Aalto-yliopisto
Perustieteiden
korkeakoulu

Kuljetuskerros

CSE-C2400 Tietokoneverkot

28.1.2014 (osa 1)

4.2.2014 (osa 2)

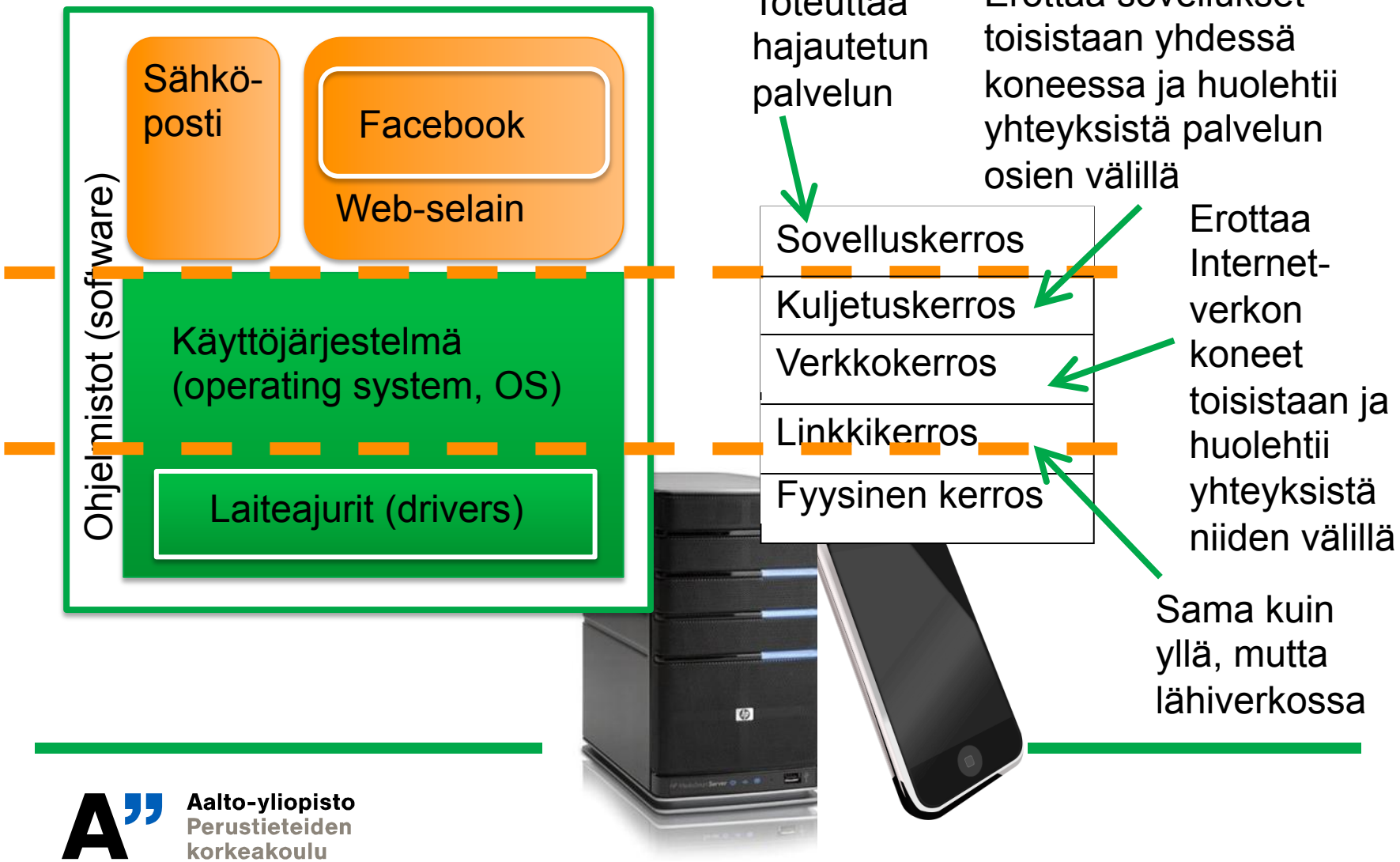
Matti Siekkinen

Osa sisällöstä adaptoitu seuraavista lähteistä: J.F. Kurose and K.W. Ross: Computer Networking: A Top-Down Approach 6th ed. -kirjan lisämateriaali

Lyhenteitä ja terminologiaa

- UDP = User Datagram Protocol: epäluotettava kulj.kerroksen protokolla
- TCP = Transmission Control Protocol: luotettava kulj.kerroksen protokolla
- RTT = Round-Trip Time: kiertoviive eli aika joka kuluu paketin lähettämisestä vastauksen saamiseen
- MSS = Maximum Segment Size: suurin sallittu TCP:n segmenttikoko
- ARQ = Automatic Repeat reQuest: kuittauksiin ja uudelleenlähetyksiin perustuva virheenkorjausmenetelmä
- FEC = Forward Error Correction: redundantin datan laskemiseen ja lähettämiseen perustuva virheenkorjaus
- ACK = acknowledgment: kuittaus onnistuneesta lähetyksestä
- NACK= Negative acknowledgment: kuittaus epäonnistuneesta lähetyksestä
- SACK = Selective acknowledgments: TCP-optio selektiiviseen kuittaukseen (kumulatiivisen lisäksi)
- AIMD = Additive Increase, Multiplicative Decrease: TCP:n perusversion ruuhkanhallinnan perusperiaate
- CA = Congestion Avoidance: ruuhkanvälttelyvaihe TCP:n ruuhkanhallinnassa
- SS = Slow Start: nopeasti lähetysnopeuden kasvattamisen vaihe TCP:n ruuhkanhallinnassa
- cwnd = congestion window: TCP:n ruuhkaikkuna

Internet-protokollapino



Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

Näiden kahden luennon jälkeen…

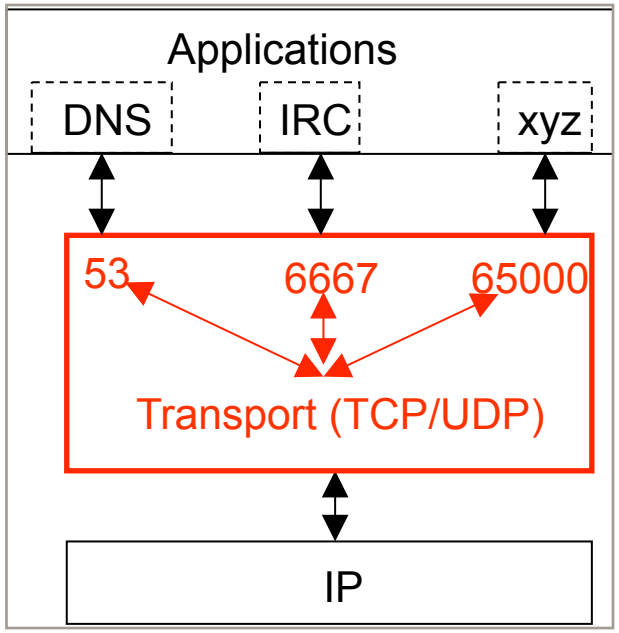
- Ymmärrätte:
  - kuljetuskerroksen tehtävän ja toiminnan
  - luotettavan tiedonsiirron erityyppiset menetelmät
  - UDP:n ja TCP:n toimintaperiaatteet
  - Mitä on ruuhkanhallinta ja miksi sitä tarvitaan
  - Minkälaista ruuhkanhallintamekanismia TCP käyttää
- Tiedostatte:
  - Perusruuhkanhallintamekanismin rajoitukset

Kuljetuskerroksen tehtävä

- Kuljetuskerros yhdistää sovelluksia
  - Viestejä päätelaitteen sovelluksesta toiseen (end-to-end)
  - Aktiivisia sovelluksia voi olla monia yhtäaikaa yhdessä päätelaitteessa
- Kuljetuskerros tarjoaa sovelluksille erilaisia palveluita
  - Luotettava/epäluotettava tiedonsiirto
  - Viestinvälitys (datagrammi) tai tavuvirta
- Kuljetuskerros toteutetaan eri protokollilla, jotka ovat vaihtoehtoisia
  - TCP tarjoaa luotettavan tavuvirran palveluna sovellukselle
  - UDP tarjoaa epäluotettavan viestinvälityksen palveluna
  - Myös muita, ei käsitellä tällä kurssilla

Kuljetuskerroksen ominaisuuksia

- Portti
  - Jokaisella päätelaitteella on osoite (IP)
  - Portti (16-bittinen numero) identifioi sovelluksen päätelaitteessa
  - Monta aktiivista yhtäaikaisesti
  - Well-known port numbers: 0-1023
    - Varattuja, esim. 80=HTTP, 53=DNS
    - Internet Assigned Numbers Authority: www.iana.org

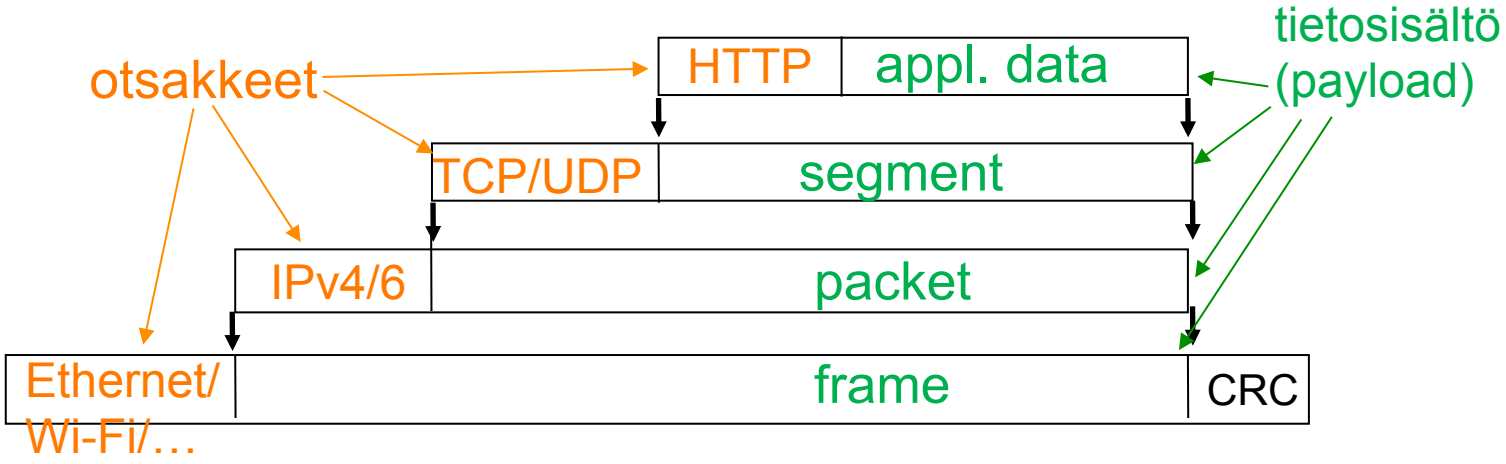


Kuljetuskerroksen ominaisuuksia

- Socket rajapinta
  - Sovelluksen ja kuljetuskerroksen protokollan välissä
  - Porttinumero määräytyy sokettia luodessa
- Data välitetään segmentteinä
  - UDP viesti, TCP tavuvirran osa
  - Kapseloidaan pakettiin alemmalla kerroksella (IP)

Kapselointi (encapsulation)

- Ylemmän kerroksen protokollan viesti ”kapseloidaan” alemman kerroksen viestin sisään
  - Otsake (header) eteen
  - Dekapseloidaan toisessa päässä
- Sovelluksen lähettämä data kapseloidaan kuljetuskerroksen (TCP tai UDP) segmenttiin



Mitä hyötyä?

Kapselointi (encapsulation)

- Miten sovellusprotokolla tietää kuljetuskerroksen protokollan?
  - Sokettia luodessa määritellään sokettityyppi
- Miten kuljetuskerroksen protokolla tietää IP version?
  - Sokettia luodessa määritellään osoiteavaruus
- Miten IP protokolla tietää oikean verkkorajapinnan (linkki +fyysinen kerros)?
  - Asiakasohjelmassa yleensä määräytyy automaattisesti
    - Esim. soketin muodostaessa yhteyden (connect())
  - Soketti voidaan myös halutessa sitoa tiettyyn rajapintaan (palvelin)
    - bind()

Sisältö

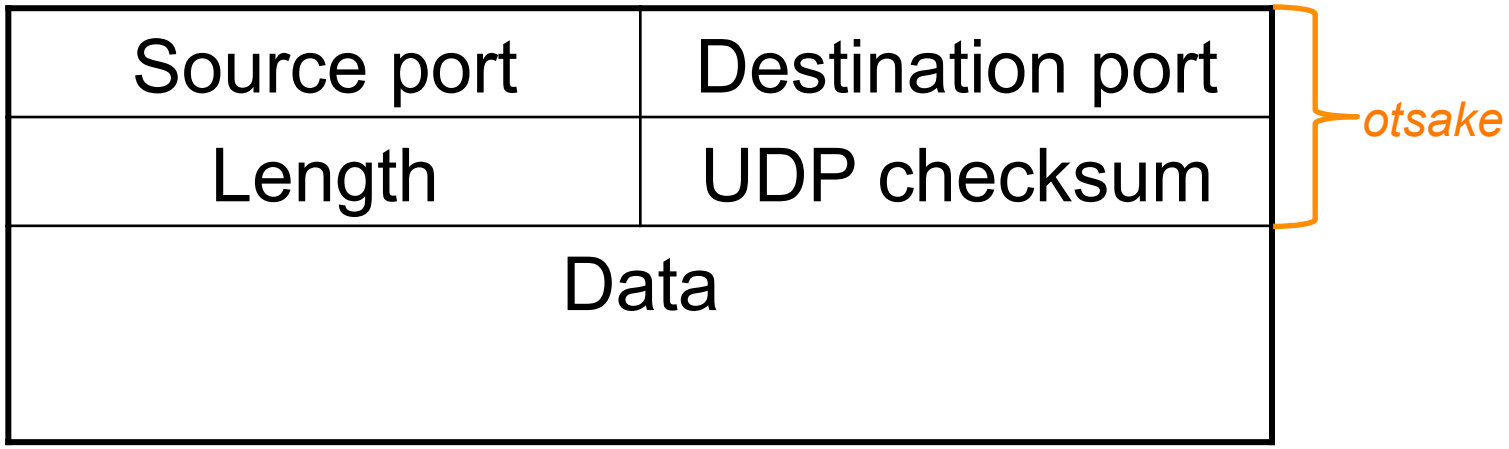
- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

UDP

- User Datagram Protocol
- Standardi RFC-768
- UDP tarjoaa epäluotettavan yhteydettömän kuljetuspalvelun
  - Kevyt, ei tilaa, ei yhteydenmuodostusta, helppo toteuttaa
- Datagrammien välitys päätelaitteessa
  - Kohdeosoitteen ja kohdeportin avulla

UDP

- UDP välittää datagrammeja (viesti)



- Tarkistussummaan lasketaan sekä otsake että data
  - Ei ole välttämätön
- UDP-sovelluksia: DNS, Radius, NTP, RTP (VoIP)

UDP-kaappaus: dig (DNS)

```
siekkine@b128-dell:~$ dig www.hs.fi

; <<>> DiG 9.4.2-P2 <<>> www.hs.fi
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 50872
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 4, ADDITIONAL: 4

;; QUESTION SECTION:
;www.hs.fi.                     IN      A

;; ANSWER SECTION:
www.hs.fi.              600     IN      A       194.137.237.63

;; AUTHORITY SECTION:
hs.fi.                  600     IN      NS      ns4.sanoma.fi.
hs.fi.                  600     IN      NS      ns3.sanoma.fi.
hs.fi.                  600     IN      NS      ns2.sanoma.fi.
hs.fi.                  600     IN      NS      ns1.sanoma.fi.

;; ADDITIONAL SECTION:
ns1.sanoma.fi.          27395   IN      A       194.137.237.33
ns2.sanoma.fi.          27395   IN      A       194.137.237.34
ns3.sanoma.fi.          58894   IN      A       195.165.77.83
ns4.sanoma.fi.          58894   IN      A       195.165.77.84

;; Query time: 54 msec
;; SERVER: 130.233.192.1#53(130.233.192.1)
;; WHEN: Thu Feb 18 22:01:29 2010
;; MSG SIZE  rcvd: 186
```

UDP-kaappaus: DNS kysely

```
Internet Protocol, Src: 130.233.194.105 (130.233.194.105), Dst: 130.233.192.1 (130.233.192.1)
User Datagram Protocol, Src Port: 36287 (36287), Dst Port: domain (53)
    Source port: 36287 (36287)
    Destination port: domain (53)
    Length: 35
    Checksum: 0x8872 [incorrect, should be 0xc8f9 (maybe caused by "UDP checksum offload"?)]
        [Good Checksum: False]
        [Bad Checksum: True]
Domain Name System (query)
    [Response In: 209]
    Transaction ID: 0xc6b8
    Flags: 0x0100 (Standard query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    Queries
        www.hs.fi: type A, class IN
```

UDP payload

UDP-kaappaus: DNS vastaus

```
Internet Protocol, Src: 130.233.192.1 (130.233.192.1), Dst: 130.233.194.105 (130.233.194.105)
User Datagram Protocol, Src Port: domain (53), Dst Port: 36287 (36287)
    Source port: domain (53)
    Destination port: 36287 (36287)
    Length: 194
    Checksum: 0xd768 [correct]
        [Good Checksum: True]
        [Bad Checksum: False]
Domain Name System (response)
    [Request In: 208]
    [Time: 0.053526000 seconds]
    Transaction ID: 0xc6b8
    Flags: 0x8180 (Standard query response, No error)
    Questions: 1
    Answer RRs: 1
    Authority RRs: 4
    Additional RRs: 4
    Queries
        www.hs.fi: type A, class IN
    Answers
    Authoritative nameservers
    Additional records
```

(sisääntulevalle liikenteelle verkkokortti jo laskenut)

UDP payload

Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

Luotettava tiedonsiirto

- Fyysiset linkit ja reitittimet eivät ole 100% luotettavia
  - Paketeissa voi esiintyä bittivirheitä
  - Esim. langattomat linkit
- Verkkokerros (IP) ei ole luotettava
  - Reittittimet tietoisesti pudottavat paketteja
- Kuljetuskerroksen protokolla varmistaa että lähetetty tieto pääsee ehjänä perille
  - Lähettävältä sovelluksesta vastaanottavalle sovellukselle
  - Segmentit virheettömiä ja oikeassa järjestyksessä

Luotettava tiedonsiirto

- Miksi toteutetaan kuljetuskerroksella?
  - Sovelluskerros
    - Redundanttia samaa toiminnallisuuden toteuttamista
  - Alemmat kerrokset
    - ”Per-hop” (linkki) luotettavuus ei aina riitä
    - Virheitä voi syntyä myös reitittimen muistissa
    - Reitittimet suunniteltu mahdollisimman yksinkertaisiksi

Luotettava tiedonsiirto

- ARQ: Automatic Repeat reQuest
  - Virheenkorjauksen konsepti
  - Oikeastaan joukko tekniikoita
  - mm. TCP hyödyntää tätä
  - Kuittaukset ja segmentin uudelleenlähetys
- Muitakin on..
  - Forward Error Correction (FEC)
  - Hybridit

