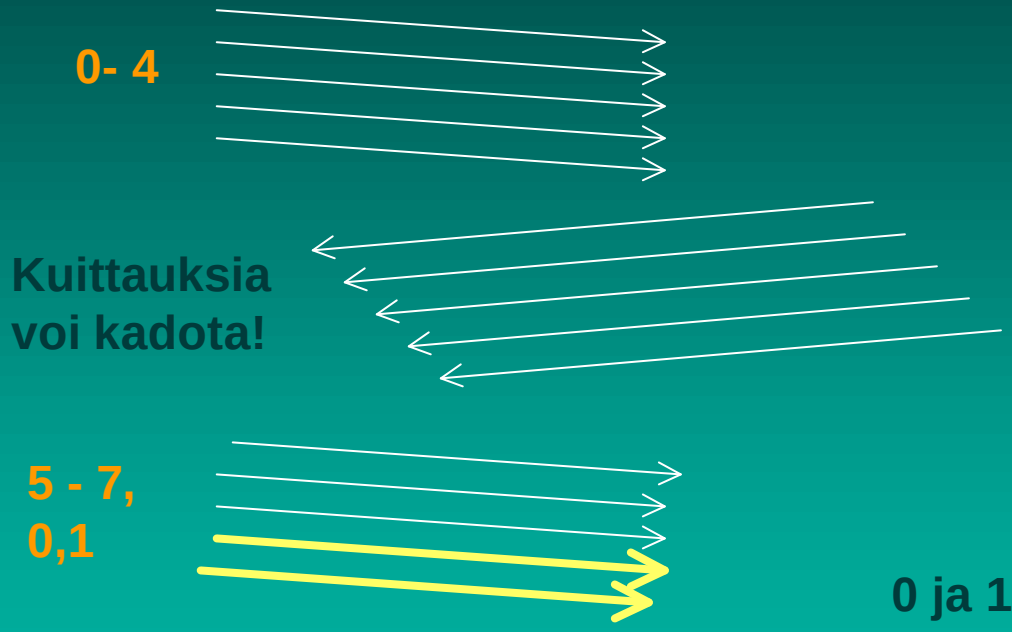
Ikkunankoko

- Kun käytetty numeroavaruus on 0, 1, .. 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 ainakin yhtä pienempi kuin numeroavaruus
- ikkunan koko valikoivassa toistossa voi olla korkeintaan (n+1)/2
  - korkeintaan puolet numeroavaruudesta

Miksi?

- Valikoiva toisto: ikkuna 5, numeroavaruus 8



Miksi?

- Valikoiva toisto: ikkuna 4, numeroavaruus 8

4, 5, 6 ja 7: täytyy olla uusi

0, 1, 2, 3: täytyy olla uudelleenlähetys

Kaksisuuntainen liikenne

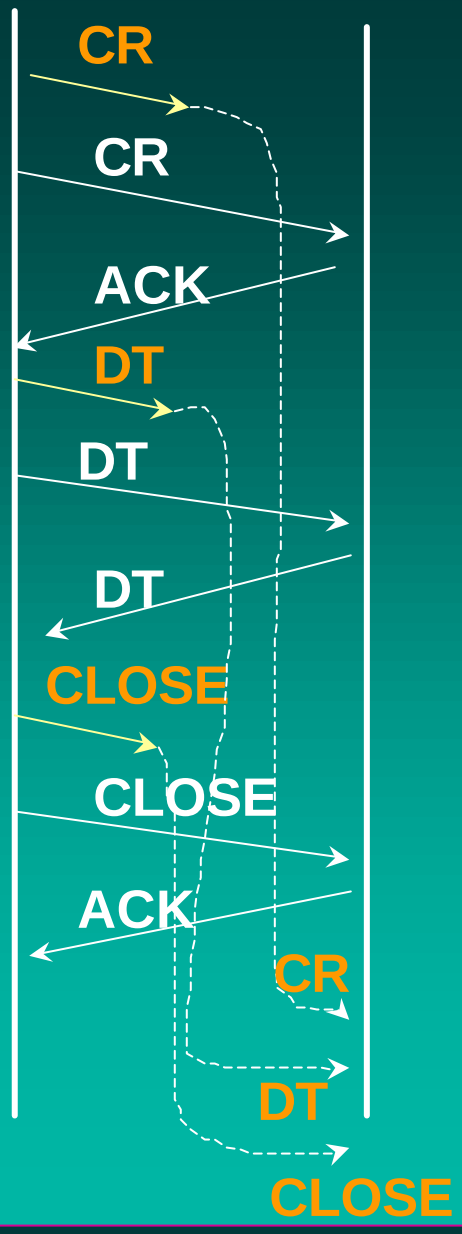
- datakehys ja kuittauskehys
- kehyksessä sekä data että kuittaus
  - ‘piggypacking’
  - tehostaa lähetystä
- ongelma: kauanko kuittaaja odottaa dataa ennen pelkän kuittauksen lähettämistä?

