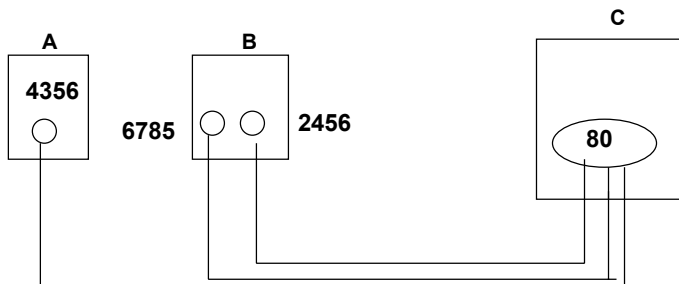
Kuljetusyhteydellä käytetään apuna myös IP-osoitetta:

=> koneilla eri IP-osoitteet, joten yhteydet pystytään
erottamaan

8/18/2003

8

Yhteydellisessä TCP:ssä tarvitaan sekä lähteen että kohteen pistokenumerot! (yhteydettömässä UDP:ssä riittää kohteen pistokenumero)



Kolme yhteyttä: A:4356 \Leftrightarrow C:80 ,B: 6785 \Leftrightarrow C:80 ja B:2456 \Leftrightarrow C:80, vaikka kohteena on sama C:80

TCP-yhteys

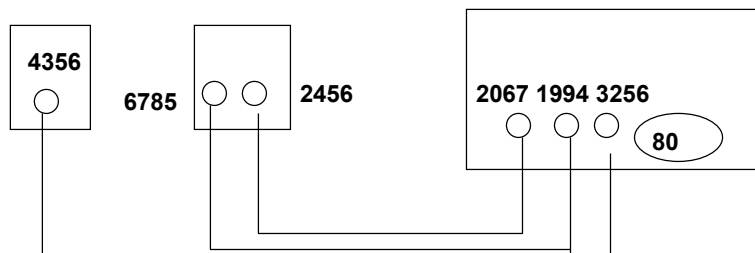
- **kaksisuuntainen** (full-duplex) **kaksipisteyhteys**
- **tunnistetaan päätepisteinä olevien pistokkeiden tunnuksista** (pistoke1, pistoke2)



- pistoke (socket)

- TCP-yhteyden päätepiste sovellukselle
 - lähettäjällä ja vastaanottajalla oma pistoke
- pistokenumero 48 bittiä
 - koneen 32 bitin IP-osoite
 - 16 bitin porttinumero

Palvelimessa yhteyksille uudet porttinumerot, jotta portti 80 voi ottaa vastaan uusia yhteyspyyntöjä



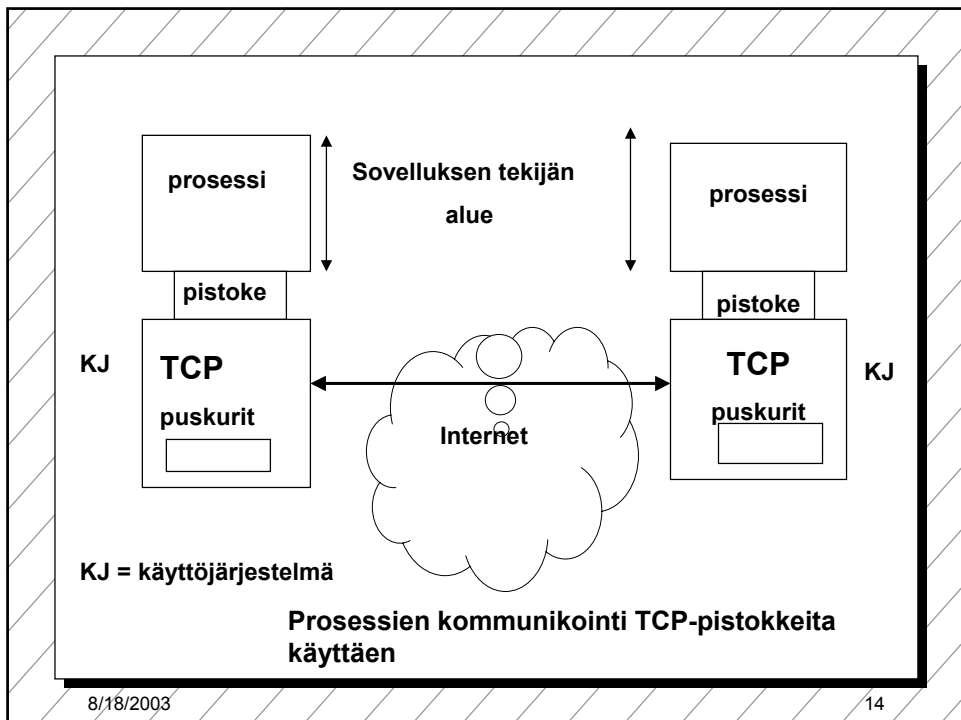
Kolme yhteyttä: 4356 \Leftrightarrow 3256, 6785 \Leftrightarrow 1994, 2456 \Leftrightarrow 2067

Pistokerajapinta (Socket interface)

- Verkkopalvelun ja sitä käyttävän sovelluksen rajapinta
 - yleensä käyttöjärjestelmän tarjoama palvelu
 - pistokerajapinta alunperin Berkeley Unixin mukana, nyt lähes kaikissa käyttöjärjestelmissä
 - miten verkkoprotokollan tarjoamiin palveluihin päästään käsiksi sovelluksesta

8/18/2003

13



8/18/2003

14

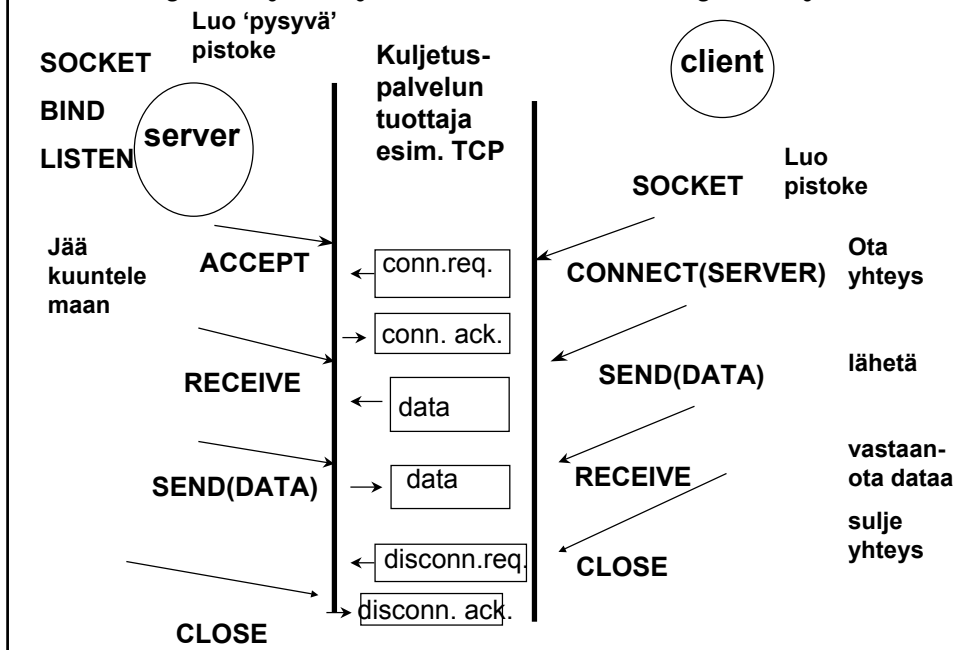
TCP:n pistokeprimitiivit

- **SOCKET** luo uusi yhteydenpäätepistoke
- **BIND** anna pistokkeelle osoite
- **LISTEN** halukas vastaanottamaan yhteyksiä
- **ACCEPT** jää odottamaan yhteysyrityksiä
- **CONNECT** yritä muodostaa yhteys
- **SEND** lähetä dataa yhteyttä pitkin
- **RECEIVE** vastaanota dataa yhteydeltä
- **CLOSE** pura yhteys (symmetrinen)

8/18/2003

15

Kuljetusyhteyden muodostus ja käyttö



3.3 UDP

■ UDP (User Data Protocol)

- voidaan lähettää sanomia ilman yhteyden muodostusta

UDP-otsake

<----- 32 bittiä ----->

Source port #	Destination port #
UDP length	UDP checksum
sovelluksen dataa	

UDP-tarkistussumma

■ Virheen havaitsemista varten otsakkeeseen liitetään tarkistussumma

- kaikki segmentin 16 bitin sanat lasketaan yhteen ja summasta otetaan yhden komplementti
 - = muutetaan ykköset nolliksi ja nollat ykkösiksi
- vastaanottaja laskee taas kaikkien segmentin sanojen (mukana myös tarkistussumma) summan jos tulokseksi saadaan 16 ykköstä, niin ok!

Esimerkki

- Lasketaan tarkistussumma kolmen tavun mittaiselle sanomalle:

■ Lähettäjä	vastaanottaja	
1011 0100	1011 0100	1011 0100
0111 0101	0111 0101	1111 0101
1000 1101	1000 1101	1000 1101
=====	0100 1001	0100 1001
1011 0110	=====	=====
↓	1111 1111	0111 1111
0100 1001	OK!	VIRHE!

Yhden komplementti

8/18/2003

19

- Miksi tarvitaan tarkistussumma?

- Kaikki siirtoyhteyskerrokset eivät suorita tarkistuksia!

- UDP-tarkistussumma ei ole kovin tehokas havaitsemaan virheitä!

- Se ei myöskään yritä toipua virheistä!