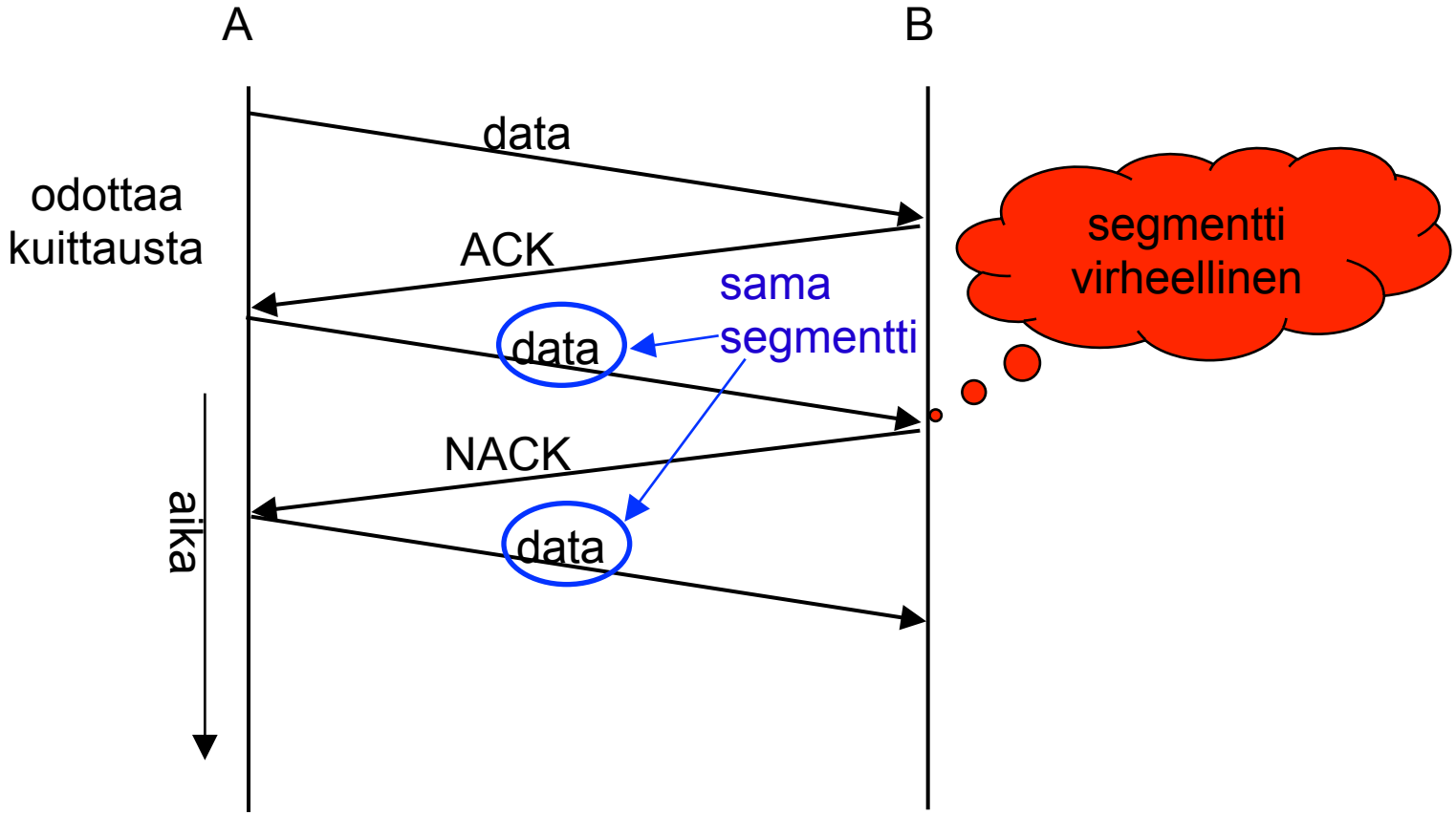
ARQ mekanismit

- Tarkistesumma
  - Virheellisen segmentin havaitseminen
- ACK: positiivinen kuittaus
  - ”Vastaanotin segmentin ok”
- NACK: negatiivinen kuittaus
  - ”Segmentti rikki, lähetä uudelleen”

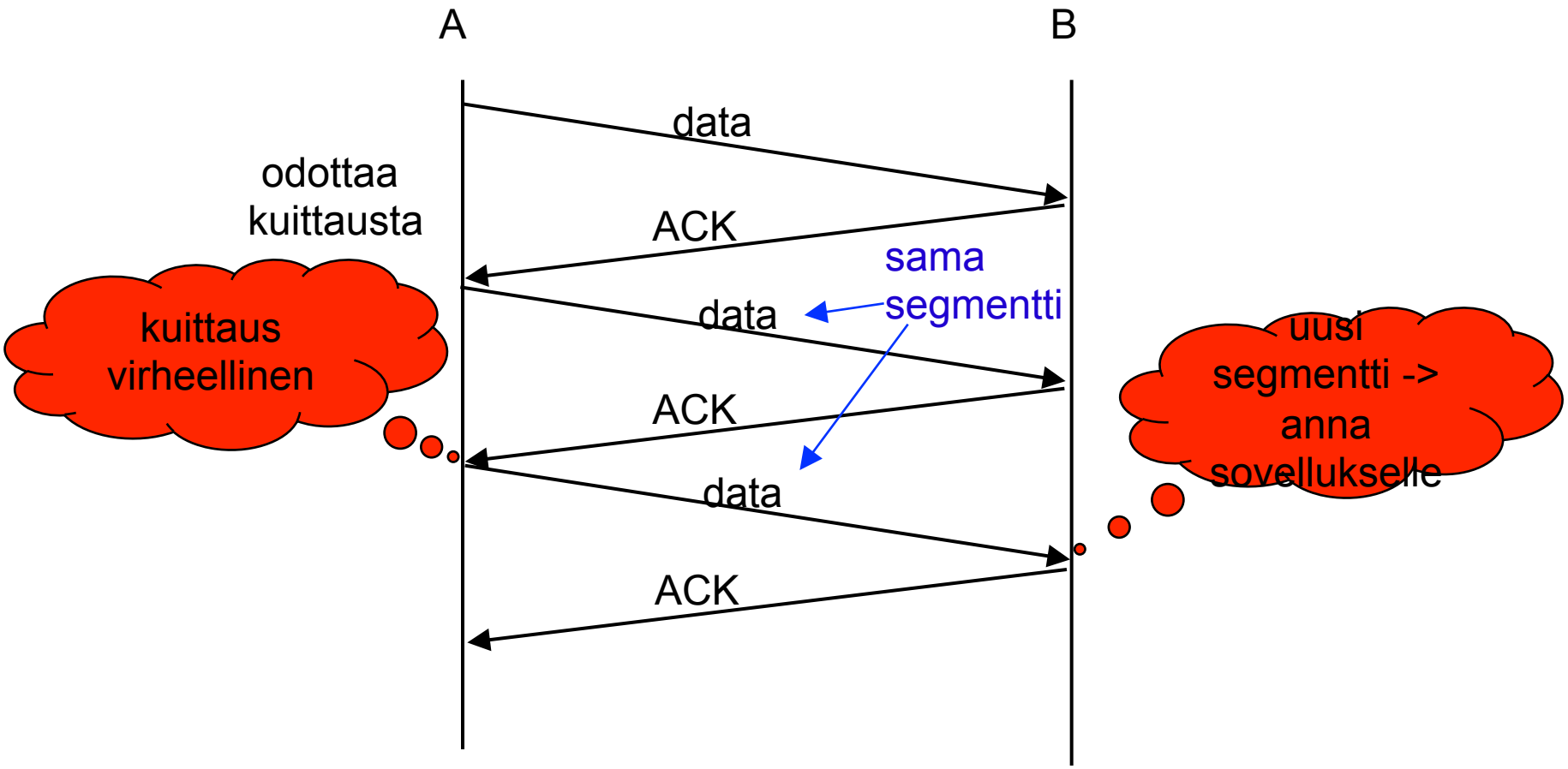
Tarkistesumma

- Havaitaan virheellinen segmentti
  - Lähettäjä laskee ja liittää segmentin otsakkeeseen
  - Vastaanottaja tarkistaa
- Lasketaan pseudo-otsakkeesta ja kuljetuskerroksen segmentistä
- Pseudo-otsake sisältää:
  - lähettäjän ja vottajan IP osoitteet
  - protokollanumero (esim. TCP tai UDP, IP-otsakkeesta)
  - TCP/UDP segmentin pituus
  - Hieman eri menetelmä IPv4 (RFC 768/793) vs. IPv6 (RFC 2460)
  - IP-osoitteet suojaavat väärinreititetyiltä segmenteiltä

Positiivinen ja negatiivinen kuittaus



Myös kuittaus voi olla virheellinen

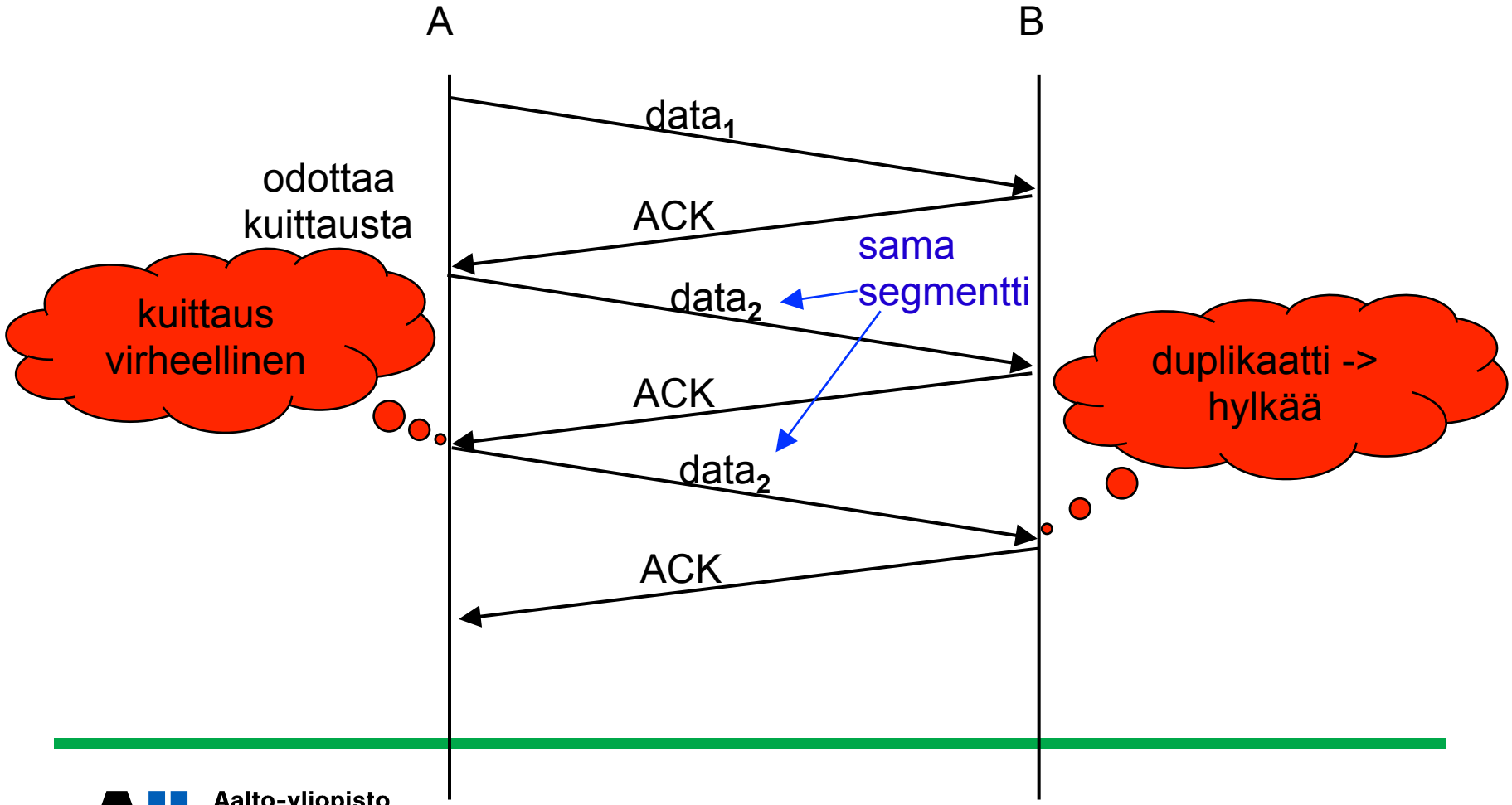


Mitä tarvitaan lisäksi?

ARQ mekanismit

- Tarkistesumma
  - Virheellisen segmentin havaitseminen
- ACK: positiivinen kuittaus
  - ”Vastaanotin segmentin ok”
- NACK: negatiivinen kuittaus
  - ”Segmentti rikki, lähetä uudelleen”
- Sekvenssinumerot
  - Erottaa uuden datan uudelleenlähetetystä
  - Esim. korruptoitunut ACK

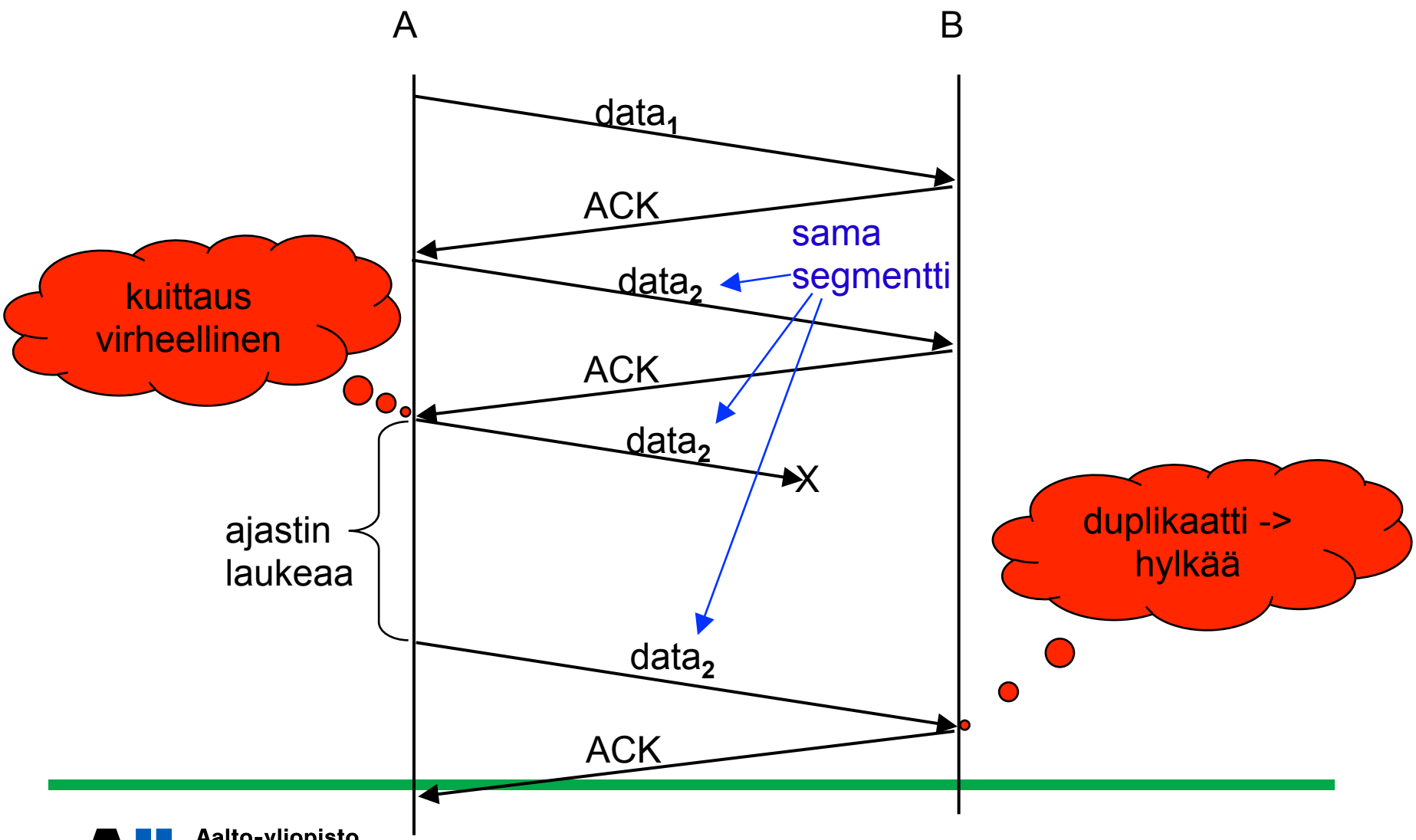
Sekvenssinumerot estävät väärinkäsityksen



Entä jos paketti katoaa matkalla?

