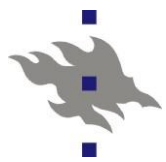




# Tietoliikenteen perusteet

## Kuljetuskerros

Kurose, Ross: Ch 3



# Sisältöä

- Kuljetuspalvelut
- Yhteydetön kuljetuspalvelu, UDP
- Luotettavan kuljetuspalvelun periaatteet
- Yhteydellinen kuljetuspalvelu, TCP
- Ruuhkanhallinta TCP:ssä

## Oppimistavoitteet:

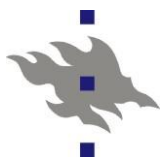
- Tuntea Internetin kuljetusprotokollien (UDP/TCP) toiminnallisuus ja periaatteet
- Osata luotettavan kuljetuspalvelun ja vuonvalvonnan periaatteet ja toteutukset
- Osata TCP-ruuhkanhallinnan





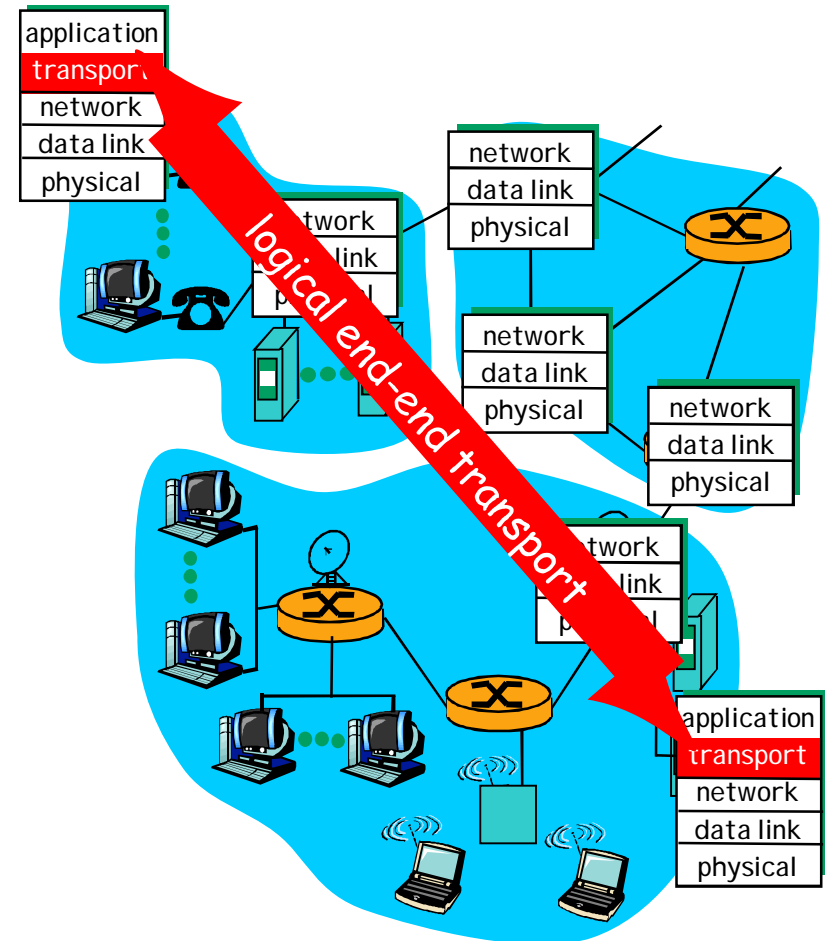
# Kuljetuskerros

## Kuljetuspalvelu

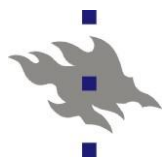


# Kuljetuskerros

- Tarjoaa kuljetuspalvelun prosessien välille
- Vain isäntäkoneissa  
Lähetys: Pilko sovelluskerroksen sanoma pienemmiksi segmenteiksi, jotka verkkokerros toimittaa perille.  
Vastaanotto: Kokoa segmentit sanomaksi, jonka sovellus lukee.
- Verkkokerros reitittää koneesta koneelle
- Segmentin koko s.e. verkkokerros pystyisi välittämään sellaisena



KuRo08: Fig 3.1



# Sovelluksen vaatimuksia

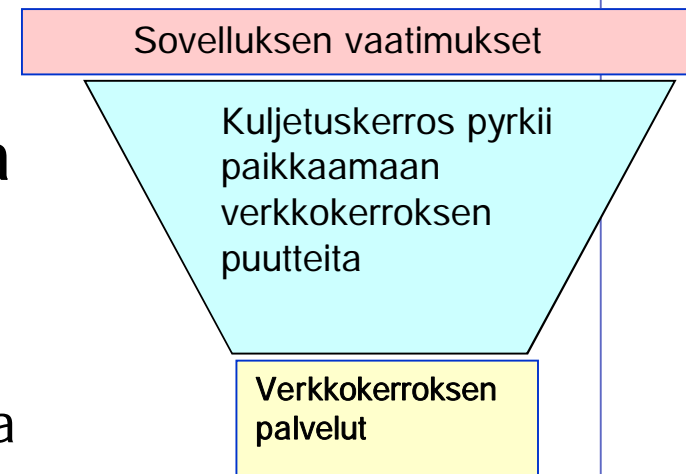
## ■ Verkkokerroksen palvelu voi

- Muuttaa segmentin bittejä tai kadottaa segmenttejä
- Toimittaa segmentit epäjärjestyksessä kuljetuskerrokselle
- Viivyttää segmenttejä satunnaisen pitkän ajan
- Luovuttaa kuljetuskerrokselle useita kopioita samasta segmentistä
- Rajoittaa segmentin kokoa

## ■ Sovellus edellyttää kuljetuspalvelulta

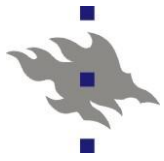
- Virheettömyyttä, luotettavuutta
- Järjestyksen säilymistä
- Kaksoiskappaleiden karsimista
- Mielivaltaisen pitkien sanomien sallimista
- Vuonvalvonnan mahdollistamista

- Kuljetuskerros peittää verkkokerroksen puutteita ja parantaa sovelluksen näkemää palvelun laatua



# Internetin kuljetusprotokollat

- **TCP: luotettava, järjestyksen säilyttävä tavujen kuljetuspalvelu**
  - **Virheenvalvonta (error control):** Huomaa ja korjaa virheet, hylkää kaksoiskappaleet
  - **Vuonvalvonta (flow control):** Älä ylikuormita vastaanottajaa
  - **Ruuhkanhallinta (congestion control):** Älä ylikuormita verkkoa
  - **Yhteyden muodostaminen ja purku**
- **UDP: Ei-luotettava, ei-järjestyksen säilyttävä sanomien kuljetuspalvelu**
  - **Välittää vain sanomia, ei pyri mitenkään parantamaan verkkokerroksen tarjoamaa palvelun laatua**
  - **Luotettavuus jää sovelluskerroksen hoidettavaksi**
- **Kumpikaan kuljetuspalvelu ei anna takuita viiveelle tai siirtonopeudelle ("best effort")**



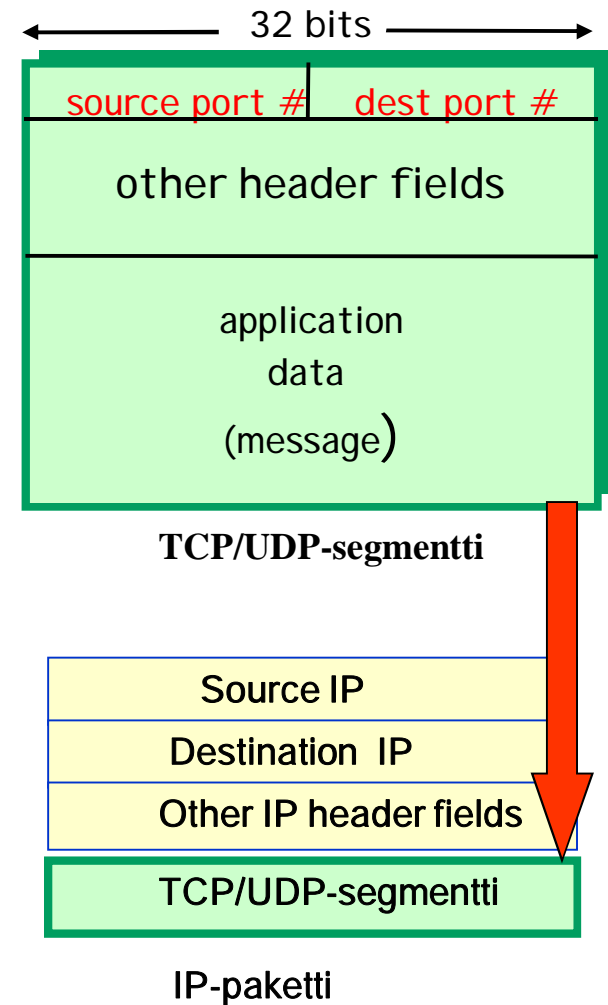
# Mikä kone /Mikä prosessi?

- Kuljetuskerros tarjoaa päästä-päähän yhteyden
  - Prosessilta prosessille ( = pistokkeesta pistokkeeseen)
  - Prosessi lukee ja kirjoittaa sanomia halutessaan
- Datana lisäksi on välitettävä osoitetietoja
  - Vastaanottajan ja lähettäjän tiedot
  - Eri koneiden prosessit voivat käyttää samaa palvelua
  - Saman koneen prosessit voivat käyttää eri palveluita
- Kuljetuskerros: mikä prosessi = mikä portti
- Verkkokerros: mikä kone = mikä IP-osoite
- Porttinumero
  - 16-bittinen: 0 – 65535
  - Portit 0 – 1024 on varattu kukin tietylle palvelulle (well known ports)
    - Esim. www-palvelulle portti 80, SMTP-postipalvelulle portti 25

# Mikä kone /Mikä prosessi?

## Lähetys (asiakas)

- **Kuljetuskerros**
  - Segmentin otsakkeessa lähde- ja kohde prosessin porttinumero
  - Antaa segmentin verkkokerroksen välitettäväksi
  - TCP: huolehtii myös luotettavuudesta
  - UDP: tarjoaa pelkän välityspalvelun
- **Verkkokerros**
  - Paketin otsakkeessa lähde- ja kohde koneen IP-osoite → reitittimet osaavat ohjata oikealle koneelle



# Mikä kone /Mikä prosessi?

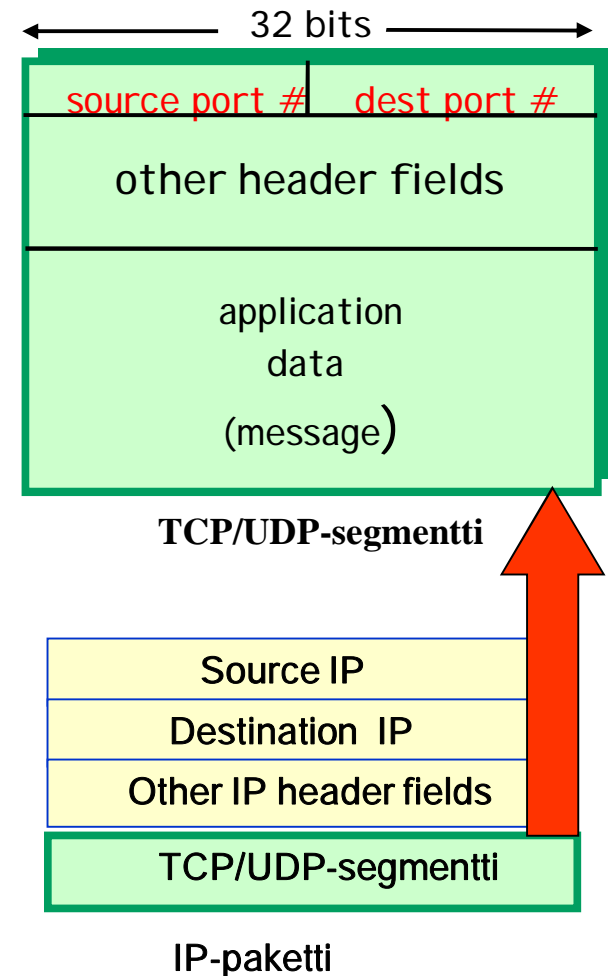
## Vastaanotto (palvelija)

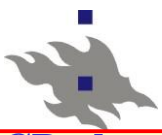
### ■ Verkkokerros

- Vastaanottaa IP-paketin
- Poistaa verkkokerroksen otsaketiedot
- Luovuttaa paketissa olleen segmentin kuljetuskerrokselle

### ■ Kuljetuskerros

- Poistaa kuljetuskerroksen otsaketiedot
- Kokoaa yhteenkuuluvat segmentit sanomiksi (tavuvirraksi)
- Ohjaa sanoman (tavuvirran) oikealle prosessille (eli oikeaan pistokkeseen) porttinumeron avulla
  - TCP: huolehtii myös luotettavuudesta
  - UDP: tarjoaa pelkän välityspalvelun





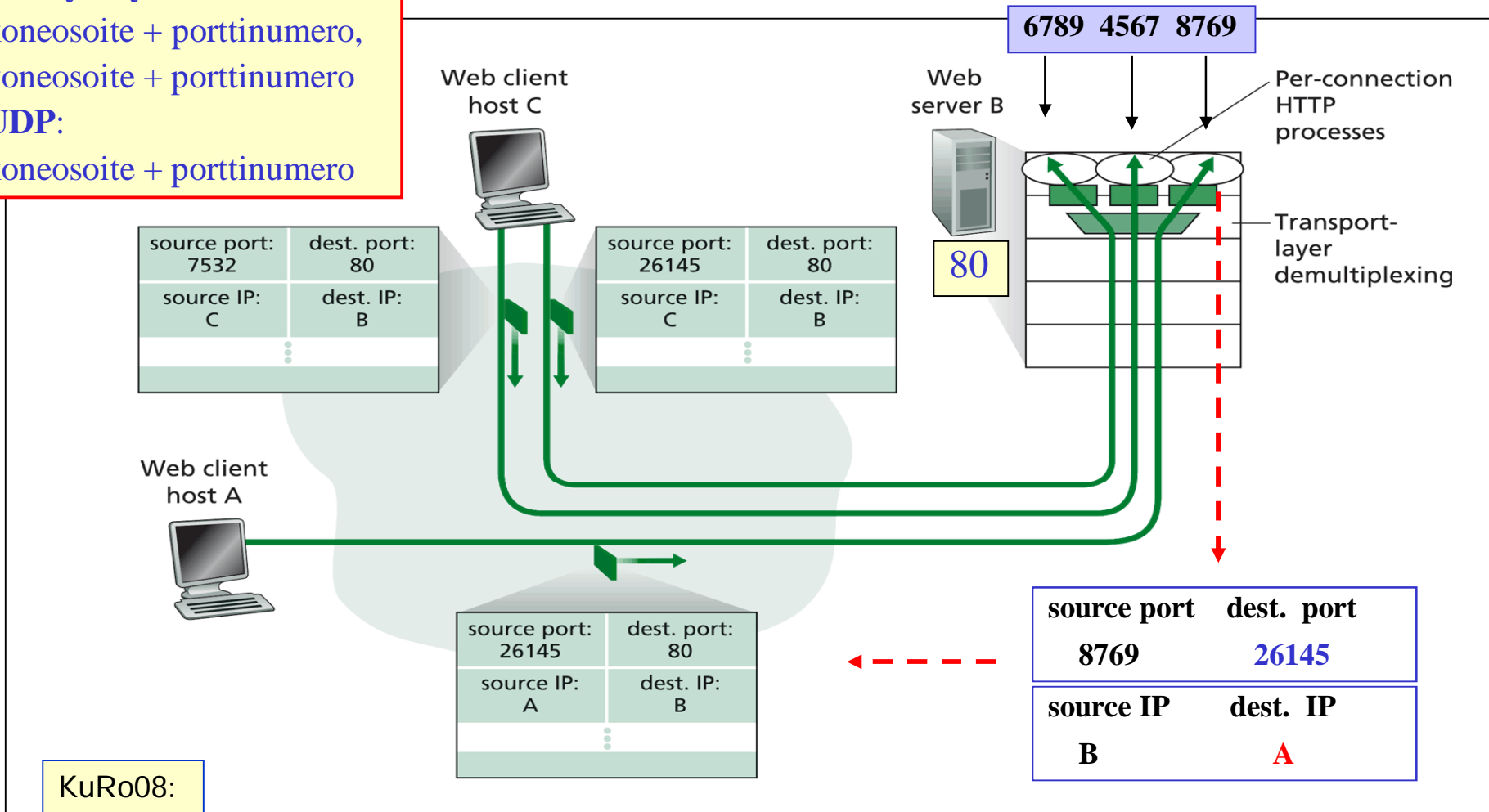
# Kaksi www-asiakasta ja palvelija

## TCP-yhteys:

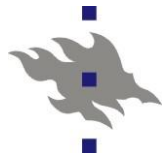
koneosoite + porttinumero,  
koneosoite + porttinumero

## UDP:

koneosoite + porttinumero



**Figure 3.5** ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application



## Yhteydetön kuljetuspalvelu UDP



# 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'?)

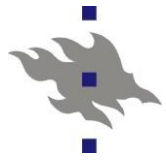
- vain minimi palvelu: mille koneelle, mihin porttiin
- voi kadottaa sanoman

## ■ Ei myöskään säilytä sanomien järjestystä

- Sovellus saa sanomat siinä järjestyksessä kuin ne tulevat perille

## ■ Vähän yleisrasitetta

- Aikaa ei kulu yhteyden muodostukseen eikä purkuun
- Ei kulu resursseja yhteyden tilatietojen ylläpitoon
- Pieni otsake (eli vähän itse protokollaan liittyviä tietoja)
- Ruuhkanhallinta ei säännöstele liikennettä



# Käyttö

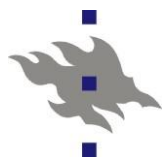
- **Vaikka UDP ei takaa luotettavuutta, se sopii silti monen sovelluksen tarpeisiin**

Alkujaan oli vain TCP, UDP luotu myöhemmin

- **Miksi UDP?**

- Pieni yleisrasite, koska pieni otsake
- Ei tilatietojen tallettamista eikä lähetys- ja vastaanottopuskureita
- Sovellus voi sietää virheitä
- Reaaliaikavaatimuksia: data lähtee heti verkkoon, koska ei yhteydenmuodostusta eikä vuon- tai ruuhkanvalvontaa

- **Tarvittava luotettavuus on räätälöitävissä sovellukseen**  
Sovellusprotokolla: oma sanomanumerointi, uudelleenlähetys



# Kuljetusprotokolla TCP

## ■ 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

segmenttinumerot, kuittaukset, uudelleenlähetykset

### Vuonvalvonta (flow control)

Lähettäjä hiljentää vauhtia, jos **vastaanottaja** ei ehdi käsitellä

### Ruuhkanvalvonta (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	<b>UDP or TCP</b>
Internet telephony	typically proprietary	<b>UDP or TCP</b>
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

Kuro08:

Miksi nämä sovellukset suosivat UDP:tä?