3.5. TCP-protokolla

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

6.2.2. 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
    - sanomien numeroinnista sopiminen
  - Bysanttilainen ongelma (two-army problem)
    - “hyökkään, jos olen varma, että sinäkin hyökkäät”
    - symmetrinen yhteyden purku = molemmat osapuolet tietävät, että toinenkin on varmasti purkanut yhteyden

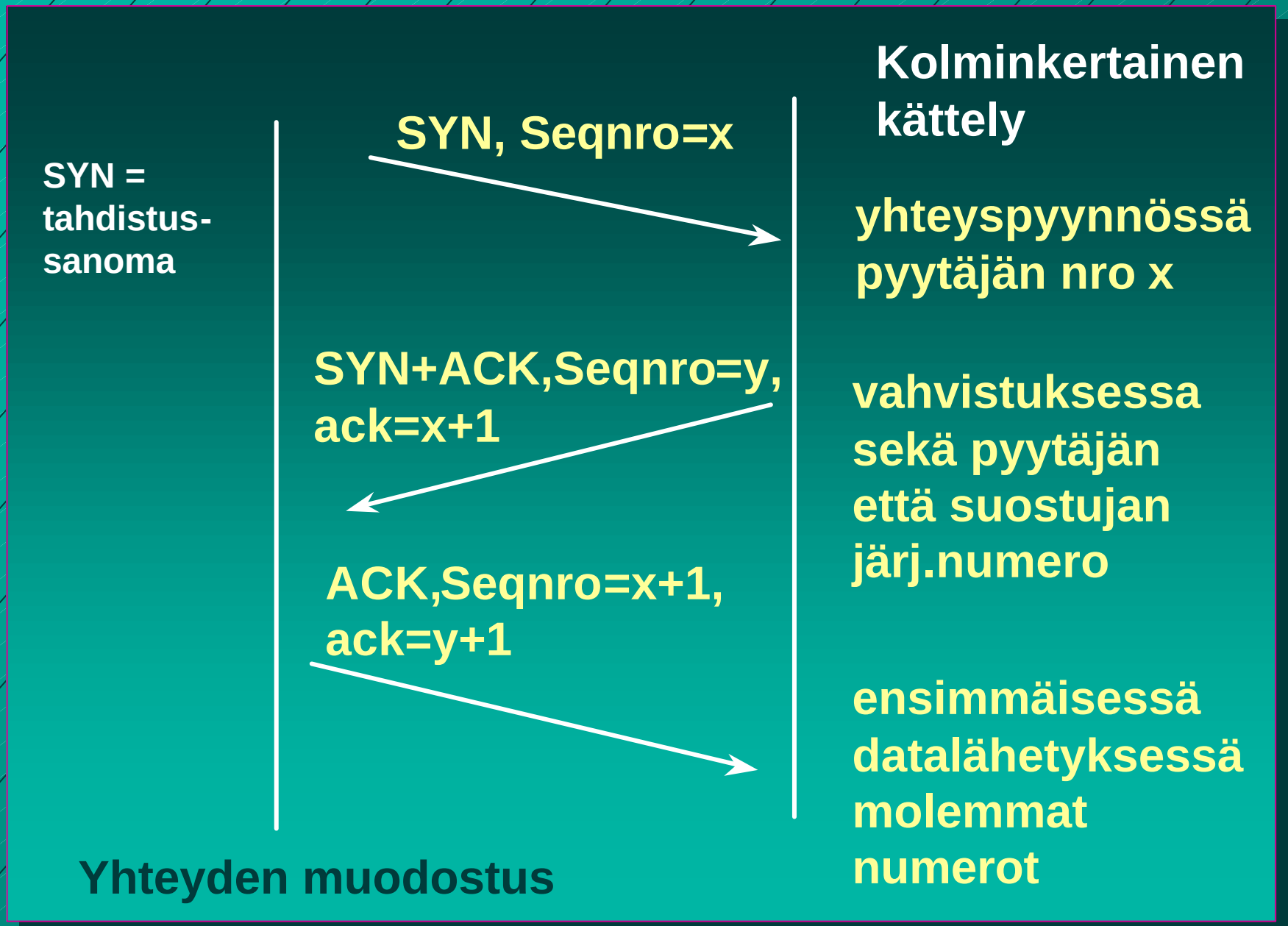


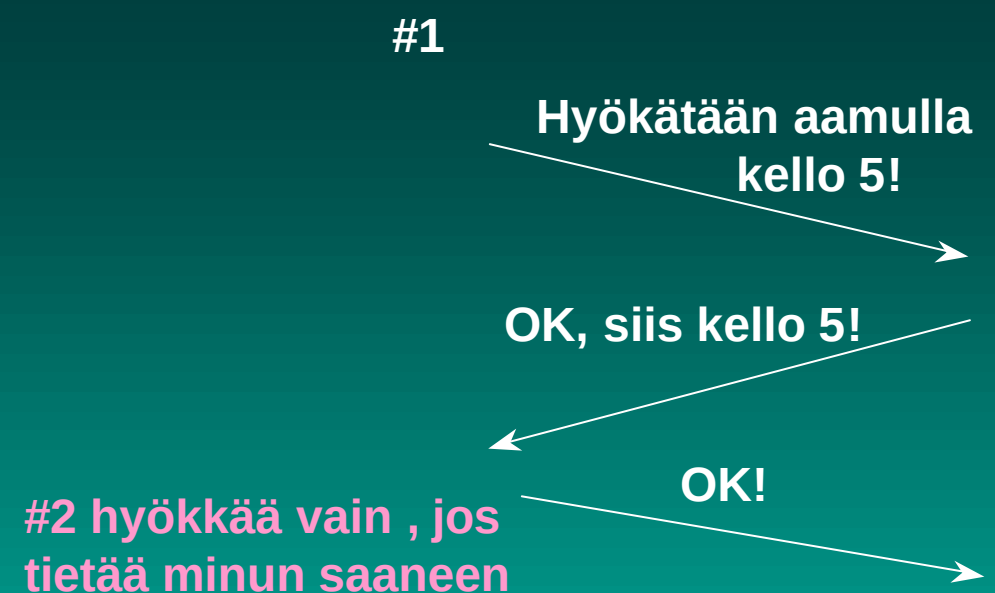
Yhteyden muodostus ruuhkaisessa verkossa

Jokainen paketti lähetetään kahteen kertaan

Kun yhteys on purettu, viivästyneet kaksoiskappaleet saapuvat

Ne tulkitaan uudeksi yhteydeksi, ja data otetaan vastaan kahteen kertaan!