ARQ mekanismit

- Tarkistesumma
  - Virheellisen segmentin havaitseminen
- ACK: positiivinen kuittaus
  - ”Vastaanotin segmentin ok”
- NACK: negatiivinen kuittaus
  - ”Segmentti virheellinen, lähetä uudelleen”
- Sekvenssinumerot
  - Erottaa uuden datan uudelleenlähetetystä
  - Esim. virheellinen ACK
- Ajastimet
  - Lähetä uudelleen paketti jollei kuulu kuittausta
  - Kadonneiden pakettien aiheuttamat tilanteet


A
B
data1
ACK
sama
segmentti
kuittaus
virheellinen
data2
ACK
data2
X
ajastin
laukeaa
duplikaatti ->
hylkää
data2
ACK

Sisältö

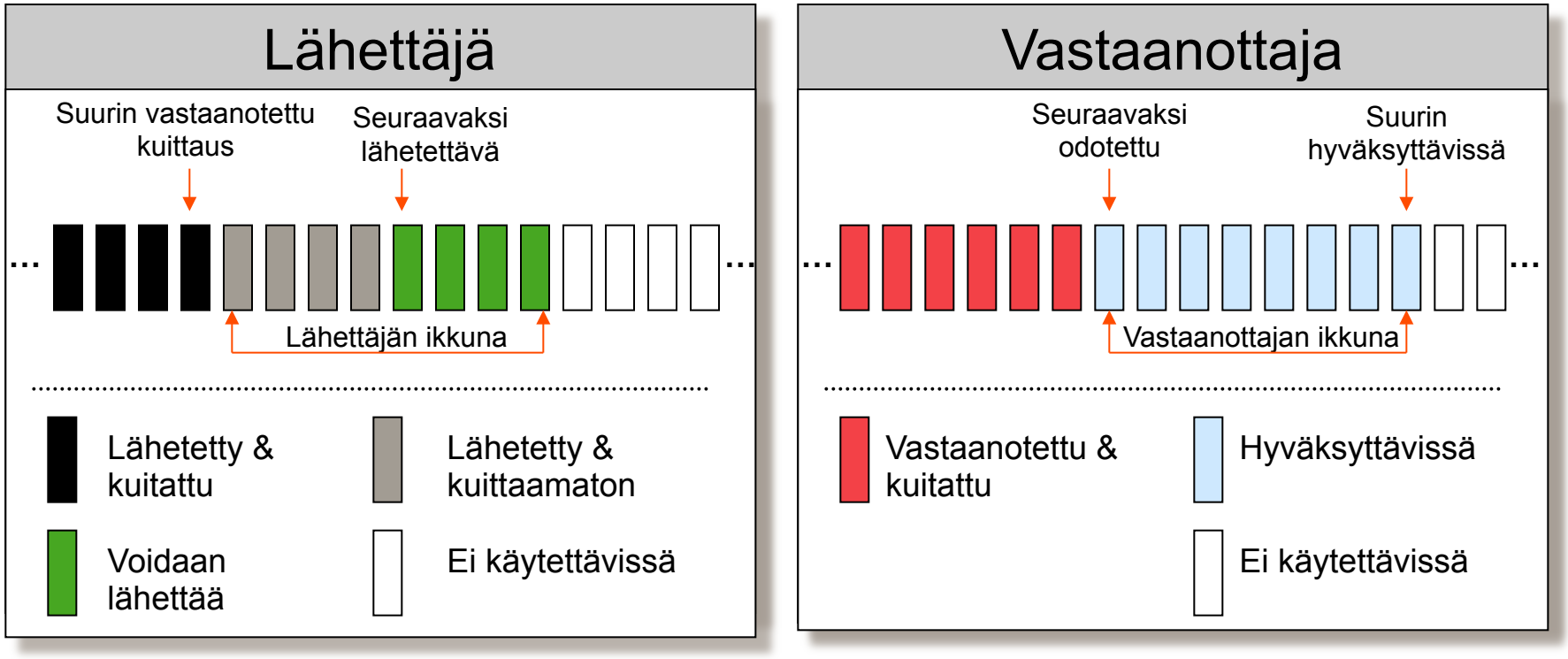
- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

Stop-and-wait

- Vain yksi segmentti matkalla kerrallaan
  - Uusi lähetetään kuittauksen tai ajastimen laukeamisen jälkeen
- 1-bittinen sekvenssinumero riittää
  - Uusi segmentti, sekvenssinro vaihtuu
  - Uudelleenlähetys, sama sekvenssinro
- Ei ole kovinkaan tehokas
  - Verkon käyttöaste jää matalaksi
  - Ratkaisu: segmenttien ”putkitus” (pipelining)

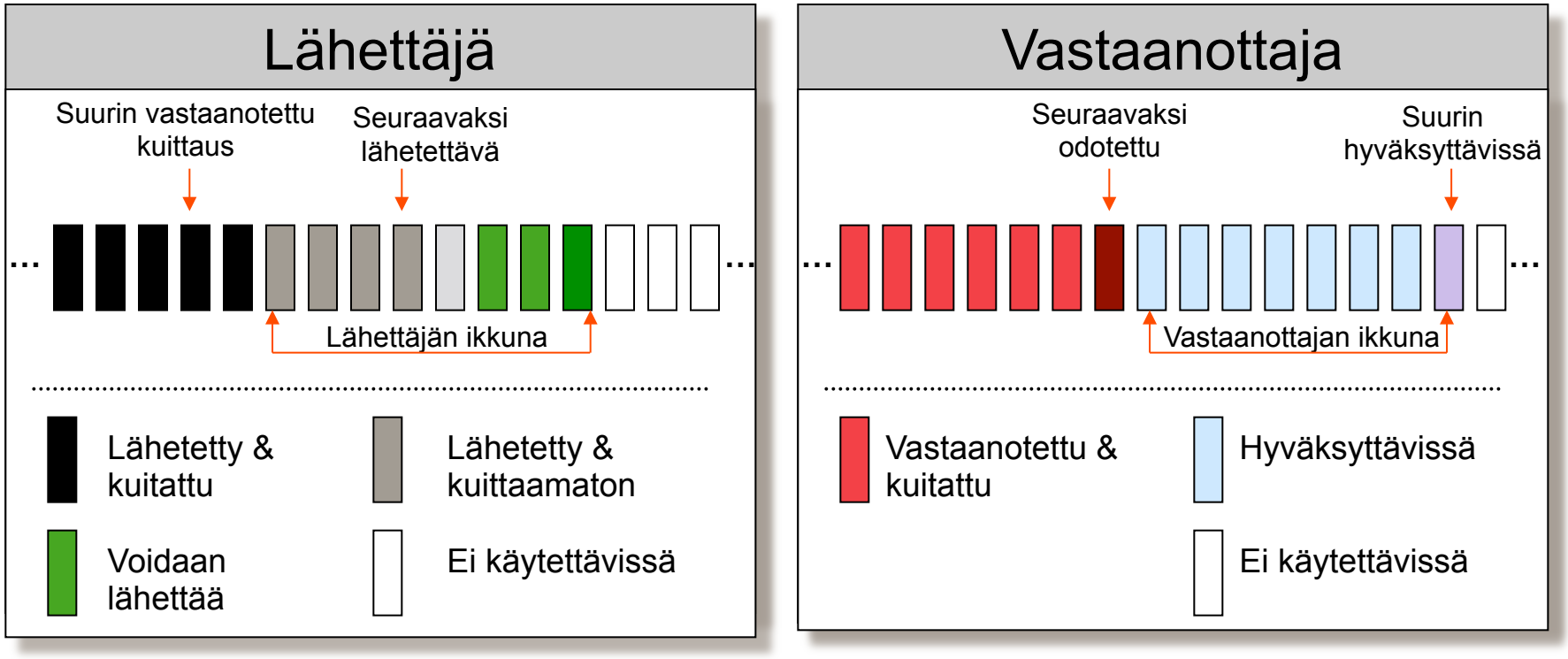
Putkitus

- Useita segmenttejä matkalla yhtäaikaisesti
- Tarvitaan riittävä numeroavaruus sekvenssinumeroille
- Liukuva ikkuna (sliding window)



Putkitus

- Useita segmenttejä matkalla yhtäaikaisesti
- Tarvitaan riittävä numeroavaruus sekvenssinumeroille
- Liukuva ikkuna (sliding window)



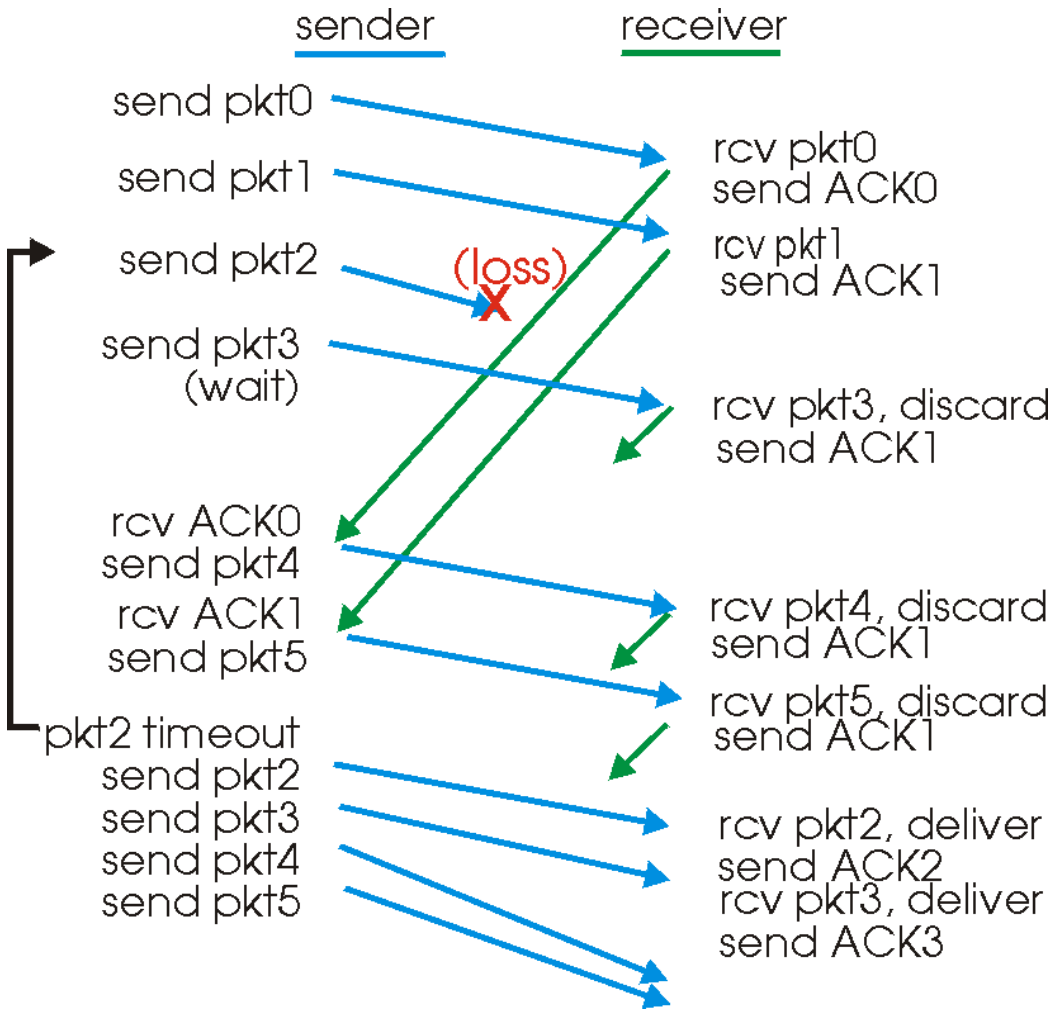
Putkitus

- Go-Back-N ja Selective Repeat protokollat
  - Huomattavasti parempi käyttöaste
  - Käyttöaste riippuu viiveestä, kaistanleveydestä ja ikkunankoosta

Go-Back-N

- Segmentin kadotessa se ja kaikki sen jälkeen jo lähetetyt uudelleenlähetetään (eli go-back-n)
  - Lähettäjä havaitsee kehyksen katoamisen aikakatkaisulla (timeout)
  - Lähettäjällä ikkunan suuruinen puskuri
  - Vastaanottaja ei puskuroi mitään
- Kumulatiiviset kuittaukset
  - Vastaanottaja kuittaa vain oikeassa järjestyksessä saapuvat segmentit
  - Ehjänä mutta väärässä järjestyksessä vastaanotetut hylätään
  - Kuittauksen katoaminen ei vaarallista
    - Myöhempi kuittaus korvaa sen

Go-Back-N

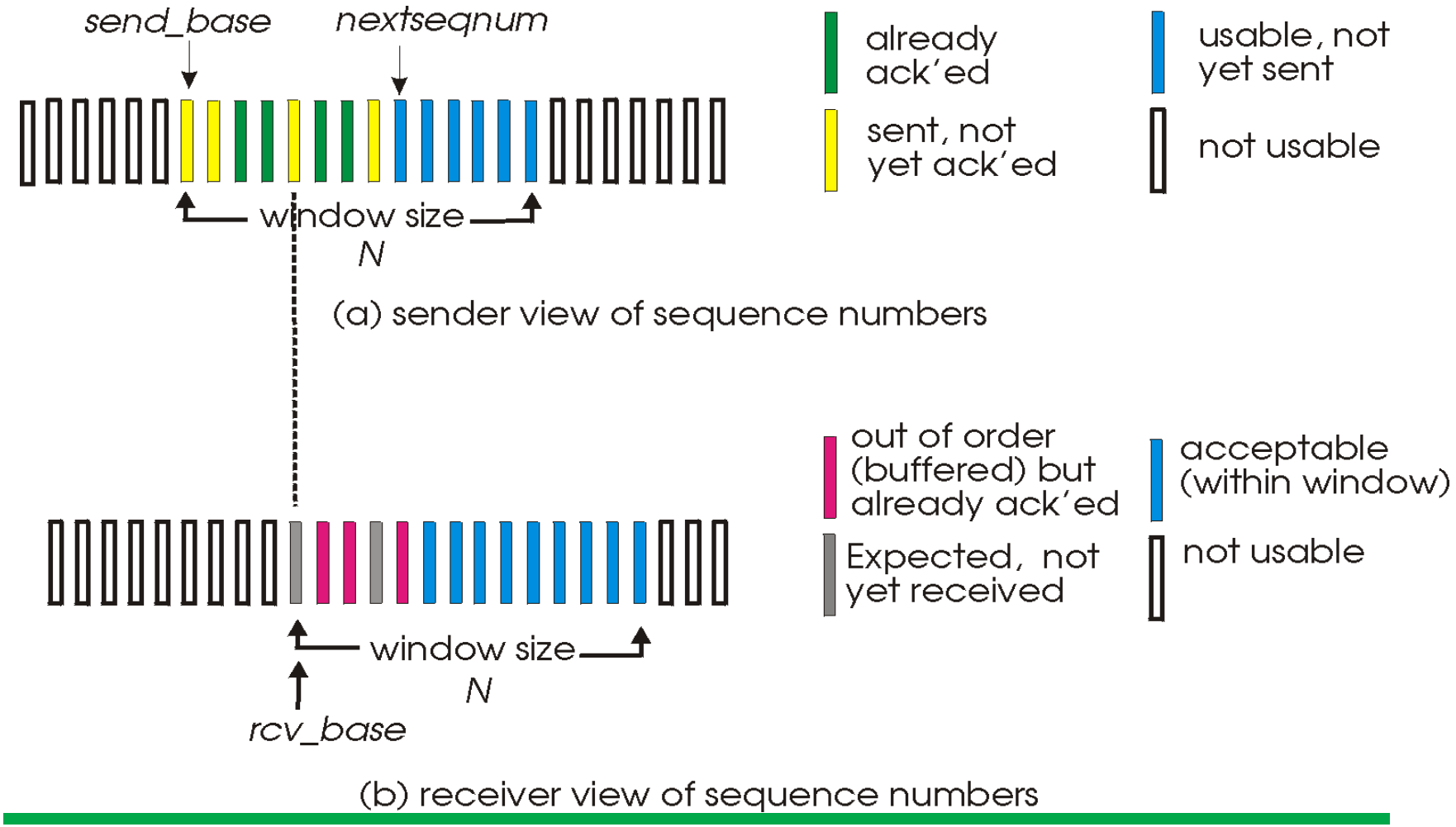


Mitä vikaa?

Selective Repeat

- Go-Back-N tehokkuus kärsii jos pitkä viive ja iso kaistanleveys
  - Yksi kadonnut segmentti aiheuttaa paljon turhia uudelleenlähetyksiä
- Selective Repeat
  - Vastaanottaja kuittaa erikseen jokaisen segmentin
  - Lähettäjä uudelleenlähettää vain kuittaamattomat segmentit
  - Vastaanottajalla on oltava riittävän suuri puskuri

Selective Repeat



Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

TCP

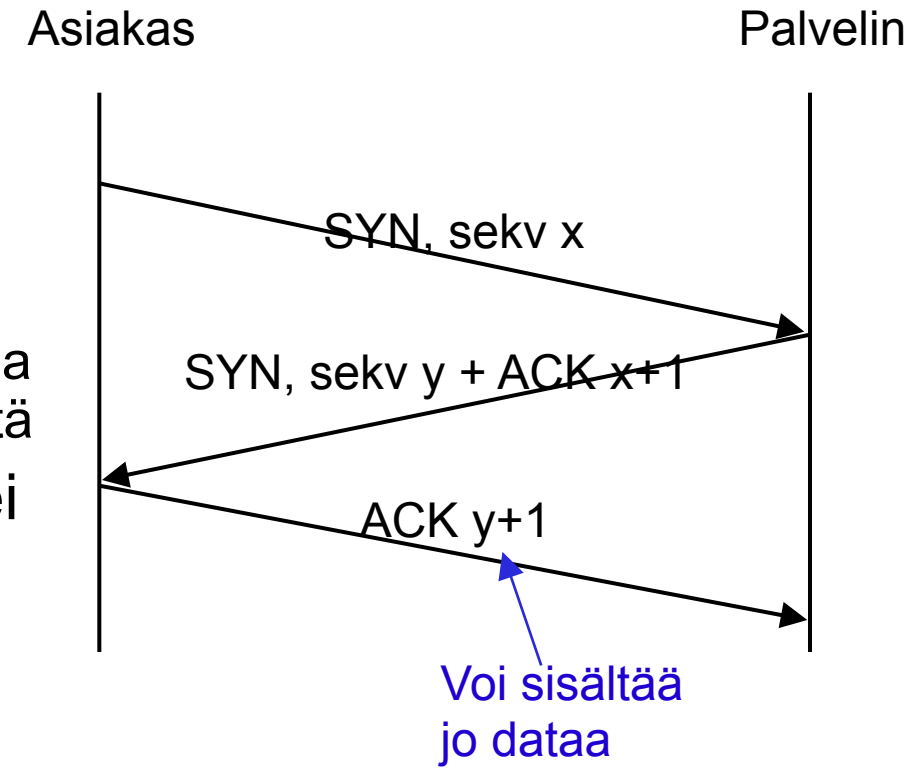
- Transmission Control Protocol
- Standardi RFC-793
- Yhteydellinen protokolla
- Full duplex
  - Sovellusdataa molempiin suuntiin samanaikaisesti
- Luotettava tavuvirta
  - Jakaa sovelluksen lähettämän tavuvirta segmentteihin
  - Nämä kapseloidaan IP-paketeiksi
- Ominaisuuksia
  - Kolmivaiheinen yhteyden muodostus
  - ARQ virheenkorjaus
  - Vuonhallinta
  - Ruuhkanhallinta
- Kuljettaa 99% (lonkalta arvio) Internetin sovellusliikenteestä

TCP yhteys

- Yksiselitteisesti identifioidaan neljällä parametrilla
  - Lähettäjän ja vastaanottajan osoitteet ja porttinumerot
- Segmenttien ohjaus (demux) päätelaitteessa
  - Kaikki neljä parametria tarkistetaan
  - Eroaa UDP:sta
- TCP soketti
  - Vastaanottava soketti palvelun portissa
    - Esim. portti 80 Web-palvelimessa
  - Uusi soketti luodaan välittömästi uutta yhteyttä muodostettaessa

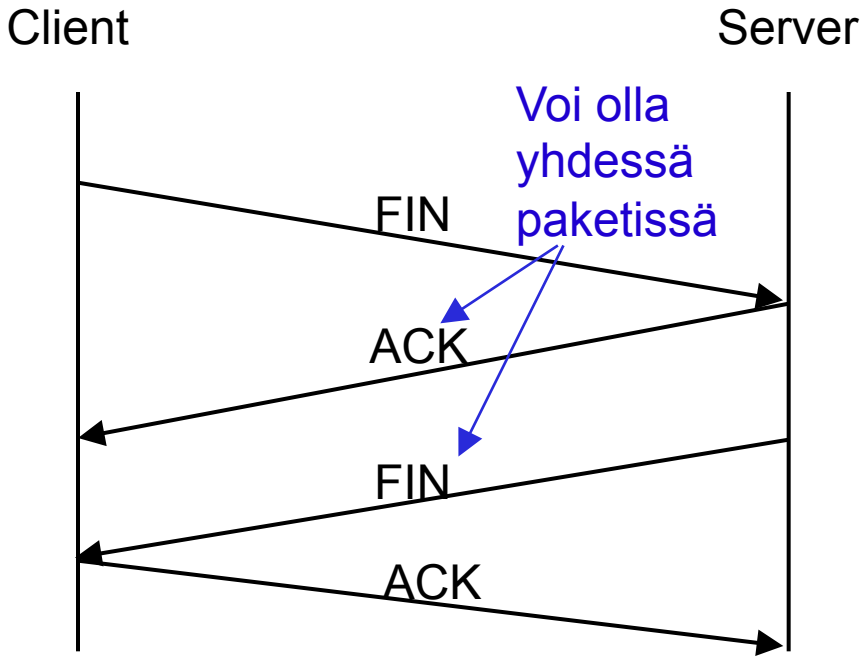
TCP-yhteyden muodostaminen

- “three-way handshake”
- SYN-paketit alustavat sekvenssinumerot satunnaisluvuilla x ja y
  - Mahdolliset vanhat vielä matkalla olevat paketit eivät sotke yhteyttä
- Kolmas paketti varmistaa ettei palvelin jää turhaan odottelemaan asiakasta
  - SYN+ACK häviää tai asiakas keskeyttää