# UDP-segmentin rakenne

## ■ Porttinumerot

- Koska on prosessien välinen palvelu

## ■ Length

- Segmentin kokonaispituus otsake (8 B) mukaanluettuna

## ■ Checksum (optionaalinen)

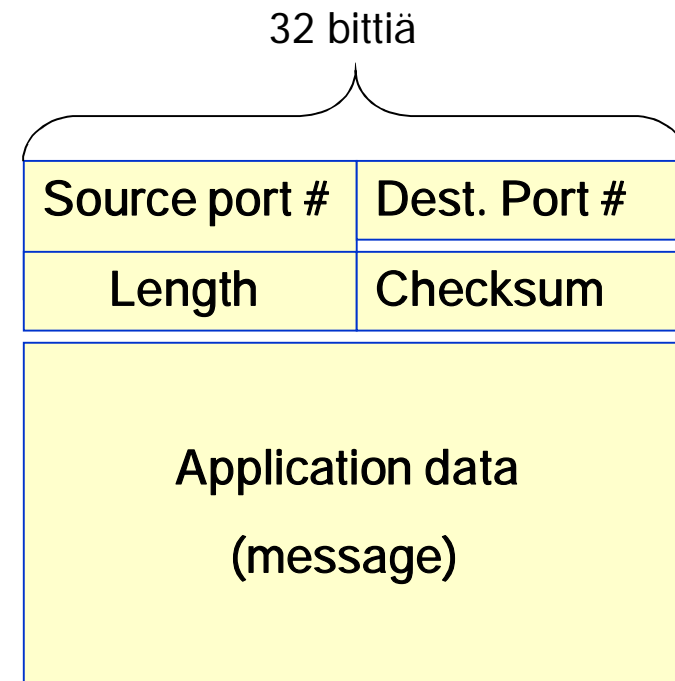
- Bittivirheen havaitsemiseen
- UDP ei yritä toipua, hävittää virheellisen segmentin

## ■ Data

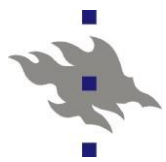
- Pitkä sanoma pilkottuna useaksi segmentiksi

## ■ IP-osoitteet vasta verkkokerroksen otsakkeessa

- Näitä tarvitaan reitityksessä

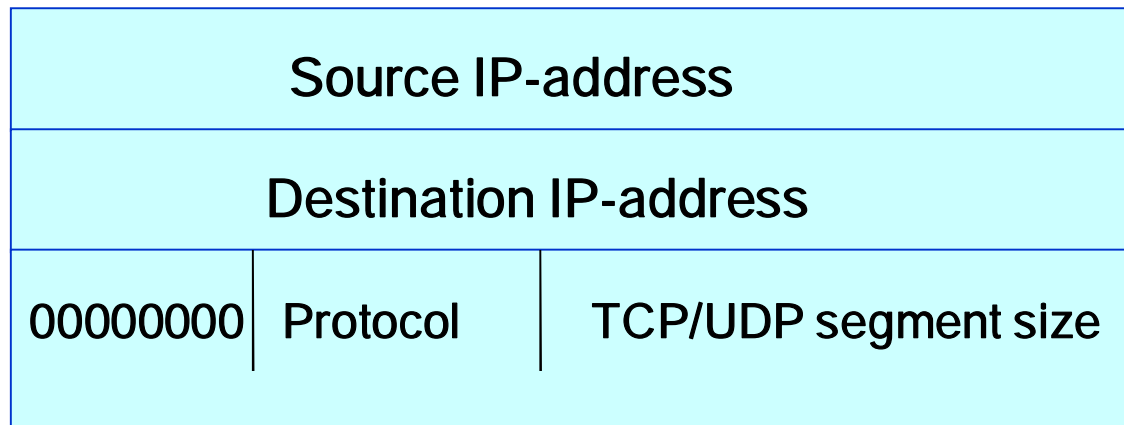


UDP-otsake



## Kuljetuskerroksen pseudo-otsake

- Käyttö vain isäntäkoneen sisäisesti. Ei lähetetä verkkoon.
  - UDP laskee tarkistussumman otsakkeelle, datalle ja ns. **pseudo-otsakkeelle**, joka sisältää IP-otsakkeen tietoja
  - Varmistus, että segmentti on tullut oikeaan koneeseen ja oikeaan porttiin



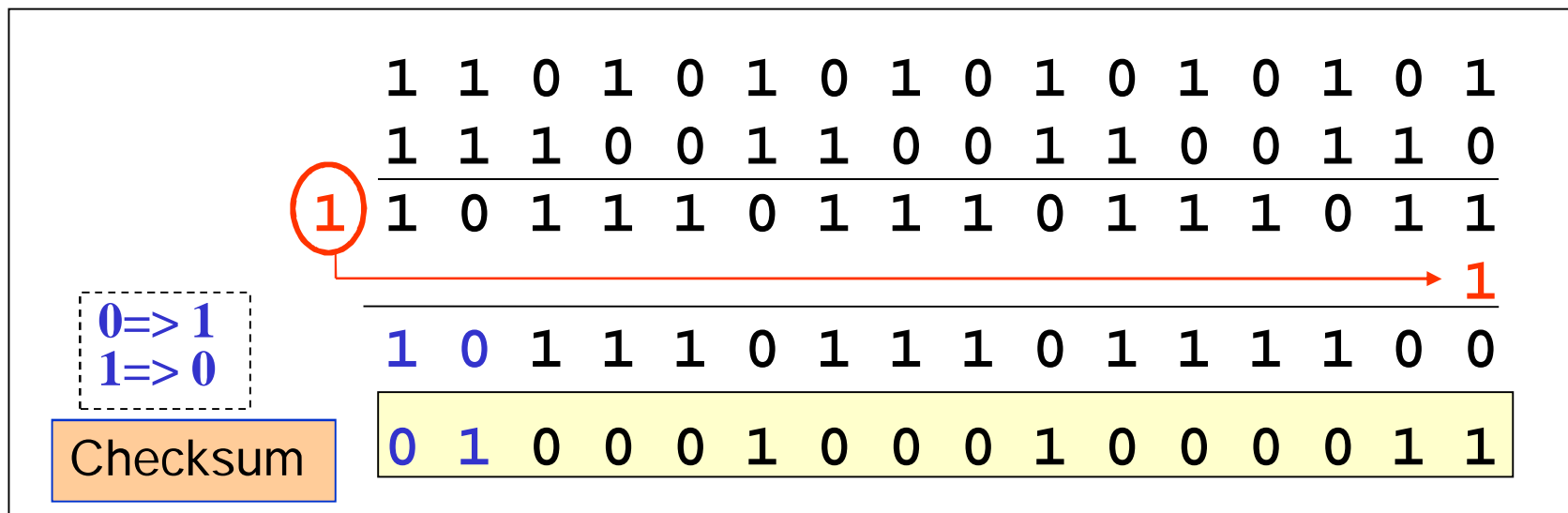
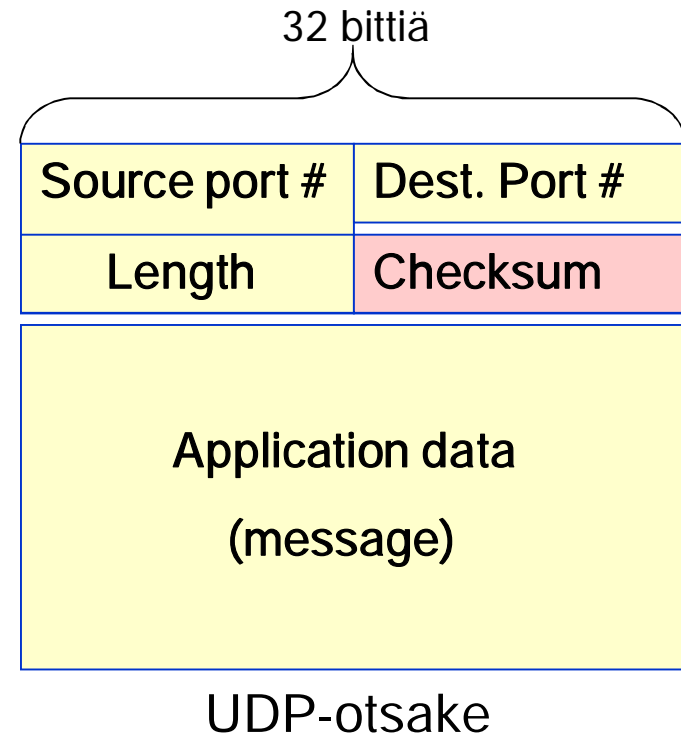
# UDP-tarkistussumma

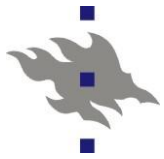
## Lähetys

- 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

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

■ Lähettäjä:

1011 0100  
0111 0101  
1000 1101  
0000 0000

checksum



=====  
1 1011 0110  
.....▶1

=====  
1011 0111



0100 1000

Yhden komplementti

vastaanottaja:

1011 0100  
0111 0101  
1000 1101  
0100 1000

=====  
1 1111 1110  
1

=====  
1111 1111

OK!

1011 0100  
1 1111 0101  
1000 1101  
0100 1000

=====  
1 0 1111 1110  
1

=====  
0 1111 1111

Virhe!



## Miksi UDP-tarkistussumma

- **Kaikki linkkikerrokset eivät suorita tarkistuksia!**
  - Ethernet huolehtii kyllä
  
- 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 tarkistusummaksi pelkkiä nolliä



## Tehtäviä:

### ■ Lähetetään 10 tavun viesti UDP:llä.

- Miten kauan kestää lähettäminen, jos lähetyksenopeus on 56 kbps?

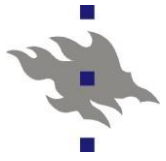
$$10 \text{ tavua} + 8 \text{ tavua} = 18 * 8 \text{ b} = 144 \text{ bittiä}$$
$$144 \text{ b} / 56\,000 \text{ b/s} = 2.57 \text{ ms}$$

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

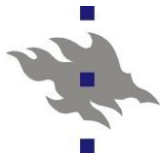
$$1000 \text{ km} / 200\,000 \text{ km/s} = 5 \text{ ms}$$

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

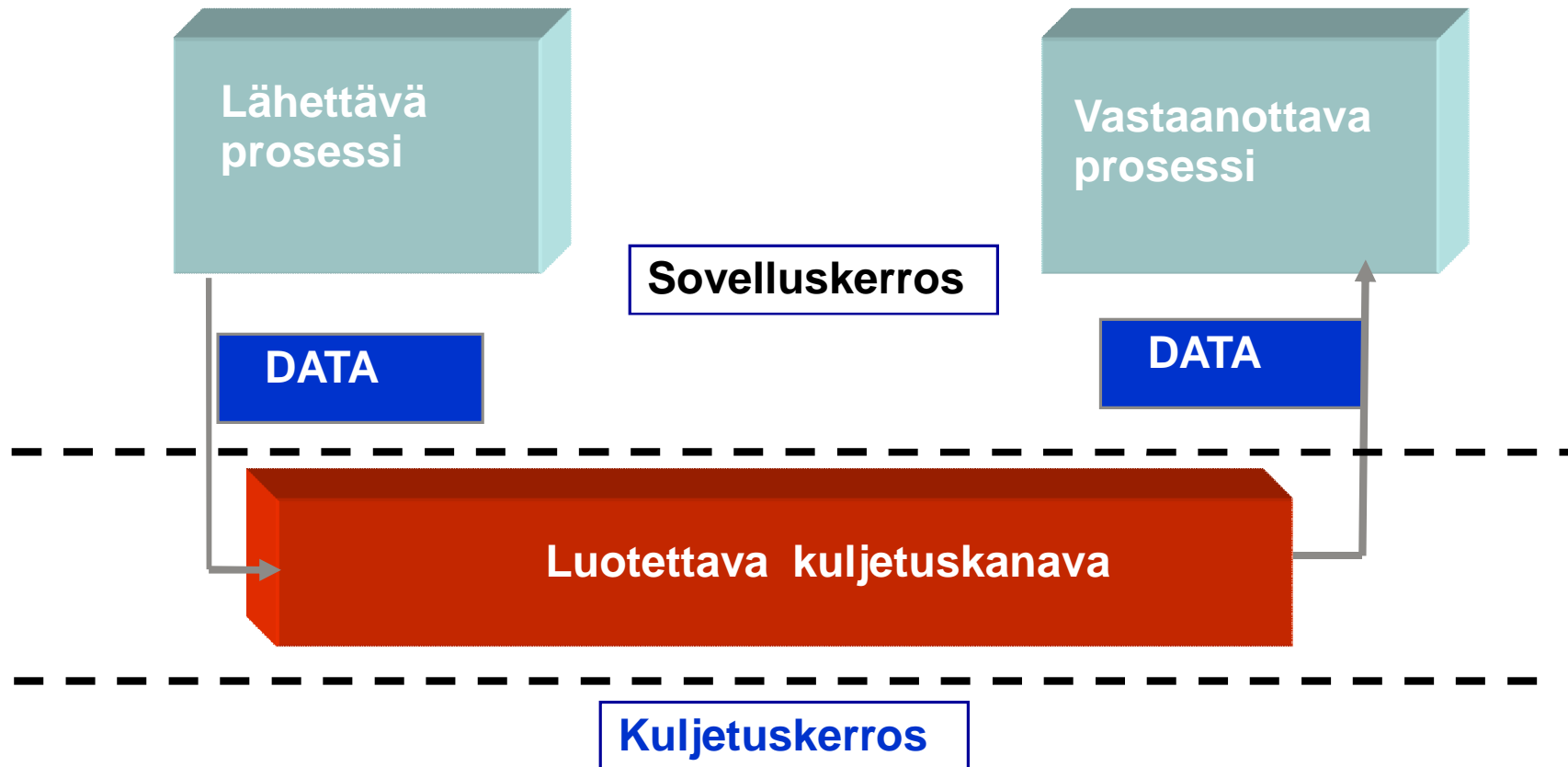
$$8/18 = 0.44 \text{ eli } 44 \%$$

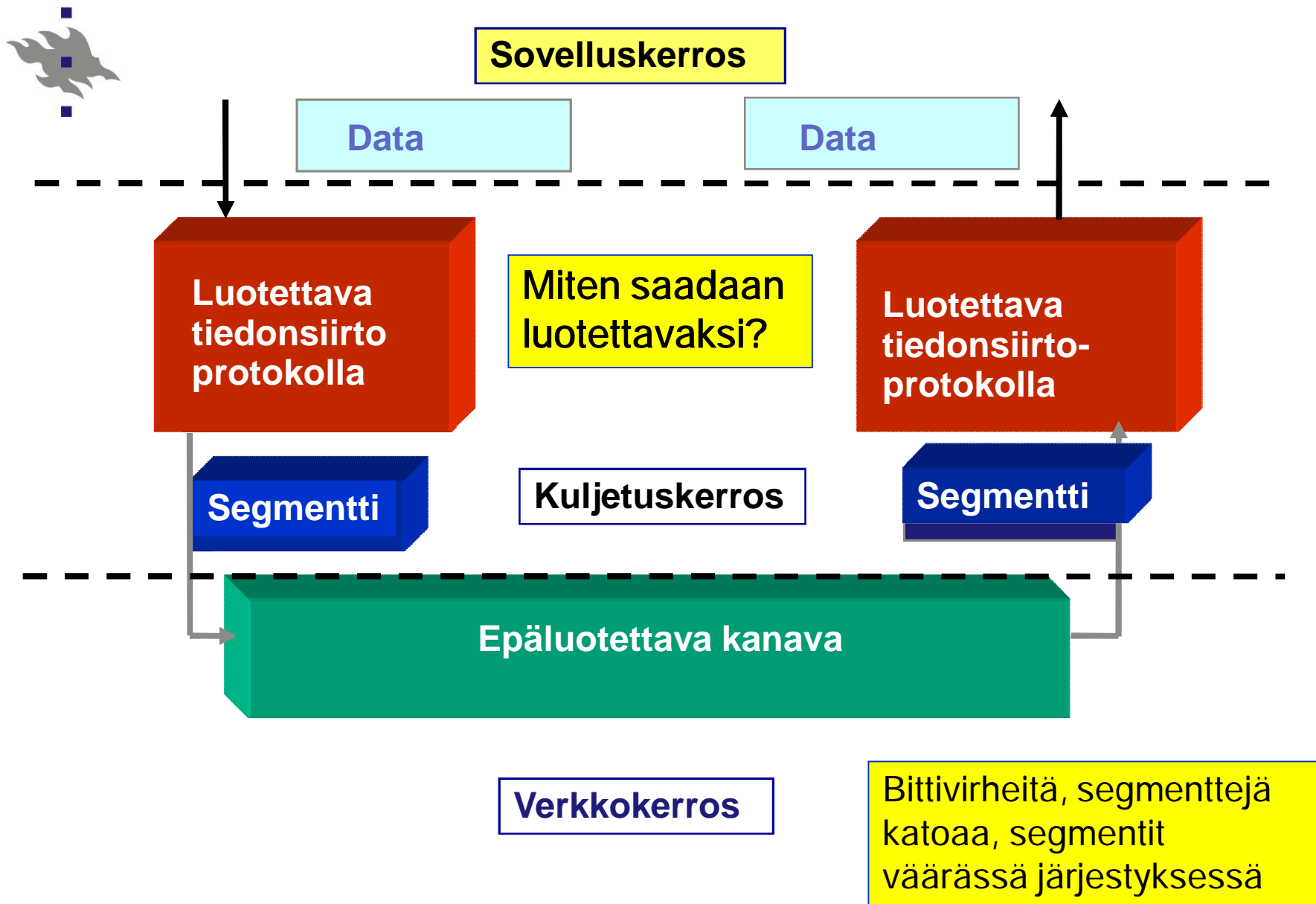


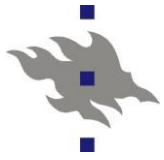
# Luotettavan kuljetuspalvelun periaatteet



# Luotettava tiedonsiirto

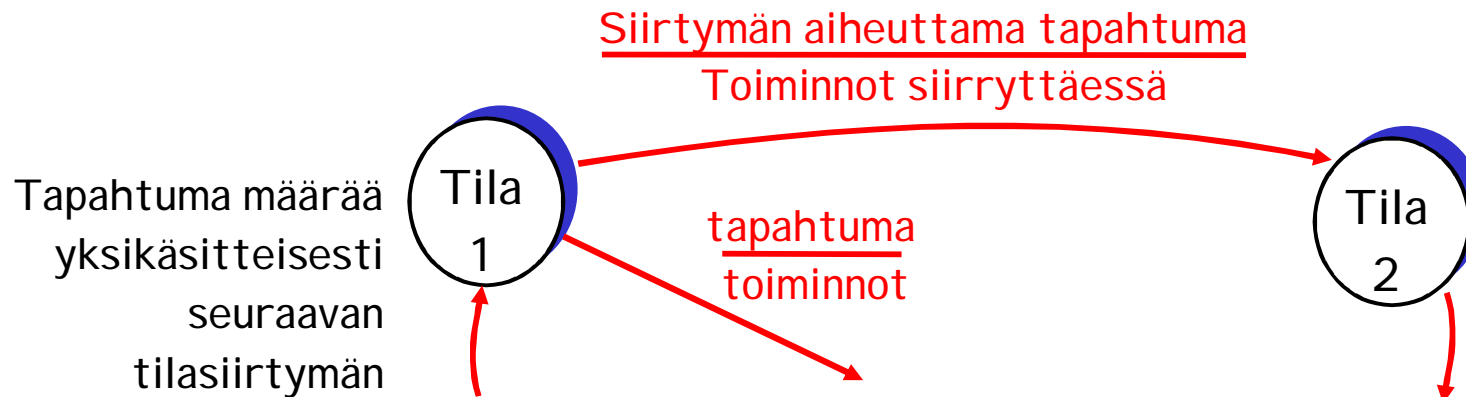






# Kuinka saada luotettavaksi?

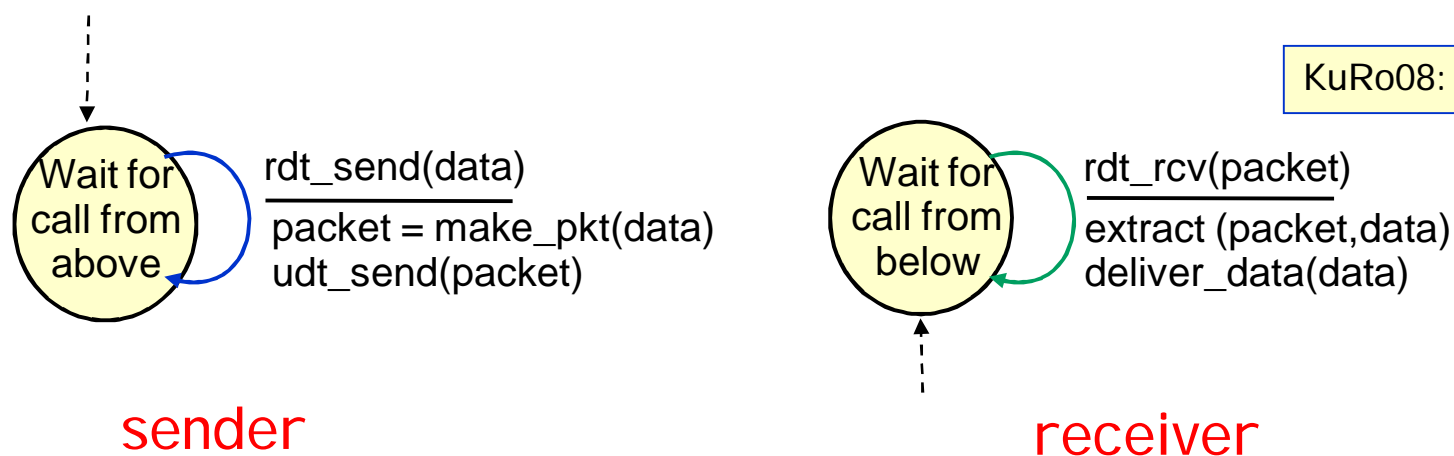
- Tarkastellaan yleisesti luotettavan tiedonsiirron ongelmia ja erilaisia ratkaisuyrityksiä
  - Edeten ideaalitalanteesta yhä ongelmaisempaan
  - Käyttäen äärellisiä tila-automaatteja lähettäjän ja vastaanottajan toiminnan kuvaamiseksi

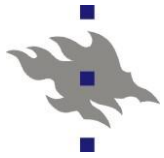




## Versio rdt1.0: Ideaalitilanne

- ❑ Oletus: **siirtokanava on täysin luotettava**
  - ❑ Ei bittivirheitä, ei katoavia paketteja, ei epäjärjestystä
- ❑ Luotettava kuljetuspalvelu =
  - ❑ Lähettäjä kirjoittaa datan kanavaan,
  - ❑ Vastaanottaja lukee datan kanavasta