- Jotkut toteutukset voivat tuhota virheellisen segmentin
- jotkut antavat se sovellukselle varoituksen kera

8/18/2003

20

UDP:n etuja:

- **Yhteydetön**
 - aikaa ei kulu yhteyden muodostamiseen ja purkamiseen
 - ei tarvita resursseja yhteyden tilatietojen ylläpitoon
- **Otsake (= 8 tavua) pieni => pieni yleisrasite => lisää tehokkuutta**
- **Ruuhkanvalvonta ei säännöstele liikennettä**

Tehtäviä:

- **Lähetetään 10 tavun viesti UDP:llä.**
 - **Miten kauan kestää lähettäminen, jos lähetyksenopeus on 56 kbps?**
 - $10 \text{ tavua} + 8 \text{ tavua} = 18 * 8 \text{ b} = 144 \text{ bittia}$
 - $144 \text{ b} / 56\,000 \text{ b/s} = 2.57 \text{ ms}$
 - **Miten suuri on etenemisviive, jos etäisyys lähettäjältä vastaanottajalle on 1000 km?**
 - $1000\text{km} / 200\,000 \text{ km/s} = 5 \text{ ms}$
 - **Miten suuri on UDP-otsakkeen aiheuttama yleisrasite (overhead)?**
 - $8/18 = 0.44$ eli 44 %

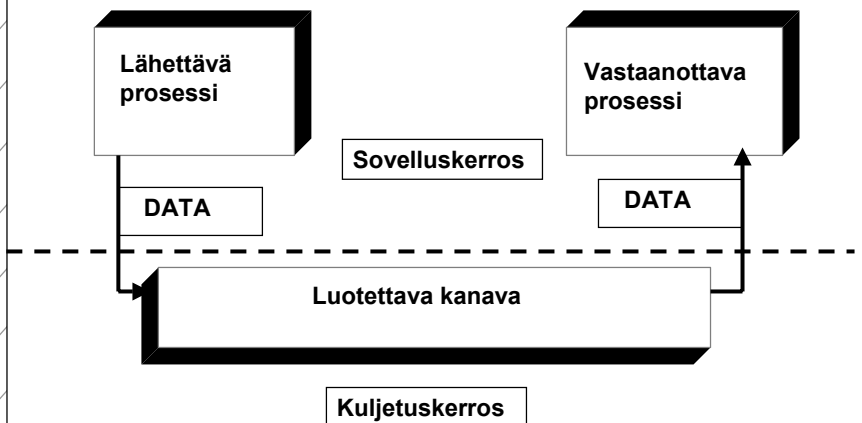
UDP:n käyttö

- **Vaikka UDP on epäluotettava, se sopii monien sovellusten tarpeisiin:**
 - Remote file server (NFS)
 - multimedia
 - Internet-puhelin
 - verkon hallinta (SNMP)
 - reititys (RIP)
 - nimipalvelu (DNS)
- **Miksi nämä sovellukset suosivat UDP:tä?**

8/18/2003

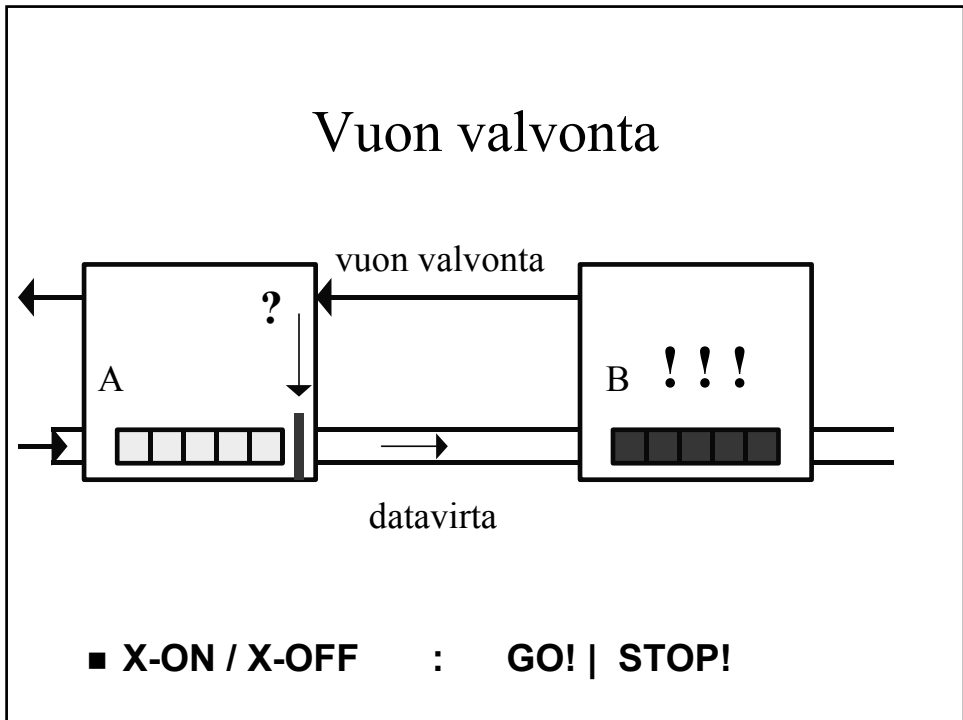
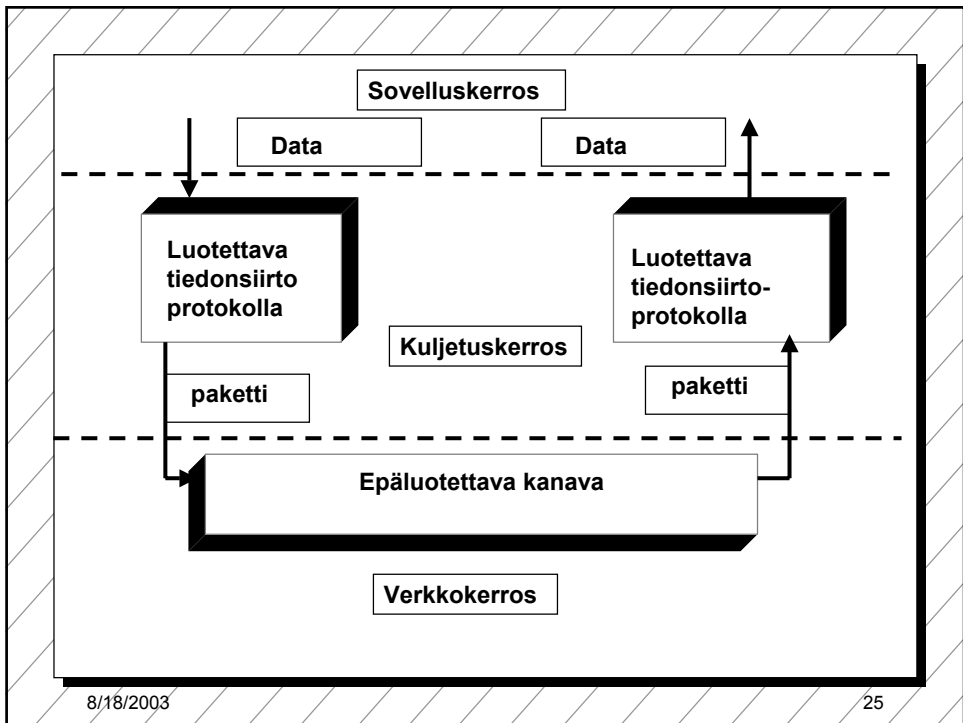
23

3.4 Luotettava tiedonsiirto

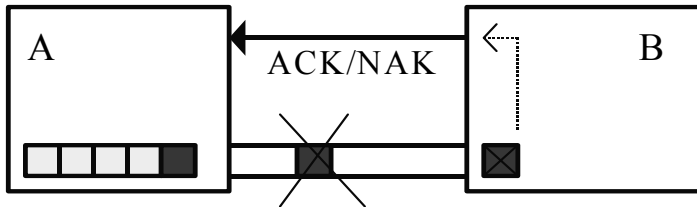


8/18/2003

24



Kohinainen kanava



- sanoma vääristyy => virhetarkistus
- sanoma katoaa => numerointi, ajastin ja uudelleenlähetys
 - duplikaattien havaitseminen
- sanoma viivästyy => rajallinen elinaika
- sanomien järjestys muuttuu => järjestäminen

Stop and wait -protokolla

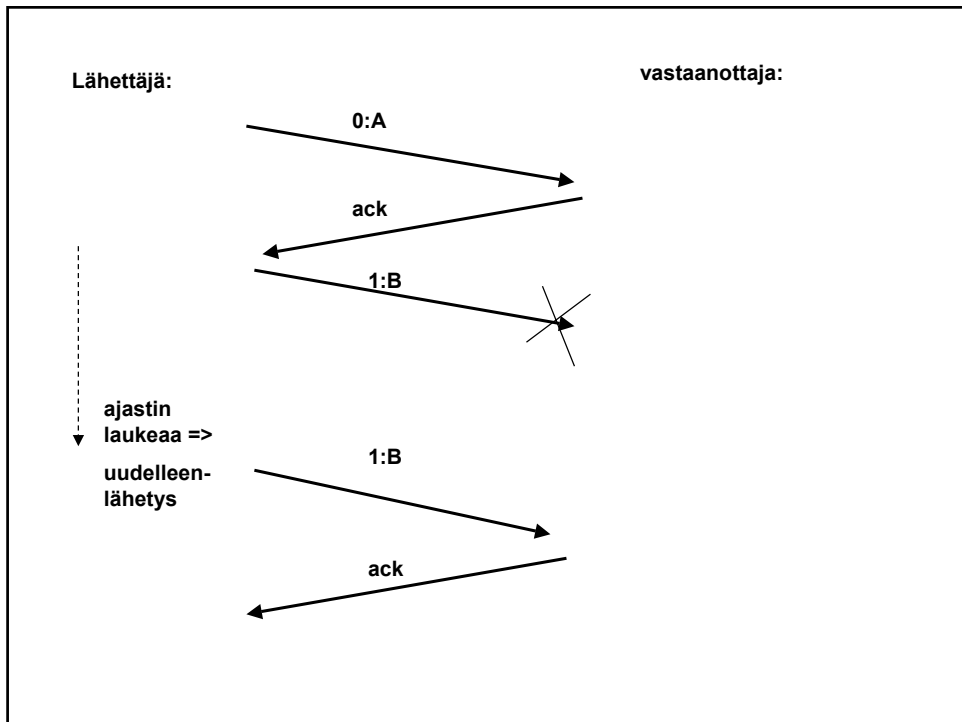
- **Oletus**
 - virheetön siirto => ei huolta virheistä, mutta vuonvalvontaa tarvitaan
- **lähettäjä**
 - lähettää sanoman
 - odottaa lupaa lähettää seuraava sanoma
- **vastaanottaja**
 - käsittelee sanoman
 - lähettää tiedon (=antaa luvan) lähettäjälle

Entä jos virheitä?