TCP-yhteyden sulkeminen

- Kumpi tahansa osapuoli voi aloittaa sulkemisen
- Molemmat ”simplex-yhteydet” suljetaan erikseen
- FIN paketin lähettämisen jälkeen ajastin käynnistyy
  - Yhteys ei jää roikkumaan auki kuittausten hävitessä

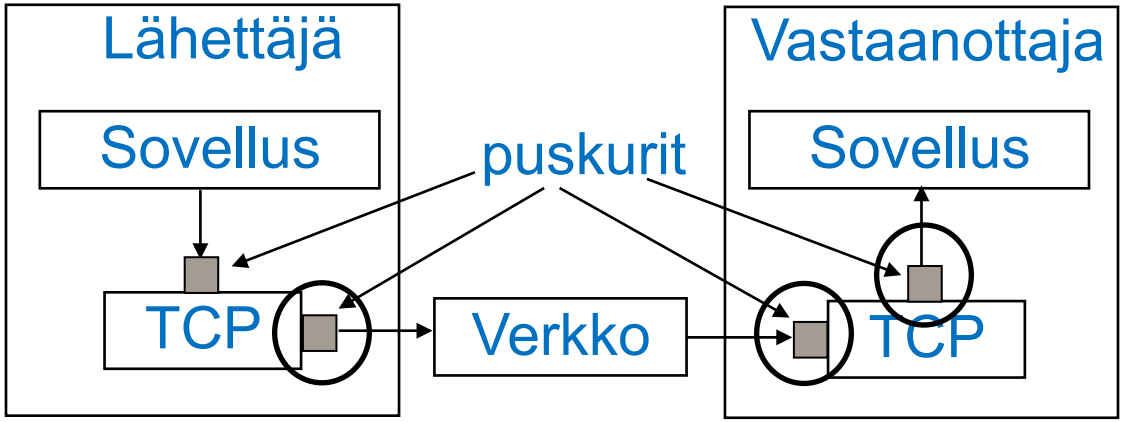


Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

TCP:n toiminta

- Kolme päätehtävää kun yhteys on muodostettu
  1. Virheenkorjaus
     - Epäluotettavan verkkokerroksen takia
  2. Vuonhallinta
     - Otetaan huomioon hidas vastaanottava sovellus
  3. Ruuhkanhallinta
     - Vältetään verkon ylikuormitustilanteita



TCP virheenkorjaus

- Go-Back-N tyyppinen ARQ
  - Ajastimet, tarkistussummat, uudelleenlähetykset
  - Suurin ero: TCP uudelleenlähettää vain kadonneet segmentit
    - Vastaanottaja puskuroi myös epäjärjestyksessä tulleita segmenttejä
- Kumulatiiviset (positiiviset) kuittaukset
  - Indikoi mitä sekvenssinumeroa vastaanottaja odottaa seuraavaksi
  - Lasketaan tavuina aloitussekvenssinumerosta
  - Viivästetyt kuittaukset (delayed ACK)
    - 1 kuittaus per 2 täyttä segmenttiä
- Lisäksi on mahdollista käyttää Selective Repeatia
  - TCP SACK (selective acknowledgments) optio
  - Virheenkorjauksesta tulee GBN+SR hybridi

TCP virheenkorjauksen ajastin

- Uudelleenlähetyksen ajastin
  - Eli Retransmission timeout (RTO)
  - Jokaisella segmentillä oma
- Ajastimen pituus
  - Pidempi kuin kiertoviive
    - Eli aika joka kestää paketilla kulkea lähettäjältä vastaanottajalle ja taas takaisin lähettäjälle (RTT: Round trip time)
  - Mahdollisimman lyhyt jotta reagoidaan nopeasti virheisiin
  - Pitää säätää koko ajan koska kiertoviive vaihtelee myös
- Viiveen jatkuva mittaus
  - Tarvitaan ajastimen säätelyyn
  - Lähettävä TCP mittaa ajan joka on kulunut aika paketin lähetyksen ja sen kuittauksen vastaanottamisen välillä
  - Jokaiselle lähetetylle paketille

TCP virheenkorjauksen ajastin: alkuperäinen algoritmi

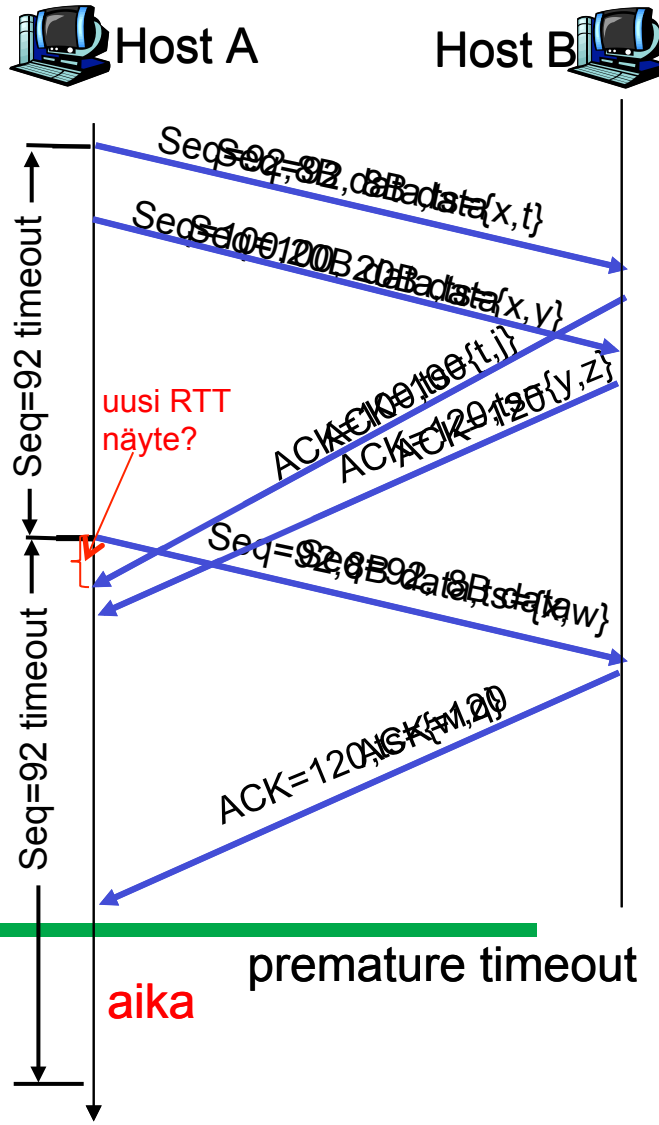
- Uudelleenlähetyksen ajastin säädetään algoritmilla
- Lasketaan kiertoviiveen (RTT) keskiarvoa koko ajan
  - Painotettu liikkuva keskiarvo mitatusta viiveestä
  - RTT = (α*oldRTT)+((1-α)*newRTTsample) (suositeltu α=0,9)
- Ajastin on lasketun kiertoviiveen keskiarvon lineaarinen funktio
  - RTO = β*RTT, β>1 (suositeltu β=2)
- Jotain vikaa?
  - Ei ota huomioon isoja kiertoviiveen vaihteluja

TCP virheenkorjauksen ajastin: parempi algoritmi

- Viiveenvaihtelu mukaan algoritmin muuttujaksi
- Nyt kaksi kiertoviiveen mittauksen (R) muuttujaa
  - kiertoviiveen painotettu keskiarvo: SRTT (smoothed round-trip time)
    - Eka mittaus: SRTT = R
    - Jatkossa: SRTT = (1 - alpha) * SRTT + alpha * R
  - Painotettu poikkeama keskiarvosta: RTTVAR (round-trip time variation)
    - Eka mittaus: RTTVAR = R/2
    - Jatkossa: RTTVAR = (1 - beta) * RTTVAR + beta * |SRTT – R|
  - alpha=1/8, beta=1/4
- Ajastin: RTO = SRTT + 4*RTTVAR
  - Jos saadaan RTO < 1s → round it up to 1s

Karnin algoritmi

- Kuittaus uudelleenlähetyksen jälkeen
  - Kuittaus uudelleenlähetetystä vai alkuperäisestä paketista??
  - Ei voi tietää…
- Karnin algoritmi: Ei päivitetä ajastinta jos uudelleenlähetys
- Tarvitaan myös ajastimen pidennys
  - Kun timeout tapahtuu: new_timeout = 2*timeout (exponential backoff)
  - Muuten voidaan ajautua turhaan lähettämään uudelleen ikuisesti!
- TCP aikaleimat auttavat erottelemaan kuittaukset

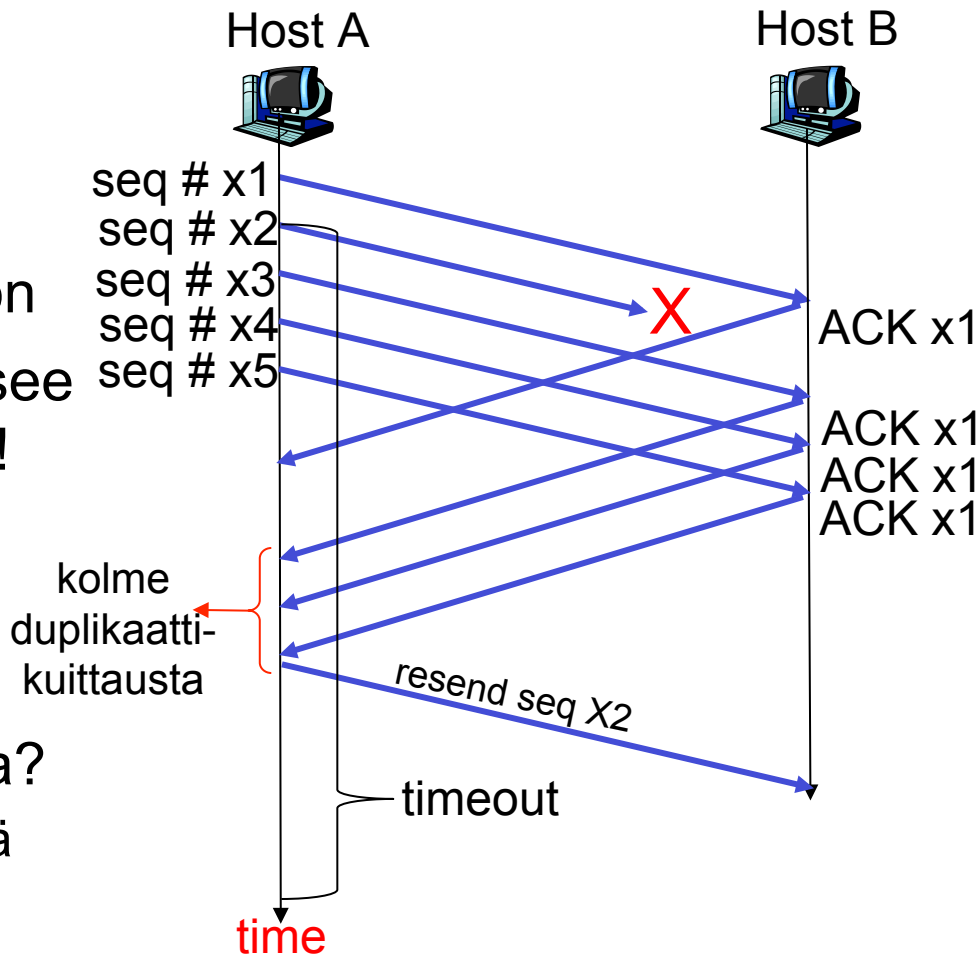


Nopea uudelleenlähetys (Fast Retransmit)

- Van Jacobson 1988
- Kuittaus kertoo aina seuraavan puuttuvan paketin sekvenssinron

→ Duplikaattikuittaukset merkitsee kadonnutta tai virheellistä pakettia!

- FR: Ei odoteta ajastinta vaan uudelleenlähetetään heti 3 duplikaattikuittauksen jälkeen
- Miksi odottaa kolme duplikaattia?
  - Joskus verkko uudelleenjärjestää paketteja → vältetään turhia uudelleenlähetyksiä

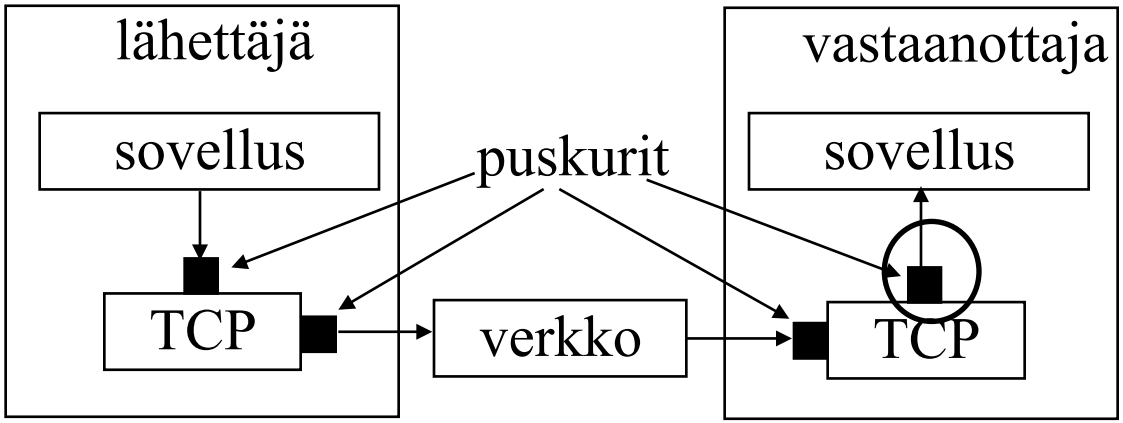


Sisältö

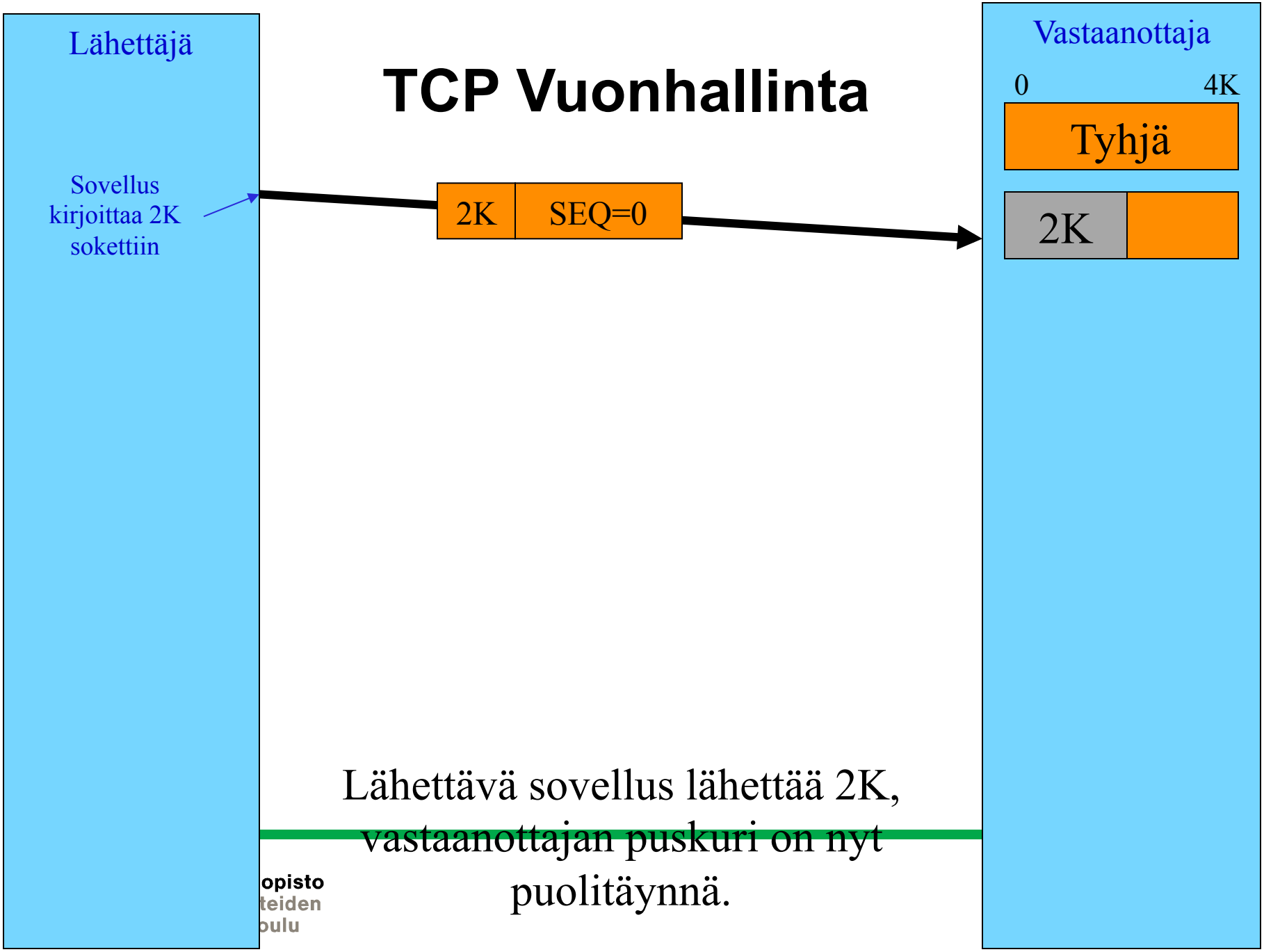
- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

TCP vuonhallinta

- Eli flow control
- Vastaanottava sovellus kuluttaa dataa tietyllä nopeudella
- TCP yhteyden yli voidaan joskus lähettää tätä nopeammin
- Vuonhallinta varmistaa ettei näin tapahdu
- Menetelmä perustuu liukuvan ikkunan koon vaihteluun