Kolminkertainen kättely
SYN, Seqnro=x
SYN = tahdistus-sanoma
yhteyspyynnössä pyytäjän nro x
SYN+ACK,Seqnro=y, ack=x+1
vahvistuksessa sekä pyytäjän että suostujan järj.numero
ACK,Seqnro=x+1, ack=y+1
ensimmäisessä datalähetyksessä molemmat numerot
Yhteyden muodostus

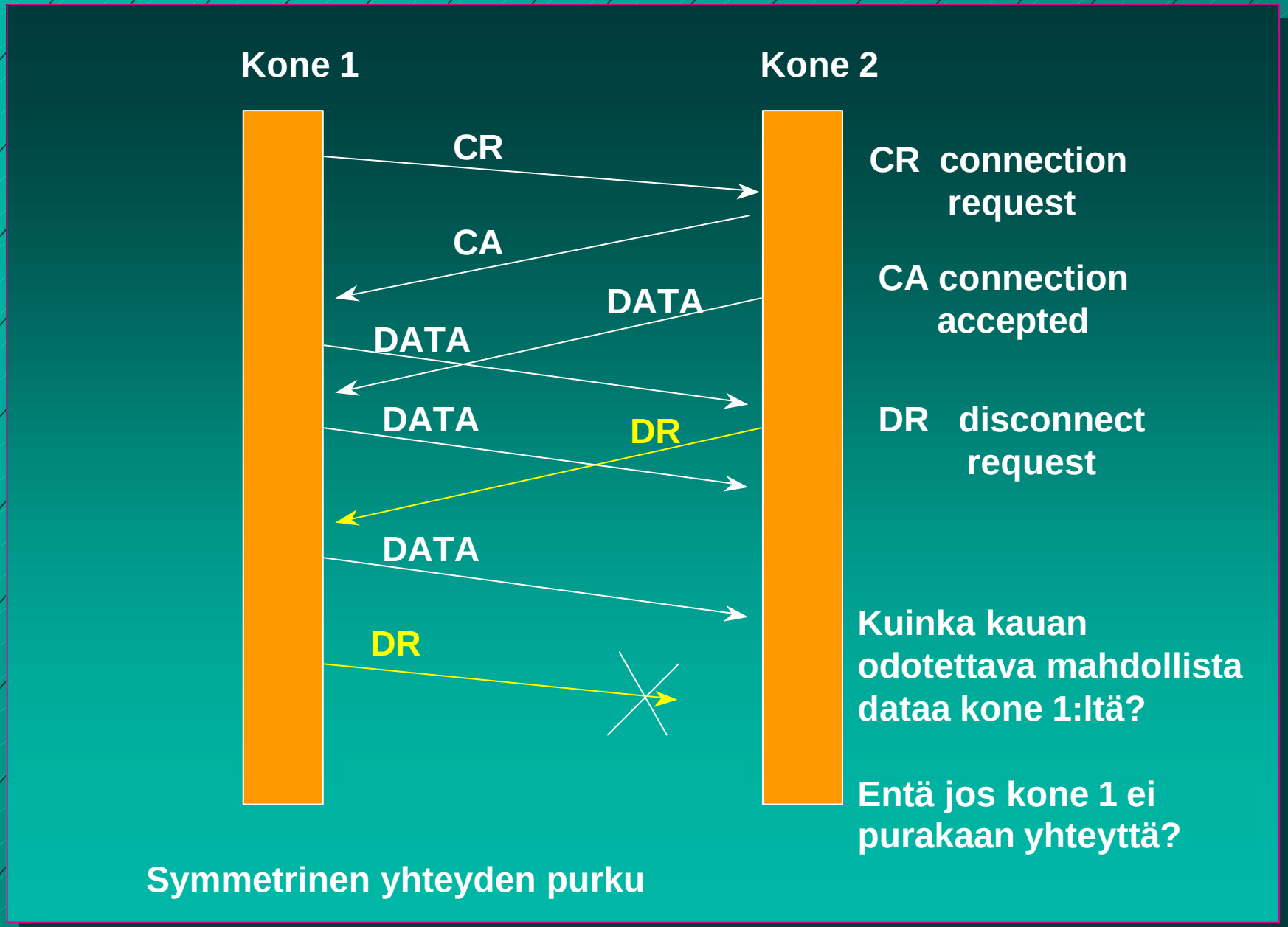


#2 hyökkää vain , jos tietää minun saaneen vastaussanoman.

Entä, jos vastaus ei mene perille? Silloin #1 ei hyökkää!

Loogisesti ratkeamaton ongelma.
Kaikki riippuu aina viimeisestä sanomasta, jonka perillemenoa ei voida taata!