- Sanomissa virheitä tai sanomat voivat puuttua kokonaan
- Myös kuittaukset voivat kadota
- Tarvitaan
 - virheen havaitseminen ja korjaaminen
 - tarkistussumma
 - Kuittaus (ACK/NAK/toistoACK)
 - uudelleenlähetykset
 - sanomien numerointi
 - uudelleenlähetyksistä

Todellisempi “stop and wait” - protokolla

- ajastin lähettäjälle
 - jos kuittausta ei kuulu, sanoma lähetetään automaattisesti uudelleen
 - kuittaus: ACK = ‘ok, lähetä seuraava’
 - uudelleenlähetykset synnyttävät kaksoiskappaleita!
- Sanomanumerointi
 - jotta vastaanottaja tunnistaa kaksoiskappaleet
 - Miten paljon numeroita tarvitaan?
 - » Numero vie tilaa sanomassa!



Stop and wait -protokollan suorituskyky

■ Esim. satelliittiyhteydellä

- 50 kbps, kiertoviive ~520 ms, sanoma 1000 bittiä
- kanavan käyttöaste < 4%

■ => lähetetään useita sanomia ja sitten vasta odotetaan kuittauksia

- ideaali: lähetykset liukuhihnalla (pipeline)
 - lähetykset ja kuittaukset limittyvät
 - ei mitään odottelua
 - lähetyiskanava koko ajan käytössä
- suorituskyky kasvaa

Liukuvan ikkunan protokolla

(Sliding Window)

- **Lähetysikkuna**
 - ikkunan koko
 - montako sanomaa saa korkeintaan olla kuittaamatta
 - järkevä koko riippuu yhteyden tyypistä ja vastaanottajan kapasiteetista
 - kiinteä koko /vaihteleva koko
 - sisältö = mitkä sanomat saa lähettää
 - sanomalla järjestyksnumero
 - rajallinen, N bittiä => 2^{*N} arvoa
 - numerot käytettävä järjestyksessä

8/18/2003

33

- **Lähetäjä joutuu odottamaan vasta, kun kaikki ikkunan sanomat on lähetetty**
 - eli numerot käytetty
- **Kun kuitaus saapuu => ikkuna liikuu**
 - seuraavat numerot tulevat luvallisiksi
- **eli**
 - lähettäjä: tietyllä hetkellä sallittujen numeroiden joukko = lähettäjän ikkuna
 - mitkä sanomat saa lähettää “etukäteen” odottamatta kuitausta

8/18/2003

34

- **Vastaanottajan ikkuna**
 - kullakin hetkellä sallittujen numeroiden joukko
 - mitä sanomia suostuu vastaanottamaan
 - **kuittaus muuttaa myös vastaanottajan ikkunan**
- **ikkuna pysäyttää sanomien lähetyksen**
 - seuraava sanomanumero ei ole lähetyksikkunassa
- **ikkuna estää sanoman vastaanoton**
 - saadun sanoman numero ei ole vastaanottoikkunassa

Kun ikkunan koko on 1

- **Aina vain yksi sanoma kuittaamattomana**
 - => One Bit Sliding Window -protokolla
 - ~ stop and wait -protokolla
- **sanomanumerot 0 ja 1 riittävät**
- **ACK-sanoma identifioi viimeksi vastaanotetun virheettömän sanoman**
 - jotta kuittausduplikaatti ei voi kuitata väärää sanomaa
 - **ACK ilmoittaa joko**
 - » seuraavaksi odotetun sanoman numeron
 - » viimeksi vastaanotetun sanoman numeron

- **Entä kun tapahtuu virhe?**
 - kaksi eri tapaa hoitaa
 - 1. toisto virheestä lähtien (go back n) (tai paluu n:ään)**
 - 2. valikoiva toisto (selective repeat)**

Toisto virheestä eli Paluu n:ään ('Go back n')

- **virheellisen sanoman havaittuaan**
 - vastaanottaja hylkää kaikkia sen jälkeiset sanomat eikä lähetä niistä kuittauksia
 - => sanomat hyväksytään vain oikeassa järjestyksessä
- **kun lähettäjä ei saa kuittauksia,**
 - sen lähetysikkuna 'täyttyy'
 - eikä se voi enää lähettää
- **lähettäjän ajastimet laukeavat aikanaan ja**
 - virheellinen sanoma
 - sekä kaikki sen jälkeen lähetetyt sanomat lähetetään uudelleen
- **tehoton, jos paljon virheitä ja iso ikkuna**

Valikoiva toisto

- vastaanottaja hyväksyy kaikki kelvolliset sanomat
 - se kuittaa sanomat
 - puskuroi ne ja toimittaa eteenpäin oikeassa järjestyksessä
 - » tarvitaan puskuritilaa
- Kun lähettäjä ei saa kuittausta virheellisestä sanomasta
 - ajastin laukeaa ja sanoma lähetetään uudelleen
 - lähettää uudelleen vain virheellisen sanoman
 - ikkuna liukuu nytkin tasaisesti
 - » yksi puuttuva kuittaus voi pysäyttää lähetyksen

Kuittaukset

- ACK
 - kumulatiivinen ACK
 - tähän saakka kaikki ok!
 - Go-Back N
 - yksittäinen ACK
 - vain tämä ok!
 - Valikoiva toisto
- NAK-kuittaus
 - sanoma virheellinen tai puuttuu

Negatiiviset kuittaukset

- **NAK-kuittauksilla voidaan nopeuttaa uudelleenlähettämistä**
 - vastaanottaja ilmoittaa heti virheellisestä tai puuttuvasta kehyksestä
 - ei ole tarpeen odottaa ajastimen laukeamista
- **hyödyllinen, jos kuittausten saapumisaika vaihtelee paljon**
 - ajastinta vaikea asettaa oikein

8/18/2003

41

- **NAK-kuittaukset voivat aiheuttaa turhia uudelleenlähetyksiä**
 - lähetys ja kuittaus menevät ristiin
- **NAK-kuittauksen katoaminen ei haittaa**
- **implisiittinen uudelleenlähetys**
 - Toistetaan samaa ACK-kuittausta (toistokuittaus)
 - ei käytetä NAK-kuittauksia
- **explisiittinen uudelleenlähetys**
 - käytetään NAK-kuittauksia

8/18/2003

42

Ikkunankoko

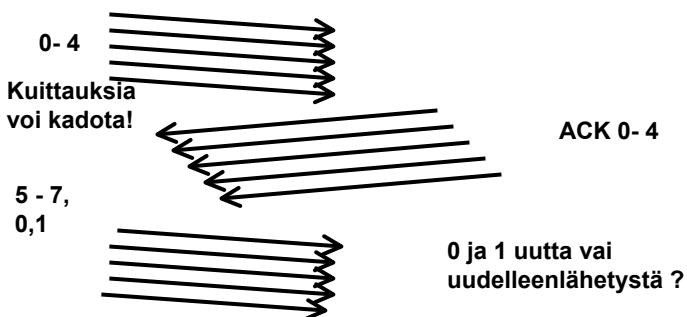
- Kun käytetty numeroavaruus on $0, 1, \dots, n$ ja eri numeroita siis käytettävissä $n+1$
 - yleensä jokin kakkosen potenssi
 - » koska numerokentän koko k bittiä => käytössä 2^{**k} numeroa
- ikkunan koko 'go back n ':ssä voi olla korkeintaan n
 - eli oltava ainakin yhtä pienempi kuin numeroavaruus
- ikkunan koko valikoivassa toistossa voi olla korkeintaan $(n+1)/2$
 - saa olla korkeintaan puolet numeroavaruudesta

8/18/2003

43

Miksi?

Valikoiva toisto: ikkuna 5, numeroavaruus 8

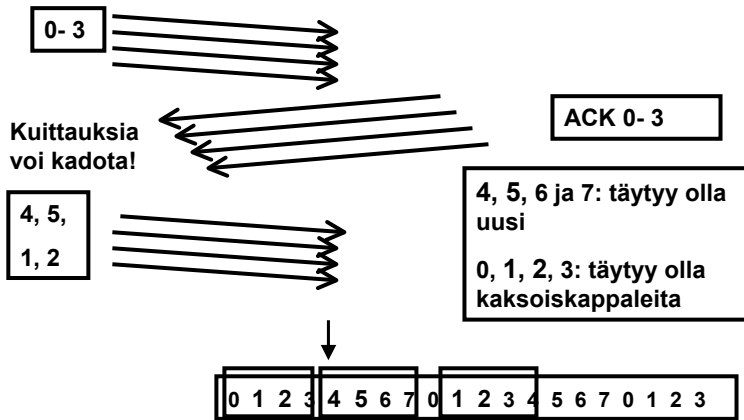


8/18/2003

44

Miksi?

Valikoiva toisto: ikkuna 4, numeroavaruus 8



8/18/2003

45

Kaksisuuntainen liikenne

- datakehys ja kuittauskehys
- kehyksessä sekä data että kuittaus
 - ‘piggybacking’
 - tehostaa lähetystä
- ongelma: kauanko kuittaja odottaa dataa ennen pelkän kuittauksen lähettämistä?