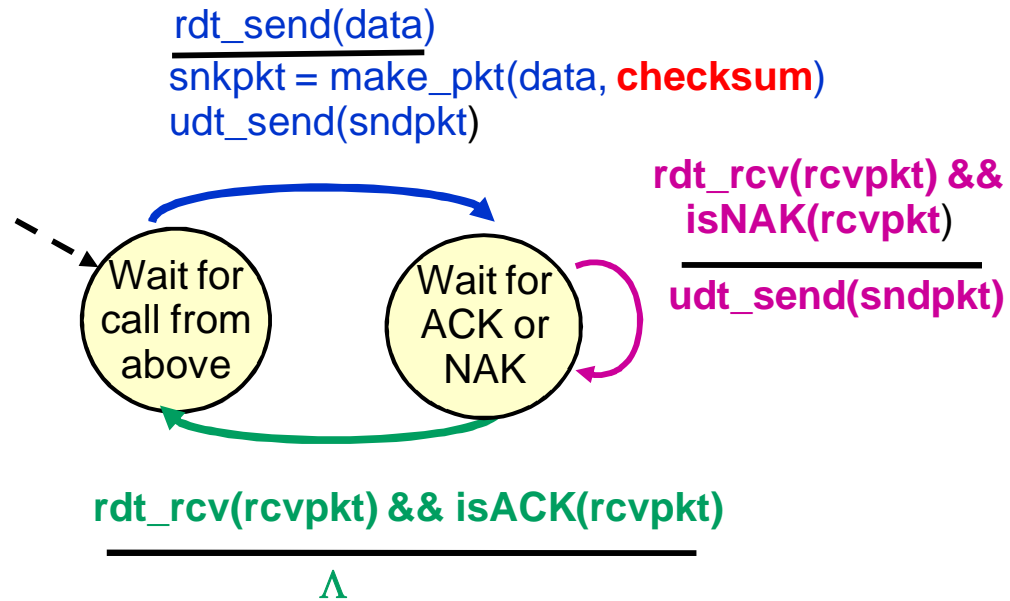
## Versio rdt2.0: Bittivirheitä

- ❑ Oletus: Voi olla **vain bittivirheitä**
  - ❑ Bitti voi kääntyä siirron aikana
  - ❑ **Siirto ei kadota paketteja**
- ❑ **Kuinka toipua?**
  - ❑ Kuittaukset: vastaanottaja kuittaa **ACK**:lla virheettömän paketin,
  - ❑ **NAK**-kuittaus, jos paketti on virheellinen + hylkää
  - ❑ Jos NAK, niin lähettäjä lähettää paketin uudelleen
- ❑ **Luotettava kuljetuspalvelu**
  - ❑ Virheen huomaaminen: **tarkistussumma**
  - ❑ Palaute vastaanottajalta: **kuittaussanoma** (ACK / NAK)
  - ❑ **Uudelleenlähetys**: dataa puskuroitava
- ❑ **Stop-and-wait-protokolla**
  - ❑ Lähettäjä odottaa kuittausta ennenkuin lähettää seuraavan

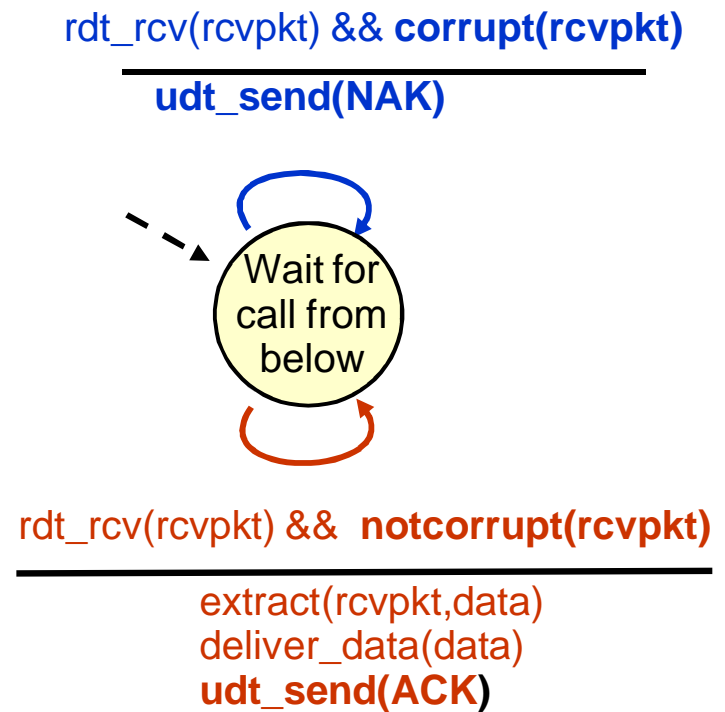


## rdt2.0:

### sender



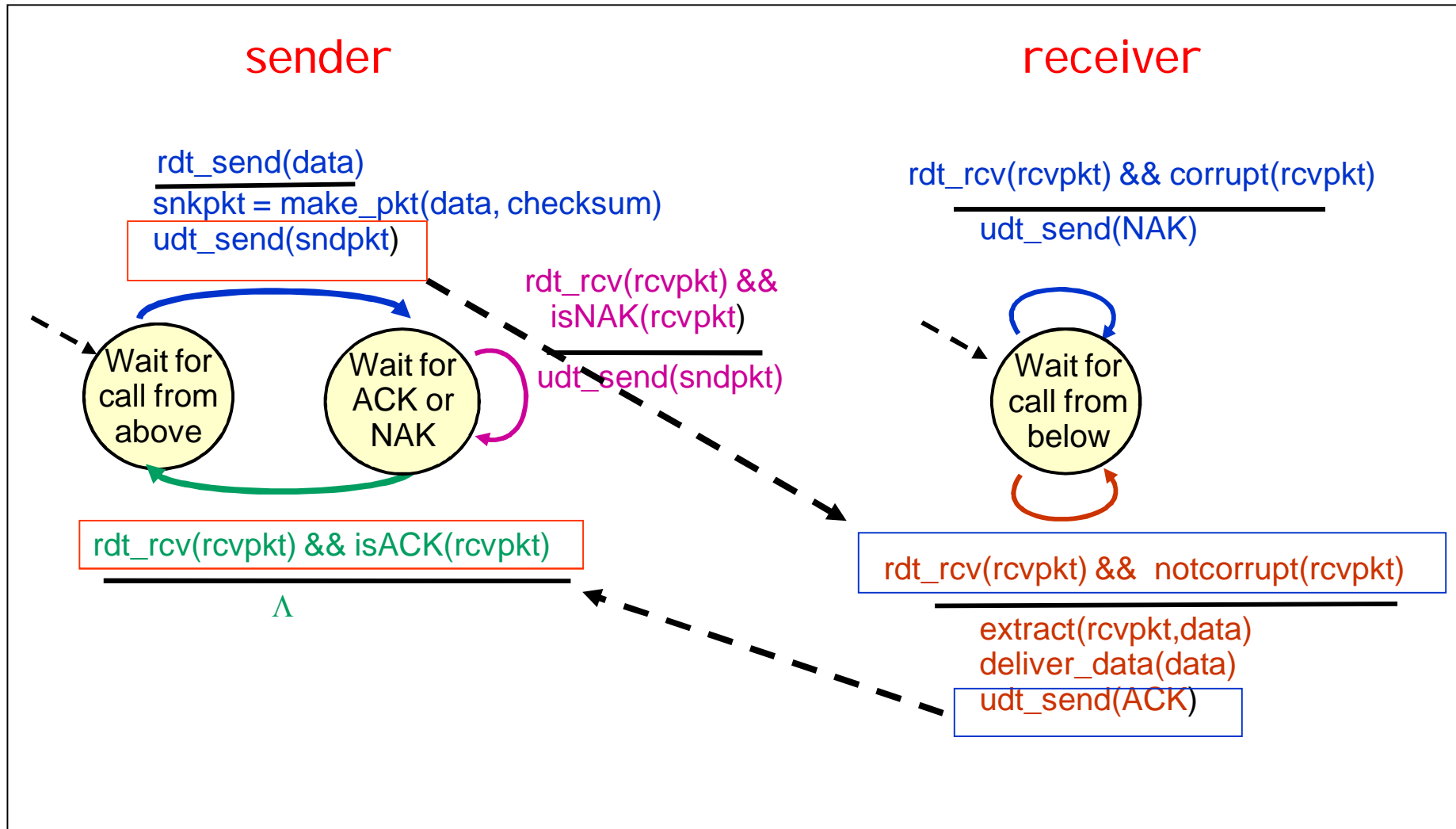
### receiver



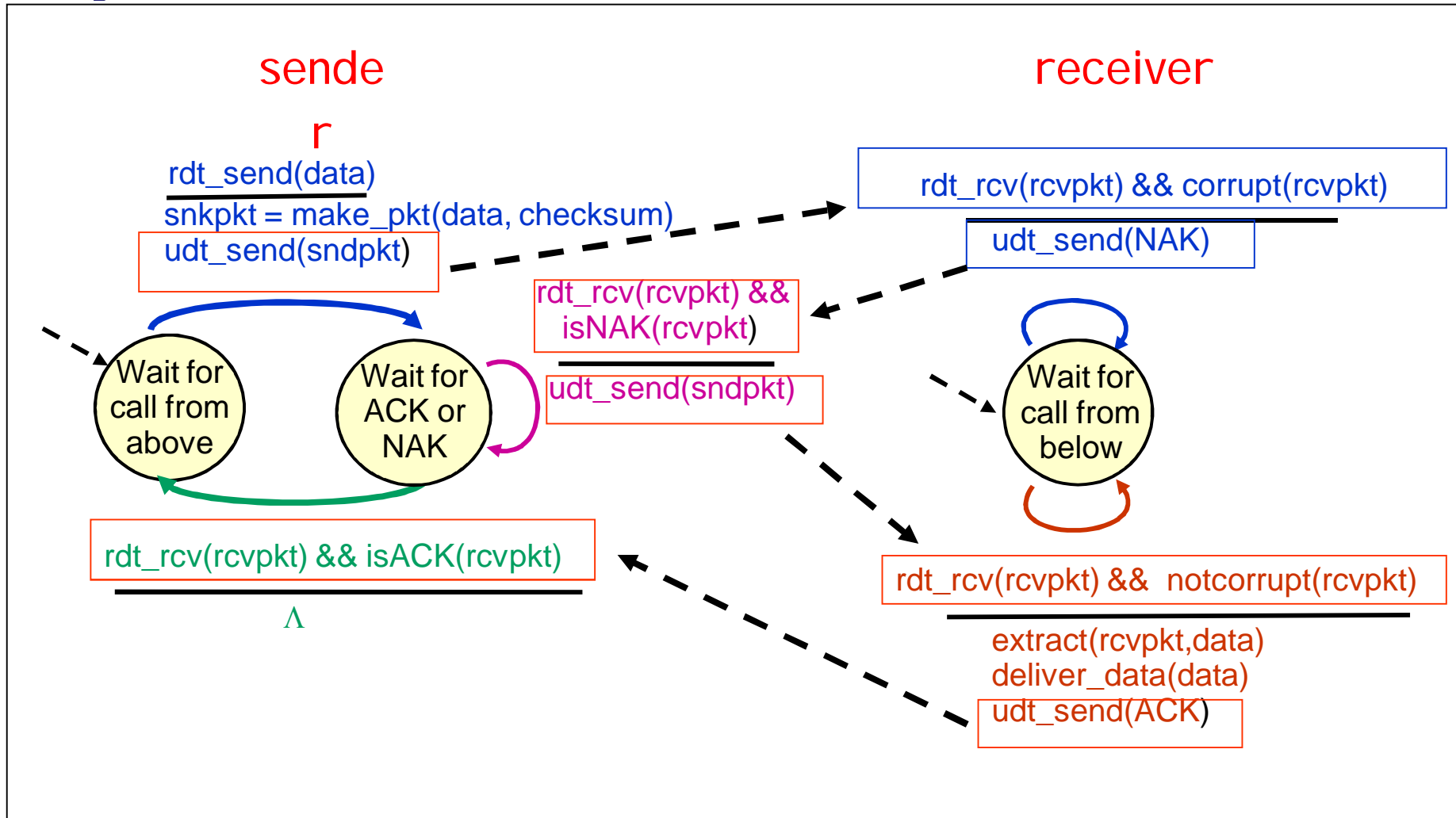
KuRo08: Fig. 3.10

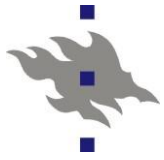


## rdt2.0: Toiminta, kun ei ole virhettä



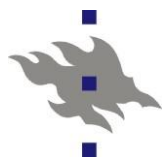
# rdt2.0: Toiminta virhetilanteessa





## rdt2.0: Missä voi mennä väärin?

- ❑ Ei toimi, jos ACK /NAK -bitit korruptoituvat!
  - ❑ Onko OK vai ei?
- ❑ Korjaus:ACK/NAK-paketteihin tarkistusbitit, jotta virhe huomataan
- ❑ Entä toipuminen?
  - ❑ Jos jos kuittauksessa virhe, uudelleenlähetetään paketti
  - ❑ Uudelleenlähetys voi tuottaa **kaksoiskappaleen, jotka on huomattava ja hylättävä**
- ❑ => **Paketteihin järjestysnumero**
  - ❑ Kaksoiskappale: jos sama numero
  - ❑ Vastaanottaja kuittaa normaalisti, mutta ei anna sovellukselle



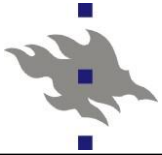
## Versio rdt2.1: enemmän bittivirheitä

### Lähettäjä:

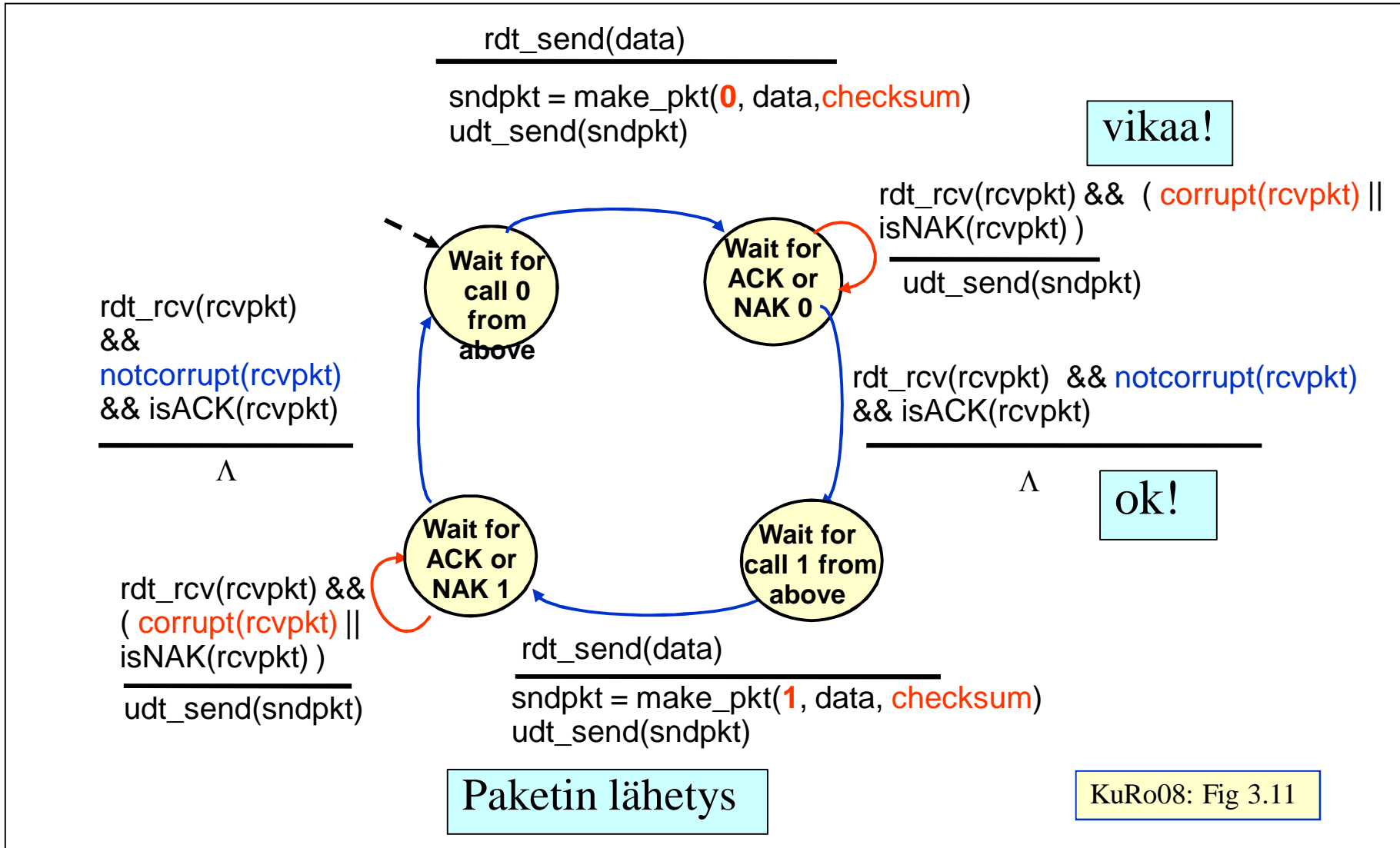
- Lisää pakettiin järjestysnumeron  
(numerot 0 ja 1 riittävät tässä protokollassa. Miksi?)  
Ns. vuorottelevan bitin protokolla
- Tarkista, että ACK/NAK ei ole korruptoitunut  
Tilakaaviossa nyt kaksinkertaisesti tiloja  
Kaavion tilan 'muistettava' paketin numero
- Säilytä kopio lähetetystä paketista

### Vastaanottaja:

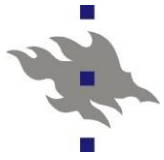
- Tarkista, ettei ole kaksoiskappale
  - Kaavion tilan 'muistettava' seuraavan paketin numero: 0 vai 1
- Myös duplikaatti kuitattava, koska ei tiedä menikö edellinen kuittaus perille



# rdt2.1: Lähettäjän tilakaavio



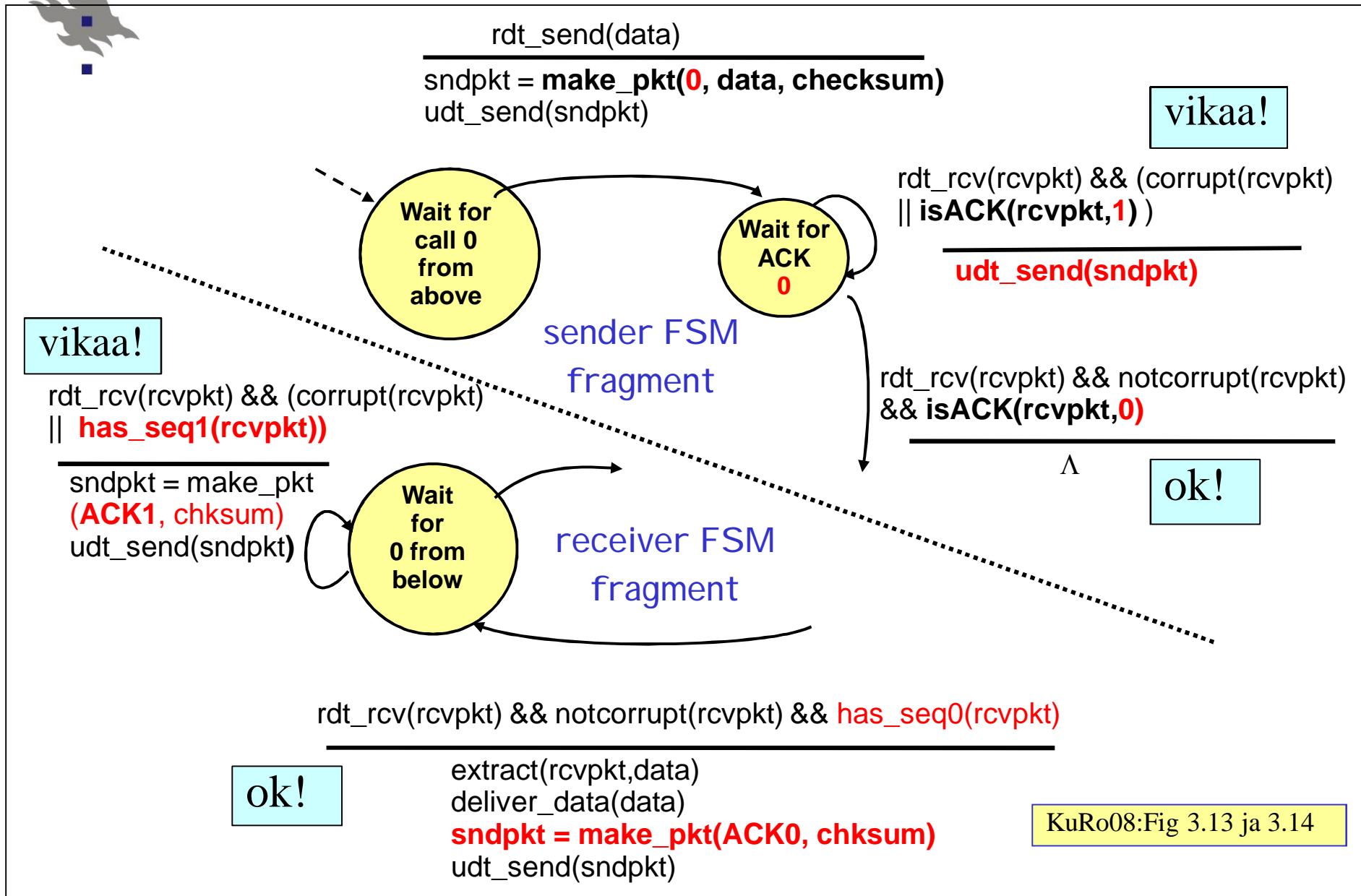


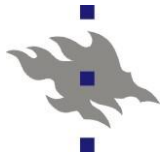


## Versio rdt2.2: vain ACK-kuittaus käytössä

- ❑ Sama toiminnallisuus kuin edellä
- ❑ Käyttää vain ACK-kuittauksia
  - ❑ Vastaanottaja kuittaa viimeksi kunnossa saamansa paketin
  - ❑ Kuittaukseen on liitettävä kuitattavan paketin numero
- ❑ Jos samalle paketille (nro X) tulee useita ACK-kuittauksia (*duplicate ACK*), niin sitä seuraava paketti (nro X+1) joko puuttuu tai on virheellinen
  - ❑ ~NAK-kuittaus
  - ❑ Lähetetään uudelleen sanoma X+1