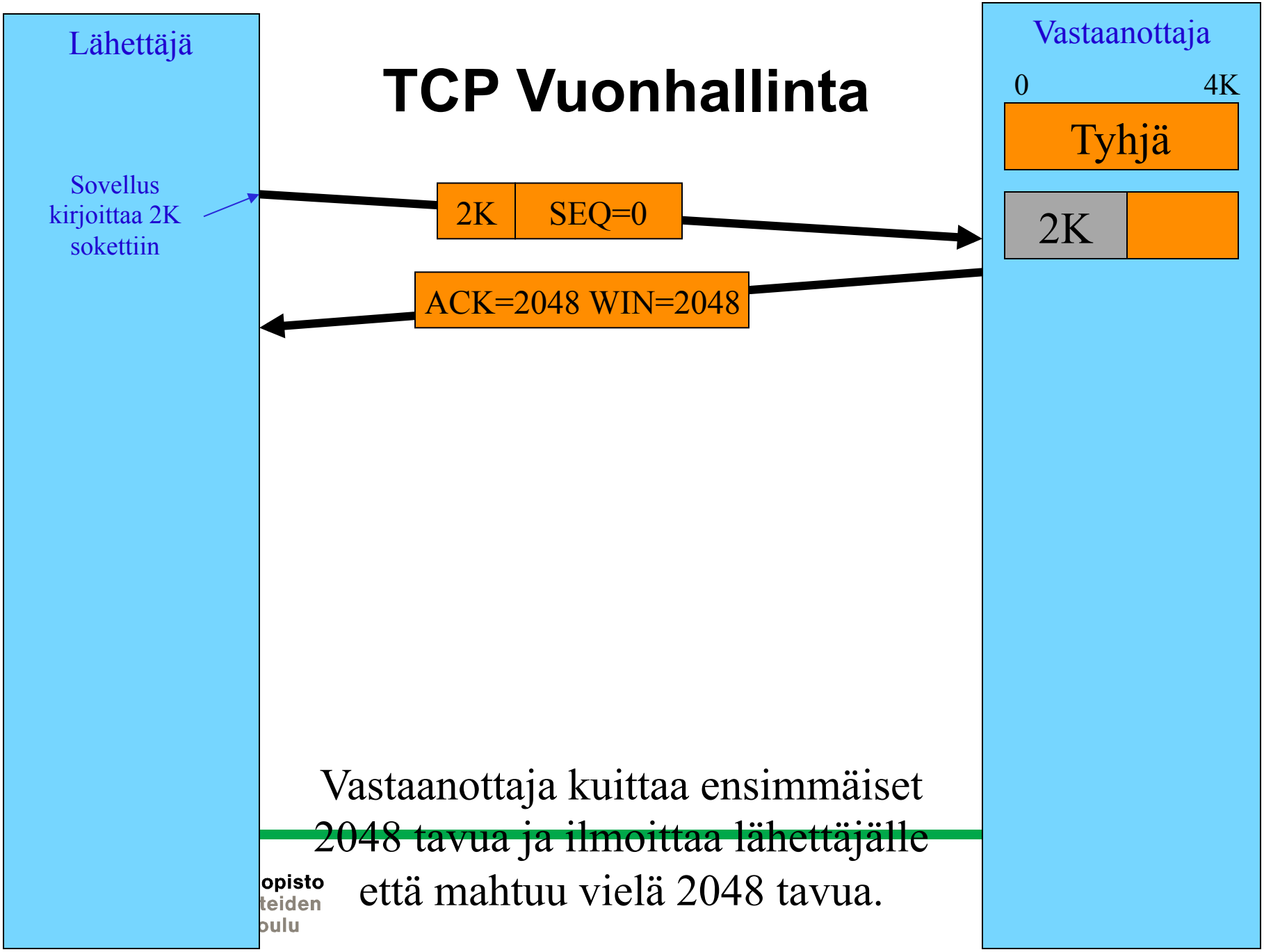


TCP Vuonhallinta



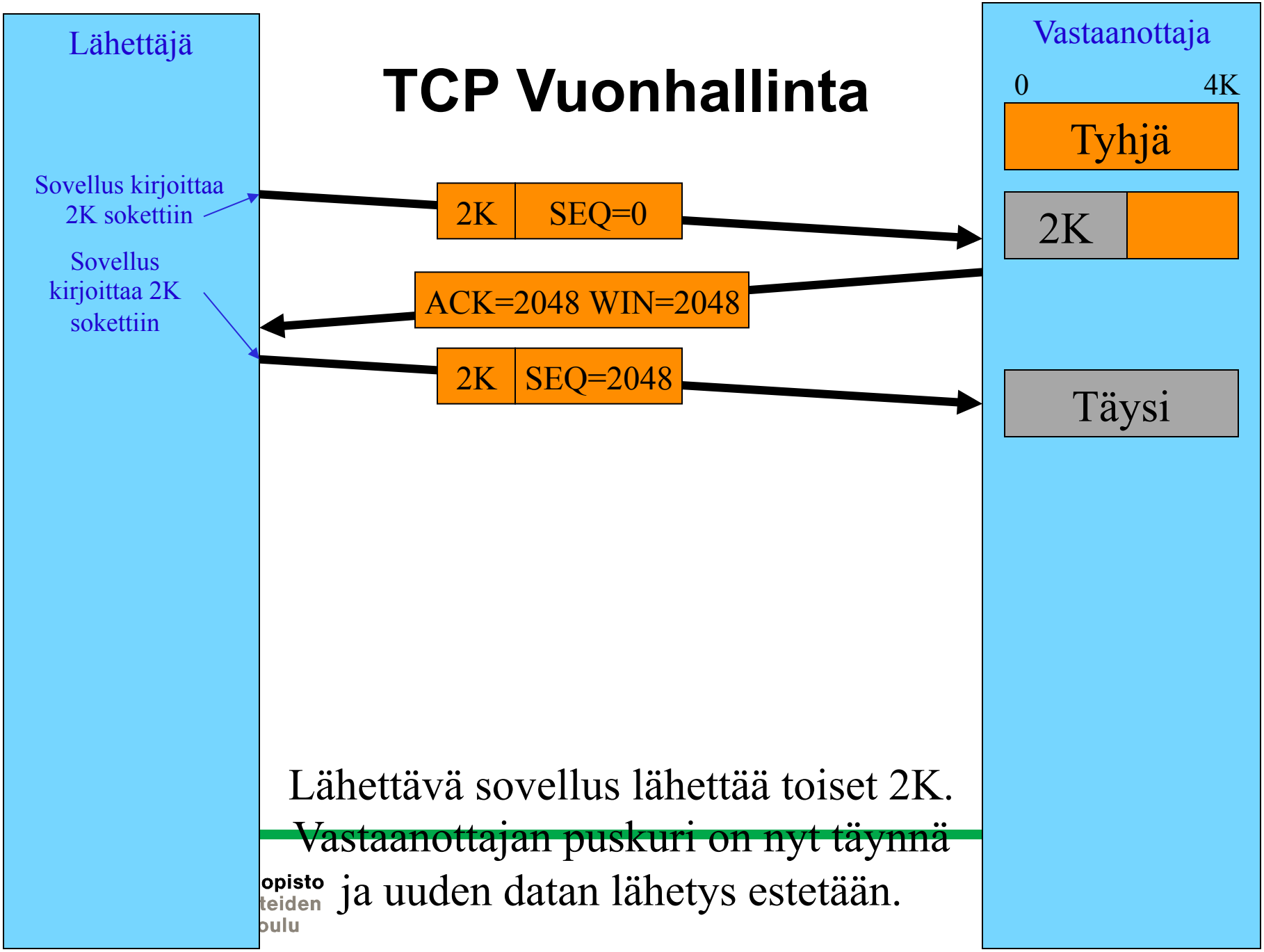
Lähettävä sovellus lähettää 2K, vastaanottajan puskuri on nyt puolitäynnä.

TCP Vuonhallinta



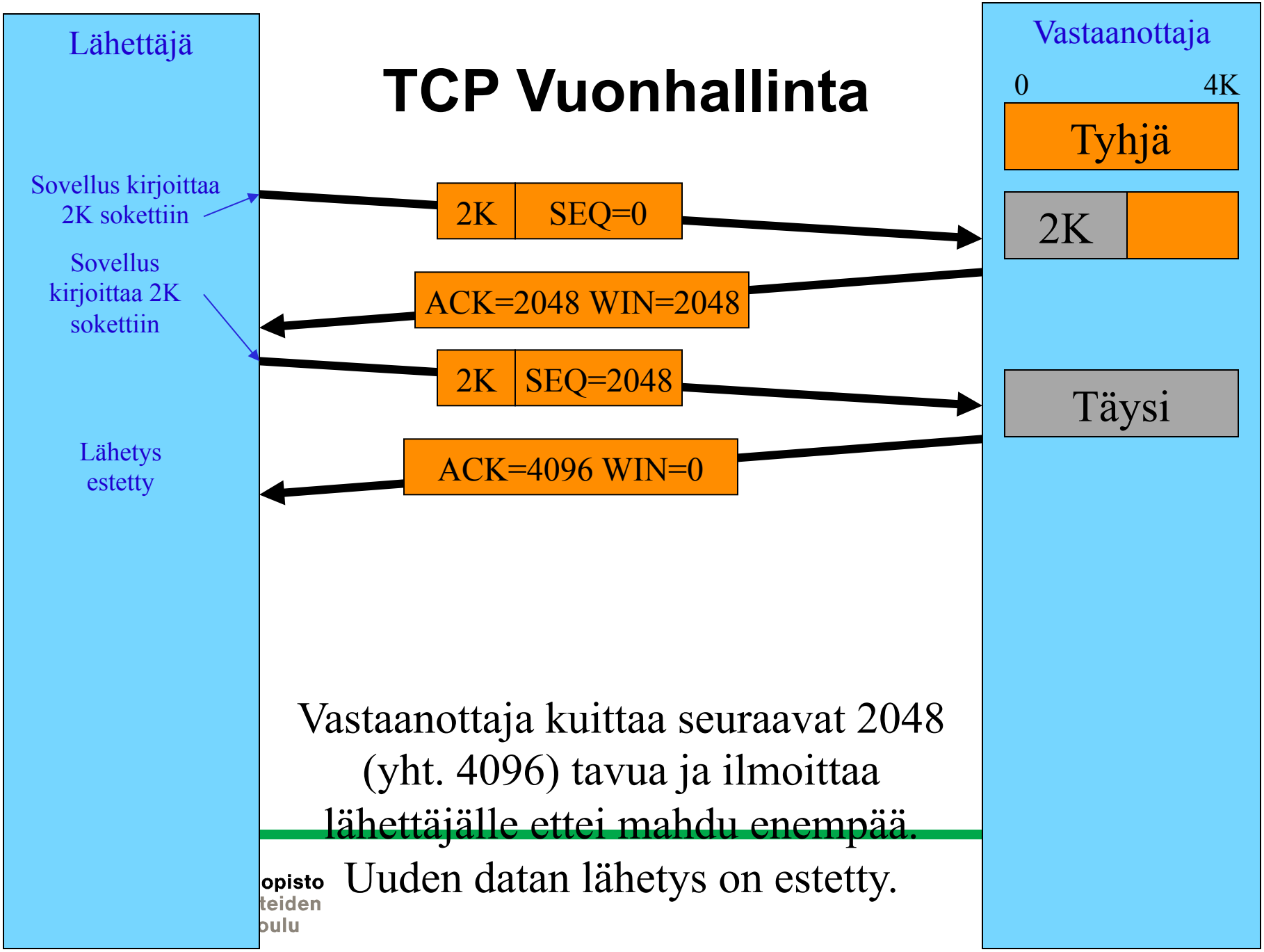
Vastaanottaja kuittaa ensimmäiset 2048 tavua ja ilmoittaa lähettäjälle että mahtuu vielä 2048 tavua.

TCP Vuonhallinta



Lähettävä sovellus lähettää toiset 2K. Vastaanottajan puskuri on nyt täynnä ja uuden datan lähetys estetään.

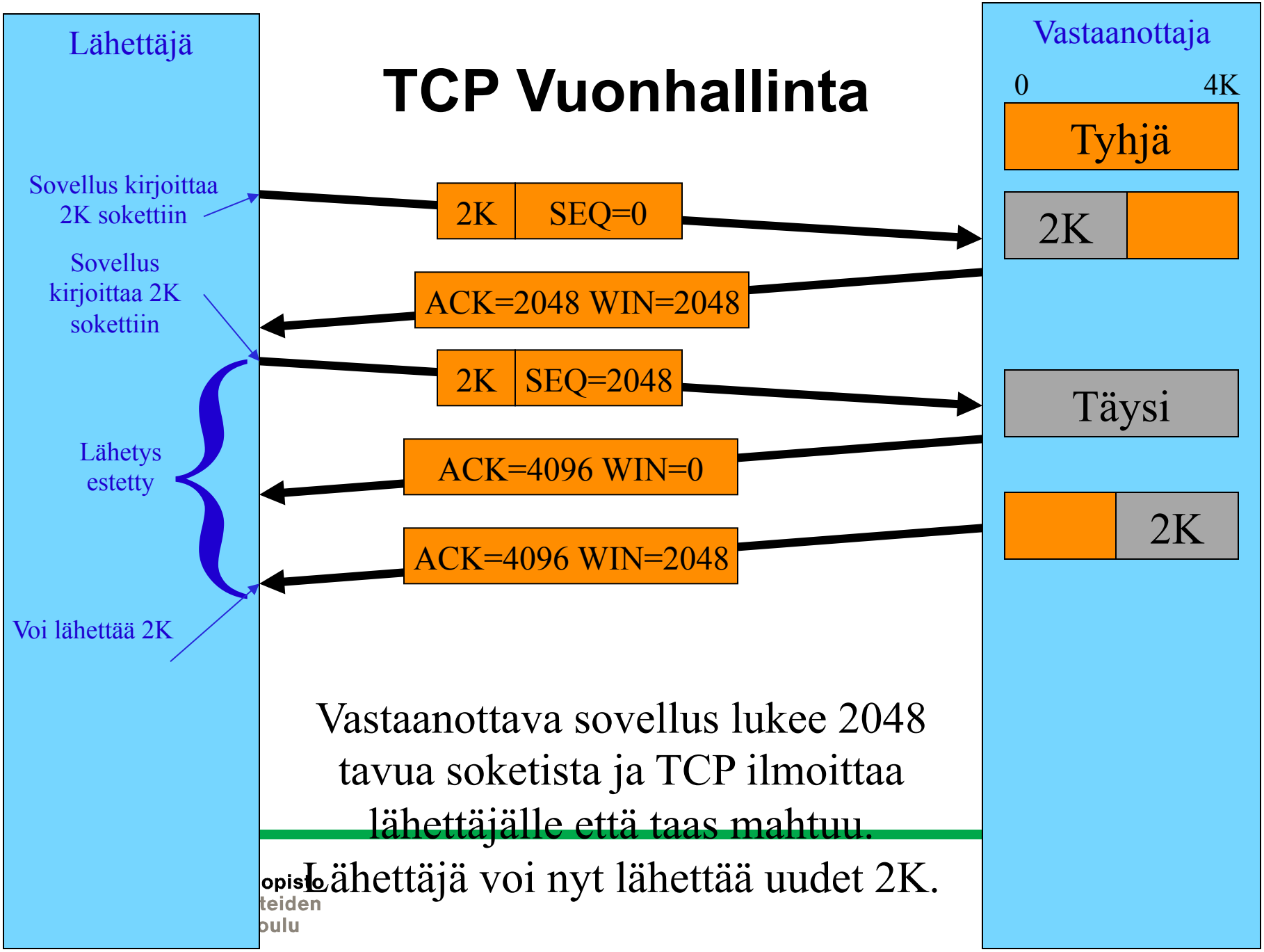
TCP Vuonhallinta



Vastaanottaja kuittaa seuraavat 2048 (yht. 4096) tavua ja ilmoittaa lähettäjälle ettei mahdu enempää.

Uuden datan lähetys on estetty.

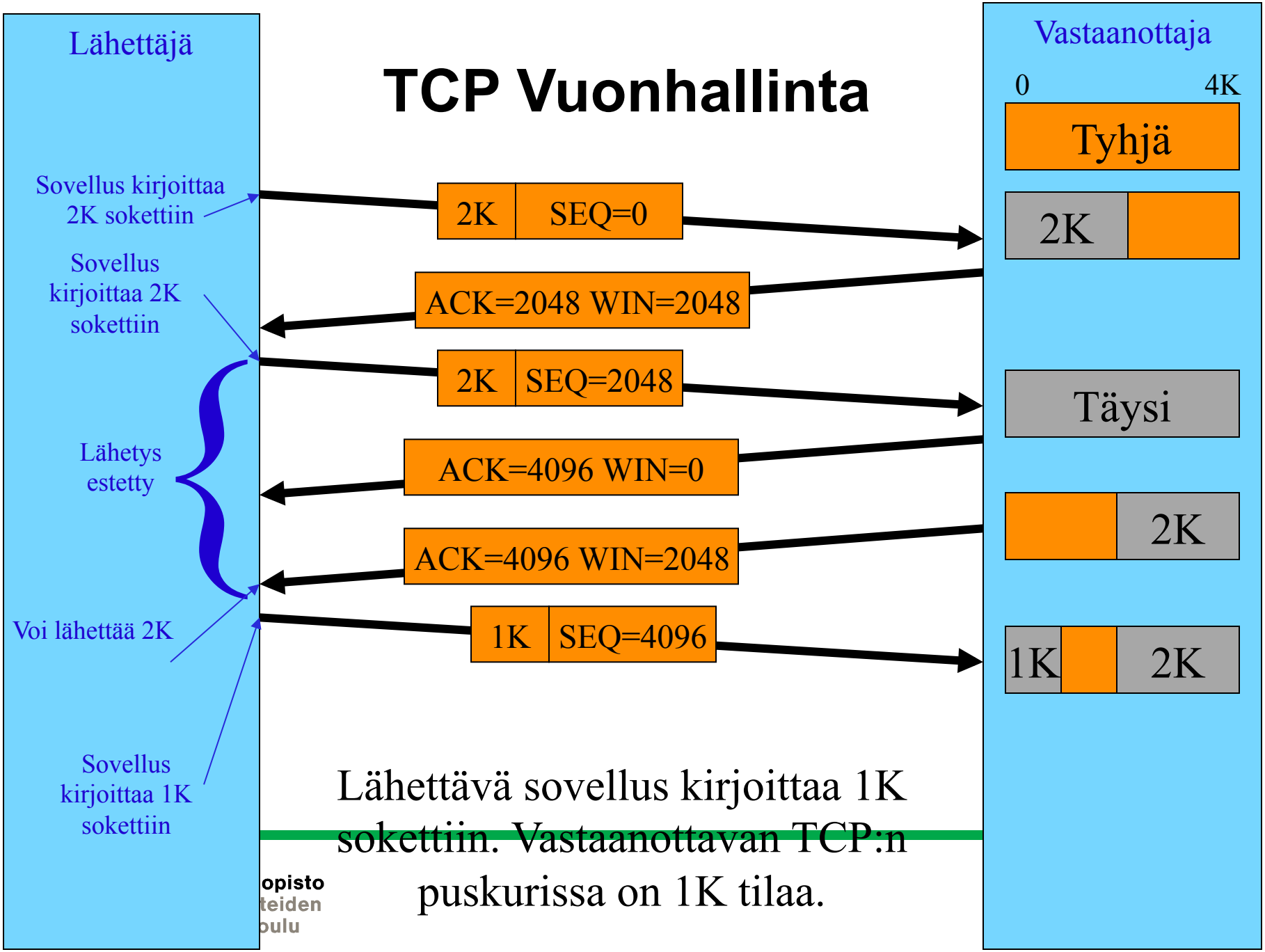
TCP Vuonhallinta



Vastaanottava sovellus lukee 2048 tavua soketista ja TCP ilmoittaa lähettäjälle että taas mahtuu.

Lähettäjä voi nyt lähettää uudet 2K.

TCP Vuonhallinta



Lähettävä sovellus kirjoittaa 1K sokettiin. Vastaanottavan TCP:n puskurissa on 1K tilaa.