8/18/2003

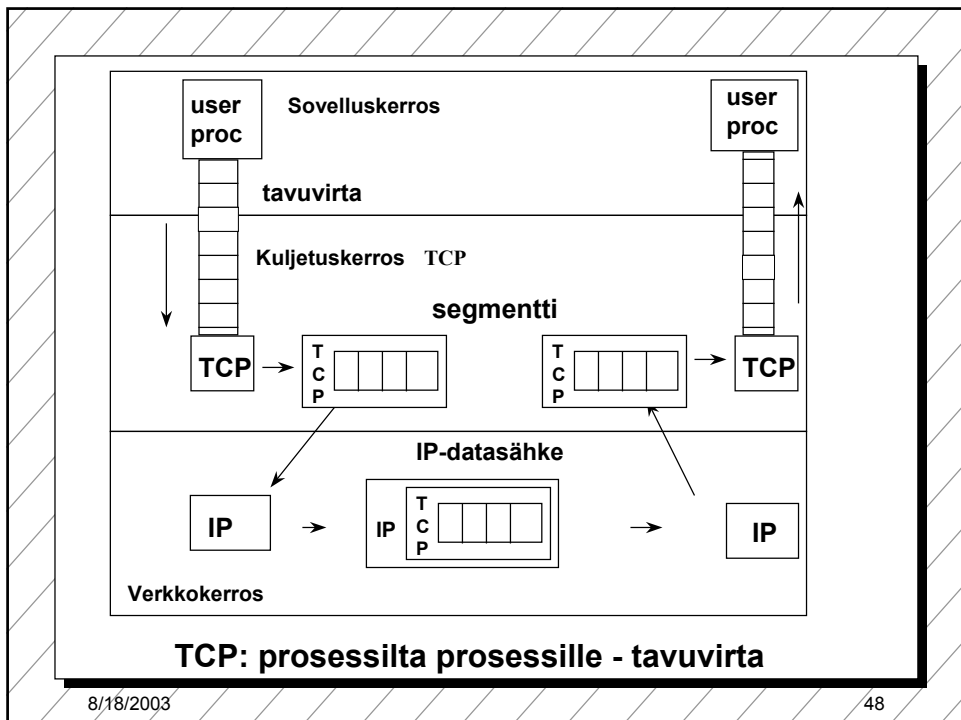
46

3.5. TCP-protokolla

- yhteyden muodostus ja purku
- luotettavan tavuvirran toteuttaminen
- vuonvalvonta
- siirron optimointi
- TCP-segmentti
- ruuhkan valvonta
- TCP-palvelun käyttö

8/18/2003

47



8/18/2003

48

Yhteyden muodostus ja purku TCP:ssä

- TCP käyttää yhteyden muodostamiseen ja purkuun ns. kolminkertaista kättelyä (three-way handshake)
 - välissä oleva verkko tekee yhteyden muodostamisen ja purun hankalaksi
 - viivästyneet sanomat => sanomille elinaika (max 3 minuuttia)
 - sanomien numeroinnista sopiminen
 - Kahden armeijan ongelma (two-army problem)
 - “hyökkään, jos olen varma, että sinäkin hyökkäät”
 - symmetrinen yhteyden purku = molemmat osapuolet tietävät, että toinenkin on varmasti purkanut yhteyden

8/18/2003

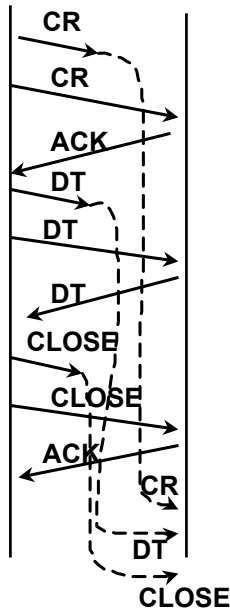
49

Yhteyden muodostus ruuhkaisessa verkossa

Jokainen paketti lähetetään kahteen kertaan

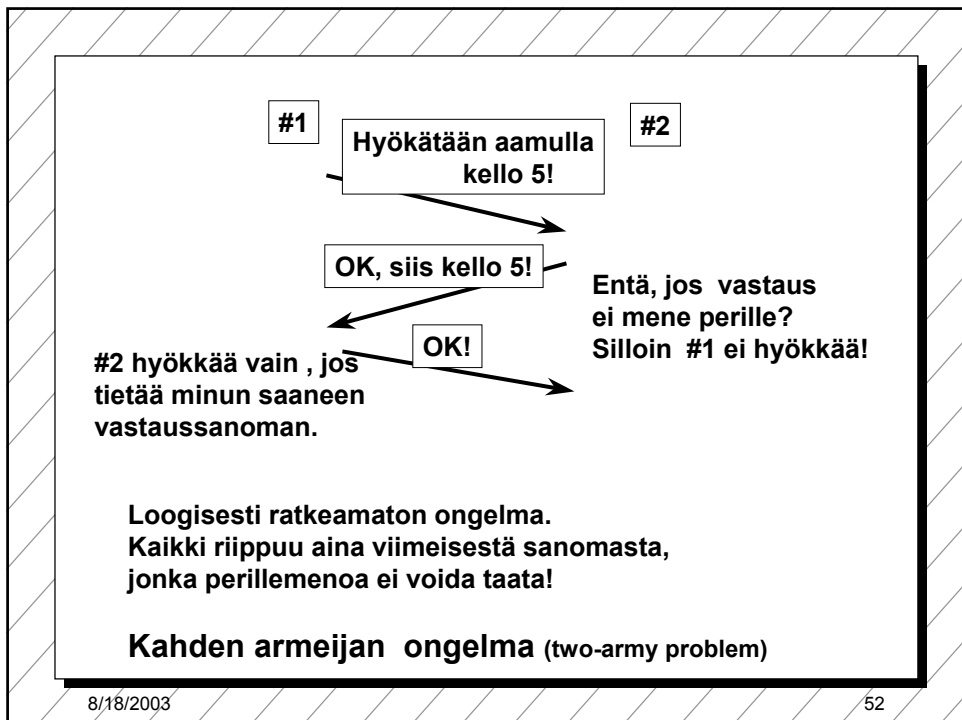
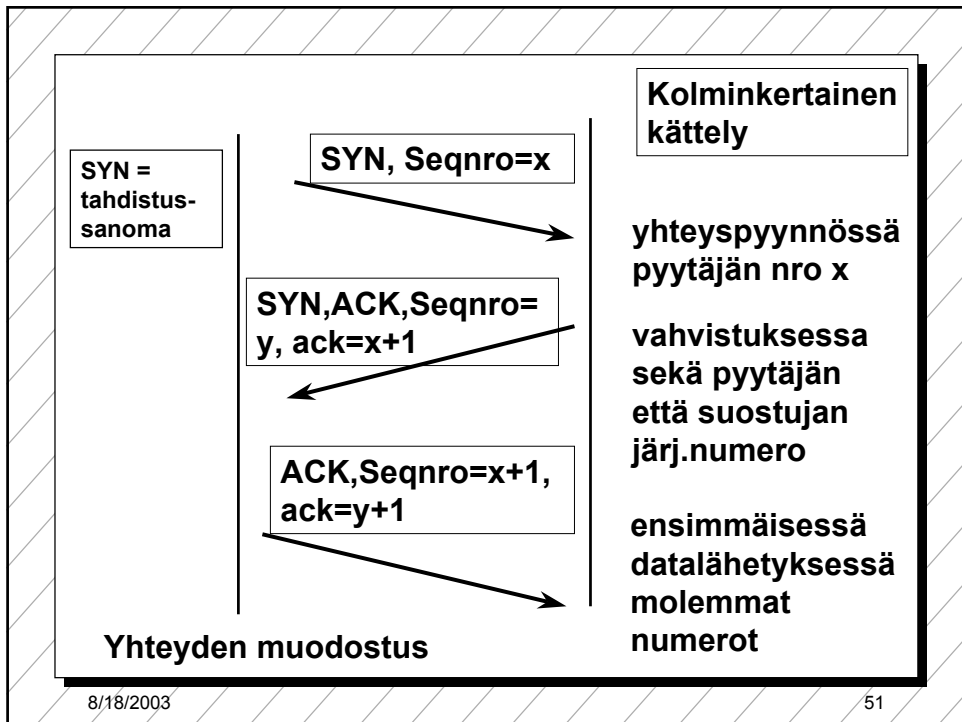
Kun yhteys on purettu, viivästyneet kaksoiskappaleet saapuvat

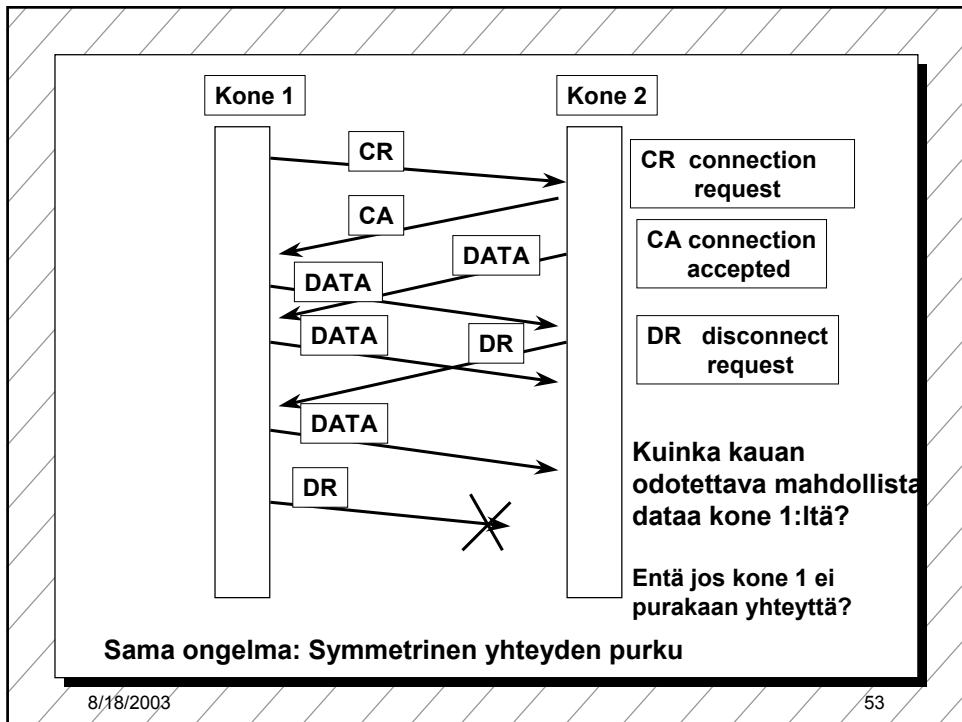
Ne tulkitaan uudeksi yhteydeksi ja data otetaan vastaan kahteen kertaan!