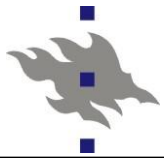
# rdt2.2



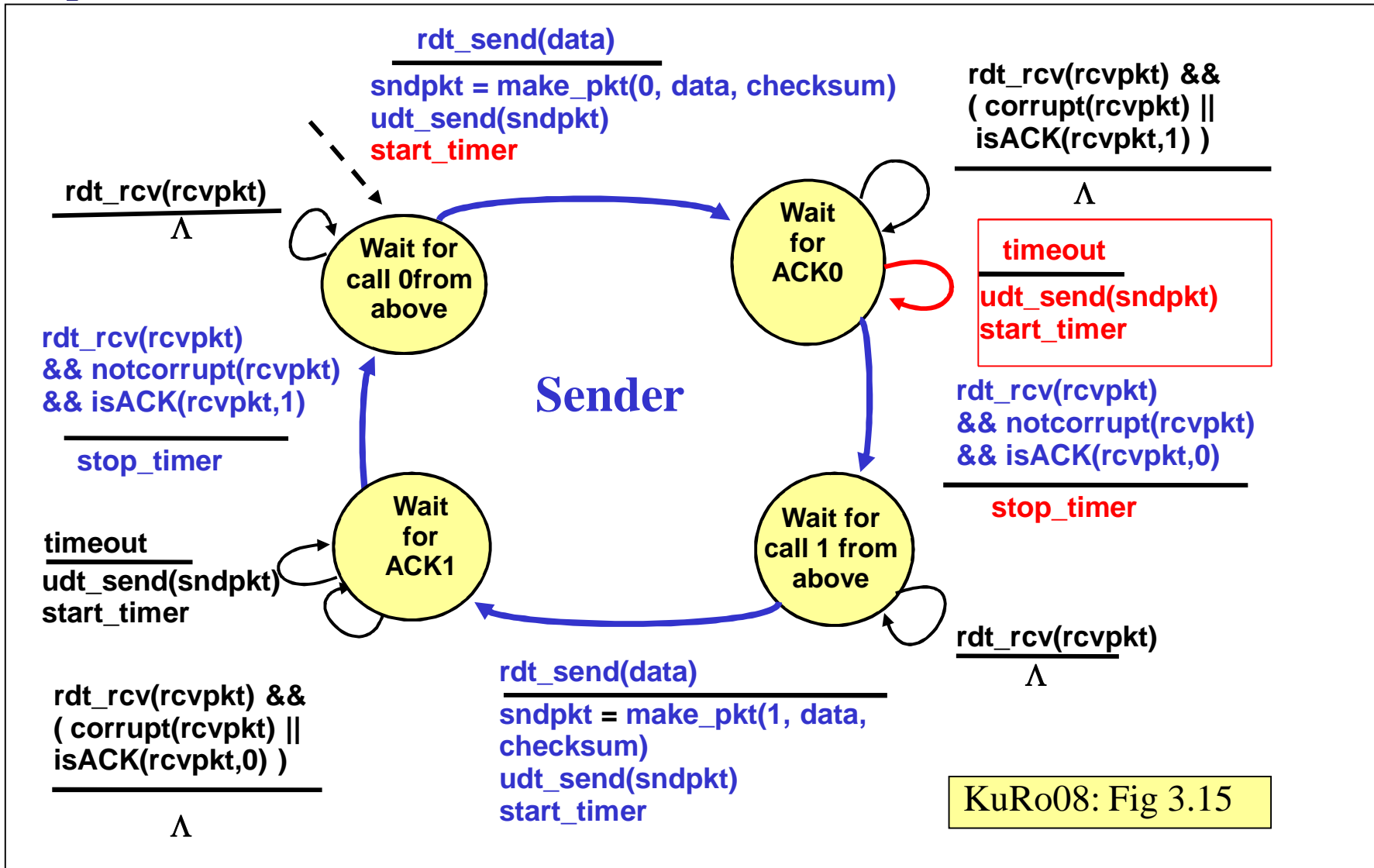


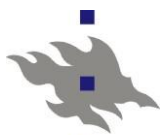
## Versio rdt3.0: paketteja voi kadota!!

- Oletus: Siirtokanava voi kadottaa paketteja**  
Sekä datapaketteja että kuittauspaketteja voi kadota.  
Tarkistussumma, pakettinumero, ACK eivät vielä riitä! **Miksi?**
- Lähettäjä odottaa jonkin aikaa kuittausta  
Jos ei saavu, lähettää paketin uudelleen  
**Ajastin** laukaisee uudelleenlähetyksen
- Jos paketti (data / kuittaus) kuitenkin vain viivästyy eikä olekaan kadonnut  
Syntyy duplikaatti, joka havaitaan sanomanumeroinnin avulla  
Kuittauksessa mukana kuitatun paketin numero

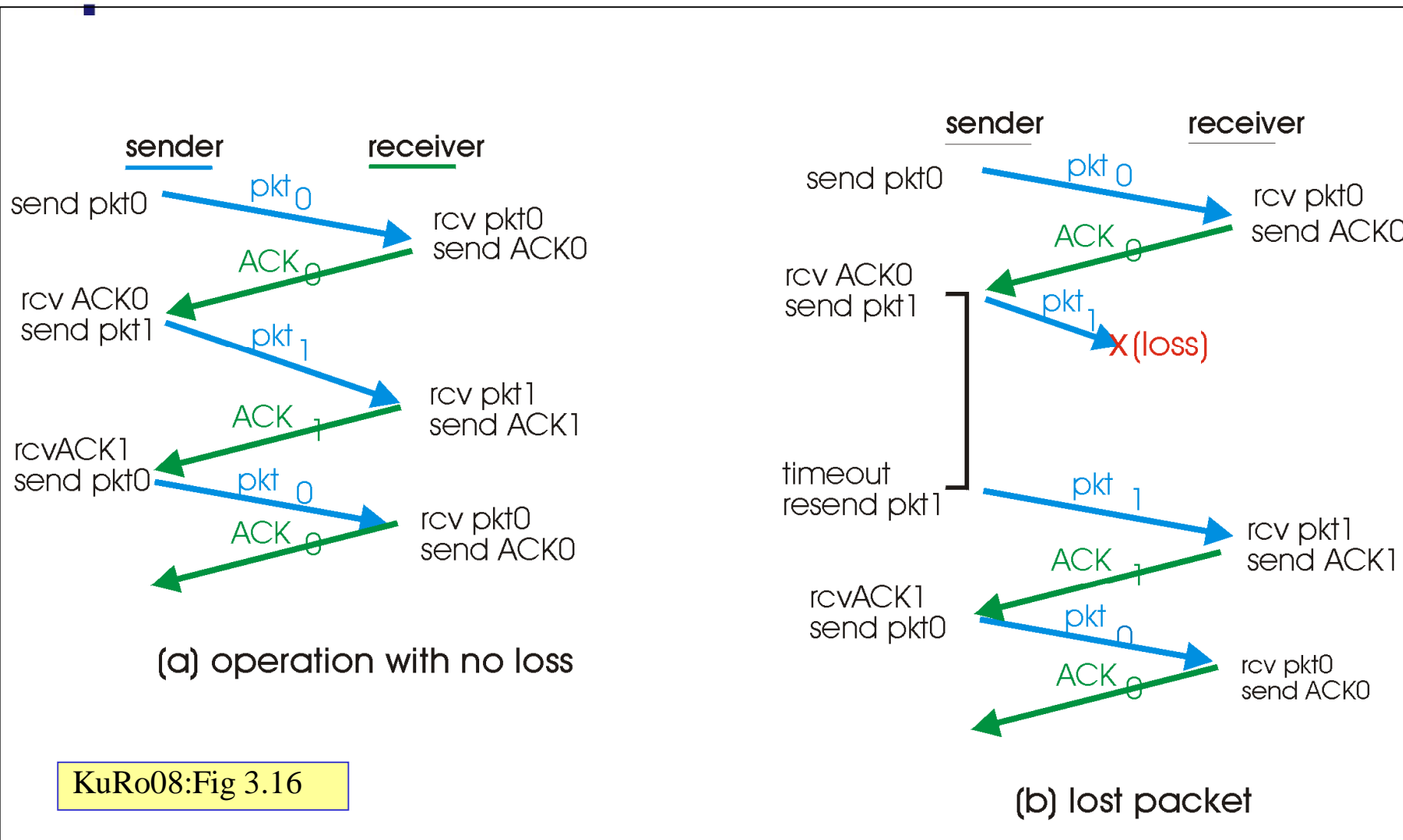


# rdt3.0





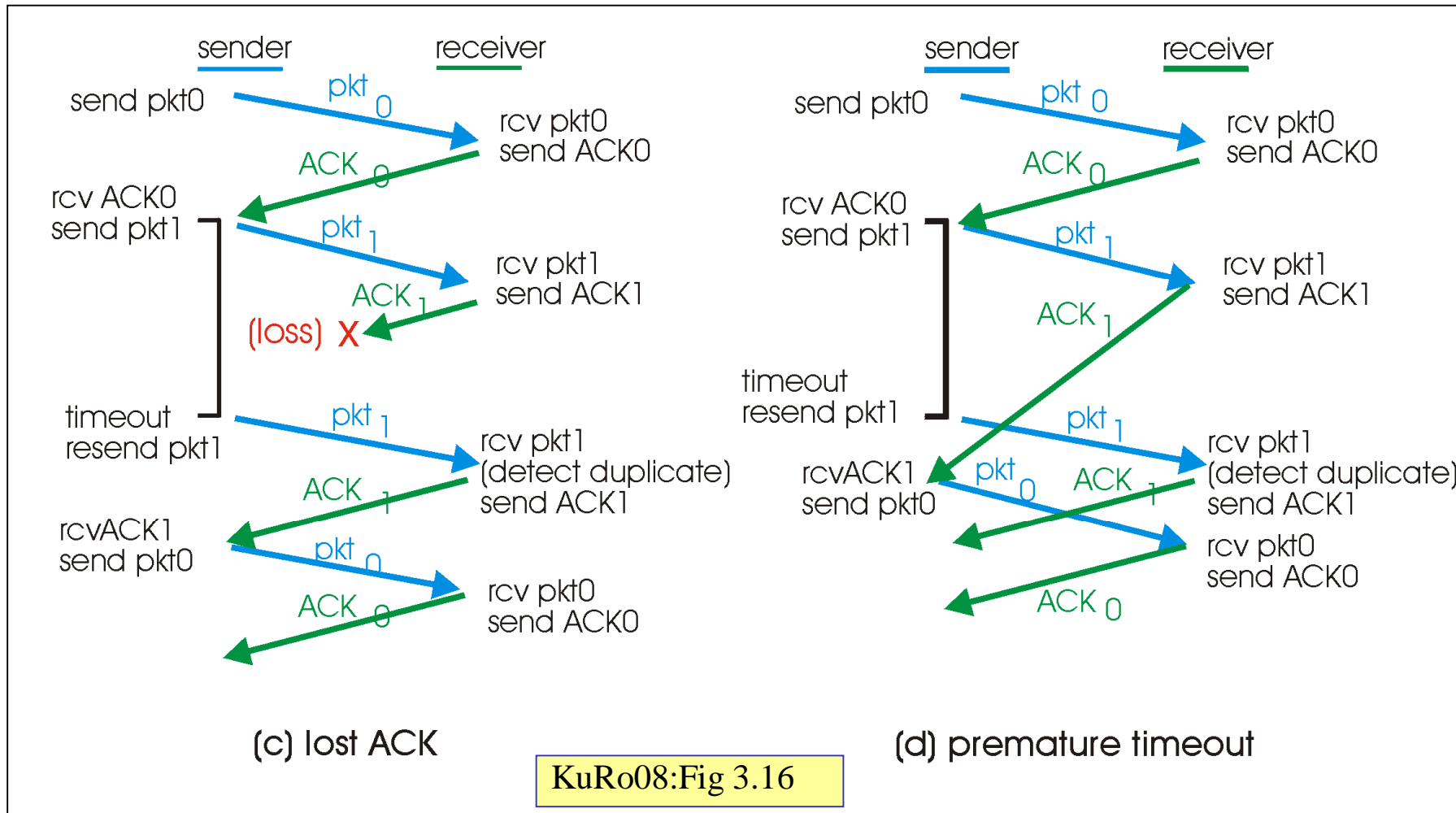
# rdt3.0 toiminnassa

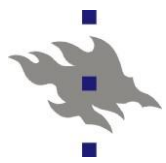


KuRo08:Fig 3.16



# rdt3.0 toiminnassa





## rdt3.0: Tehokkuus?

Esim: 1 Gbps linkki, 15 ms päästä-päähän etenemisviive eli  
RTT = 30 ms, 1 KB:n paketti

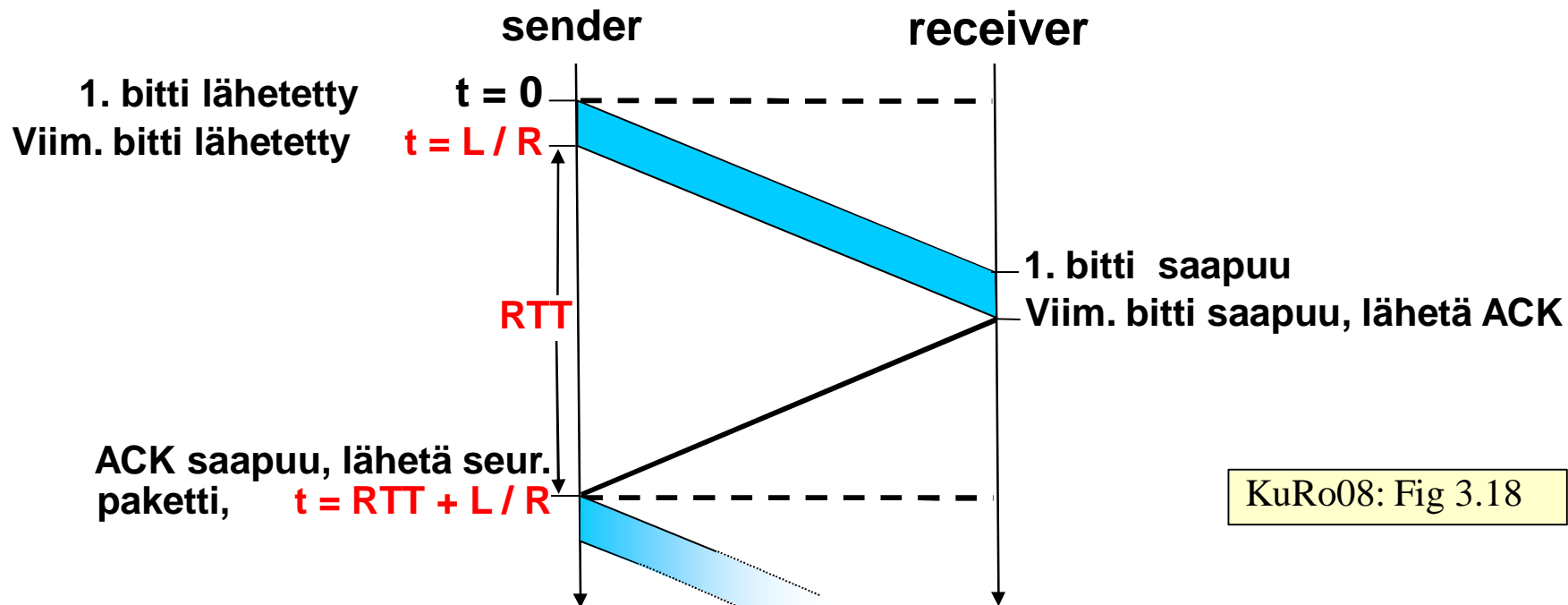
$$T_{\text{transmit}} = \frac{L \text{ (packet length in bits)}}{R \text{ (transmission rate, bps)}} = \frac{8\text{kb/pkt}}{10^{**}9 \text{ b/sec}} = 8 \text{ microsec}$$
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

**Käyttöaste** (utilization): se osa kokonaisajasta, jolloin lähettäjä lähettää

- 1KB paketti 30 ms:n välein -> 33kB/s nopeus 1 Gbps linkillä.
- Stop-and-wait-protokolla rajoittaa,  
ei linkin kyky siirtää dataa (linkin siirtonopeus)

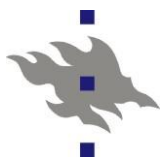


## rdt3.0: stop-and-wait tehokkuus



KuRo08: Fig 3.18

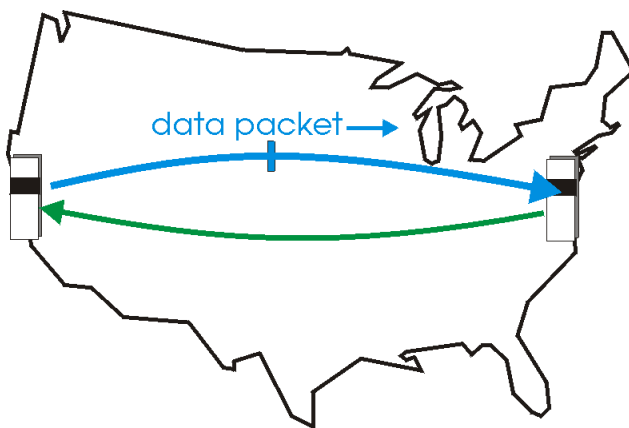
$$U_{\text{sender}} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$



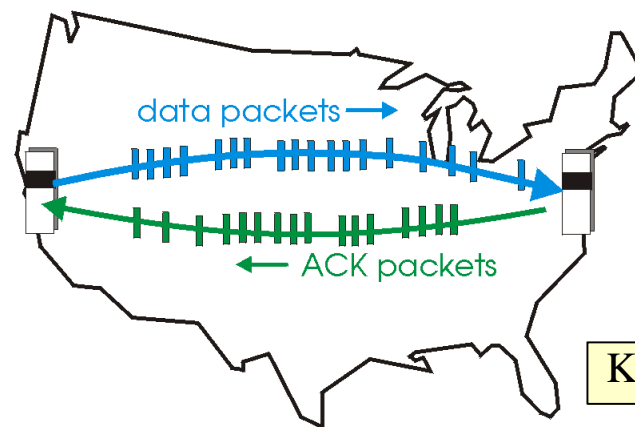
## rdtX.X: Liukuhihnaprotokollat

Lähettäjä saa lähettää useita paketteja, vaikka ei ole saanut kuittauksia edeltäviin

- Numerointi (0,1) ei enää riitä, lisää numeroita tarvitaan
- Tarvitaan puskurointia molemmissa päissä

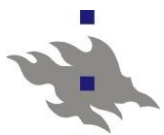


(a) a stop-and-wait protocol in operation

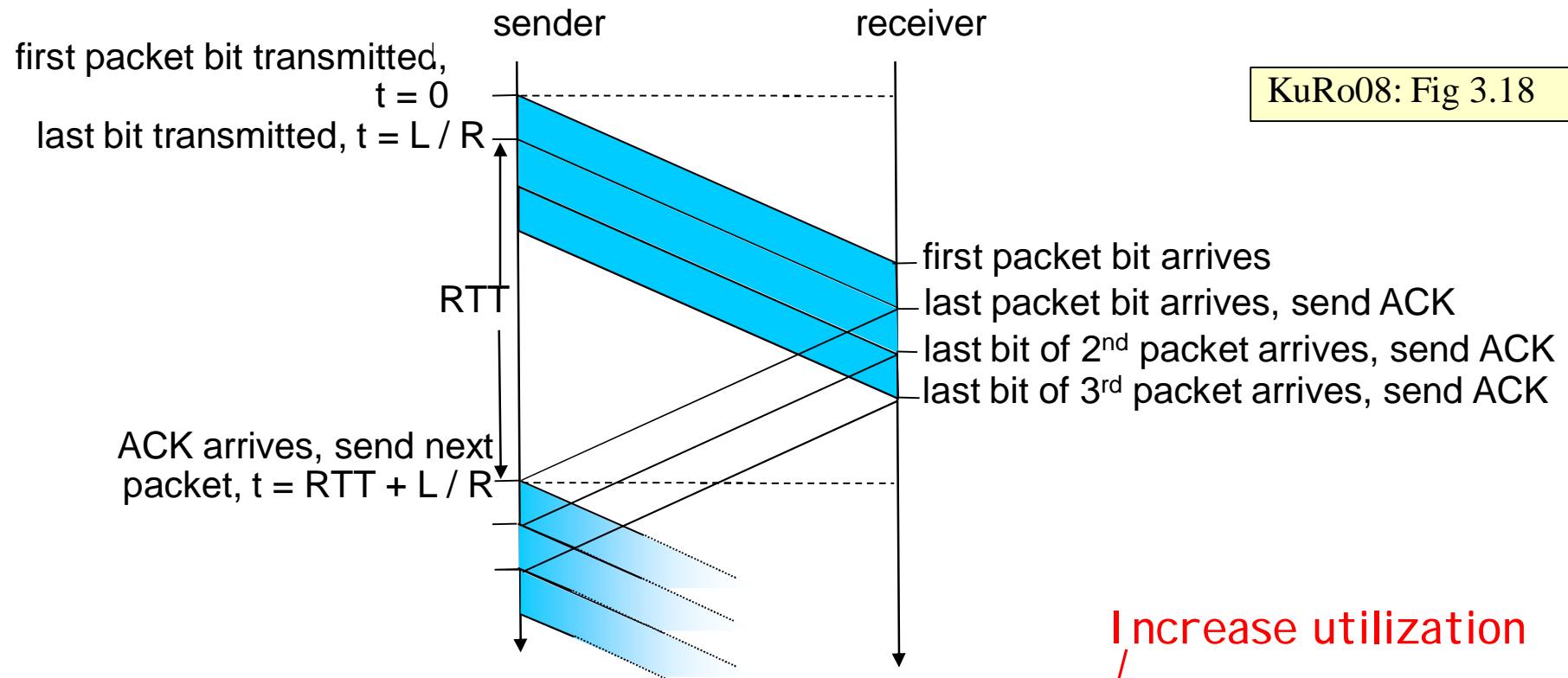


(b) a pipelined protocol in operation

KuRo08: Fig 3.17



# Liukuhhnoitus: käyttöasteen kasvattaminen

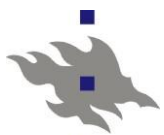


KuRo08: Fig 3.18

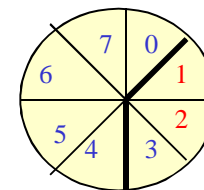
Increase utilization  
by a factor of 3!