Vuonhallintaongelmat

- Silly Window –syndrooma (SWS)
  1. Vastaanottavan TCP:n ja sovelluksen välissä oleva puskuri täyttyy
  2. Soketin vastaanottopuskuri
  3. Sovellus lukee soketista pienen määrän (esim. 1 tavu)
  4. TCP avaa vastaanottoikkunaa saman verran
  5. TCP lähettäjä lähettää uuden yhtä pienen segmentin
  6. Sama toistuu uudelleen
- Lähettäjän puolella vastaavanlainen tilanne
  - Sovellus kirjoittaa pieniä määriä dataa sokettiin → TCP lähettää pieniä paketteja
- Molemmissa karmiva overhead TCP/IP-otsakkeiden takia

Vuonhallintaongelmien välttäminen

- Ei sallita pieniä vastaanottoikkunan kokoja
  - Täysin kiinni tai…
  - vähintään MSS verran auki tai…
  - puolet puskurikoosta verran auki (jos pienempi kuin MSS)
  - Ratkaisee SWS:n
- Lähettäjän puolen ratkaisu: Naglen algoritmi
  - Viivästetään lähetystä kunnes MSS verran lähetettävää
  - Ajastin varmistaa ettei odoteta ikuisesti
- Nagle huono interaktiivisille sovelluksille → Push-mekanismi
  - TCP_NODELAY sokettioptio kertoo TCP:lle että data lähetettävä heti
  - Push-lippu TCP otsakkeessa kertoo vastaanottavalle TCP:lle että data puskettava heti sovellukselle
  - Jokseenkin epävarmaa onko toteutettu ja miten…

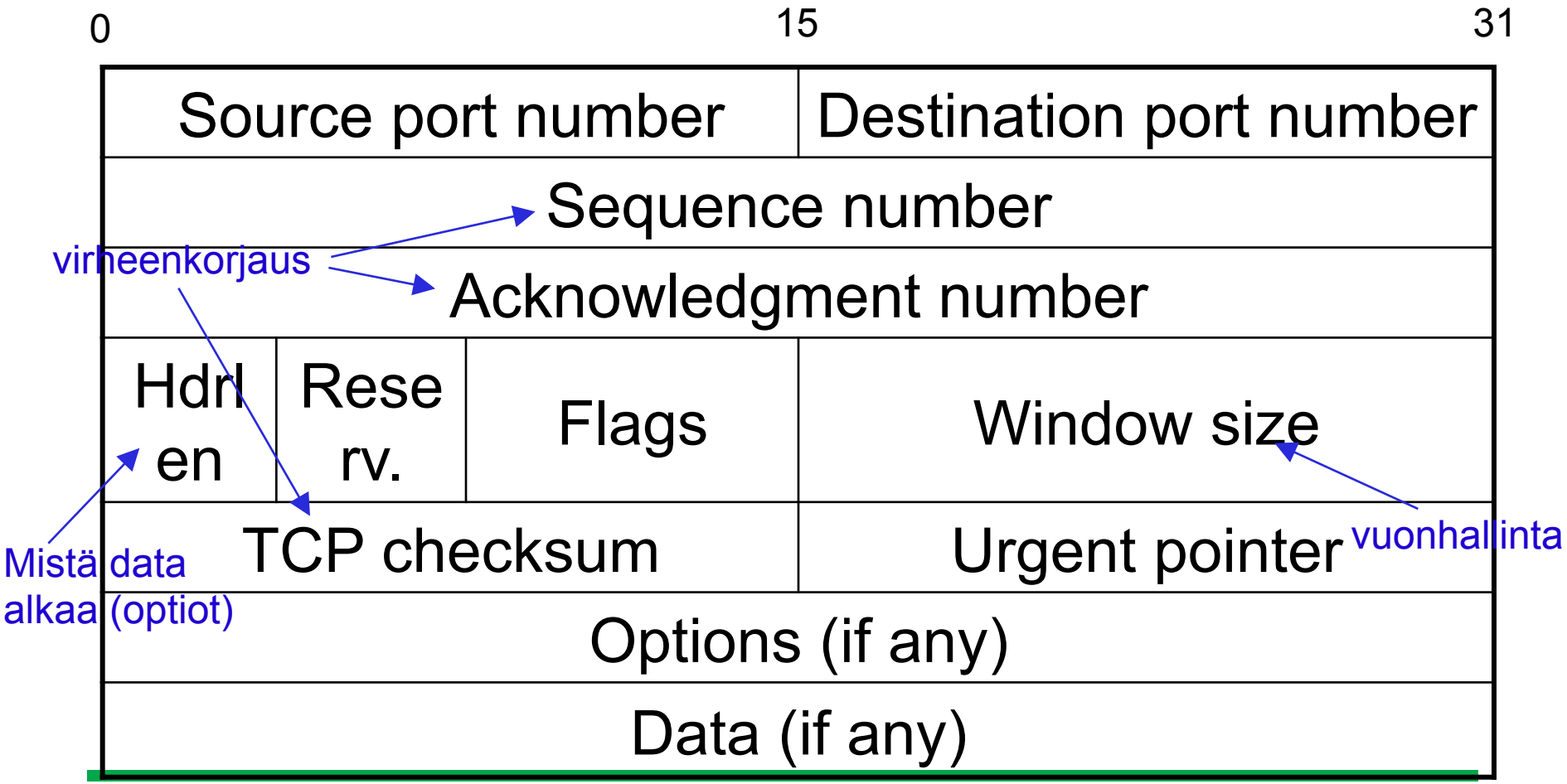
Isot vastaanottoikkunat

- Vo:n ikkunankoolle varattu 16 bittiä otsakkeessa
  →max koko noin 65 kt
- Esim: 10Mbit/s polku Euroopasta USA:n länsirannikolle
  - 0.15s * 10^7/8 ≈ 190KBytes
  - 16 bittiä eri riitä määrittämään riittävän suurta ikkunaa!
- Käytetään Window Scaling –optiota
  - Molemmat päät määrittää kertoimen yhteydenmuodostuksen aikana (SYN-segmentit)
  - Kertoimen ja otsakkeen määrittämän luvun avulla lasketaan lopullinen ikkunankoko tiedonsiirron aikana

kaistanleveys

kiertoviive

TCP-otsake


0
15
31
Source port number
Destination port number
Sequence number
Acknowledgment number
Hdrl en
Rese rv.
Flags
Window size
TCP checksum
Urgent pointer
Options (if any)
Data (if any)
virheenkorjaus
Mistä data alkaa
(optiot)
vuonhallinta

TCP-otsake: liput (flags)

- Liput ovat bittejä
- URG viestin urgent pointer osoittaa dataan, joka on aiheellista lukea ohi jonossa olevan datan
  - Esim. käyttäjä näppäilee interrupt-komennon kesken telnet-istunnon
- ACK: kuittausnumero on aktiivinen
- PSH: tämän jälkeen ei toistaiseksi uutta dataa, välitä heti eteenpäin
  - ei odoteta segmentin täyttymistä
- RST: resetoi yhteys
- SYN: yhteyden avaus
- FIN: yhteyden sulkeminen

TCP-otsake: optiot

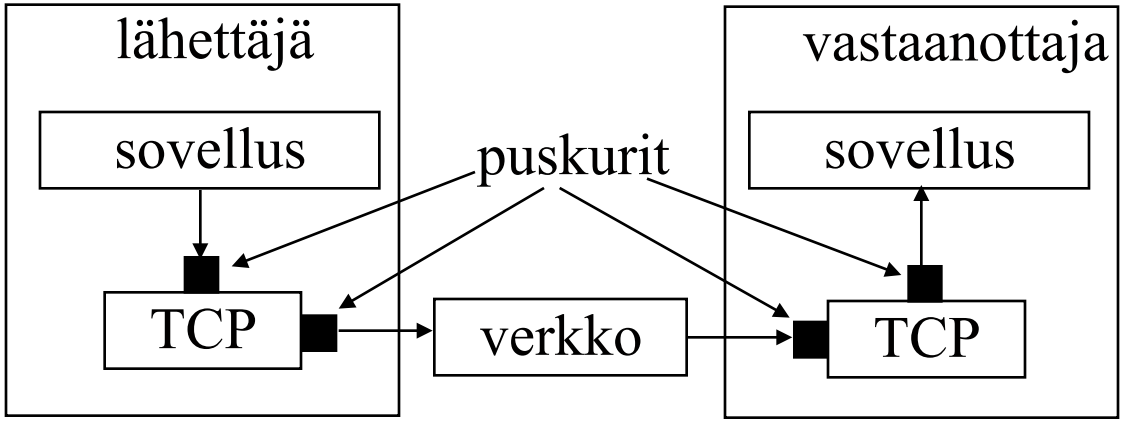
- MSS: suurimman sallitun segmentin koko
- Window scaling: sovitaan kerroin jolla vastaanottajan ikkunan koko kerrotaan
  - Tarvitaan koska window kenttä on usein liian pieni (max ~65KB) nykyaikaisille kaistanleveyksille
  - Saadaan käyttöaste korkeammaksi
- SACK: Selective Repeat –tyyliset kuittaukset
- Timestamps eli aikaleimat (ks. kalvo 49)
- Joitain muita vielä löytyy...

Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

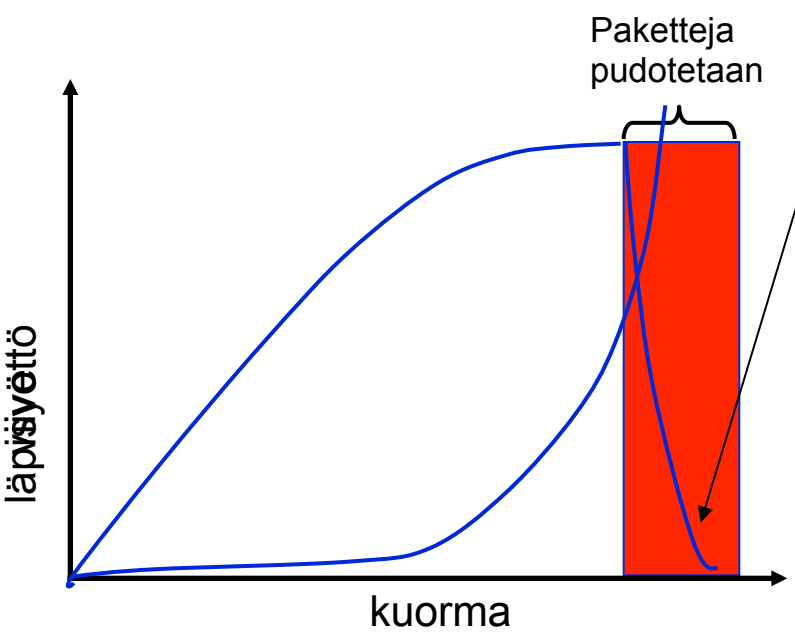
Ruuhkanhallinta: Miksi?

- Verkon hetkellinen jäljellä oleva vapaa kaista vaihtelee
  - Voi olla vähemmän kuin lähettävän ja vastaanottavan sovellusten kapasiteetti → pelkkä vuonhallinta ei riitä
  - Monta lähettäjää jakaa samoja verkkoresursseja
  - Verkko voi siis olla pullonkaula
- Ruuhkanhallinta varmistaa että verkkoa ei ylikuormiteta



Ruuhkanhallinta: Miksi?

- Verkkoelementeissä (reitittimet) on puskurit
  - FIFO+drop tail
  - Puskurin täyttyessä paketit joutuvat jonottamaan → viive kasvaa
  - Puskurien ollessa täynnä uudet paketit pudotetaan



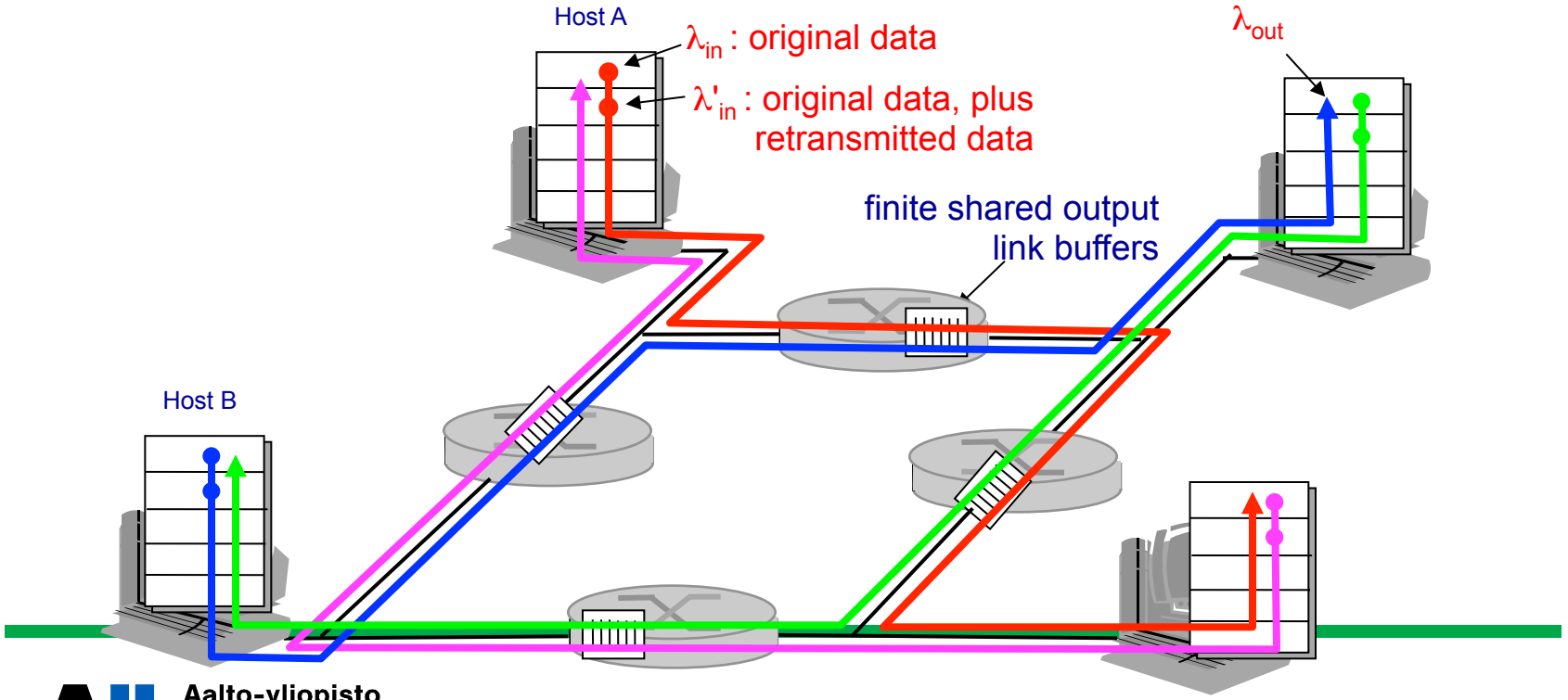
"Congestion collapse":

- Pudotettujen pakettien uudelleenlähetys
  - Kasvattaa lisää kuormaa
- Uudelleenlähetetään tarpeettomasti vielä matkalla olevia paketteja
  - Viive kasvaa nopeasti → RTO ei pysy perässä
  - Viive kasvaa yli maksimi RTOn
  - Lisää edelleen kuormaa!
- Reitittimet tekevät enenevästi turhaa työtä
  - Esim. paketin pudottaa loppusuoralla oleva reititin

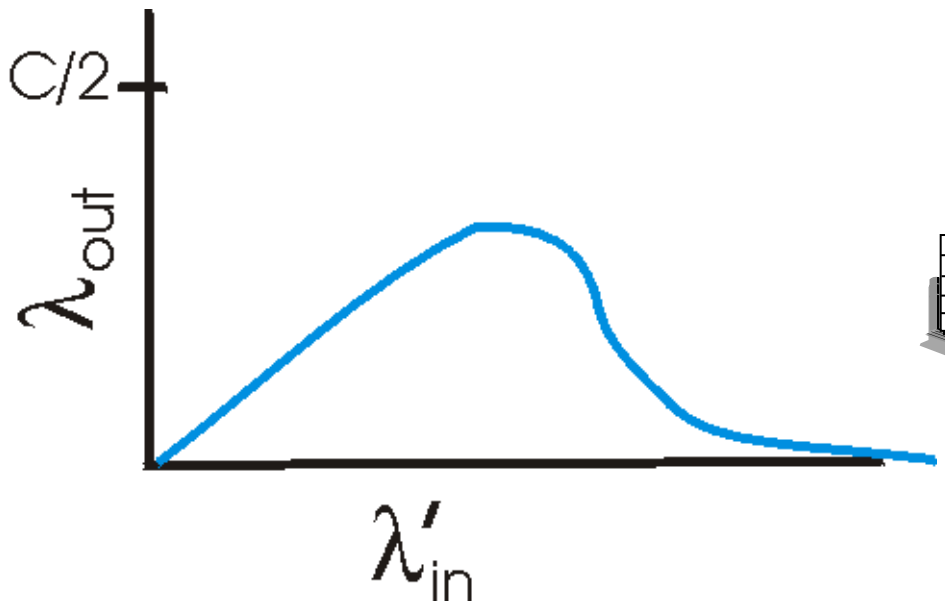
Ruuhkan muodostuminen ja vaikutus

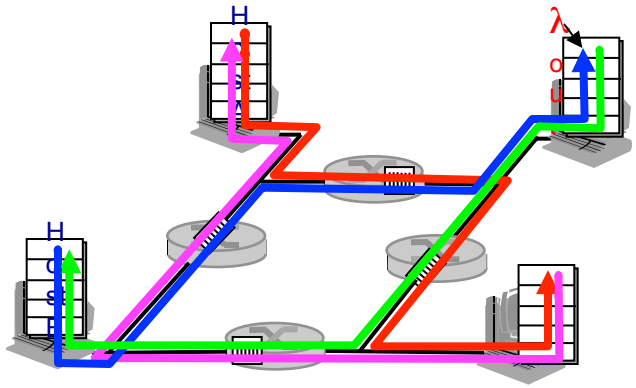
- neljä lähettäjää
- polut useamman reitittimen kautta
- ajastinpohjainen uudelleenlähetys

Mitä tapahtuu kun λ_{in} ja λ'_{in} kasvaa?



Ruuhkan muodostuminen ja vaikutus





- Kun paketti pudotetaan, sen välittämiseen jo kulutettu verkon kapasiteetti menee haaskuun!