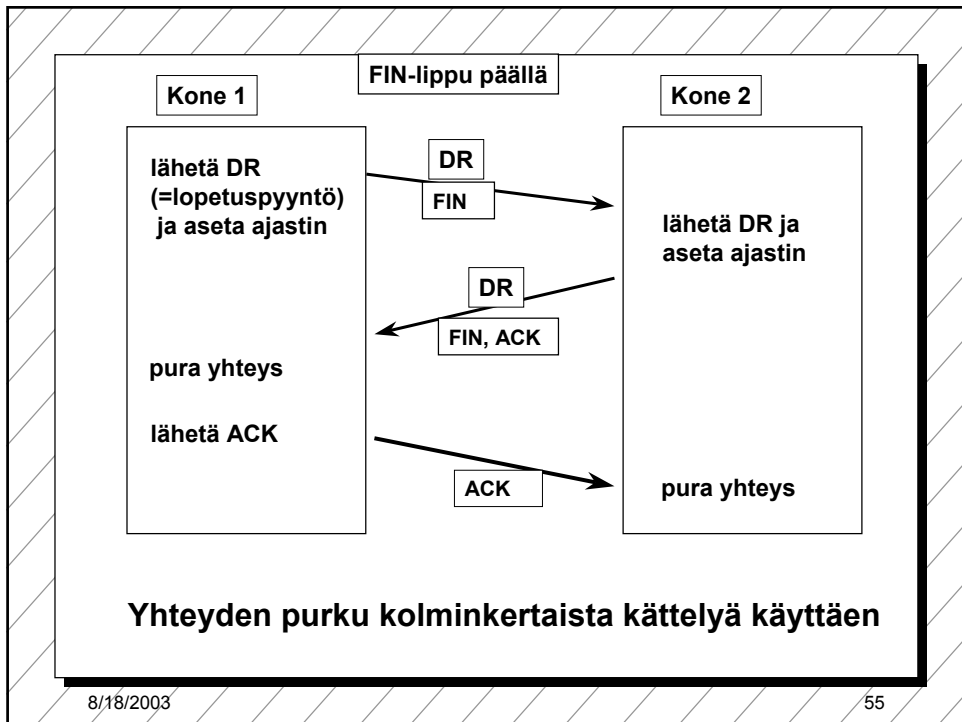
8/18/2003

50





- ## Yhteyden purku
- molemmat suunnat puretaan erikseen
 - TCP-segmentti
 - FIN = 1
 - ei enää dataa lähetettävä
 - kun saadaan kuittaus => yhteys tähän suuntaan purettu
 - yhteys kokonaan purettu, kun molemmat suunnat purettu
 - purussa käytetään ajastimia
 - 2 * paketin maksimaalinen elinikä
- 8/18/2003
- 54



- ## TCP: Virheettömyys ja järjestys
- **Järjestysnumerot**
 - tavuvirta => tavunumerointi
 - segmentin 1. tavun järjestysnumero
 - yhteyden alussa satunnaiset numerot
 - **kuittaukset**
 - kumulatiivinen ACK, ei NAK-kuittausta
 - kuittauksessa seuraavaksi odotettava tavu
 - kuitataan 'tiheästi'
 - vähintään joka toinen
- 8/18/2003 56

- **Go Back N -tyyppinen**
 - virheellisiä tai väärässä järjestyksessä tulleita ei hyväksytä
 - ne voidaan myös tallettaa
 - mutta ei välttämättä lähetä kaikkia virheellisestä lähtien uudestaan
- **Myös ehdotettu valikoivan toiston tyyppistä kuittaamista**
 - SACK-kuitaus, joka kertoo, mitkä segmentit on vastaanotettu ok

Toistokuittaukset

- **Ensikuittaus**
 - ensimmäinen vastaanotettu sanoman kuittaus
 - ACK(i): sanomaan i saakka kaikki OK!
- **toistokuittaus (duplicate ACK)**
 - väärässä järjestyksessä saatu segmentti tai virheellinen segmentti => toistetaan uudestaan jo annettu kuittaus
 - NAK-kuitauksen korvike
 - 3 toistokuittautusta => segmentti kadonnut tai virheellinen

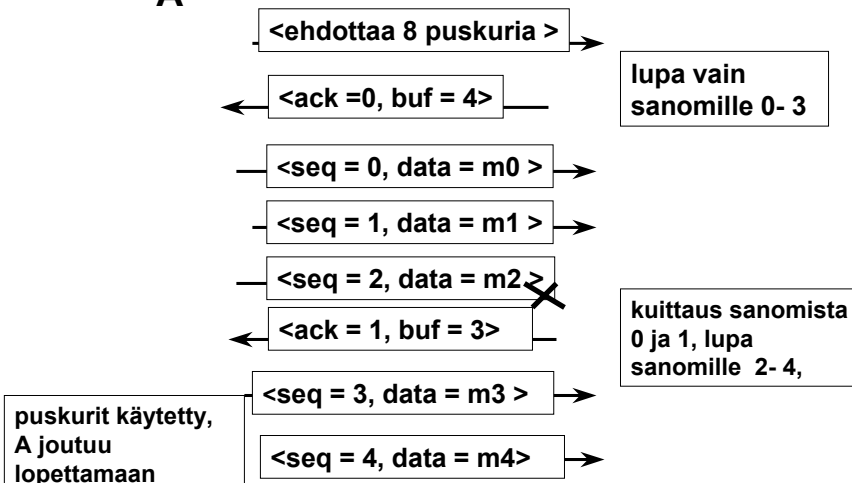
TCP:n vuonvalvonta

- **'joustava' liukuva ikkuna** (sliding window) ("credit-vuonvalvonta")
- **vastaanottaja kertoo, kuinka paljon suostuu vastaanottamaan**
 - => **kuittaus irroitettu vuonvalvonnasta**
 - puhtaassa liukuvassa ikkunassa kuittaus siirtää ikkunaa
 - **AdvertisedWindow-kenttä**
 - paljonko saa lähettää = paljonko vastaanottajan puskureihin mahtuu
- **myös ruuhkan valvonta rajoittaa lähettämistä**

8/18/2003

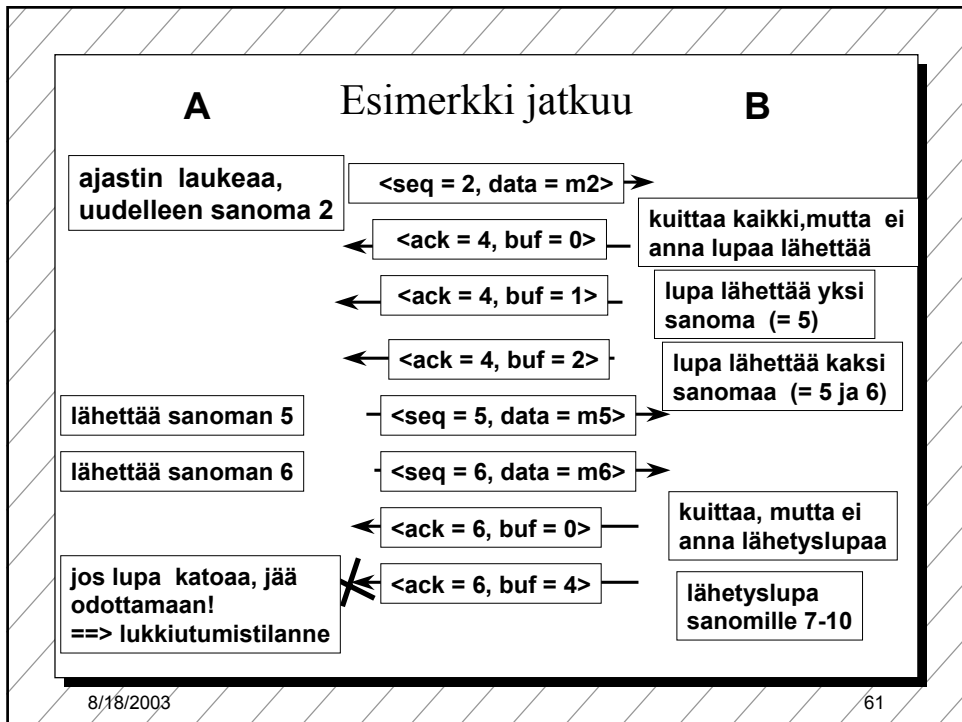
59

A Esimerkki B



8/18/2003

60



- jos ilmoitus lisäpuskureista katoaa, lähettäjä lukkiutuu odotustilaan
 - vastaanottaja voi luulla, ettei ole lähetettävää
 - lukkiutumisen estämiseksi
 - kun ikkunankoko = 0 lähettäjä ei saa lähettää, paitsi
 - erityistä pikadataa (URG)
 - yhden tavun 'kyselyn', jonka vastaanottaja kuittaa ja samalla ilmoittaa ikkunan koon => estää turhat lukkiutumiset
- 8/18/2003 62

Siirron optimointi

- **TCP saa optimoida lähettämisiään**
 - ei tarvitse lähettää heti kun data on tullut
 - dataa kerätään puskuriin ja lähetetään sopivassa tilanteessa
 - **PUSH-lipun avulla sovellus ilmoittaa, että data on lähetettävä heti**

Optimointi on usein tarpeen:

- **Interaktiivinen editori => merkki lähetetään heti**
 - 21 tavun TCP-segmentti => 41 tavun IP-paketti
 - joka kuitataan 40 tavun IP-paketilla
 - ilmoitus uudesta ikkunan koosta 40 tavun IP-paketilla
 - kaiutetaan merkki vielä 41 tavun IP-paketilla
- yhden merkin käsittely =>
 - 162 tavun siirtäminen
 - ja neljän segmentin lähettäminen

TCP-segmentti

■ segmentti

- 20 tavun otsake
 - + optionaalinen osa
- dataosa
 - voi puuttua

■ segmentin kokoa rajoittaa

- MTU (Maximum transfer unit)
 - verkon rajoitus maksimikoolle (muutama tuhat tavua)
- IP-paketin dataosa korkeintaan 65535 tavua

■ liian isot segmentit paloitellaan

- joka palalle IP-otsake => yleisrasite kasvaa

TCP-otsakkeen kentät

Source port		Destination port	
Sequence number			
Acknowledgement number			
TCP head. length		U R G K	A P R S F
		H T N N	Window size
Checksum		Urgent pointer	
Options (0 or more 32 bit words)			
Data (optional)			