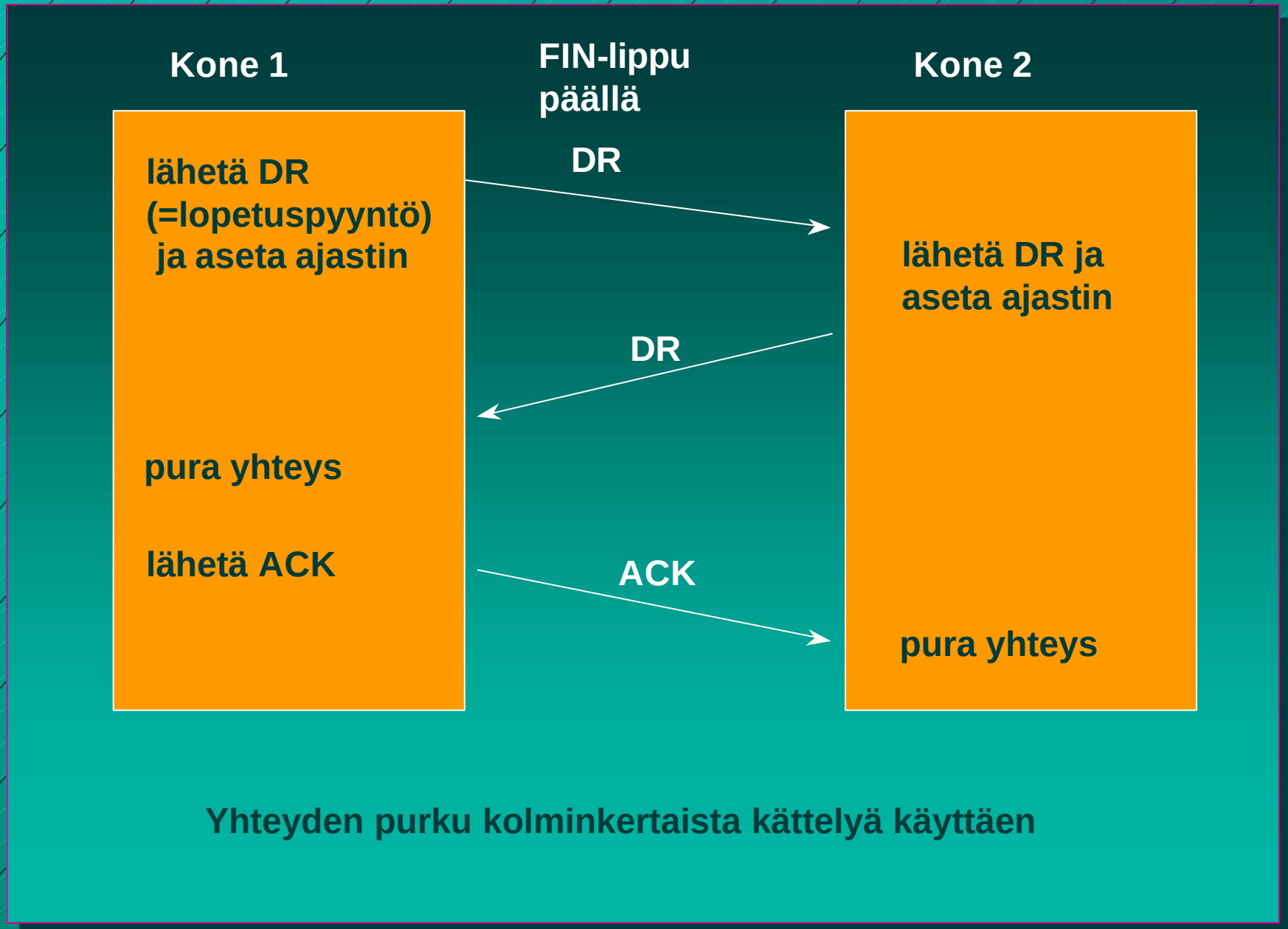
Bysanttilainen ongelma (two-army problem)



Symmetrinen yhteyden purku

Yhteyden purku

- molemmat suunnat puretaan erikseen
- TCP-segmentti
  - FIN = 1
  - ei enää dataa lähetettävä
  - kun saadan kuittaus => yhteys tähän suuntaan purettu
  - yhteys kokonaan purettu, kun molemmat suunnat purettu
- purussa käytetään ajastimia
  - 2 * paketin maksimaalinen elinikä


Kone 1
FIN-lippu päällä
Kone 2
lähetä DR (=lopetuspyyntö) ja aseta ajastin
DR
lähetä DR ja aseta ajastin
DR
pura yhteys
lähetä ACK
ACK
pura yhteys
Yhteyden purku kolminkertaista kättelyä käyttäen