$$U_{\text{sender}} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

(0.00028)

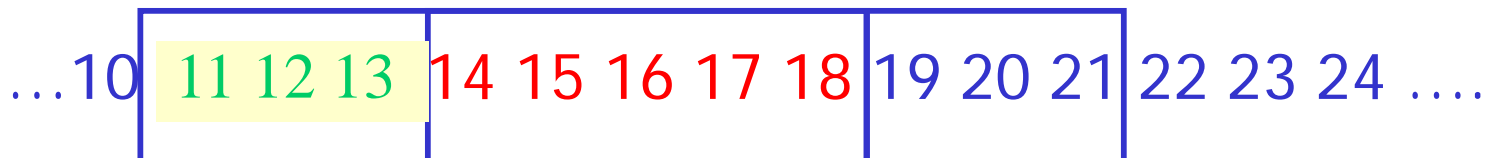


# Liukuva ikkuna (sliding window)



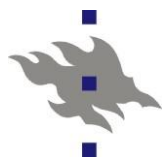
## ■ Säätää pakettien lähettämistä ja vastaanottoa

- Kertoo millä pakettinumeroilla on lähetetty/vastaanotettu, mistä saatu/lähetetty kuittaukset ja millä numeroilla voi vielä lähettää/vastaanottaa paketteja
- Ikkunan koko riippuu yhteyden tyypistä ja puskurien koosta



## ■ Lähetysikkuna (sender window)

- Ikkunan koko = montako pakettia saa olla kuittaamatta
- Mitkä pakettinumerot on käytetty, mutta kuittaamatta
- Mitä pakettinumeroita voi vielä käyttää
- Lähettäjän on odotettava, jos kaikki ikkunan numerot on käytetty
- Kun kuittaus saapuu, ikkuna liukuu
  - Seuraavat numerot tulevat luvallisiksi



## Liukuva ikkuna (2)

- **Vastaanottoikkuna** (receiver window)
  - Mitkä pakettinumerot otettu vastaan, mutta kuittaamatta
  - Mitä pakettinumeroita lähettäjä saa vielä käyttää eli mitkä hyväksytään
  
- Jos saadussa paketissa on ikkunan viimeinen numero
  - Ikkuna pysäyttää pakettien lähetyksen vastapäätä
  - Ikkuna estää uusien pakettien vastaanoton
  
- Paketin kuittaus liu'uttaa myös vastaanottajan ikkunaa
  - Hyväksytään uusia pakettinumeroita



## Kun ikkunan koko on 1

- Vain yksi paketti kuittaamattomana
  - One Bit Sliding Window –protokolla
  - = stop-and-wait –protokolla
- Pakettinumerot 0 ja 1 riittävät
  
- ACK ilmoittaa
  - Joko seuraavaksi odotetun paketin numeron (**esim.TCP**)
  - Tai viimeksi vastaanotetun virheettömän paketin numeron
  
- ACK sisältää paketin numeron
  - Kuittausduplikaatti ei voi kuitata väärää paketteja



# Virhetilanteen käsittely

- Entä, kun huomataan virhe?
  - Monta muuta pakettia jo matkalla!
  
- Pakettiin ei tule kuittausta
  - Paketti katosi tai virheellinen
  - Kuittaus katosi tai virheellinen
  - => Ajastin laukeaa aikanaan
  
- "Go-Back-N" (paluu N:ään)
  - Paketit uudelleenlähetetään virheellisestä lähtien
  
- Selective Repeat (Valikoiva toisto)
  - Lähetetään vain virheelliset paketit



## Go-Back-N

- Vastaanottaja **hyväksyy paketit vain järjestyksessä**
  - Kuittaa järjestyksessä tulleen virheettömän paketin
  - Hylkää kaikki puuttuvan tai virheellisen paketin jälkeiset paketit eikä lähetä niistä kuittauksia
  
- Kun lähettäjä ei saa pakettiin kuittausta
  - Lähetysikkuna täyttyy ja estää uusien pakettien lähettämisen
  - Lähettäjän ajastimet laukeavat
  - Lähettäjä lähettää uudestaan kaikki viimeisen kuittauksen jälkeiset paketit
  - Näiden kuittaukset siirtävät taas lähetysikkunaa
  
- **Tehoton, jos paljon virheitä ja iso ikkkuna**



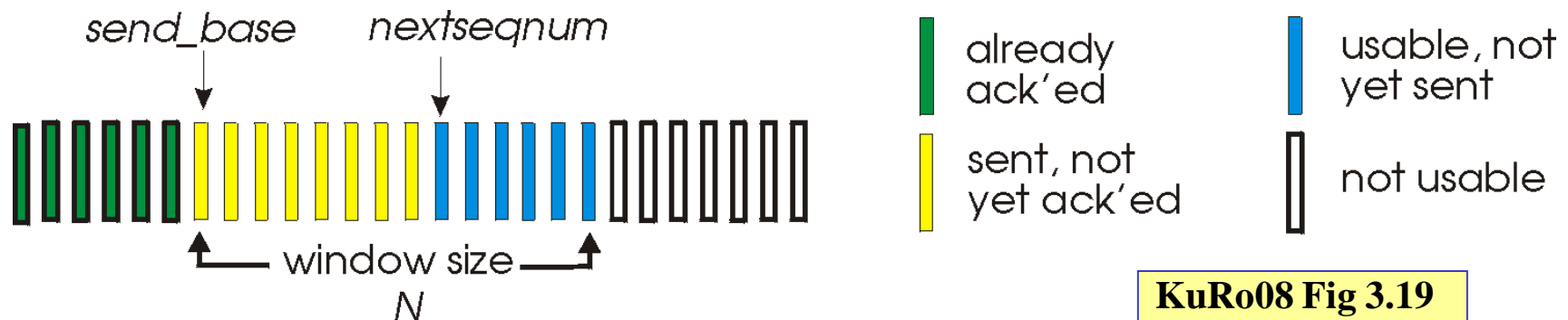
# Go-Back-N

## ■ Kumulatiivinen ACK

- Lähetä ACK, jossa korkein järjestyksessä saadun kelvollisen paketin numero
- Tämä kuittaa kaikki pienemmällä numerolla lähetetyt paketit

## ■ Jos välistä puuttuu paketti

- Lähetä uudestaan ACK, jossa korkein järjestyksessä saadun paketin numero (~ NAK)
- **Tuplakuittaus** (duplicate ACK)
- Parannus: => nopeampi reagointi puuttuvaan

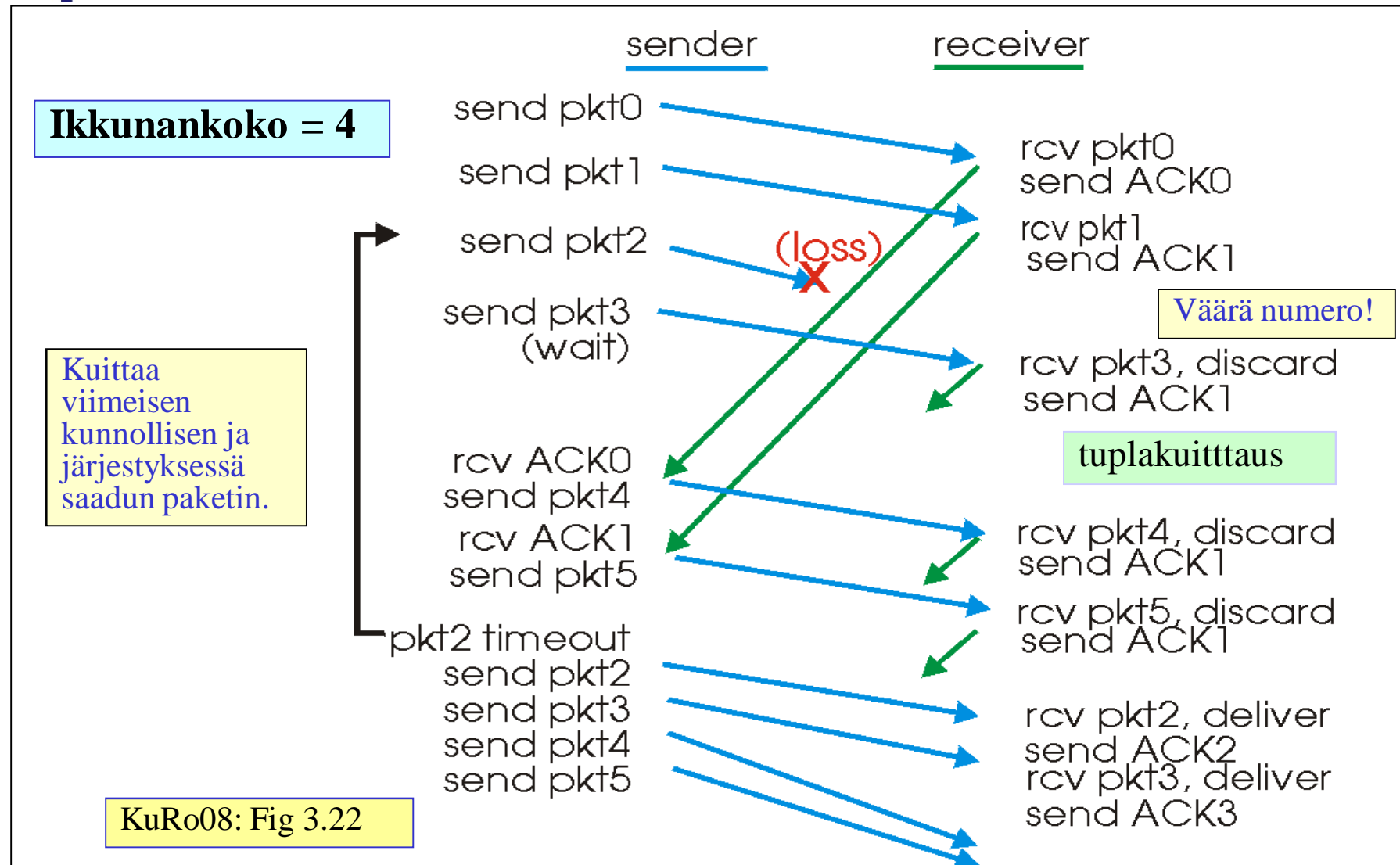


**KuRo08 Fig 3.19**



# Go-Back-N: Esimerkki

Kumulatiivinen kuittaus



# Go-Back-N: lähettäjän tilakaavio

Lähetyspyyntö sovellukselta: lähetetään, jos ikkuna sallii

rdt\_send(data)

```
if (nextseqnum < base+N) {
    sndpkt[nextseqnum] = make_pkt(nextseqnum,data,chksum)
    udt_send(sndpkt[nextseqnum])
    if (base == nextseqnum)
        start_timer
    nextseqnum++
}
else refuse_data(data)
```

Tässä vain yksi ajastin!

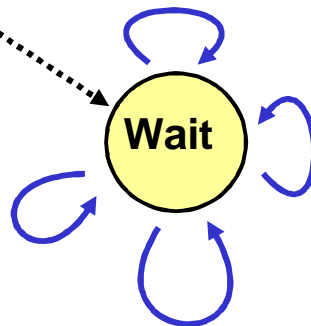
alkuarvot

$\Lambda$   
 $\frac{\text{base}=1}{\text{nextseqnum}=1}$

Ajastin laukeaa, lähetä kaikki kuittaamattomat uudestaan

rdt\_rcv(rcvpkt) && corrupt(rcvpkt)

$\Lambda$



timeout

```
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
...
udt_send(sndpkt[nextseqnum-1])
```

Korruptoitunut kuittaus hylätään

```
rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)
base = getacknum(rcvpkt)+1
If (base == nextseqnum)
    stop_timer
else start_timer
```

Kuittaus siirtää ikkunaa

KuRo08: Fig 3.20

Sender



## Go-Back-N: Vastaanottajan tilakaavio

receiver

```
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
&&
hasseqnum(rcvpkt,expectedseqnum)
```

Paketti ok ja  
odotettu numero

```
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum,ACK,chksum)
udt_send(sndpkt)
expectedseqnum++
```

Data sovellukselle,  
kuittaus lähettäjälle  
Seuraava pnumero++

Λ

Wait

default  
udt\_send(sndpkt)

Muuten sama  
kuittaus  
uudestaan

```
expectedseqnum=1
sndpkt =
make_pkt(expectedseqnum,ACK,chksum)
```

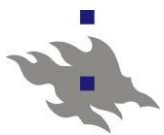
alkuarvot

KuRo08: Fig 3.21

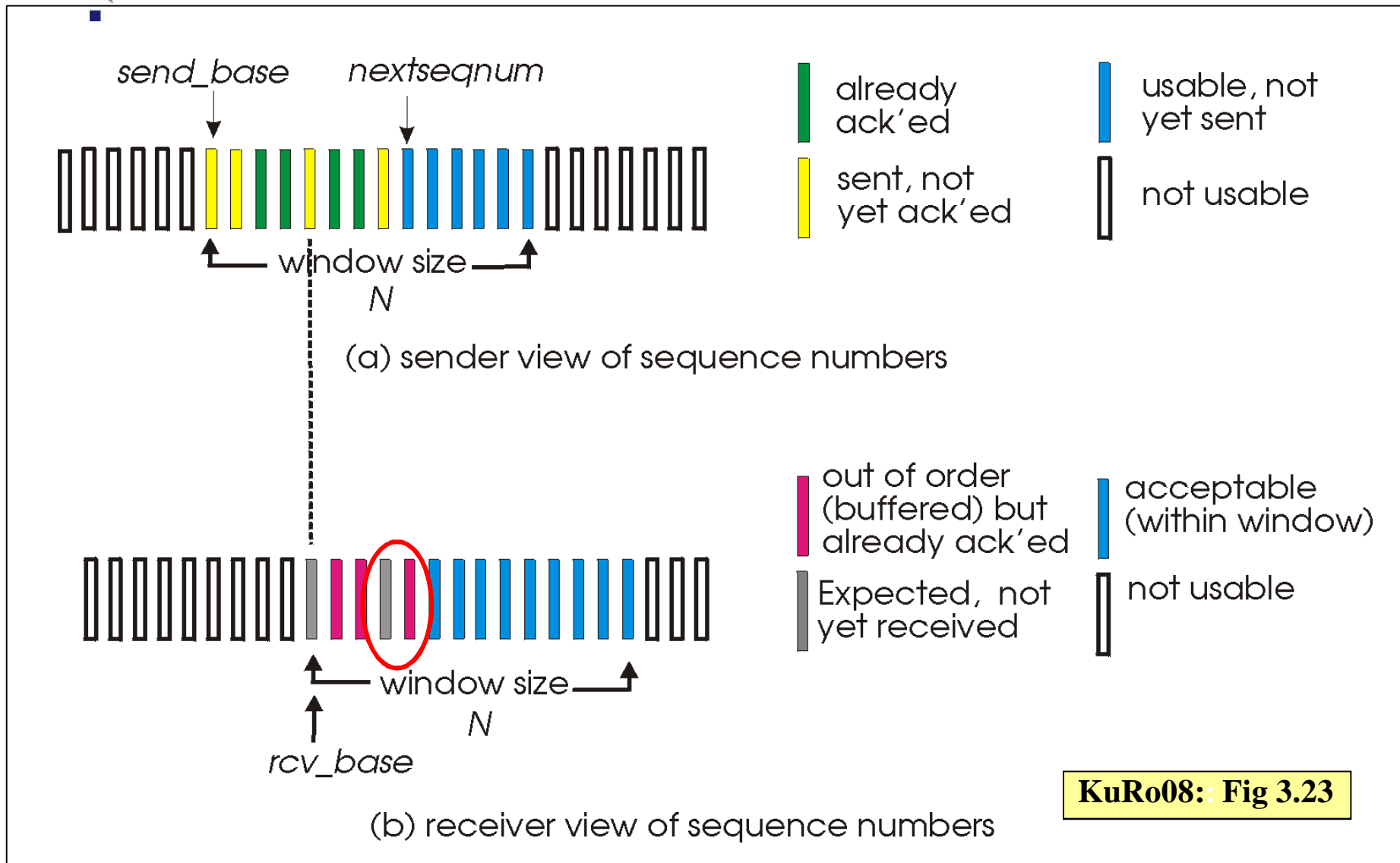


## Valikoiva toisto (Selective Repeat)

- **Valikoiva uudelleenlähetys**
  - Lähetä **uudelleen vain virheellinen /puuttuva paketti**
- **Kuittaus jokaiselle kelvolliselle paketille**
- **Paketit sovellukselle oikeassa järjestyksessä**
  - Vastaanottajalla oltava puskuritilaa pakettien järjestämiseen
- **Jos lähettäjä ei saa kuittausta paketista**
  - Lähetysikkunan täyttyminen pysäyttää lähettämisen
  - Aikanaan ajastin laukeaa ja aiheuttaa uudelleenlähetysten
  - Jokaisella paketilla on oma ajastin
- **Ikkuna liikuu nytkin tasaisesti**
  - Yksi puuttuva kuittaus voi pysäyttää lähetyksen
  - Kun puuttuva paketti saatu, ikkuna liikuu kaikkien kuitattujen yli

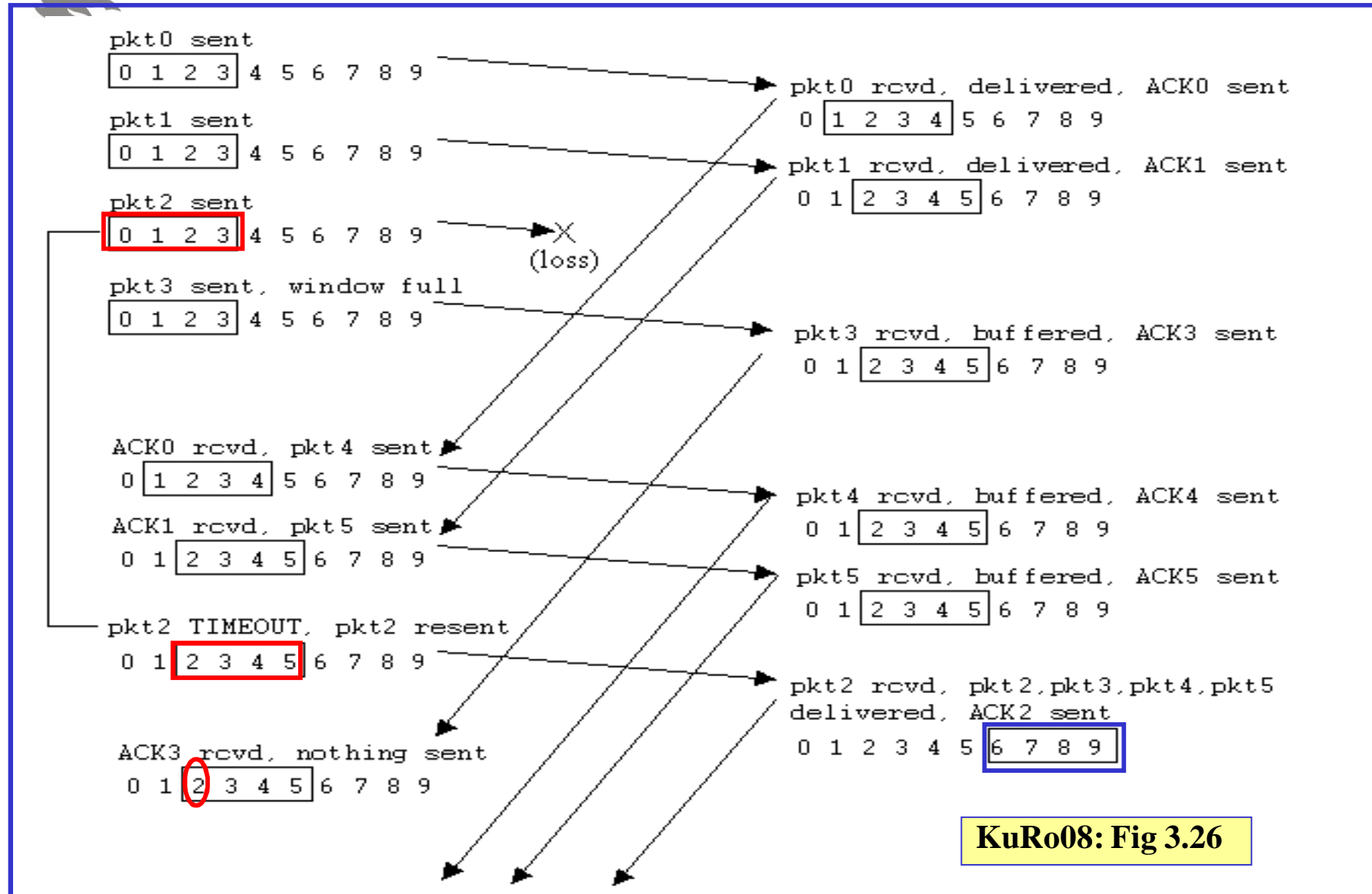


# Valikoiva toisto



# Esimerkki: valikoiva toisto

Jokainen sanoma  
kuitattava erikseen



KuRo08: Fig 3.26

# ■ Ikkunankoko

- Pakettinumeroille varatun kentän koko vaikuttaa myös ikkunankokoon
  - Yleensä jokin kakkosen potenssi
  - Kentän koko  $k$  bittiä  $\Rightarrow$  käytössä  $2^{**}k$  pakettinumeroa
- Kun paketit numeroidaan  $0, 1, \dots, n$ , niin ikkunan koko saa olla korkeintaan

$n+1$  kpl

**Go-Back-n:**  $n$  ( $m$  bit sekvenssinumeroilla  $< 2^{m-1}$ )

**Valikoiva toisto:**  $(n+1)/2$  ( $m$  bit sekvenssinumeroilla  $< 2^{m-1}$ )

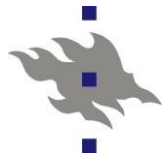
MIKSI NÄIN?