TPC-segmentin otsakekentät

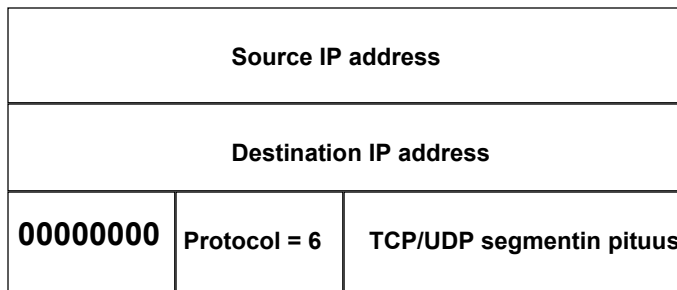
- **Lähde- ja kohdeportit** (Source port, Destination port)
 - yhteyden päätepisteet
 - portti + koneen IP-osoite => 48 bittinen TSAP
- **Järjestysnumero** (Sequence number)
 - tavut numeroidaan => 32 bittiä
 - segmentin ensimmäisen tavun numero
- **Kuittausnumero** (Acknowledgement number)
 - seuraavaksi odotettu tavu
- **TCP-otsakkeen pituus** (TCP header length)
 - mahdollisten optiokenttien takia
- **6 bitin käyttämätön kenttä**

- **6 lippubittiä**

- **URG** onko pikadataa
pikadatan sijainnin ilmoittaa
pikadatakenttä (Urgent pointer)
- **ACK** onko kuittauskenttä käytössä
- **PSH** onko hetilähetettävää (pushed) dataa
- **RST** yhteyden uudelleenalustuspyyntö (reset),
yleensä ongelmatilanne
- **SYN** käytetään yhteyttä muodostettaessa
SYN =1, ACK = 0 connection request
SYN =1, ACK = 1 connection accepted
- **FIN** käytetään yhteyden purkuun
FIN =1 ei enää lähetettävää

- **Ikkunan koko** (window size)
 - vaihteleva ikkunankoko
 - kuittaus irroitettu lähetysluvasta
- **Tarkistussumma** (Checksum)
 - lasketaan otsakkeelle, datalle ja ns. pseudo-otsakkeelle

pseudo-otsake



Auttaa havaitsemaan väärään osoitteeseen toimitetut paketit.

Sisältää IP-otsakkeen tietoja!

■ Optiokenttä (options)

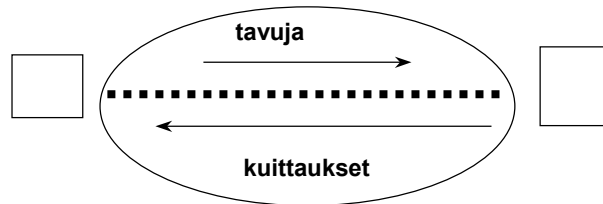
- voidaan lisätä piirteitä, joita ei ole varsinaisessa otsakkeessa
 - suurin hyväksyttävä datakenttä
 - ikkunan koon moninkertaistaminen (window scale)
 - nopeille ja pitkän viipeen linjoille 64 ktavun ikkunan koko on liian pieni
 - valikoivan toiston käyttö 'go back N':n tilalla
 - vähentää turhia uudelleenlähetysiä

3.6. TCP:n ruuhkan valvonta

- Liikaa kuormitusta => verkko ruuhkautuu => hidastetaan lähettämistä
- Ruuhkan havaitseminen
 - nykyisin siirtovirheet harvinaisia
 - poikkeuksena langattomat verkot
 - => uudelleenlähetykset johtuvat ruuhkasta
 - uudelleenlähetysajastimen laukeaminen on merkki ruuhkasta

■ ruuhkaikkuna

- “paljonko tavuja (segmenttejä) lähettäjällä saa korkeintaan olla verkossa liikkeellä”
 - paljonko lähettäjä saa kuormittaa verkkoa
- kuittaus => ko. tavut jo poistuneet verkosta



8/18/2003

73

■ Ruuhkaikkunan koko?

- Lähettäjän on itse pääteltävä ja arvioitava sopiva ruuhkaikkunan koko
 - kukaan muu ei sitä kerro!
 - uudelleenlähetyksistä laukeaa => on ruuhkaa
 - kuittaukset tulevat tasaisesti => ei ole ruuhkaa
- Internet-verkon kuormitus voi vaihdella paljon

■ Dynaaminen ruuhkaikkunan koko:

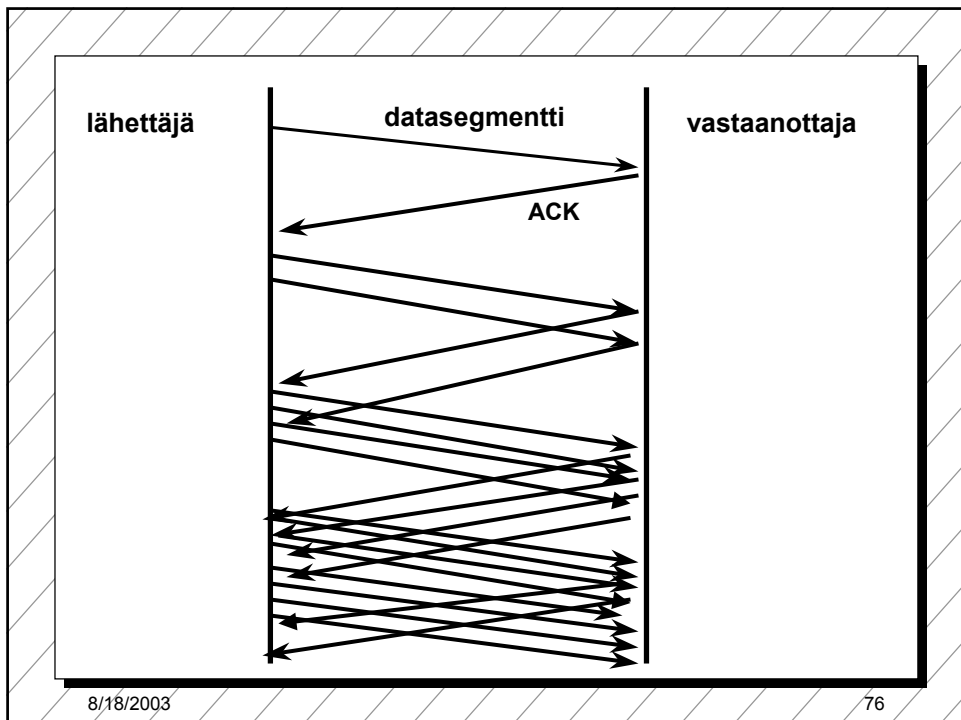
- ruuhkaikkunaa kasvatetaan, kunnes törmätään ruuhkaan
 - ensin kasvatetaan melko nopeasti, sitten varovaisemmin
- sen jälkeen ruuhkaikkunaa pienennetään reilusti
- ja aletaan uudestaan kasvattaa ruuhkaikkunaa

8/18/2003

74

Hitaan aloituksen algoritmi (slow start)

- Algoritmi pyrkii löytämään sopivan ikkunan koon yhteyden alussa tai ruuhkatilanteen jälkeen mahdollisimman nopeasti
 - ei ole niin kovin hidas, vaan alussa eksponentiaalinen!
 - alussa ruuhkaikkuna = yksi segmentti
 - kuitattu ruuhkaikkunallinen kasvattaa ruuhkaikkunan kaksinkertaiseksi



■ kynnysarvo (threshold)

- aluksi 64 K
- 'varoitussarvo' = tästä lähtien syytä varoa ruuhkaa
- kynnysarvoon saakka voidaan kasvattaa ruuhkaikkunaa eksponentiaalisesti
- kynnysarvon saavuttamisen jälkeen kasvatetaan ruuhkaikkunaa vain lineaarisesti
 - = kasvatetaan kuittausten jälkeen vain yhdellä
 - edetään hyvin varovaisesti!

■ jos ajastin ehtii laueta => ruuhkatilanne

- kynnysarvoksi puolet nykyisestä ruuhkaikkunan arvosta
- hitaalla aloituksella etsitään taas uusi sopiva ruuhkaikkunan arvo
 - ruuhkaikkunan arvoksi 1 segmentti
 - ruuhkaikkunaa kasvatetaan aluksi eksponentiaalisesti eli kaksinkertaistetaan kun ikkunallinen on kuitattu
- kynnysarvon saavuttamisen jälkeen kasvatetaan vain segmentti kerrallaan
- kunnes taas havaitaan ruuhka ja aloitetaan ruuhkaikkunan uuden arvon etsiminen

Uudelleenlähetysjastimen hallinta

- **uudelleenlähetysajastin** (retransmission timer)
 - asetetaan aina kun segmentti lähetetään
 - ruuhkaa, jos kuittaus ei saavu ajoissa
- **mikä on sopiva ajastimen aika?**
 - kuittaus aika vaihtelee suuresti
 - vaihtelu on myös nopeaa
- **dynaaminen arvo**
 - saadaan jatkuvien verkon suorituskykymittauksien perusteella

■ RTT

- arvio kiertoviiveelle (round-trip time)
 - mitataan jokaisen lähetetyn segmentin kiertoviive M
- $$RTT = \alpha RTT + (1-\alpha)M, \text{ tyypillisesti } \alpha = 7/8$$

■ uudelleenlähetysajastimen arvo βRTT

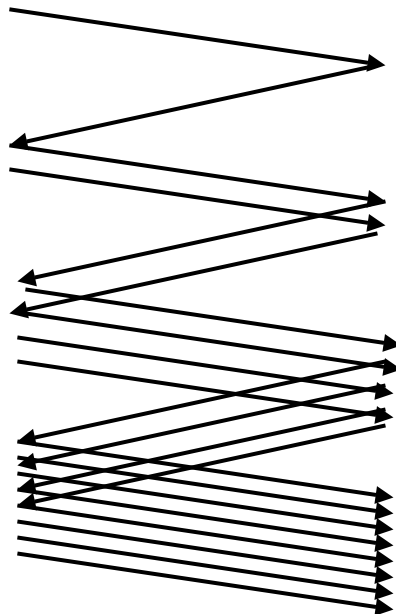
- aluksi β oli aina 2
 - parannus: otetaan huomioon myös poikkeama D (deviation) oletetun ja saadun kiertoviiveen välillä $|RTT-M|$
- $$D = \alpha D + (1-\alpha)|RTT-M|$$
- ajastimen arvo = $RTT + 4 \cdot D$