Ruuhkanhallintamenetelmät

kaksi menetelmää:

päästä-päähän (end-to-end) ruuhkanhallinta:

- verkko ei raportoi ruuhkatilasta
- päätelaitteet päättelevät ruuhkatilan pakettihäviöstä tai viiveestä
- TCP:n perusmekanismi

verkon avustama ruuhkanhallinta:

- reitittimet kertovat ruuhkasta päätelaitteille
- yleensä bittilippu otsakkeessa joka kertoo ruuhkan (esim. ECN)

Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

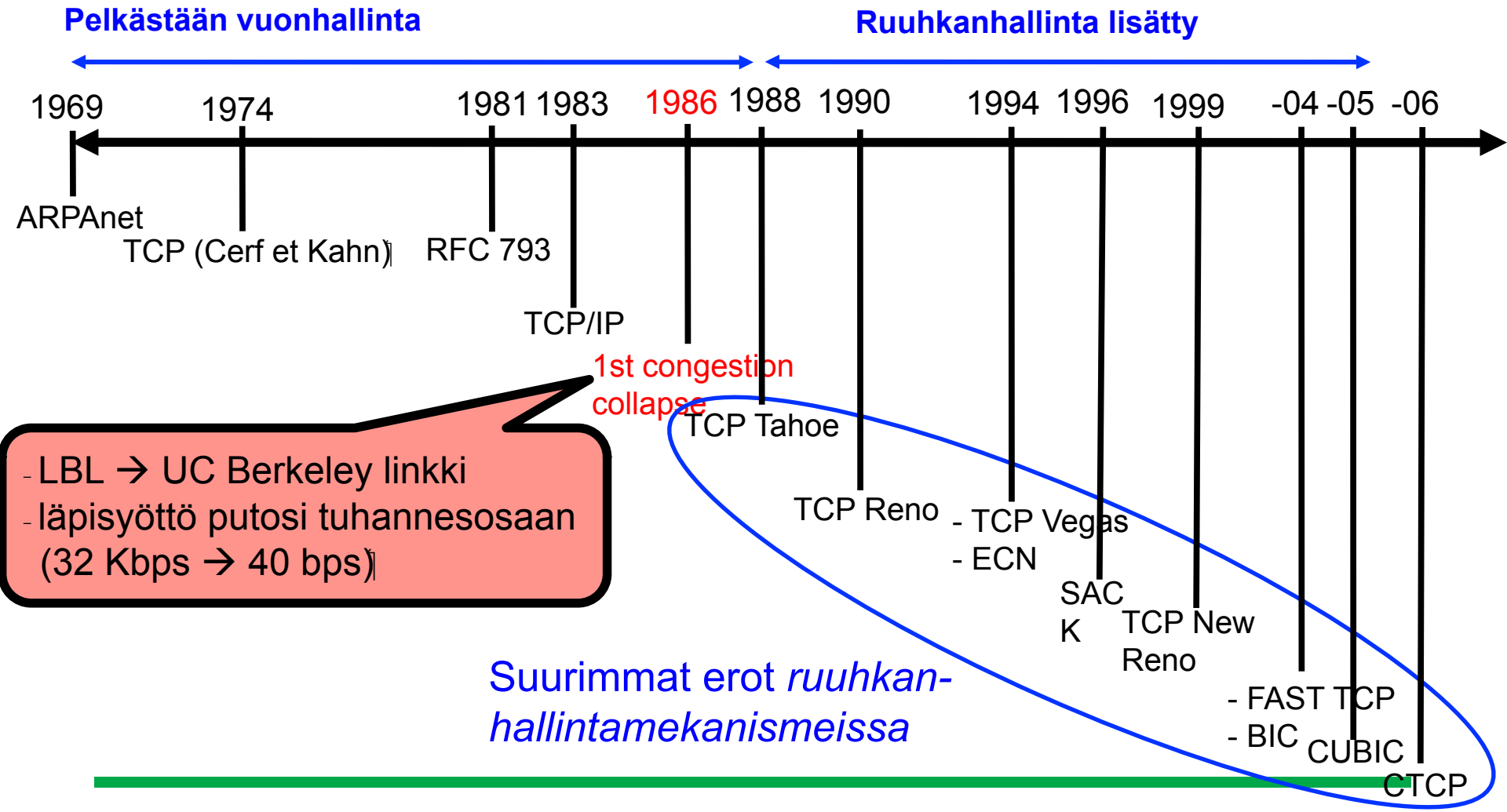
TCP Ruuhkanhallinta

- Periaate:
  - Kontrolloi jatkuvasti TCP lähetysnopeutta
  - Kasvata nopeutta kun kaikki menee hyvin
  - Pienennä nopeutta kun havaitaan ruuhkaa
    - Segmenttejä katoaa
- Miten?
  - Ruuhkaikkunan (eli congestion window=cwnd) avulla:

    `lähetetysikkunan koko ≤ min(cwnd, rwnd)`
- Lähetysnopeutta säädellään muuttamalla ruuhkaikkunan kokoa

pitää ottaa vuonhallinta huomioon

Hieman taustaa…

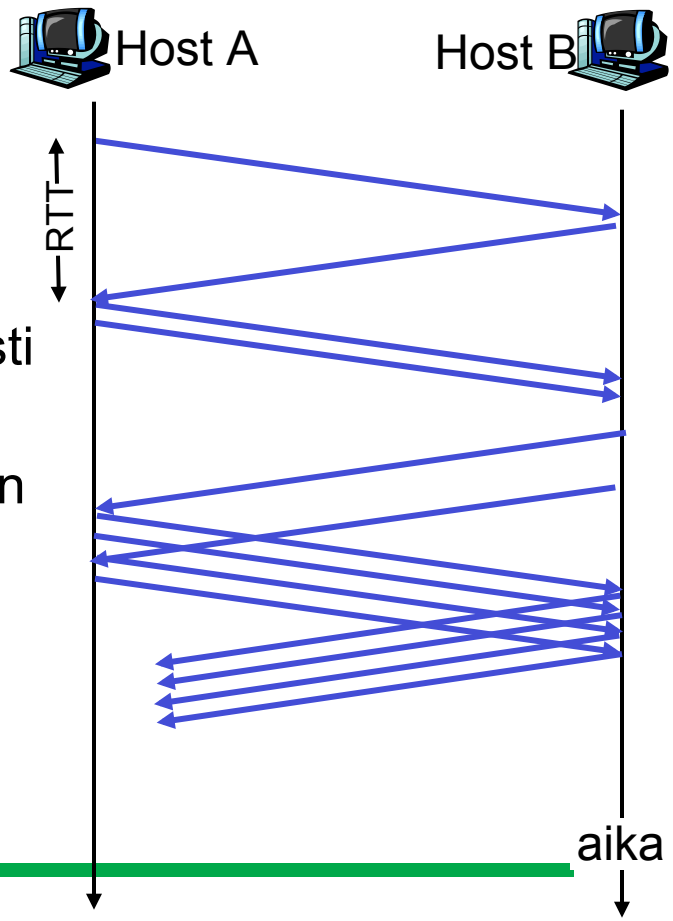


TCP Tahoe

- 1988 Van Jacobson
- Ensimmäiset TCP:n ruuhkanhallintamekanismit
- Pakettihäviö indikaattori ruuhkasta
  - Jos havaitaan ajastinten avulla
- Kaksivaiheinen:
  - Slow Start
  - Congestion Avoidance
- Tässä TCP versiossa myös mukana:
  - Uusittu ajastinalgoritmi (ks. dia 48)
  - Fast Retransmit (ks. dia 50)

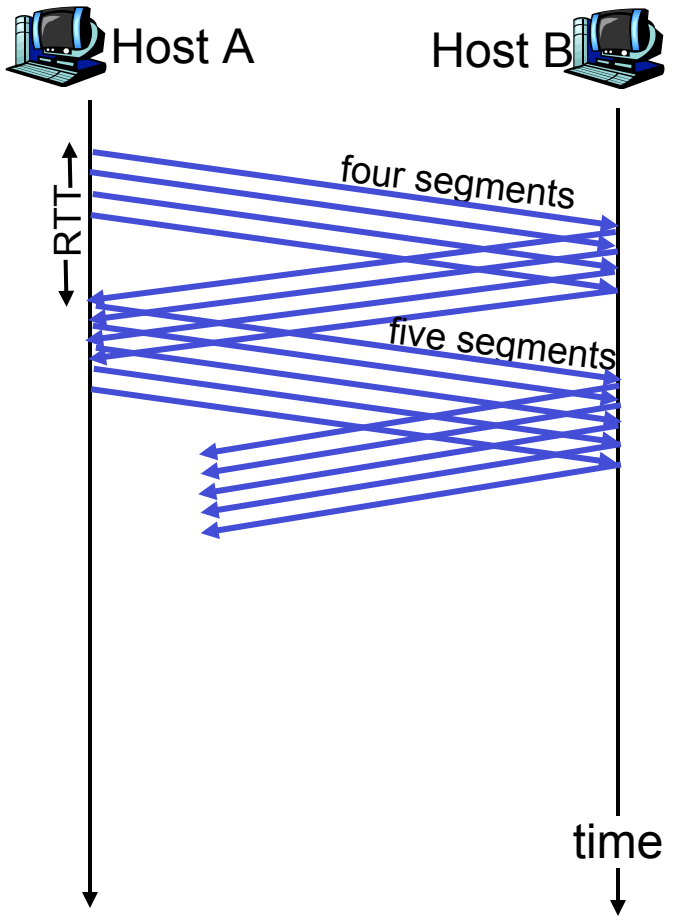
Slow Start (SS)

- Aloitetaan hitaasti: lähetetään vain yksi segmentti (cwnd=1)
- Jokainen uusi kuittaus kasvattaa ruuhkaikkunaa yhdellä
  - Ikkunan koko kasvaa exponentiaalisesti
  - Ideana on uuden yhteyden lähetysnopeuden nopea kasvattaminen
- SS käytössä:
  - Uuden TCP-yhteyden alussa
  - RTO ajastimen laukeamisen jälkeen
    → ruuhkatilanne



Congestion Avoidance (CA)

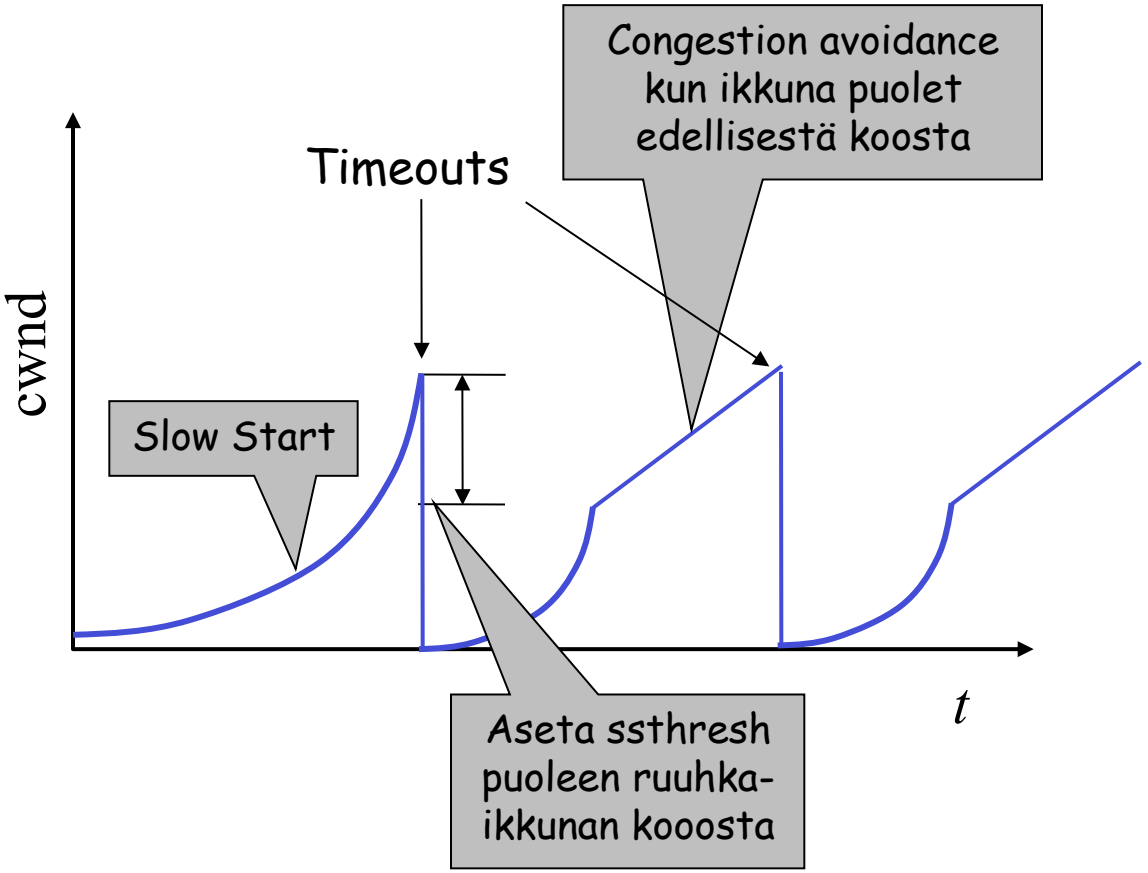
- Lähestytään verkon (polun) kapasiteettia varovaisemmin
- Kasvatetaan ruuhkaikkunaa yhdellä kun vastaanotettu ikkunan verran kuittauksia (eli +1 per RTT)
- Periaate on välttää pahoja ruuhkatilanteita
  - Verkon vaikea toipua (uudelleenlähetykset)



SS ja CA yhteentoiminta

- Määritellään raja-arvo ikkunankoolle
  - Slow start threshold (ssthresh)
- Timeout:
  - ssthresh = 0.5 x cwnd
  - cwnd = 1 segmentti
- Uusi kuittaus:
  - cwnd < ssthresh → SS
  - cwnd >= ssthresh → CA

TCP Tahoe: ruuhkaikkunan koon vaihtelu



TCP Reno

- Van Jacobson 1990
- Fast retransmit ja Fast recovery
  - Duplikaatikuittaukset kertovat että jotain paketteja menee perille
  - Peruutetaan vähemmän
    - ssthresh = 0.5 x cwnd
    - cwnd = ssthresh + 3 segmenttiä
    - Kasvata ikkunaa yhdellä aina kun tulee uusi duplikaattikuittaus
    - Uuden kuittauksen jälkeen: cwnd = ssthresh

Fast recovery

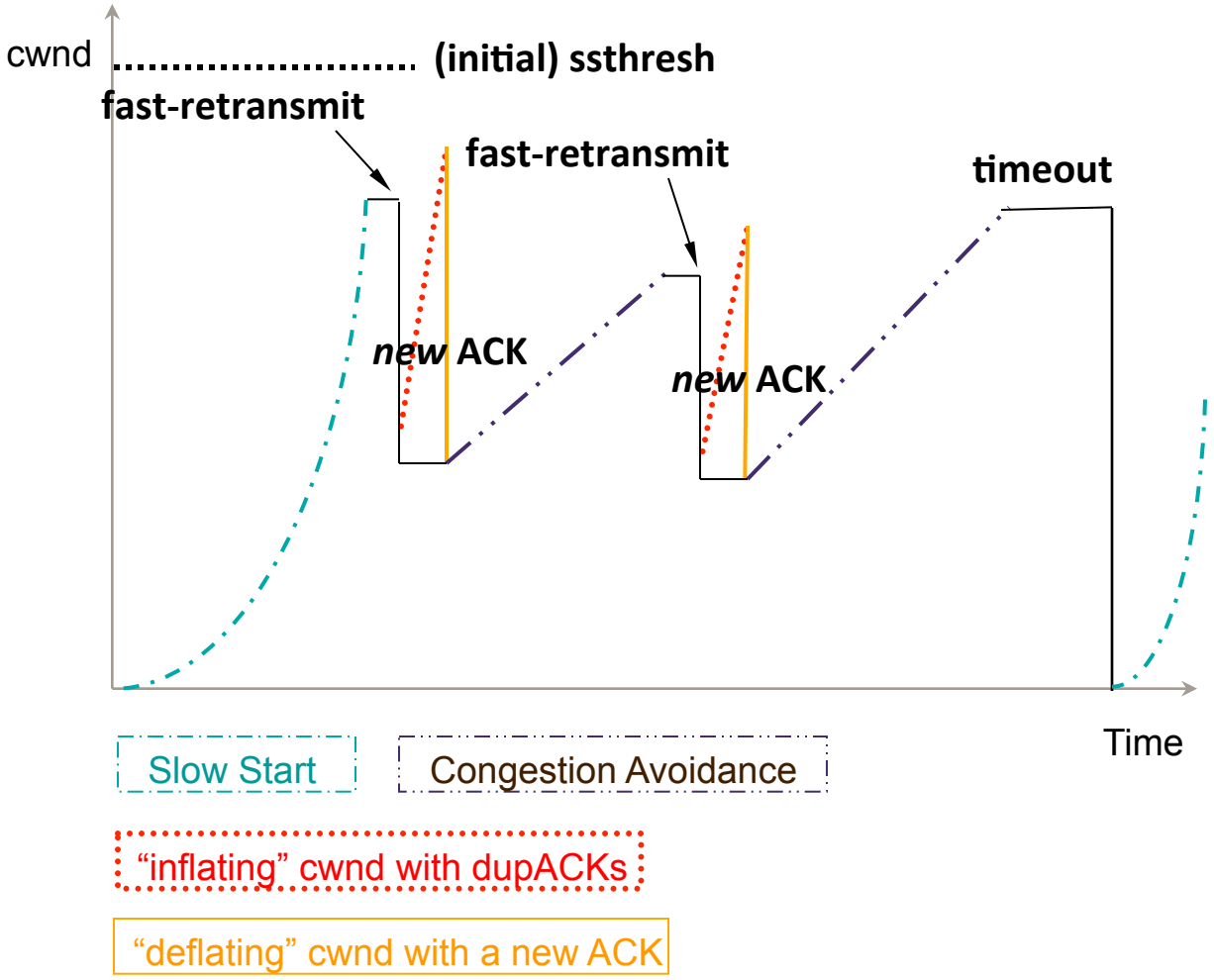
Perille menneiden pakettien lkm

AIMD

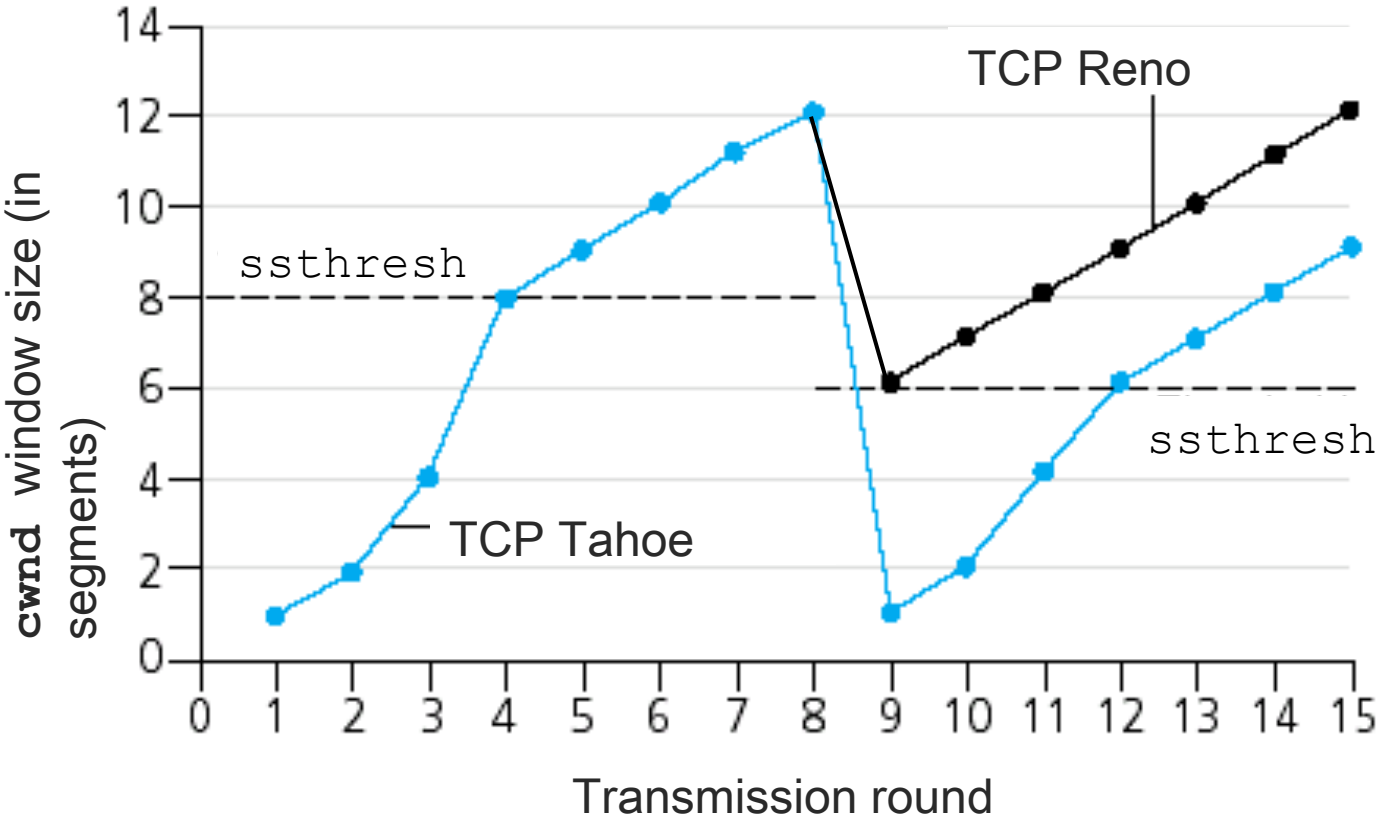
- kuittaukset: kasvattavat ruuhkaikkunaa 1 MSS per RTT: *additive increase*
- pakettihäviö: puolittaa ruuhkaikkunan (ei timeout häviö): *multiplicative decrease*