Harjoitustehtävinä!



# Yhteenveto menetelmistä

- Ks. KuRo08 Table 3.1
  
- Tarkistussumma
- Ajastin
- Järjestysnumero
- Kuittaukset
  - Positiiviset ACK, tuplakuittaukset
  - Negatiiviset NAK
- Ikkunat, liukuhihnoitus

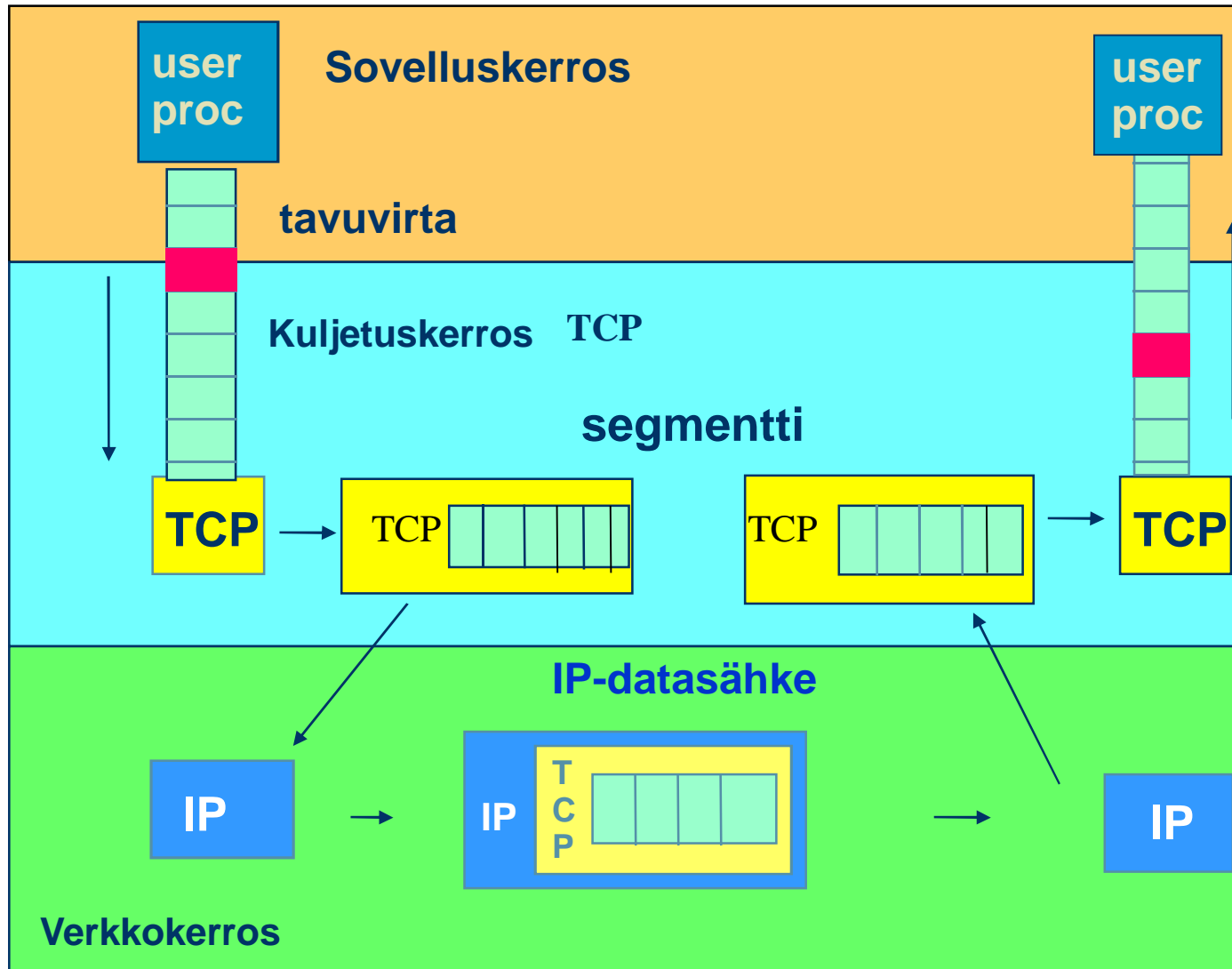
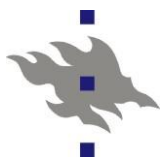


# Kuljetuskerros

# 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)

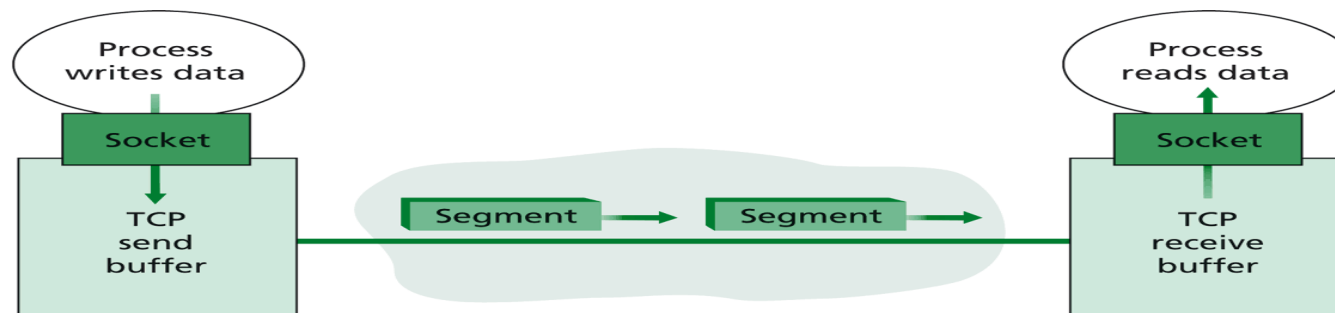
- Yksi yhteys, jossa yhtä aikaa liikennettä molempiin suuntiin

## ■ Luotettava, järjestyksen säilyttävä tavuvirta

- Ei sanomarajoja
- Tavunumerointi
- Kumulatiiviset kuittaukset

# TCP-protokolla

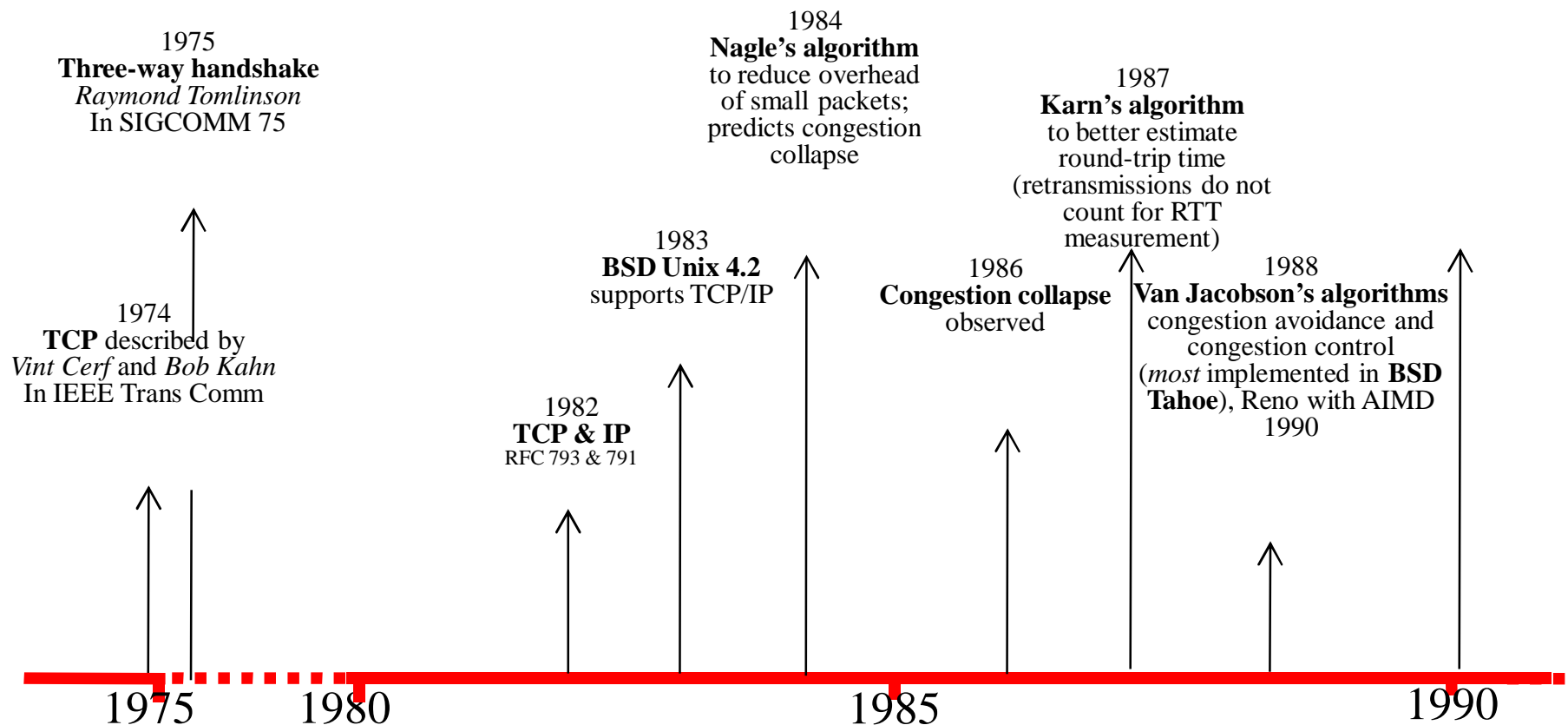
- Vuonvalvonta, ruuhkanhallinta (-valvonta)
  - Lähettäjä ei voi tukahduttaa vastaanottajaa eikä reitittäjiä
- Liukuvan ikkunan protokolla
  - Vuonvalvonta ja ruuhkanhallinta vaikuttavat lähetyssikkunankokoon
- Puskurointia molemmissa päissä
  - Uudelleenlähetystä varten
  - Jotta saadaan annettua sovellukselle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



**Figure 3.28** ♦ TCP send and receive buffers

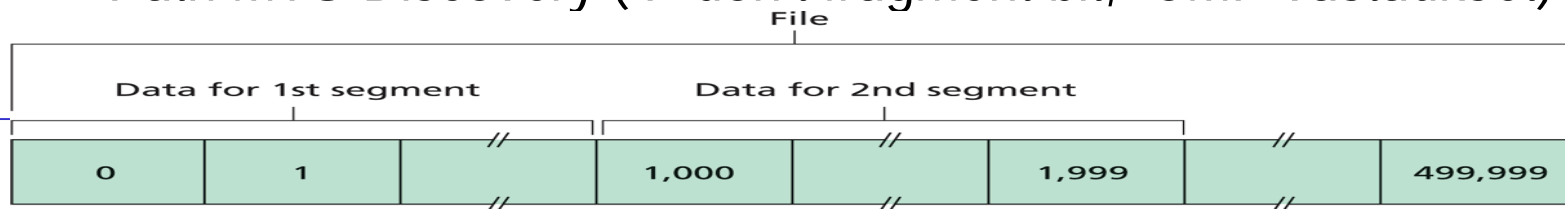


# TCP:n evoluutio



# TCP-protokolla

- 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) mikä menee linkeistä läpi
  - Ethernet MTU = 1500 => MSS = 1460, sillä TCP:n ja IP:n osoitteet (20 + 20 tavua) vievät oman tilansa
  - Path MTU Discovery (IP don't fragment bit, ICMP vastaukset)



**Figure 3.30** ♦ Dividing file data into TCP segments

# TCP-segmentti

Otsake aina vähintään 20 B  
Optio-osa tarvittaessa

Segmentti- ja kuittaus-  
numerot **tavunumeroina**

Ikkunankoko: paljonko tilaa  
vastaanottopuskurissa (tavua)

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

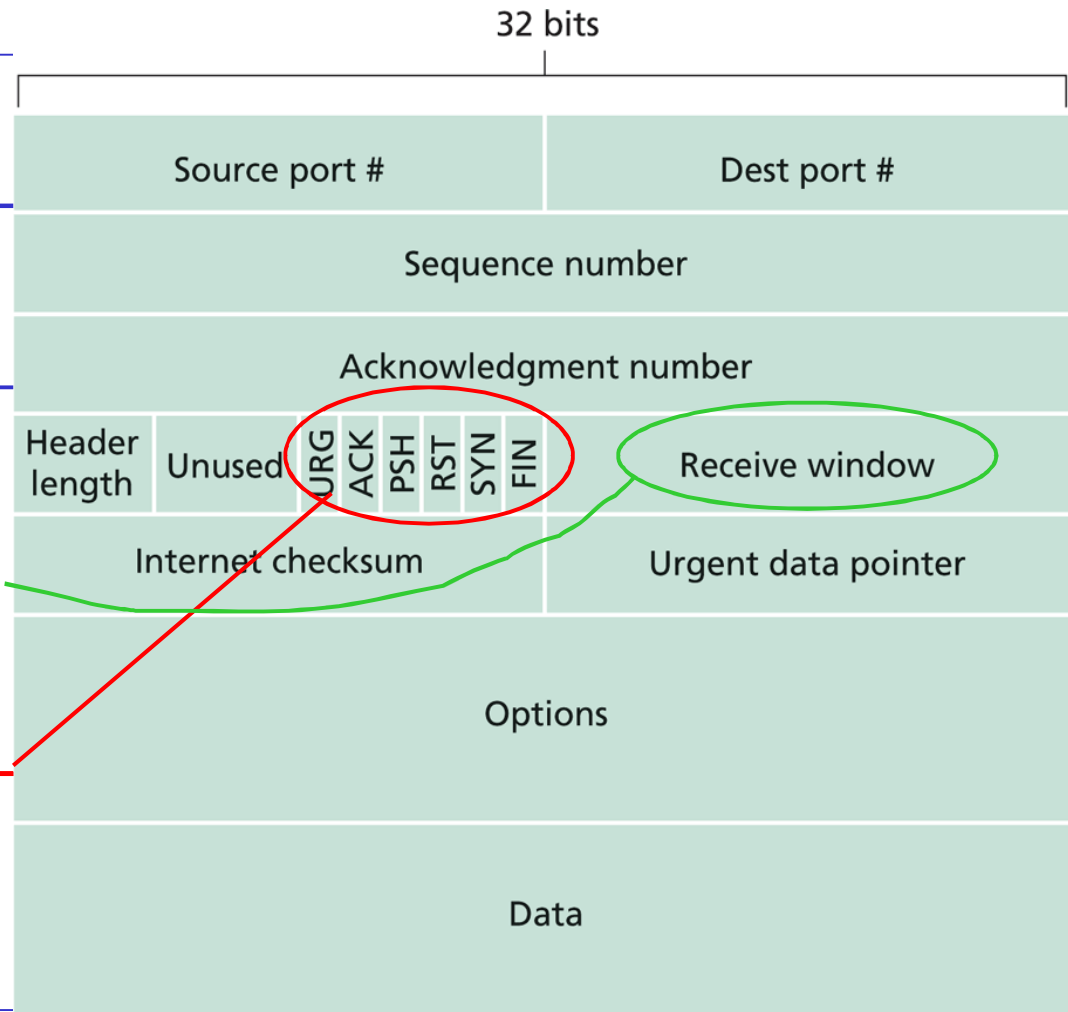


Figure 3.29 ♦ TCP segment structure

# Tavunumerointi

Kuittauksia ei kuitata ja ne eivät siirrä numerointia!

Niissä ei siirretä tavuvirtaa.

Tavuvirtaa ...

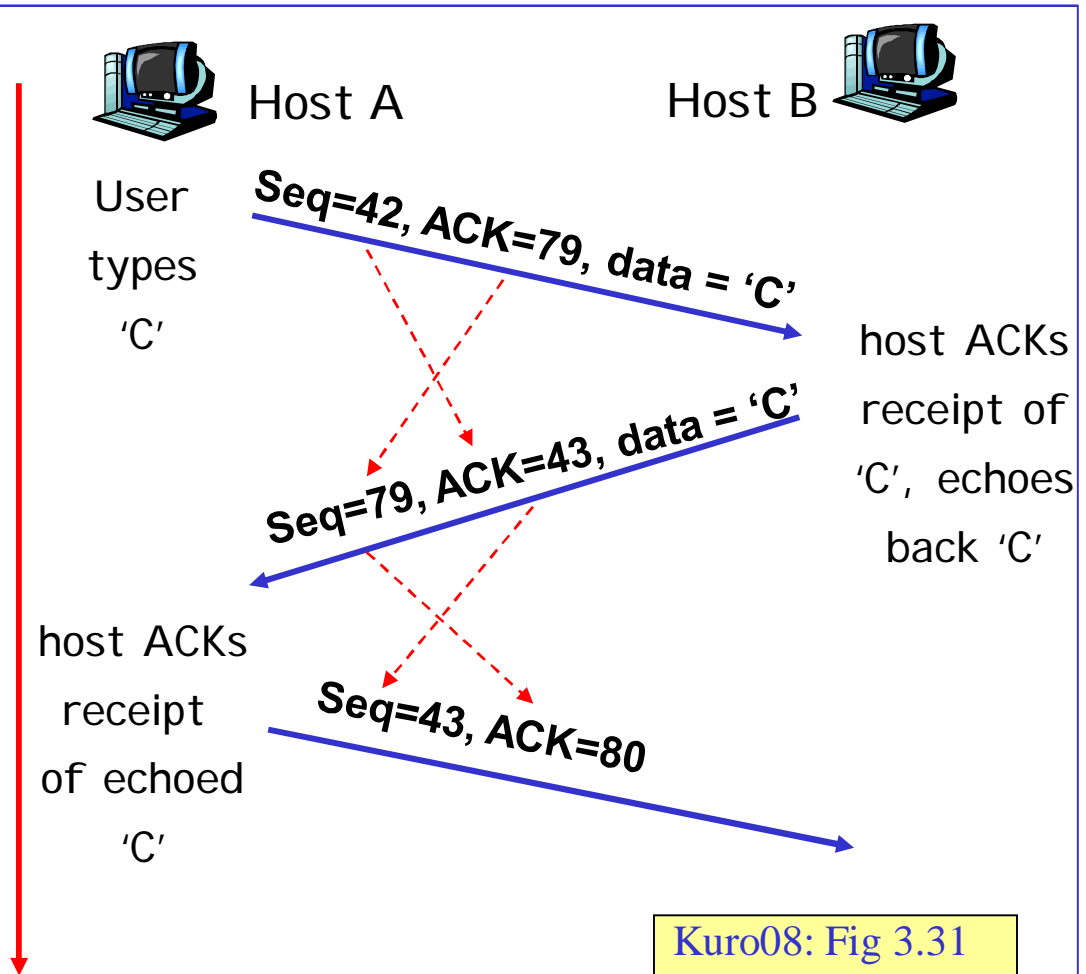
Segmentit voivat olla erikokoisia ( $\leq$  MSS)

Segmentin 'numero' =

- ensimmäisen tavun numero
- alkuarvot sovitaan yhteyttä muodostettaessa

Kuittaus

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



simple telnet scenario



Oletus: ruuhkanhallinta tai vuonvalvonta ei rajoita, data jo valmiiksi sopivina paloina ( $\geq$  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 \*/**

Kuro08: Fig 3.33

## TCP: lähetys (simplified)

Comment:

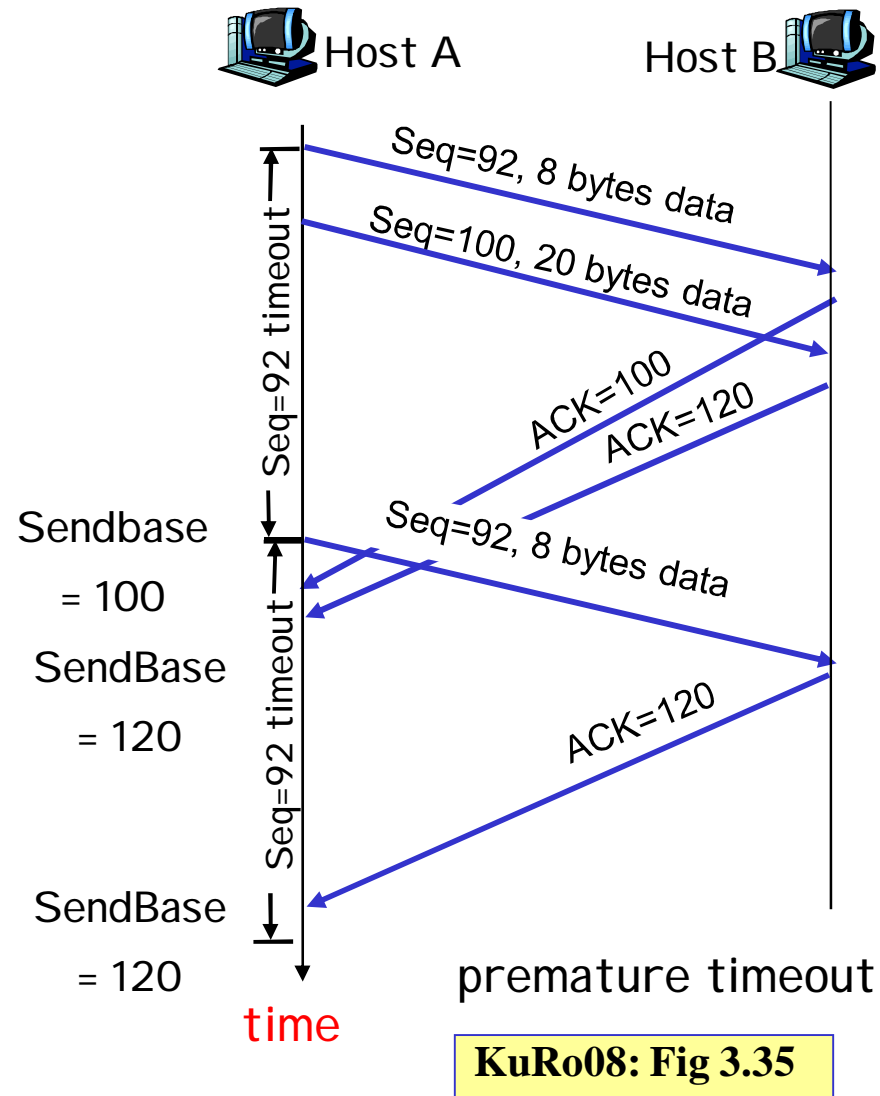
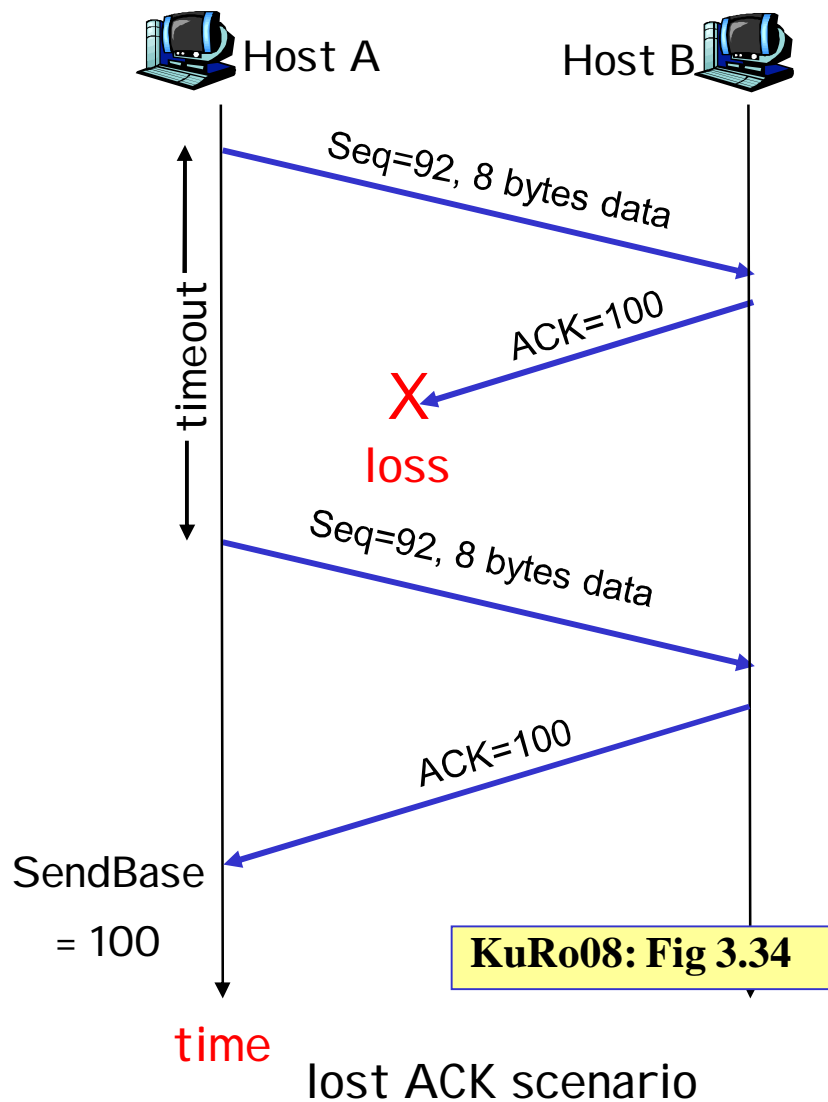
- SendBase-1: last cumulatively ack'ed byte

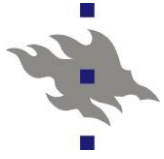
Example:

- SendBase-1 = 71; y = 73, so the rcvr wants 73+ ; y > SendBase, so that new data is acked

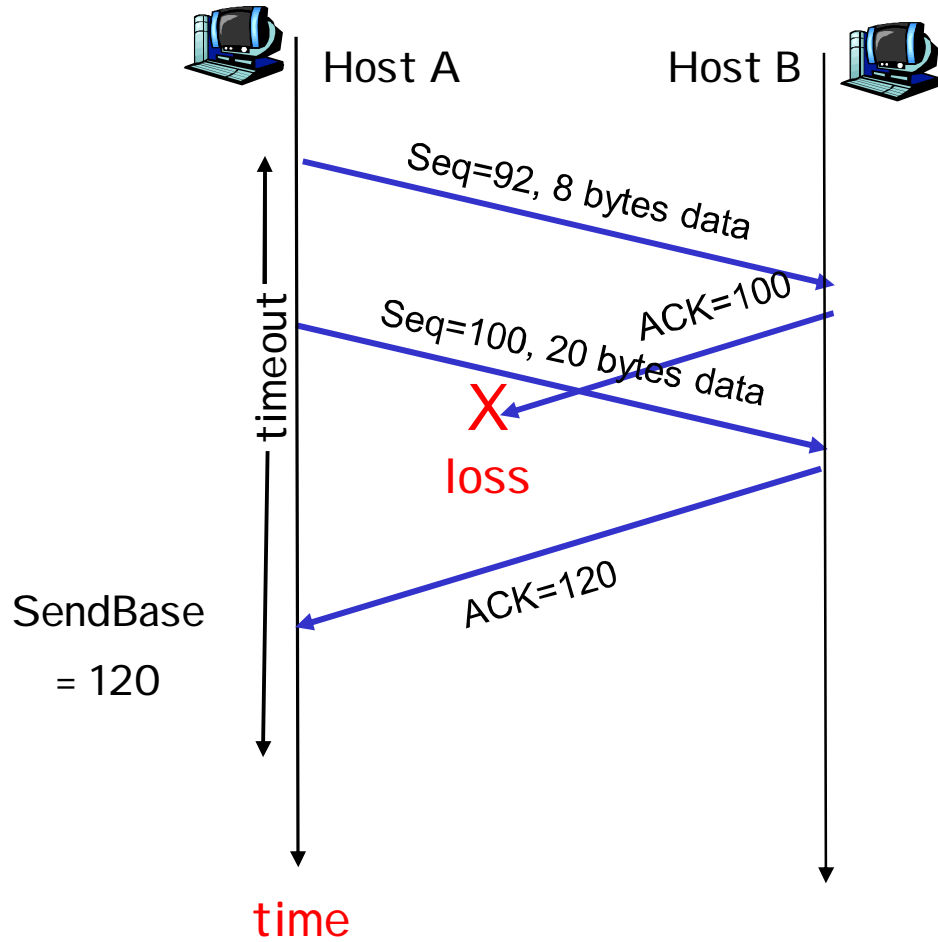


# TCP: uudelleenlähetyks





# TCP: uudelleenlähetys (2)



Cumulative ACK scenario

KuRo08: Fig 3.36



## TCP ACK generation [RFC 1122, RFC 2581]

### 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-expected seq. # . Gap detected

Immediately send *duplicate ACK*, indicating seq. # of next expected byte

Arrival of segment that partially or completely fills gap

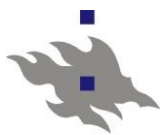
Immediate send ACK, provided that segment starts at lower end of gap

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 **toistokuittausta**, 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

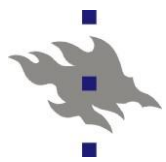
Sender

**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), ei yleisesti käytössä

# Vuonvalvonta

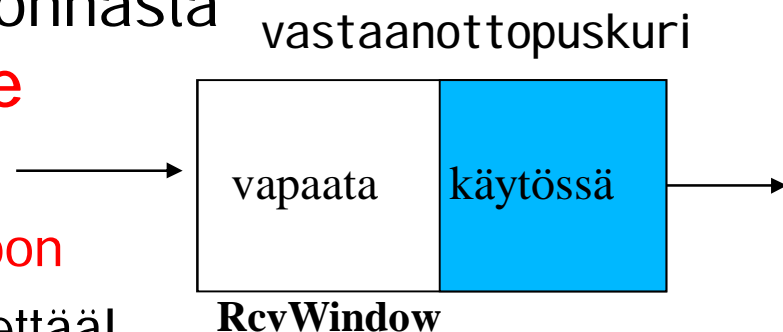
- Jotta lähettäjä ei tukahduta vastaanottoa

- 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 puskureihin vielä mahtuu

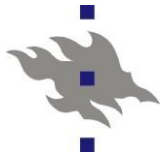
- TCP-segmentin otsakkeen kenttä **Receive window**
- Sovellus lukee tavut silloin kun haluaa
- Koko on mukana jokaisessa TCP-segmentissä (molempiin suuntiin)

- **Myös ruuhkanhallinta rajoittaa lähettämistä**



# Vuonvalvonta

- Kun ikkunankoko ==0, milloin voi taas lähettää?
  - Jatka **lähettämällä tavun kokoisia segmenttejä**
    - Kysely (ns. persist ajastin)
- Kuittaus antaa ajantasalla olevan tiedon vastaanottajan puskuritilasta
  - Edellisen ACK:n toistokuittaus => ei tilaa
  - Normaali ACK, kun vapaata vähintään täydelle TCP-segmentille
  - Silly window syndrome, pienenevät vastaanottoikkunat
  - Naglen algoritmi (puskurointi)
- 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**

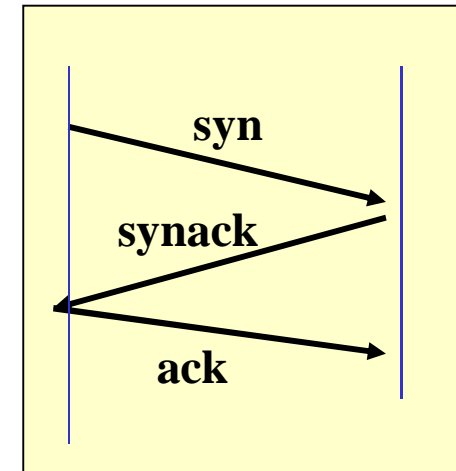
- Lähettäjä puskuroid uudelleenlähetyksi varten
- Vastaanottaja saatujen pakettien järjestämistä varten

## ■ **Sovitaan tavunumeroinnin alkuarvoista**

- Kuittauksia varten

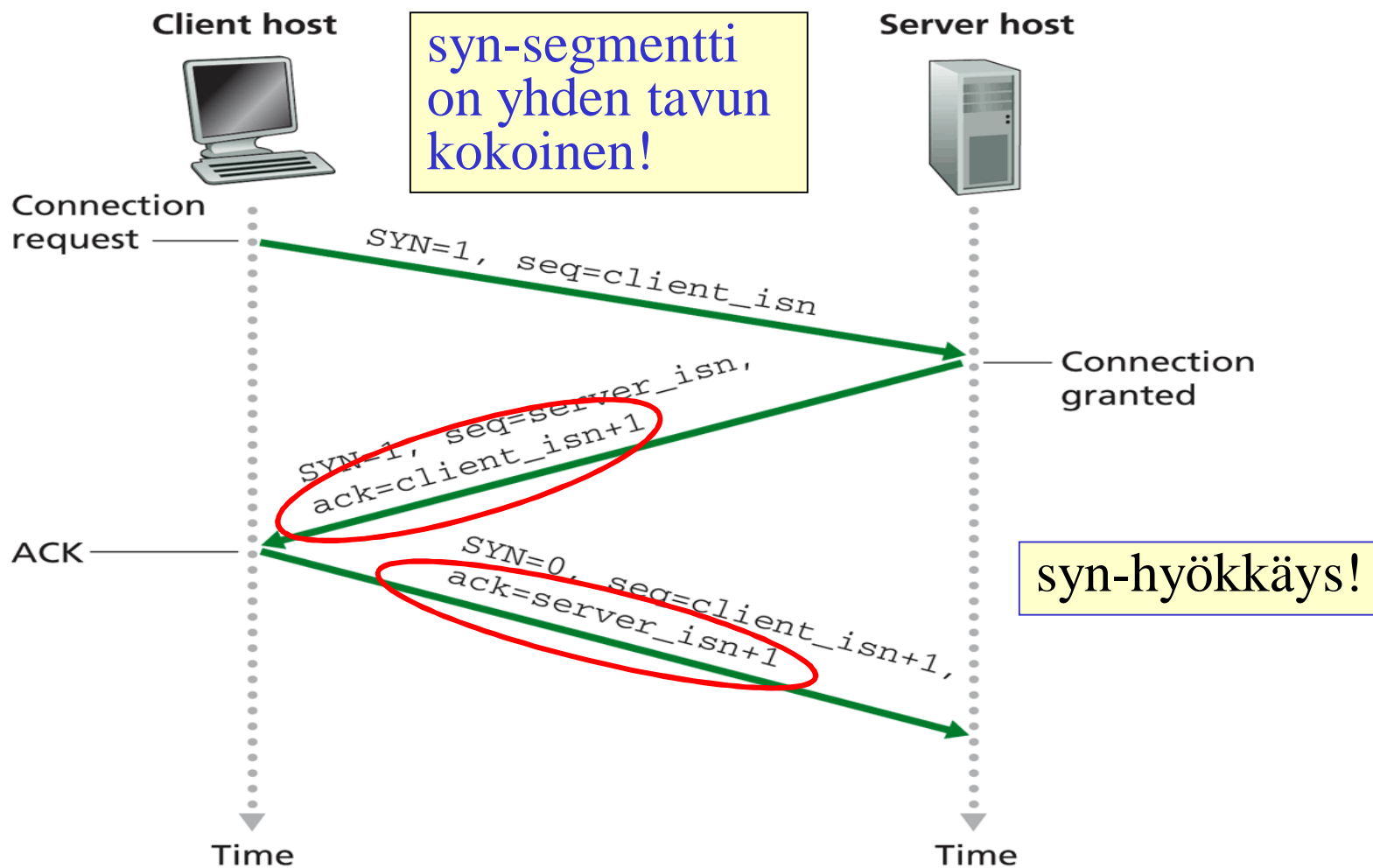
## ■ **Ilmoita oma vastaanottoikkunan koko**

- Vuonvalvontaa varten

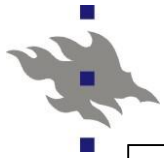




# Yhteyden muodostus

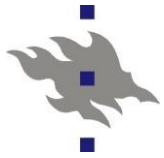


**Figure 3.39** ♦ TCP three-way handshake: segment exchange



## Yhteyden purku

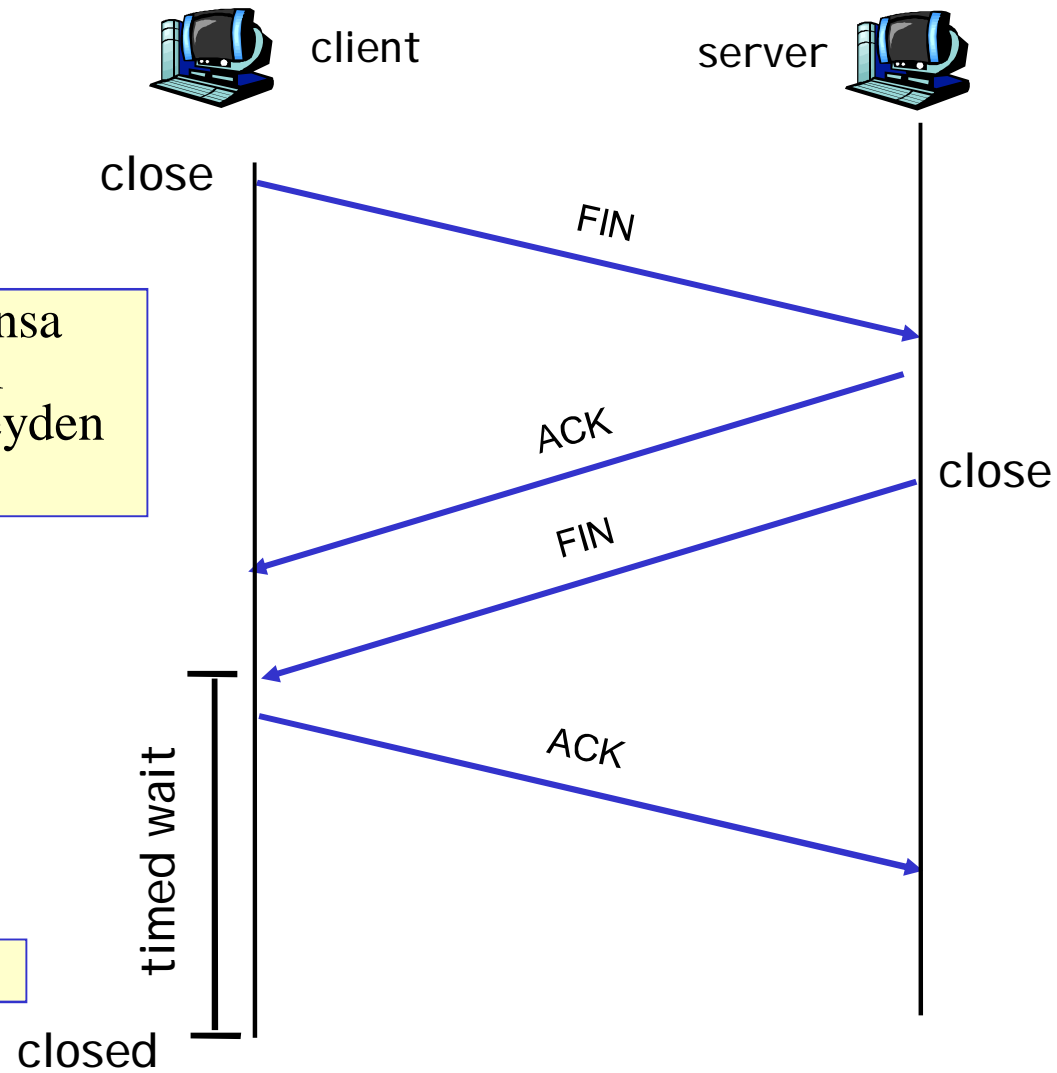
- Molemmat suunnat puretaan erikseen
  - 4 segmenttiä: FIN – ACK, FIN – ACK
- Yhteys on kokonaan purettu, kun molemmat suunnat purettu
- Purku käyttää ajastimia
  - Joidenkin erikoistilanteiden hallintaan
  - Noin 2 \* paketin maksimaalinen elinikä
  - Tyypillisesti 30, 60 tai 120 s
- Puoliavoin yhteys (half-open)
- FIN+ACK voi yhdistää, 3 vaihetta
- Myös FIN+ACK samanaikaisesti 2:lla vaiheella

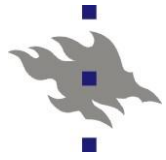


# Yhteyden purku

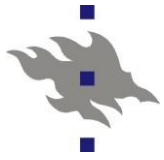
Kumpi tahansa osapuoli voi aloittaa yhteyden purun.