- uudelleenlähetysten vaikutus ajastimeen

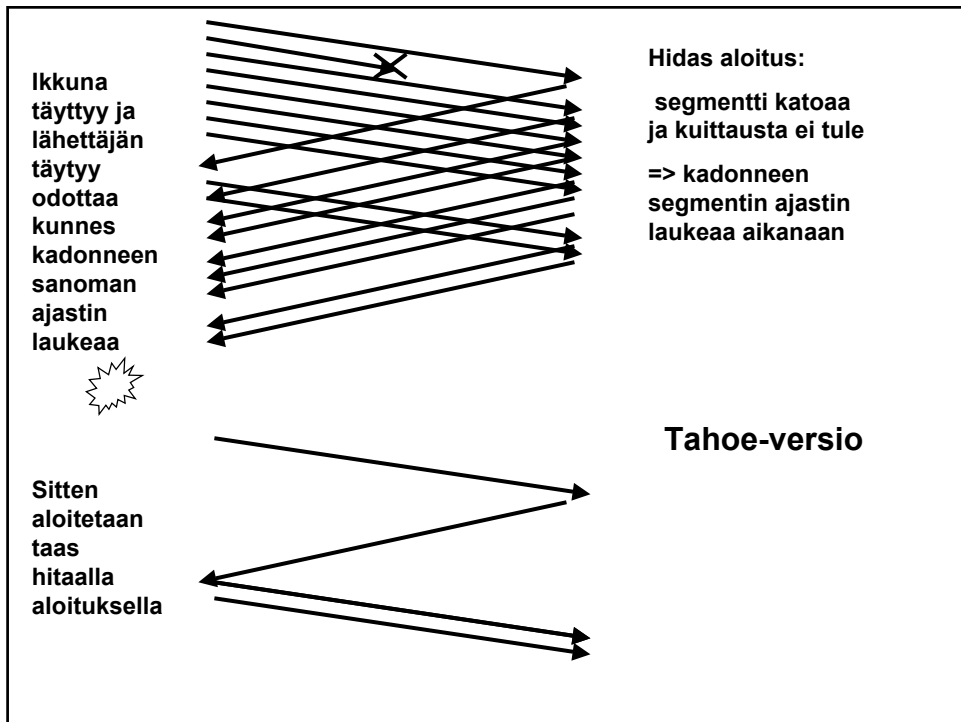
- kumpaan segmenttiin kuittaus kohdistuu?

- Karnin algoritmi

- ei oteta huomioon uudelleenlähettyjen segmenttien kuittauksia RTT:n laskemisessa



Hidas aloitus:
Lähetysmäärä kasvaa
eksponentiaalisesti



Parannuksia ruuhkanvalvontaan

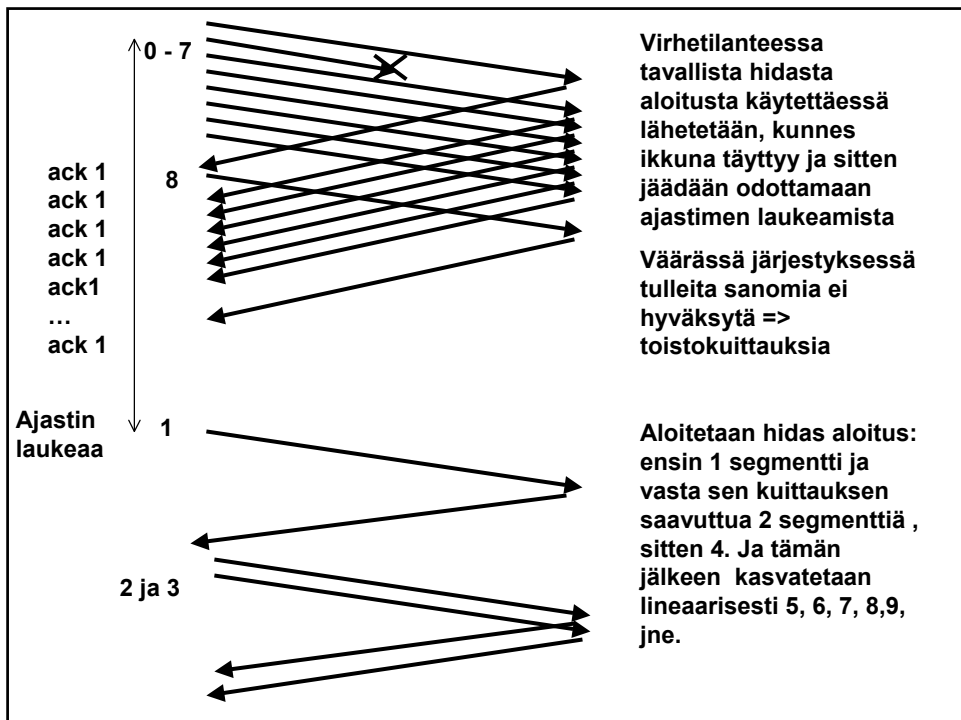
- **Nopea uudelleenlähetys** (Fast Retransmit)
 - ei odoteta ajastimen laukeamista ennen uudelleenlähetystä
 - vastaanottaja kuittaa jokaisen paketin
 - kun vastaanottaja huomaa puuttuvan paketin, se lähettää uudelleen edellisen paketin kuittauksen
 - Duplicate ACK (~ NAK)
 - kun lähettäjä saa useita (3) peräkkäisiä saman paketin toistokuittauksista=> se havaitsee tästä paketin puuttuvan ja lähettää sen heti uudelleen
 - => nopeampi uudelleenlähetys

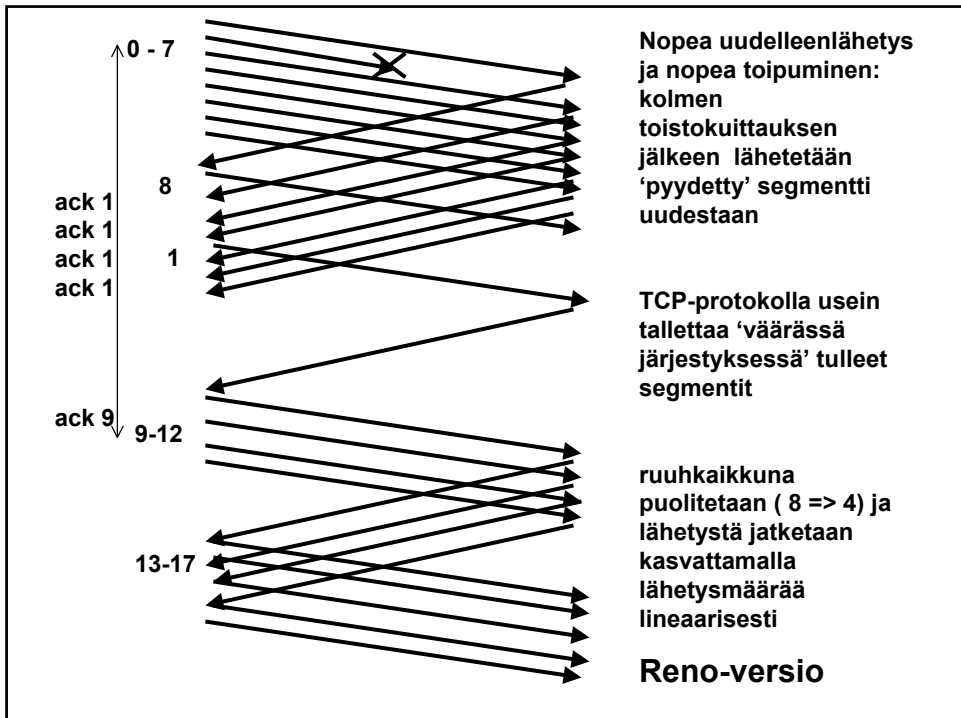
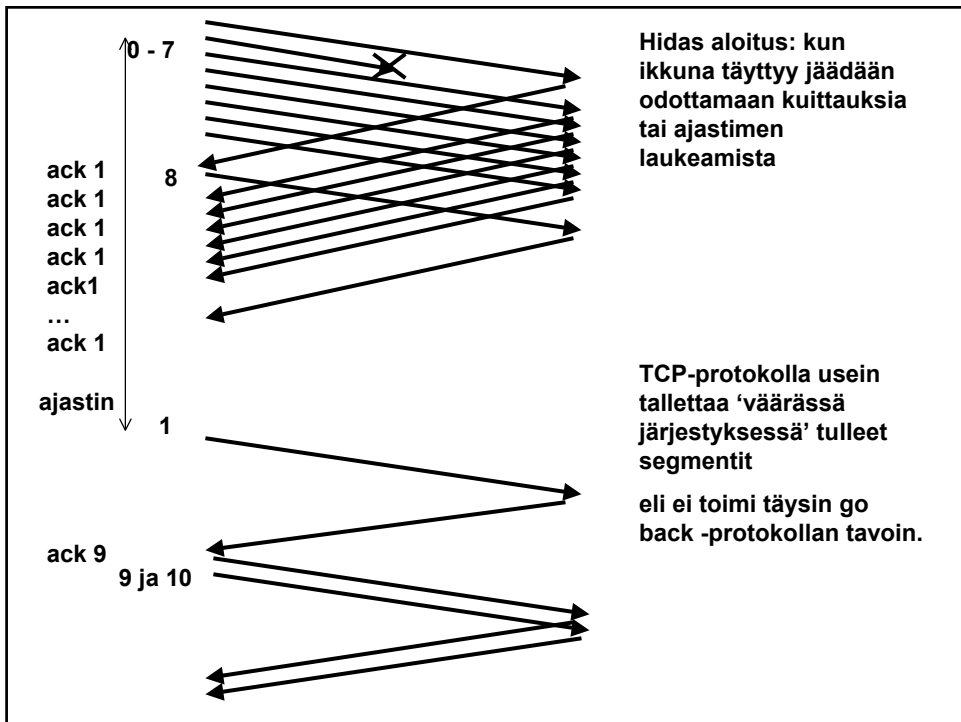
■ Nopea toipuminen (Fast Recovery)

- kun kadonnut paketti huomataan nopealla toipumisella, ei aloiteta alusta hitaalla aloituksella
 - vaan pudotetaan ruuhkaikkuna puoleen
 - ja jatketaan normaalilla lineaarisella kasvattamisella
- Mitä hyötyä tästä on?
- Miksi voidaan huoletta tehdä näin?