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

- 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

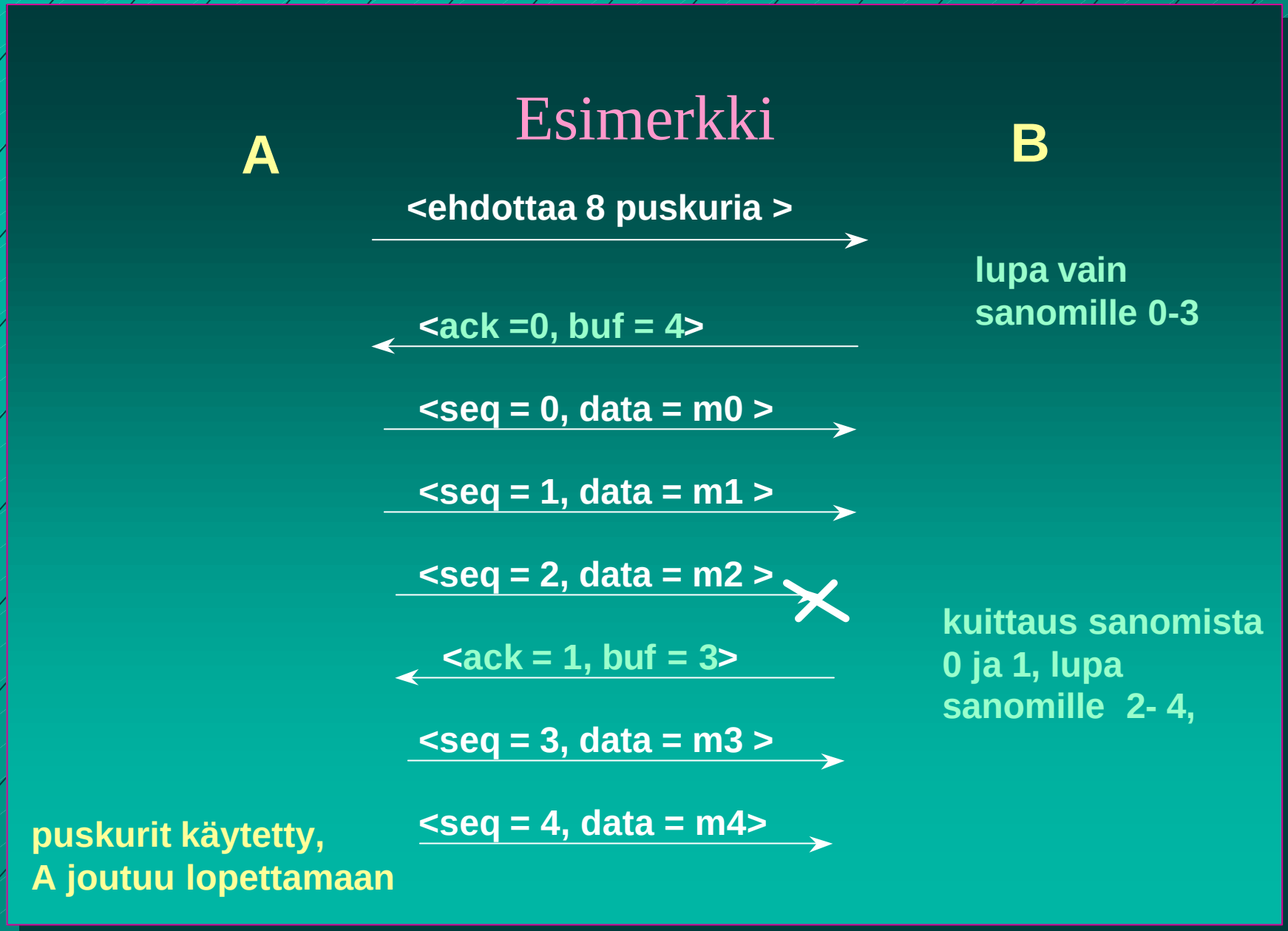
Toistokuittaukset

- Ensikuittaus
  - tähän saakka kaikki OK!
  - ensimmäisen kerran saatava
- toistokuittaus (duplicate ACK)
  - väärässä järjestyksessä saatu segmentti tai virheellinen segmentti => toistetaan uudestaan jo annettu kuittaus
    - NAK-kuittauksen korvike
    - 3 toistokuittausta => segmentti kadonnut tai virheellinen