AIMD: Additive Increase Multiplicative Decrease

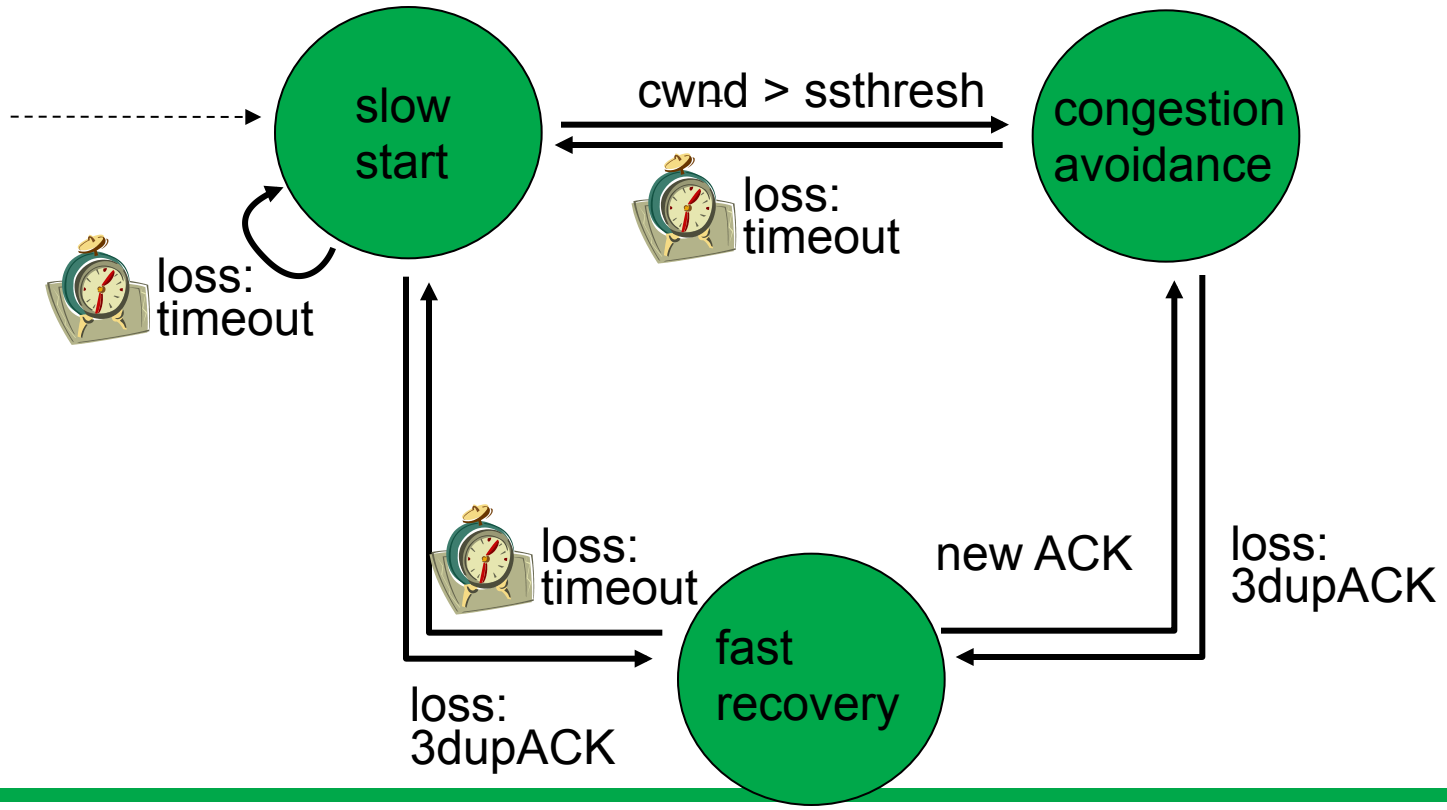
TCP Reno: ruuhkaikkunan koon vaihtelu



Tahoe vs. Reno

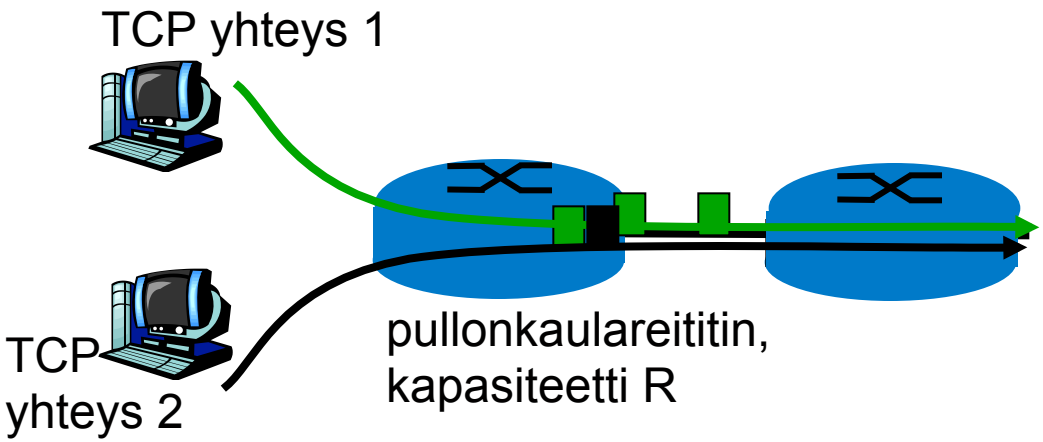


Reno: Ruuhkanhallinnan tilakone



TCP:n tasapuolisuus (fairness)

tavoite: jos K TCP yhteyttä jakaa saman pullonkaulalinkin (kapasiteetti R), jokaisen pitäisi pystyä lähettämään nopeudella R/K

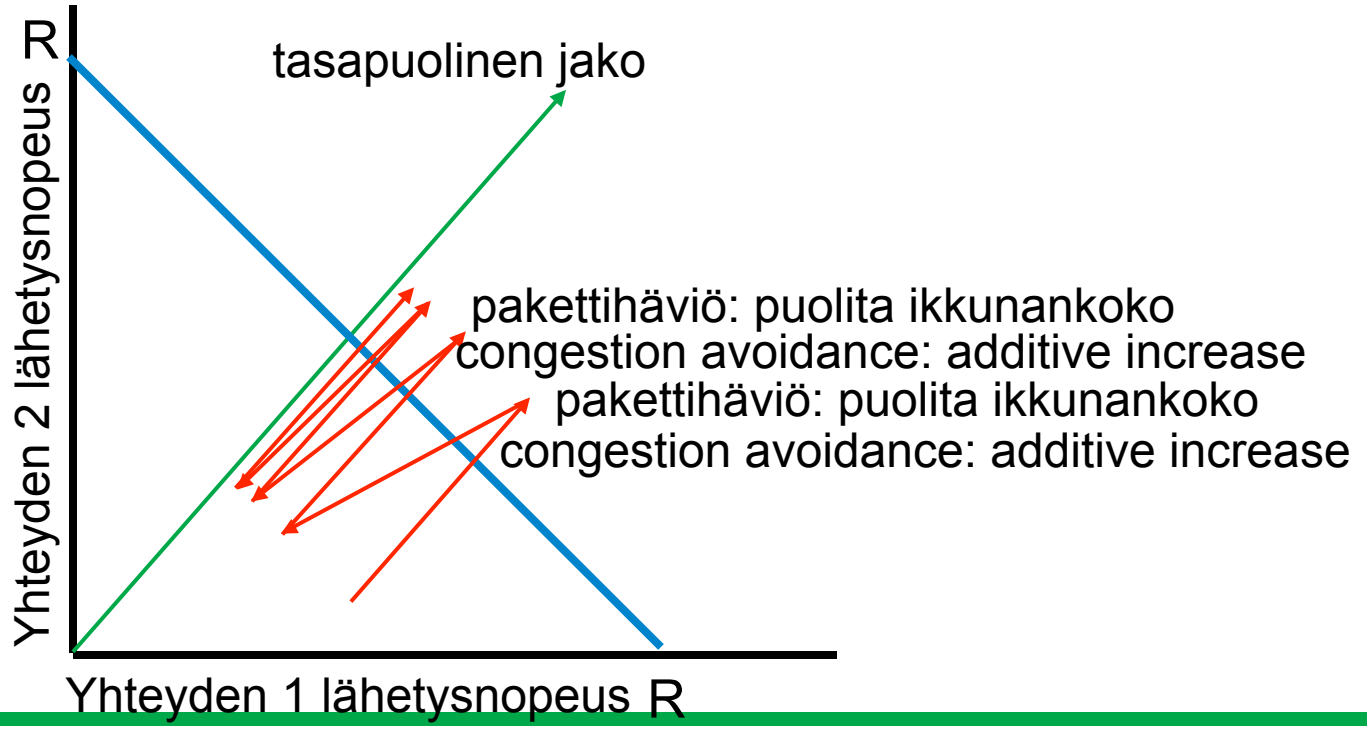


Onko TCP tasapuolinen?

Miksi TCP on tasapuolinen?

Kaksi kilpailevaa yhteyttä:

- *Additive increase:* johtaa kasvukertoimeen 1 kun läpisyöttö kasvaa
- *Multiplicative decrease:* leikkaa lähetysnopeutta suhteessa senhetkiseen nopeuteen



TCP:n tasapuolisuusongelmat

Tasapuolisuus ja kiertoviive

- Entä jos yhteyksillä eri kiertoviiveet?
  - Lyhyempi kiertoviive → nopeampi ruuhkaikkunan kasvaminen (1 per RTT)
- Reno (AIMD) ei siis tasapuolinen kiertoviiveen suhteen

Samanaikaiset yhteydet

- Sovellus voi avata monta yhteyttä samanaikaisesti palvelimelle
  - selaimet tekevät yleensä näin
- esim: pullonkaula (kap. R) josta menee läpi 9 yhteyttä
  - uusi sovellus avaa yhteyden → saa lähetettyä nopeudella R/10
  - uusi sovellus avaa 9 yhteyttä → saa lähetettyä nopeudella R/2 !

Tasapuolisuus UDP:n kanssa

- Jotkut sovellukset käyttävät UDP:ta
  - UDP ei reagoi ruuhkaan
  - UDP voi viedä kaistan TCP:ltä
- Yleensä tällaiset sovellukset eivät pyri mahd. suureen lähetysnopeuteen
  - Sovellus rajoittaa lähetysnopeutta
  - Esim. VoIP tai videopuhelu

Muita TCP versioita

- Viiveeseen perustuva ruuhkanhallinta
  - Esim. TCP Vegas
- Langattomat verkot
  - Pyritään erottelemaan langattoman linkin aiheuttamat pakettihäviöt ruuhkahäviöistä
  - Esim. TCP Veno
- Verkkopolut joissa paljon kaistaa ja pitkä(hkö) viive
  - Tarvitaan iso ruuhkaikkunan koko jotta saadaan koko kaista käyttöön
  - SS ja (varsinkin) CA kasvattavat ikkunaa liian hitaasti
  - TCP CUBIC
  - Compound TCP (CTCP)

Käyttöönotto

- Windows
  - Laajalti käytössä Compound TCP (CTCP)
    - Microsoftin kehittämä
  - Varsinkin palvelinpuolella
- Linux
  - TCP BIC oletus kerneleissä 2.6.8 - 2.6.18
  - TCP CUBIC versiosta 2.6.19 eteenpäin

Explicit Congestion Notification (ECN)

- Verkon avustama ruuhkanhallinta
- Reitittimet merkkaavat paketteja jos ruuhka päällä
  - Jos jonossa jatkuvasti paljon paketteja
  - Eräänlainen aktiivinen jononhallintamekanismi (AQM)
- Vastaanottajan pitää tehdä yhteistyötä
  - Välittää kuittauksissa ECN merkkaustietoa lähettäjälle
- Lähettäjä reagoi pienentämällä lähetysnopeutta
  - Pienentää ruuhkaikkunan kokoa

Explicit Congestion Notification (ECN)

- Käyttöönotto ollut valitettavan hidasta
  - Standardoitu jo 2001
  - Alun ongelmat (palomuurit ym.) söivät luottamusta
- Nykyään suht laajalti tuettu menetelmä
  - Noin 30% of top 100K web palvelimista tuki elokuussa 2012 vs. alle 5% 2008
  - Yli 90% poluista näille palvelimille myös tukivat ECN:a
- Mutta ECN käyttö silti matalaa tasoa
  - Selvästi alle 1% liikenteen lähettäjistä käyttää

TCP versioista vielä…

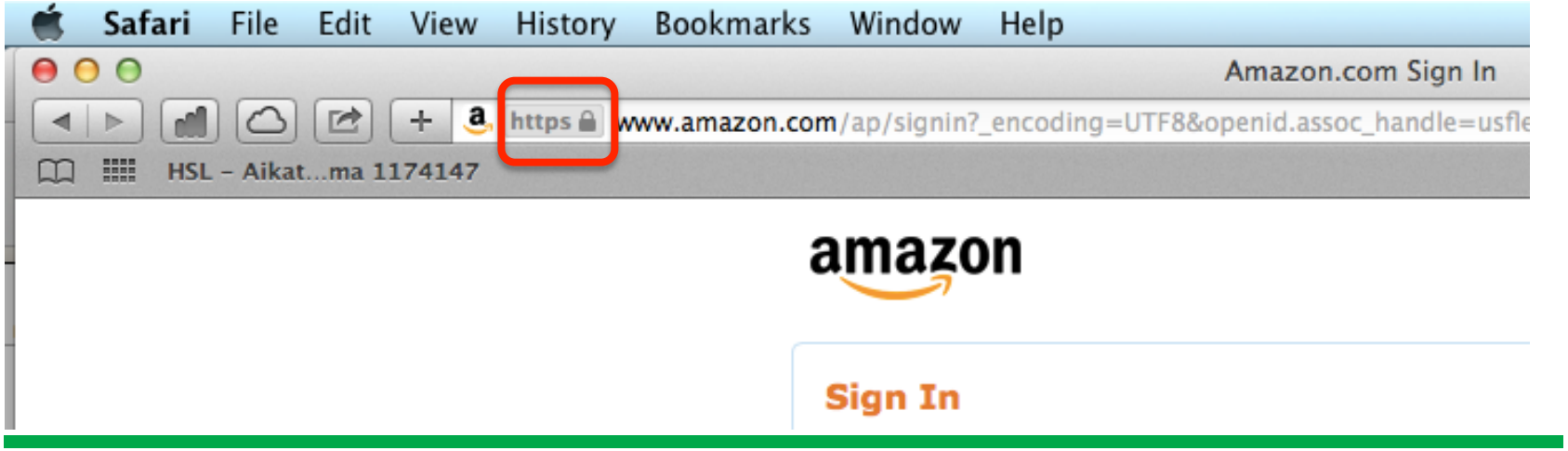
- Ruuhkanhallinta on TCP lähettäjän mekanismi
  - Vastaanottaja tekee “normaalia” virheenhallintaa (kuittaukset)
  - Vain ECN yms. tekevät poikkeuksen → vaatii vottajan tuen
- Eri TCP versiot ovat siis yhteensopivia
  - Suoraviivaista ottaa uusi TCP versio käyttöön jos muutokset ruuhkanhallinnassa
- Mutta eri kuljetuskerroksen protokollat eivät ole yhteensopivia!
  - Esim. SCTP ei toimi TCP:n kanssa
  - Käyttöönotto hankalaa

Sisältö

- Kuljetuskerroksen tehtävä ja ominaisuudet
- UDP (User Datagram Protocol)
- Luotettava tiedonsiirto
  - Virheenkorjaus
  - Putkitus
- TCP (Transmission Control Protocol)
  - Yhteydenhallinta
  - Virheenkorjaus
  - Vuonhallinta
- Ruuhkanhallinta
  - Perusteet
  - TCP:n ruuhkanhallinta
- Tietoturva: TLS

Turvallinen web-selaus: HTTPS eli Transport Layer Security (TLS)

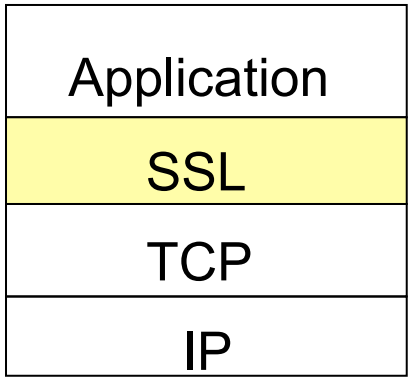
- Kuljetuskerroksen ja sovelluskerroksen välille webbiselailua turvaamaan Secure Sockets Layer (SSL)
  - standardoitu nimellä TLS
  - RFC 4346
- Lähes kaikki nettiostokset hoidetaan TLS:n avulla



SSL/TLS ohjelmointi

- SSL/TLS tarjoaa ohjelmointirajapinnan sovelluksille
  - Esim. C ja Java koodille löytyy kirjastot
- Näyttää sokettiohjelmoinnilta sovelluksen näkökulmasta:

```
SSL_write(SSL* client, const void* buffer, int size);
SSL_read(SSL* client, const void* buffer, int size);
```



SSL/TLS toiminta

- Ensin kättelyvaihe (handshake)
  - Palvelin tunnistetaan sertifikaatin avulla
  - Käyttäjä voidaan tunnistaa sertifikaatin avulla tai muuten
  - Osapuolet neuvottelevat käytettävät kryptoalgoritmit
    - Cipher suite
  - Jaetaan Pre-Master Secret (PMS)
- Yhteyttä varten muodostetaan istuntoavaimet
  - Generoidaan PMS:sta
  - Tiedonsiirto turvataan niiden avulla
- Yhteyden aikana
  - Data salataan (symmetrinen algoritmi)
  - Varmistetaan eheys sekä aitous (MAC)
  - Data lähetetään recordeina jotta MAC voidaan tarkistaa väliajoin
    - Ei tarvitse odottaa tiedoston loppuun asti

Yhteenveto

- Kuljetuskerros mahdollistaa sovellusten välisen viestinnän
  - TCP luotettava bittisiirtopalvelu
  - UDP epäluotettava viestinvälitys
- TCP ylivoimaisesti käytetyin protokolla
  - Virheenkorjaus (ARQ) ja vuonhallinta perusmekanismeja
- Ruuhkanhallinta
  - TCP:ssä tullut mukaan vasta myöhemmin
  - Monta TCP versiota → eri ruuhkanhallintamekanismit
- Tasapuolisuus
  - TCP:n ruuhkanhallinta johtaa tietyissä tapauksissa tasapuolisuuteen muttei aina (eri RTT, monta yhteyttä)
- SSL/TLS
  - Kattava tietoturvaprotokolla sovelluksen ja TCP välissä

Internetin protokollapino

Ensi viikolla Sanna jatkaa tästä