8/18/2003

85





- **hidas aloitus ja ruuhkan valvonta ongelmallisia langattomassa yhteydessä**

- Miksi?

- **Lisäparannuksia ruuhkanhallintaan**

- esim. Vegas

- ruuhkan ennustaminen ennen ajastimen laukeamista
 - ruuhkaikkunaa ei kasvateta aina ruuhkaan asti
 - RED (random early detection)

- entä UDP?

TCP langattomassa verkossa

- **monet TCP-toteutukset optimoitu luotettaville lankaverkoille => suorituskyky langattomissa verkoissa erittäin huono**

- ruuhkanvalvonta-algoritmi olettaa ajastimen laukeamisen johtuvan ruuhkasta

- lähettämistä hidastetaan, jotta verkon kuormitus pienenee ja ruuhkaa ei syntyisi

- langattomat yhteydet ovat epäluotettavia ja paketteja katoaa

- kadonneet paketit syytä lähettää nopeasti uudelleen

- lähetystä pitäisi päinvastoin nopeuttaa!

TCP-yhteyden hallinta

- yhteys muodostetaan kolminkertaisella kättelyllä
- passiivinen osapuoli kuuntelee
 - SOCKET
 - BIND
 - LISTEN
 - ACCEPT
- aktiivinen osapuoli aloittaa yhteydenmuodostuksen
 - CONNECT

8/18/2003

91

■ CONNECT-primitiivi

– parametreina

- IP-osoite ja porttinumero
- suurin hyväksyttävä segmentin koko
- muuta tietoa, esim. salasana

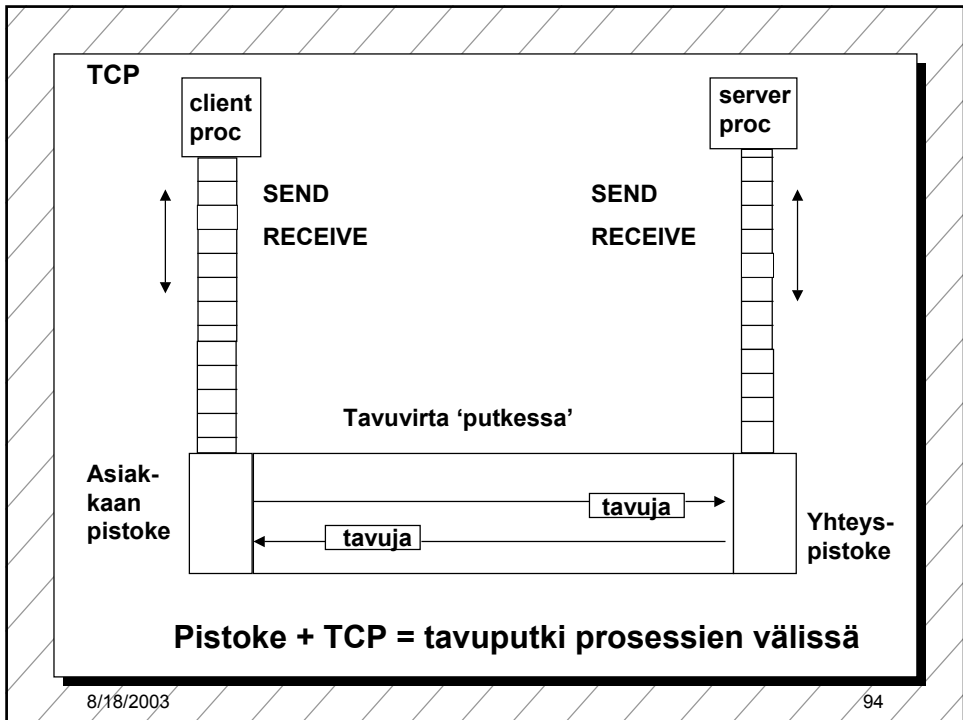
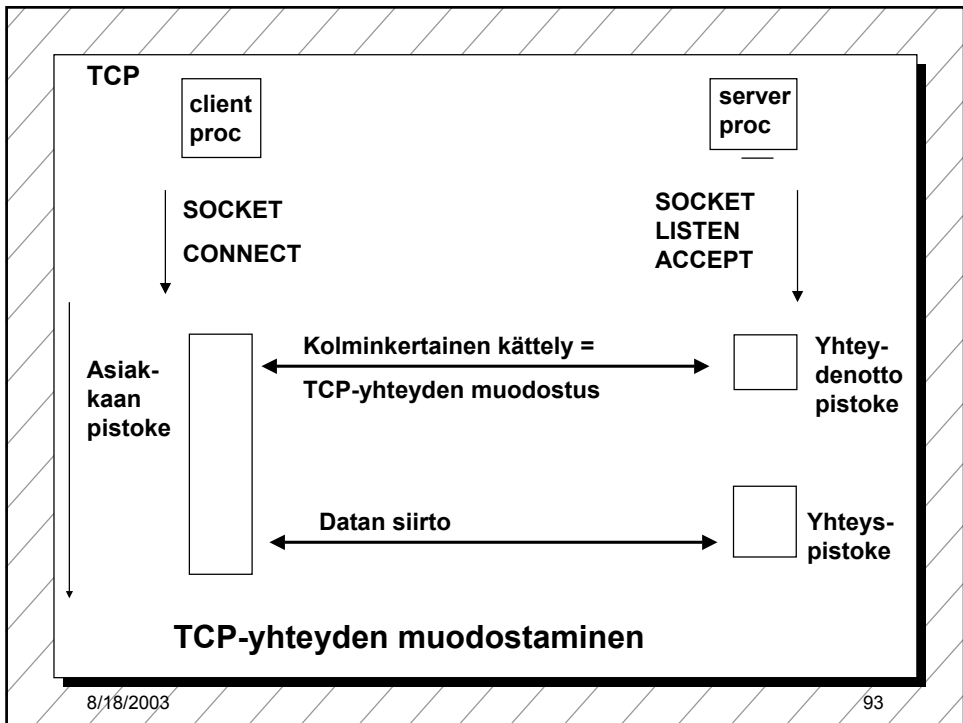


■ TCP-segmentti, jossa SYN-segmentti

- SYN = 1
- ACK = 0

8/18/2003

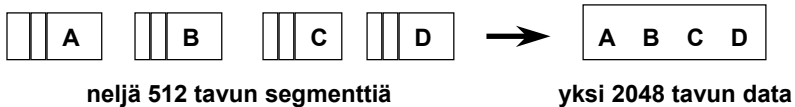
92



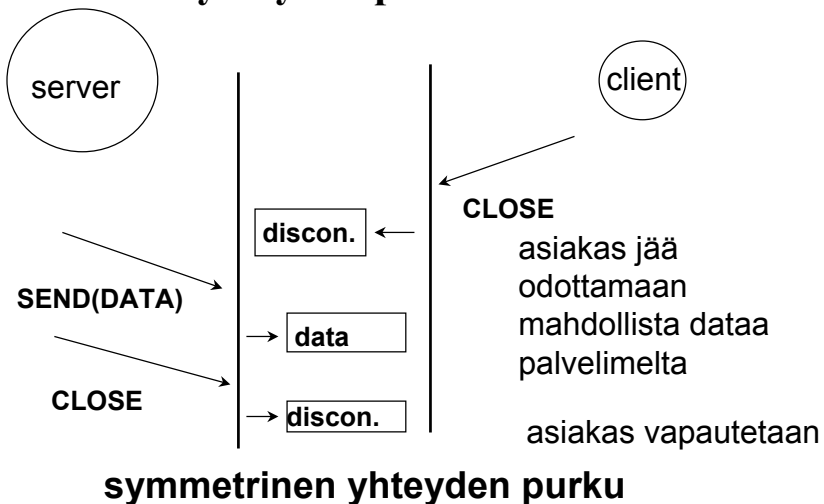
- **TCP-yhteys on tavuvirtaa, ei sanomavirtaa**
 - lähetettäessä neljä 512 tavun pätkää vastaanottaja saa joko
 - neljä 512 tavun pätkää
 - kaksi 1024 tavun pätkää
 - yhden 2048 tavun pätkän

Segmentit lähetetään neljänä eri IP-pakettina

Ne luovutetaan vastaanottajalle yhdellä READ-kutsulla



yhteyden purkaminen



C-rutiineina

int socket(int domain, int type, int protocol)

palvelin:

**int bind (int socket, struct sockaddr *address,
int addr_len)**

int listen(int socket, int backlog)

**int accept(int socket, struct sockaddr *address,
int *addr_len)**

asiakas:

**int connect (int socket, struct sockaddr *address,
int addr_len)**

**int send(int socket, char *message, int msg_len,
int flags)**

sanoman lähetys annetun pistokkeen kautta

**int recv(int socket, char *buffer, int buf_len, int
flags)**

**sanoma vastaanotto annetusta pistokkeesta
ilmoitettuun puskuriin**

Pistokeohjelmointia Javalla

- `Socket clientSocket = new Socket("hostname", 6789);`
- `clientSocket.close();`
- `ServerSocket welcomeSocket = new ServerSocket(6789);`
- `Socket connectionSocket = welcomeSocket.accept();`
- (esimerkki kirjassa Kurose, Ross, Computer Networking, A Top-Down Approach Featuring the Internet)

Pistokeohjelmointi

- **Pistokeohjelmointia ja yleensä hajautettujen verkkosovellusten tekemistä opetellaan erillisellä kurssilla**
 - **Verkkosovellusten toteuttaminen (järjestetään keväällä 2003)**

Yhteenveto

■ Kuljetuskerroksen palvelut

– UDP

– TCP

■ luotettava tavuvirta

– yhteyden muodostus ja purku

– numerointi, tarkistussumma,

– kuittaus, uudelleenlähetyt, Go-back N

– vuonvalvonta: vastaanottoikkuna (liukuva ikkuna)

– ruuhkanhallinta: hidas aloitus

– pistokeohjelmointi