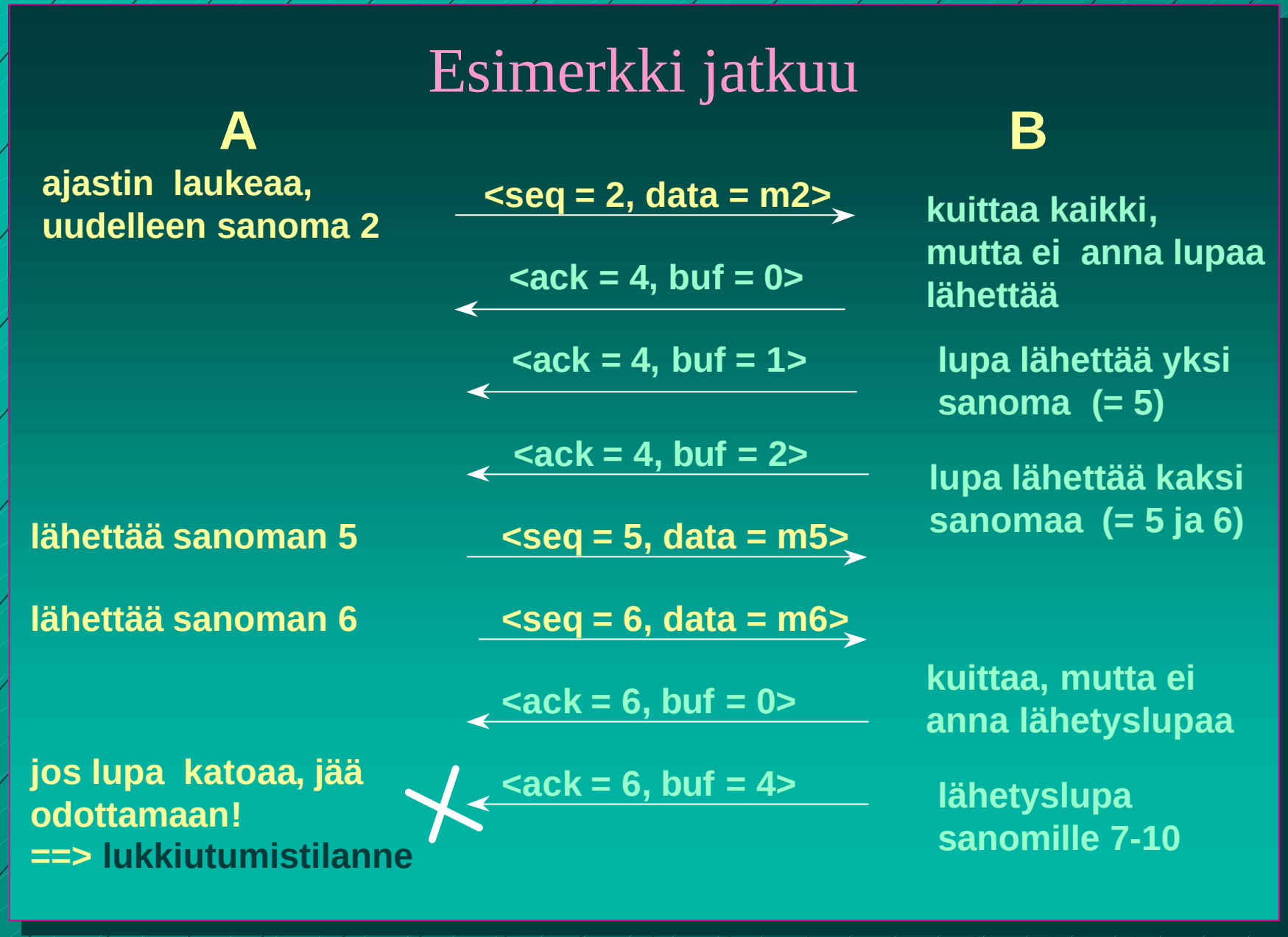
TCP:n vuonvalvonta

- ‘joustava’ liukuva ikkuna (sliding window) (credit-vuonvalvonta)
- vastaanottaja kertoo, kuinka paljon suostuu vastaanottamaan
  - => kuittaus irroitettu vuonvalvonnasta
    - AdvertisedWindow-kenttä
      - paljonko saa lähettää = paljonko vastaanottajan puskureihin mahtuu
- myös ruuhkan valvonta rajoittaa lähettämistä

Esimerkki


A
B
<ehdottaa 8 puskuria >
lupa vain sanomille 0-3
<ack =0, buf = 4>
<seq = 0, data = m0 >
<seq = 1, data = m1 >
<seq = 2, data = m2 >
kuittaus sanomista 0 ja 1, lupa sanomille 2- 4,
<ack = 1, buf = 3>
<seq = 3, data = m3 >
<seq = 4, data = m4>
puskurit käytetty, A joutuu lopettamaan

Esimerkki jatkuu



- 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

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

- Ratkaisu: Naglen algoritmi
  - jos data tulee tavuttain
    - lähetä 1. tavu
    - kerää sitä seuraavat tavut puskuriin ja lähetä vasta kun edellinen lähetys on kuitattu
    - paitsi jos lähetettävää on suurimman segmentin verran tai puolet ikkunan koosta
  - hankala, jos hiirtä liikutellaan Internetin kautta!

Silly window syndrome

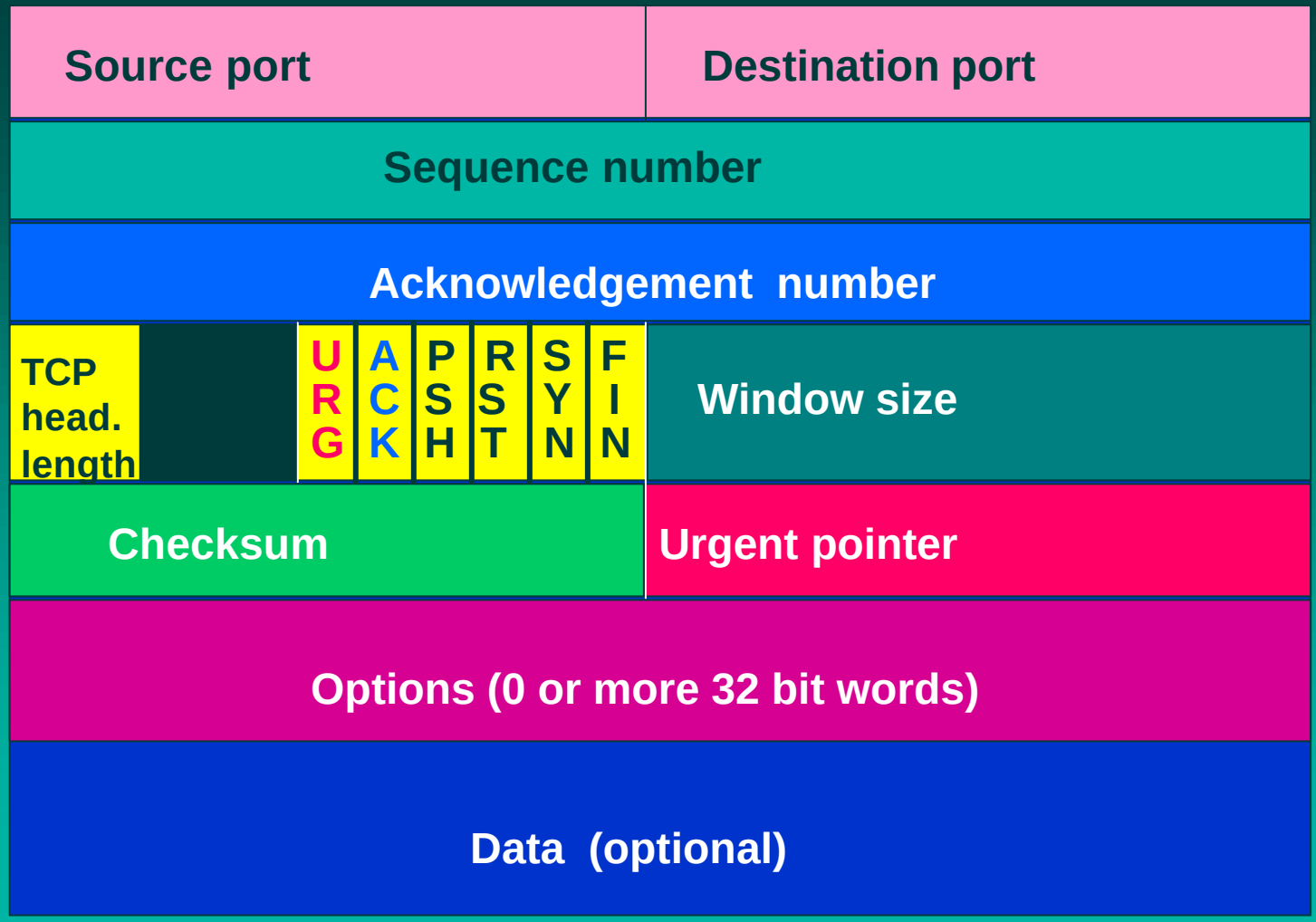
- Tilanteessa, jossa
  - lähettäjältä dataa TCP:lle suurina lohkoina
  - vastaanottajalle mahtuu vain tavu kerrallaan
- voi tuhota TCP:n suorituskyvyn
  - koko data lähetetään tavu kerrallaan
  - joka tavun välissä ilmoitus ikkunan koon kasvattamisesta yhdellä
- Siis: ei ilmoitusta yhdestä tavusta, lähettäjä ei lähetä yhtä tavua
  - koko segmentti
  - puolet puskurin koosta