KuRo08: Fig 3.40



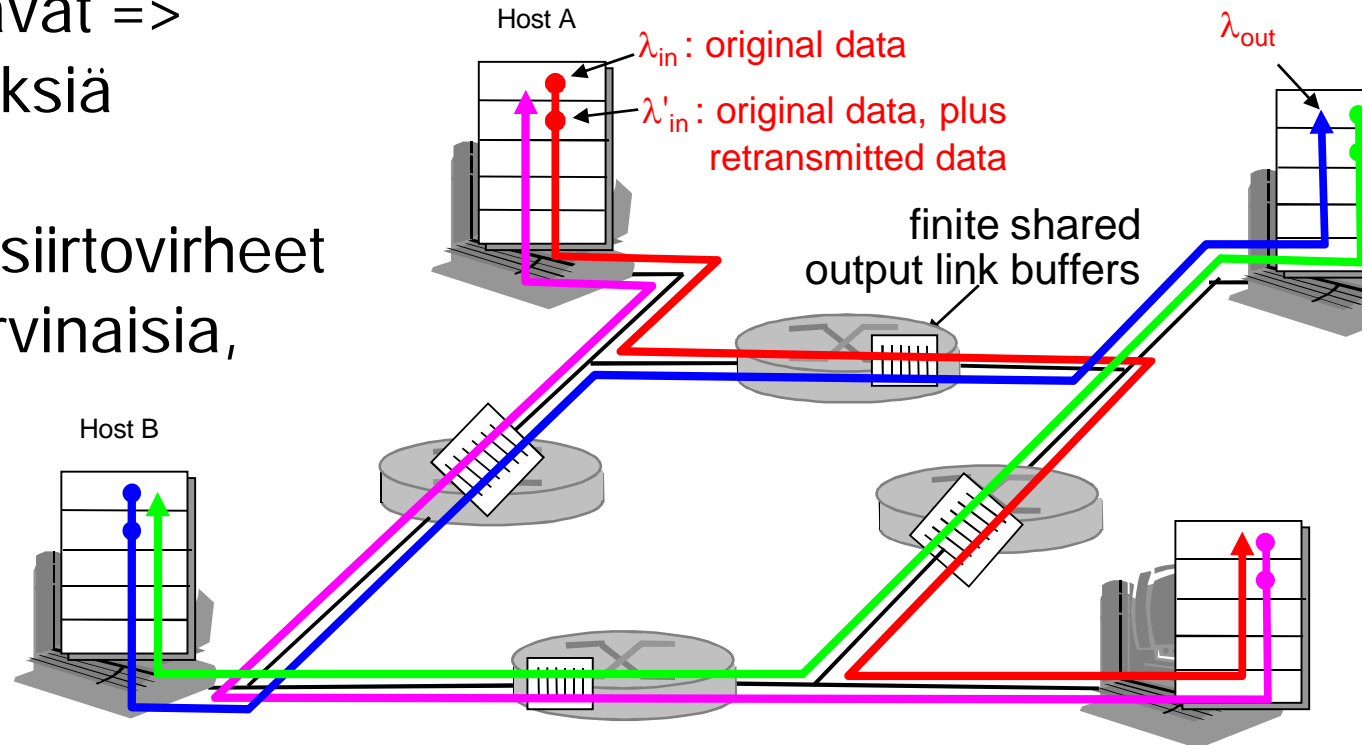


# Ruuhkanhallinta TCP:ssä



# Ruuhka

- Reitittimelle tulee paketteja nopeammin kuin se ehtii välittää niitä eteenpäin
  - Pitkiä jonotusviipeitä
  - Pakettien hävittämistä (puskuritila loppuu)
- Ajastimet laukeavat => uudelleenlähetystyksiä
- Lankaverkoissa siirtovirheet ovat nykyisin harvinaisia, paketin katoamisen syynä lähes aina ruuhka





## Miten ratkaista ongelma?

- Lähettäjän on hidastettava vauhtia
- Verkkoavusteinen ruuhkanvalvonta (network assisted congestion control)
  - Reitittimet antavat tietoa ruuhkasta
  - Lisäbitit kertomassa ruuhkasta tai kenttä, joka ilmoittaa yhdeydelle sallitun lähetysnopeuden (explicit rate), Explicit Congestion Notification
- **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 uudelleenlähetyksajastin 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(lähetysikkuna, ruuhkaikkuna)**

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

- Aluksi ruuhkaikkuna = yksi segmentti

- Alussa hidas siirtonopeus =  $MSS/RTT$

- Kukin kuittaus kasvattaa yhdellä ruuhkaikkunan kokoa

hidas aloitus

- Eksponentiaalinen kasvu
- Ikkuna kaksinkertaistuu yhden RTT:n aikana

- Jos uudelleenlähetys, ruuhkaikkunan kooksi 1 segmentti

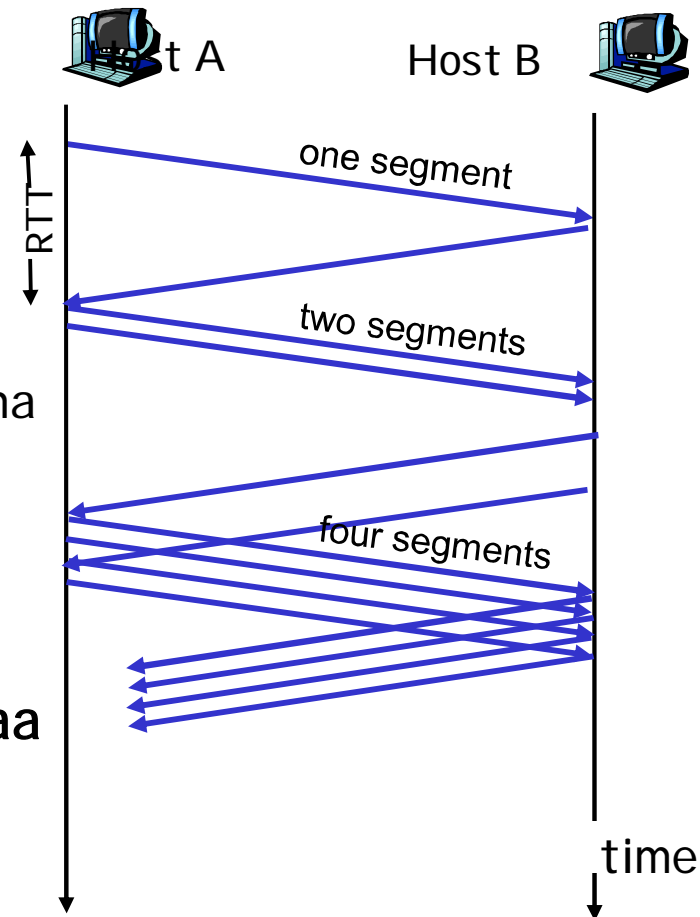
- Multiplicative decrease

- Tietyn kynnsarvon jälkeen kasvata ikkunaa yksi segmentti/RTT

ruuhkanvälttely

- Lineaarinen kasvu (Additive increase)
- Ruuhkan välttely (congestion avoidance)

- Siirtonopeus =  $CongWin / RTT$  tavua/sek





## Kynnysarvo (threshold)

- = ~ **varoitus**: tästä lähtien syytä varoa ruuhkaa
  - Aluksi threshold = 64 KB
- Ajastimen laukeamisen (timeout) jälkeen
  - Threshold = CongWin / 2
  - Myös Renon toipuessa kolmen tuplakuittauksen jälkeen
- Kynnysarvoon saakka ikkunan kasvatus eksponentilaalista
  - 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
- **Miksi näin?**

ruuhkanvälttely

# TCP Reno: Tarkennus

- Saatu 3 ACK-kaksoiskuittausta (double ACK) (4 samaa kuittausta!)
  - Verkkopysty välittämään dataa!
  - Ei siis (paha) ruuhkaa, ehkä paketissa bittivirhe tai paketti kadonnut jostain muusta syystä
- Nopea uudelleenlähetyks (fast retransmit)
- Nopea toipuminen (fast recovery)
  - Puolita ruuhkaikkunan arvo ja kasvata sitten lineaarisesti

kynnysarvo?

## ■ Timeout (= uusi hidas aloitus)

- Verkkopahasti ruuhkautunut!
- Pudota ikkunankoko arvoon 1
- Kasvata eksponentiaalisesti kynnysarvoon asti
- Kasvata sitten lineaarisesti

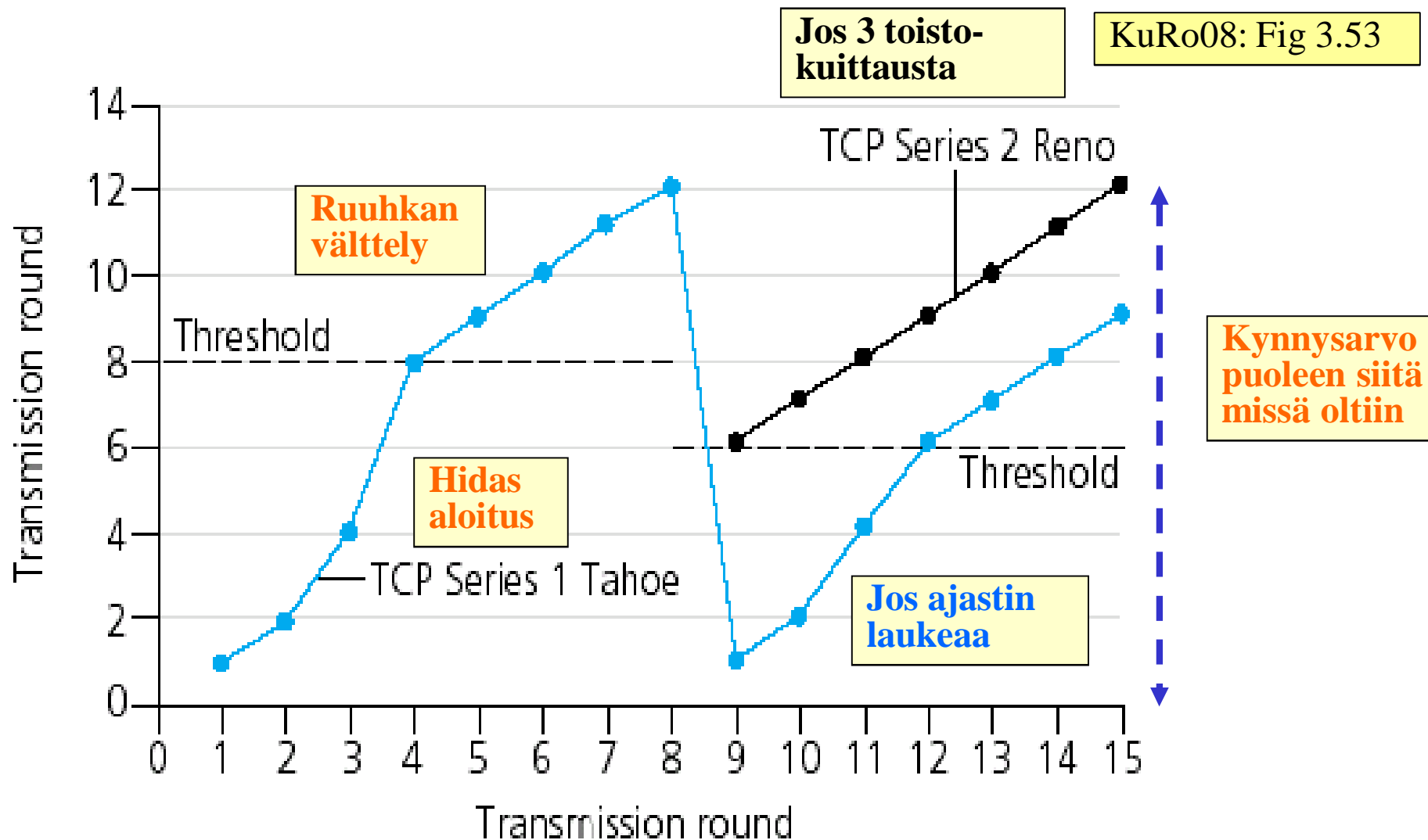
Hidas aloitus

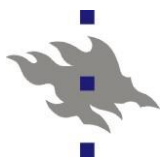
ruuhkanvälttely

Vanha TCP Tahoe pudotti aina kokoon 1.



# TCP Tahoe vs. TCP Reno





# TCP Reno Ruuhkanhallinta

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin + MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

KuRo08: Table 3.3



## Ajastimen arvo?

- Aseta ajastin, kun segmentti on lähetetty
  - Retransmission queue, järjestetty ajan mukaan
- 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, tuplaa Timeout*
  - *Exponential backoff*

## ■ ■ ■ Ajastimen arvo?

■ **Timeoutinterval = EstimatedRTT + 4 DevRTT**

■ **Arvioi kiertoviive** eli EstimatedRTT

■ Mittaa jokaisen lähetetyn segmentin kiertoviive tai noin RTT:n välein normaalien lähetysten aikana.

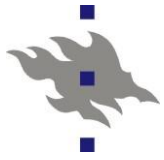
■ Laske painotettu arvo, tyypillisesti  $\alpha = 1/8 = 0.125$

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

■ **Huomioi poikkeama**

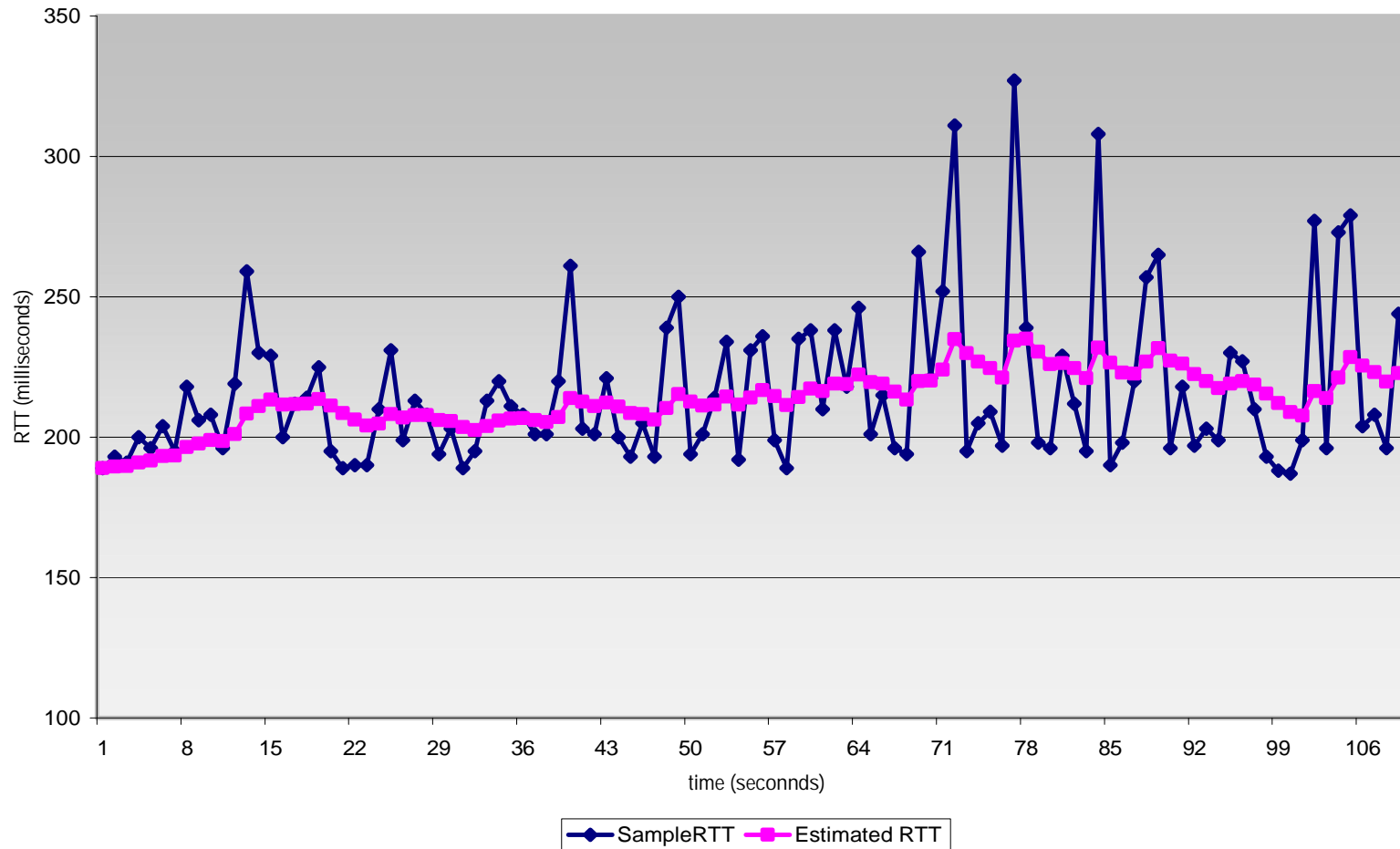
tyypillisesti  $\beta=0.25$

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



# Esimerkki

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



**Kuro05:Fig 3.32 RTT samples and RTT estimates**



## 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 useaan kertaan uudestaan, niin ajastimen arvo aina kaksinkertaistetaan.

**Esimerkki:** Lähetetään segmentti 100 ja ajastimen arvo 2.5 sekuntia.

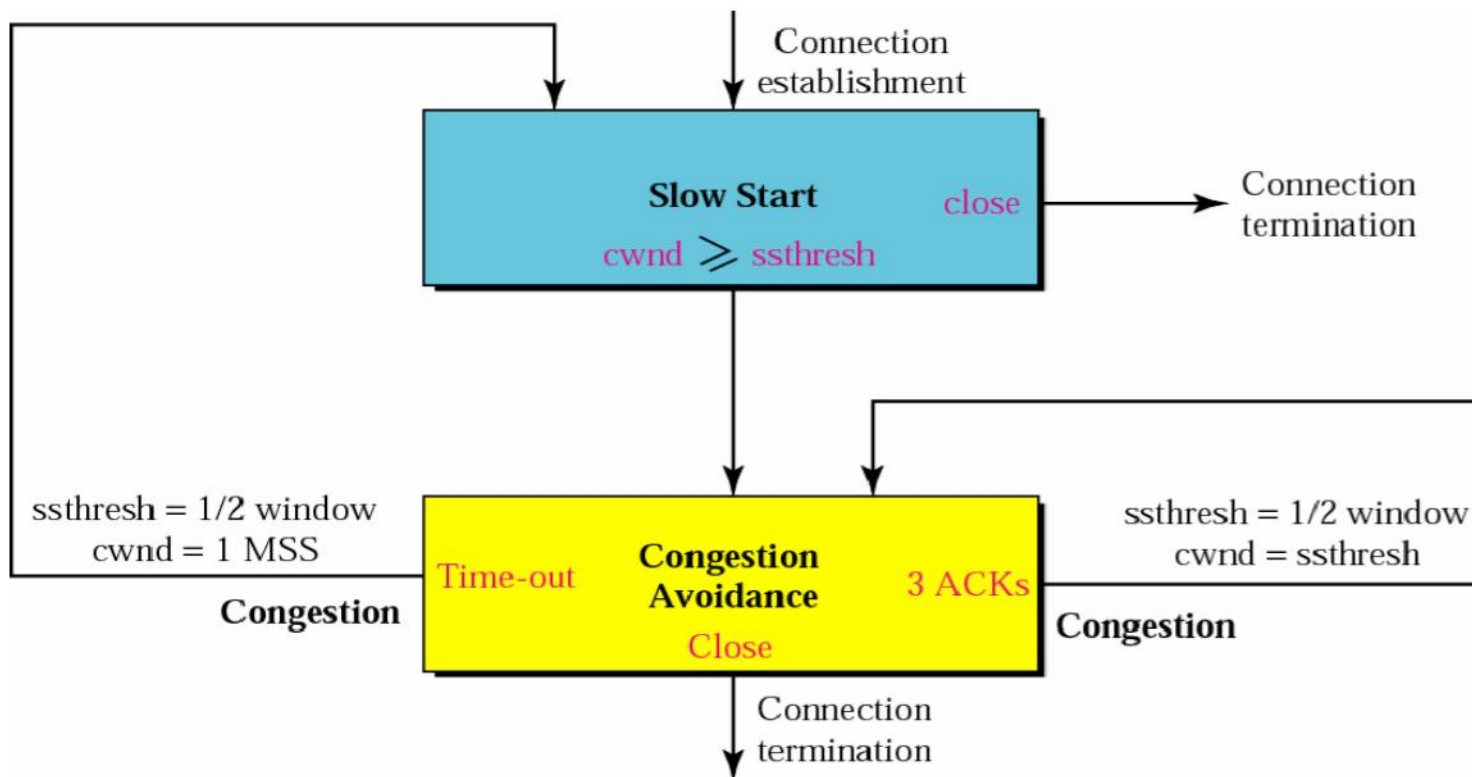
Ajastin laukeaa ja lähetään segmentti 100 uudestaan ja nyt ajastimen arvo on 5 sekuntia.

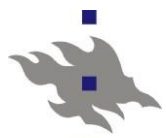
Kun kuittausta ei tule 5 sekunnin sisällä, niin ajastin taas laukea ja segmentti 100 lähetetään vielä kerran. Nyt ajastimen arvoksi asetetaan 10 sekuntia.

Tähän saadaan kuittaus ajoissa ja siirrytään lähettämään segmenttiä 1100. Ajastimen arvoksi asetetaan tuorein estimoitu arvo 3.2 sekuntia.



# TCP Ruuhkanhallinta





## 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? Explicit congestion notification



## Kertauskysymyksiä

- TCP vs. UDP?
- Miten tehdä kuljetuspalvelusta luotettava?
- Keskeisimmät TCP:n otsakkeessa olevat tiedot?
- Mitä tapahtuu TCP:n yhteydenmuodostuksessa?
- Vuonvalvonta?
- Ruuhkanhallinta?

ks. Kurssikirja s. 293





# Lisämateriaalia

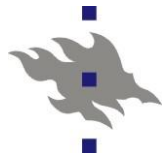


- Additional information about “holes” in sequence number space
  - TCP option that reports discontinuous blocks of received data
  - Sender gets better information about which segments are lost
  - Allows more efficient retransmissions
  - Without SACK sender can retransmit only one segment in round-trip time
  - With SACK more retransmissions can be made in a round-trip time
  - Allows more efficient tracking of number of outstanding segments
  - SACK option specified in RFC 2018
  - SACK-based retransmission algorithm specified in RFC 3517



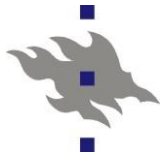
# Timestamps

- Specified in RFC 1323
- TCP option for sender to include timestamp in every packet
- TCP receiver echoes the timestamp back to sender
- Retransmissions have different timestamp than original
- Allows round-trip time measurement for retransmitted segments
  - Not allowed without timestamps
  - Allows detection of spurious retransmissions [Ludwig00]
  - Allows protection against wrapped sequence numbers



# Explicit Congestion Notification

- Sender marks a bit in IP header if transport is ECN capable
  - Routers to indicate congestion with a congestion bit in IP header
  - Used with Active Queue Management
  - Reduces the number of packet losses
  - Transport layer receiver echoes congestion notification to sender
- In transport header
  - When receiving notification, sender reduces its transmission rate
  - Implemented in many end-hosts, but not too many routers
- Problem: Some devices in network drop IP packets with ECN bits



# SYN Cookies

- Client
  - sends SYN packet and ACK number to server
  - waits for SYN-ACK from server w/ matching ACK number
- Server
  - responds w/ SYN-ACK packet w/ initial SYN-cookie sequence number
  - Sequence number is cryptographically generated value based on client address, port, and time.
- Client
  - sends ACK to server w/ matching sequence number
- server
  - If ACK is to an unopened socket, server validates returned sequence number as SYN-cookie
  - If value is reasonable, a buffer is allocated and socket is opened