Silly winwow syndrome

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

TCPv4-otsakkeen kentät



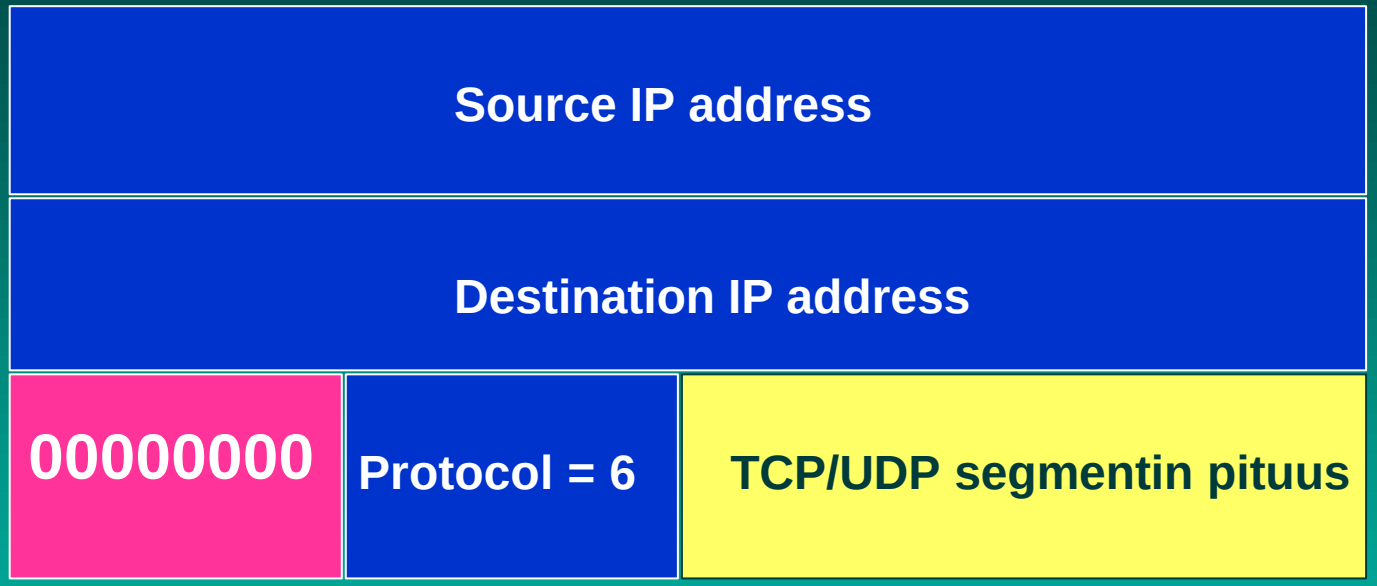
TPC-segmentin otsakekentät

- **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.

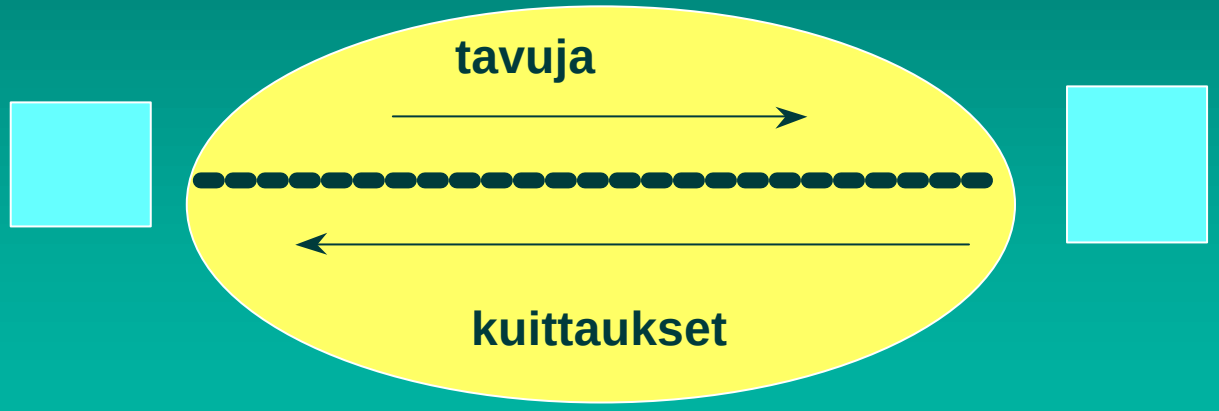
Sisältää IP-otsakkeen tietoja!

- **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 uudelleenlähetyksiä

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

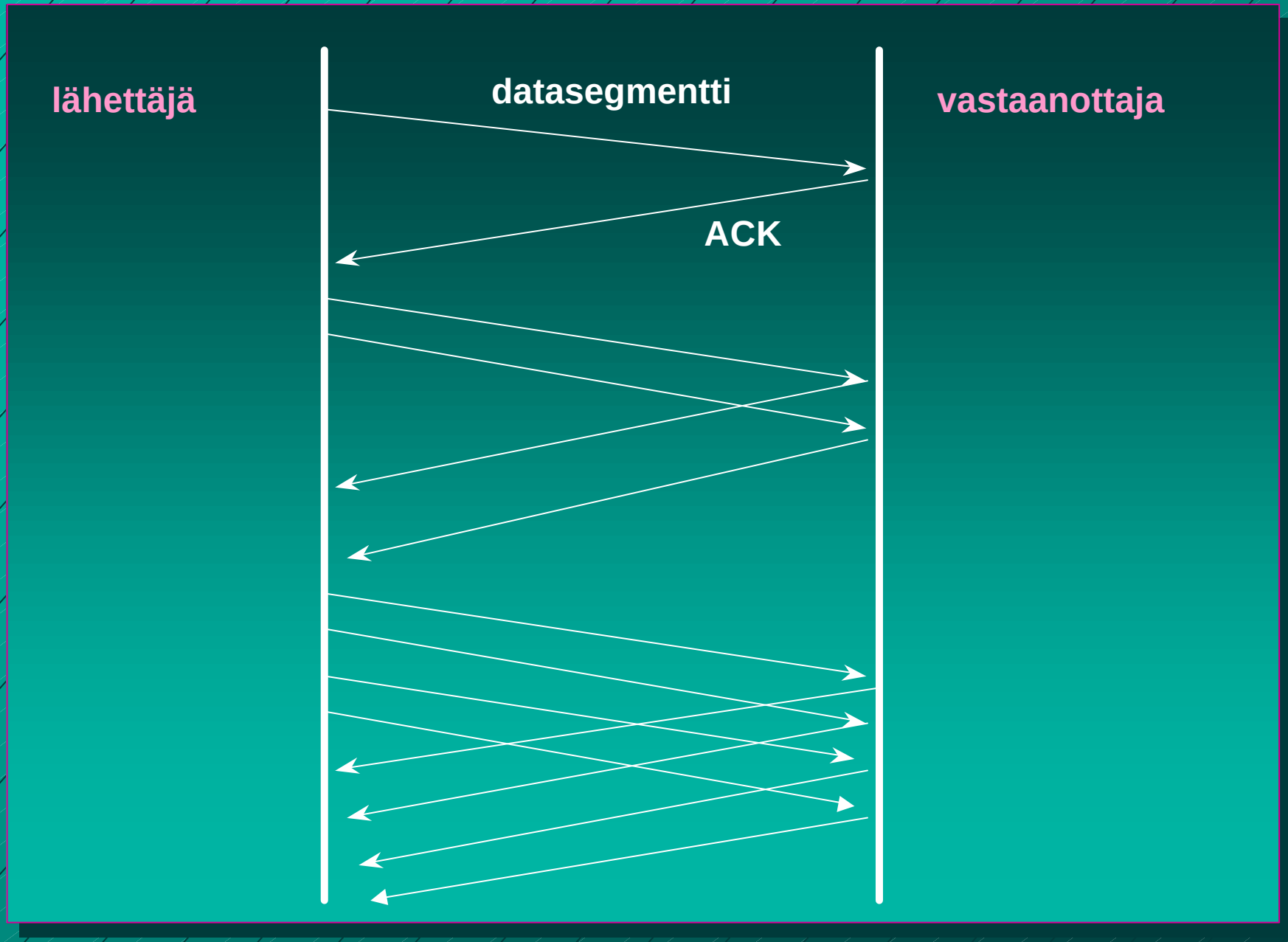
- ruuhkaikkuna
  - “paljonko tavuja (segmenttejä) lähettäjällä saa korkeintaan olla verkossa liikkeellä”
  - kuittaus => ko. tavut jo poistuneet verkosta



- Ruuhkaikkunan koko?
  - Lähettäjän on itse pääteltävä ja arvioitava sopiva ruuhkaikkunan koko
    - kukaan muu ei sitä kerro!
    - timeout => on ruuhkaa
    - kuittaukset tulevat tasaisesti => ei ole ruuhkaa
- Dynaaminen ruuhkaikkunan koko:
  - ruuhkaikkunaa kasvatetaan kunnes törmätään ruuhkaan
  - sen jälkeen ruuhkaikkunaa pienennetään reilusti
  - ja aletaan uudestaan kasvattaa ruuhkaikkunaa

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


lähettäjä
datasegmentti
vastaanottaja
ACK