- 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:

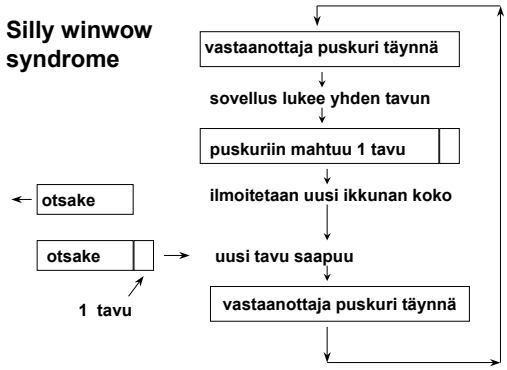
- Interaktiivinen editori => merkki lähetetään heti
  - 21 tavun TCP-segmentti => 41 tavun IP-paketti
  - joka kuitataan 40 tavun IP-paketilla
  - ilmoitus uudesta ikkunan koosta 40 tavun IP-paketilla
  - kaiutetaan merkki vielä 41 tavun IP-paketilla
  - yhden merkin käsittely =>
    - 162 tavun siirtäminen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

TCP-otsakkeen kentät

Source port								Destination port
Sequence number								
Acknowledgement number								
TCP head. length		URG	ACK	PSH	RST	SYN	FIN	Window size
Checksum								Urgent pointer
Options (0 or more 32 bit words)								
Data (optional)								

TPC-segmentin otsakekentät

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

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

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

pseudo-otsake

Source IP address		
Destination IP address		
00000000	Protocol = 6	TCP/UDP segmentin pituus

Auttaa havaitsemaan väärään osoitteeseen toimitetut paketit.

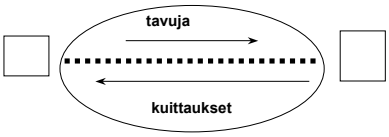
Sisältää IP-otsakkeen tietoja!

- **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**

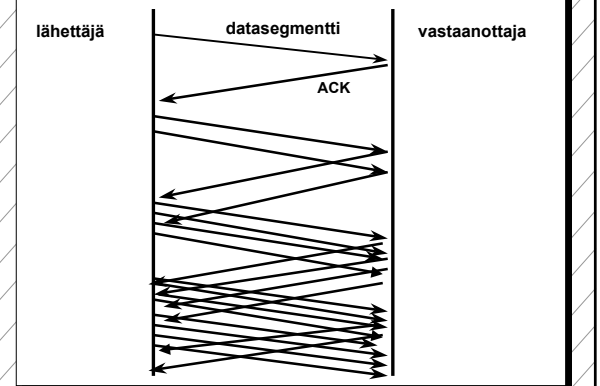
- **ruuhkaikkuna**
  - **"paljonko tavuja (segmenttejä) lähettäjällä saa korkeintaan olla verkossa liikkeellä"**
    - **paljonko lähettäjä saa kuormittaa verkkoa**
  - **kuittaus => ko. tavut jo poistuneet verkosta**



- **Ruuhkaikkunan koko?**
  - **Lähettäjän on itse pääteltävä ja arvioitava sopiva ruuhkaikkunan koko**
    - **kukaan muu ei sitä kerro!**
    - **uudelleenlähetysajastin laukeaa => on ruuhkaa**
    - **kuittaukset tulevat tasaisesti => ei ole ruuhkaa**
  - **Internet-verkon kuormitus voi vaihdella paljon**
- **Dynaaminen ruuhkaikkunan koko:**
  - **ruuhkaikkunaa kasvatetaan, kunnes törmätään ruuhkaan**
    - **ensin kasvatetaan melko nopeasti, sitten varovaisemmin**
  - **sen jälkeen ruuhkaikkunaa pienennetään reilusti**
  - **ja aletaan uudestaan kasvattaa ruuhkaikkunaa**

Hitaan aloituksen algoritmi (slow start)

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



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

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

Uudelleenlähetysjastimen hallinta

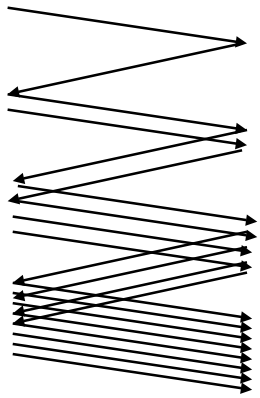
- uudelleenlähetysajastin (retransmission timer)
  - asetetaan aina kun segmentti lähetetään
  - ruuhkaa, jos kuittaus ei saavu ajoissa
- mikä on sopiva ajastimen aika?
  - kuittaus aika vaihtelee suuresti
  - vaihtelu on myös nopeaa
- dynaaminen arvo
  - saadaan jatkuvien verkon suorituskykymittauksien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

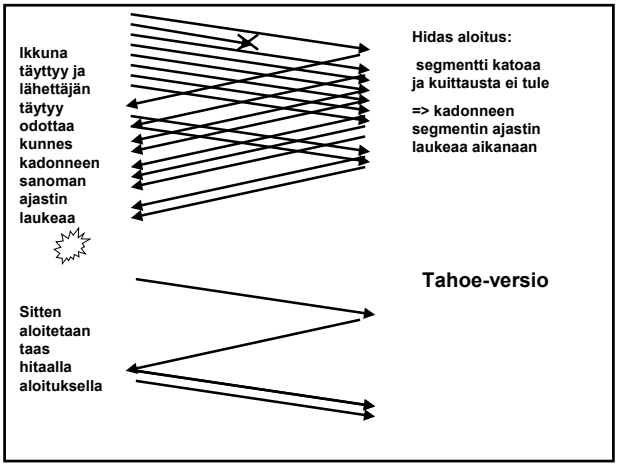
- RTT
  - arvio kiertoviiveelle (round-trip time)
  - mitataan jokaisen lähetetyn segmentin kiertoviive M

    RTT = αRTT + (1-α)M, tyypillisesti α = 7/8
- uudelleenlähetysajastimen arvo βRTT
  - aluksi β oli aina 2
  - parannus: otetaan huomioon myös poikkeama D (deviation) oletetun ja saadun kiertoviiveen välillä |RTT-M|

    D = α D + (1- α)|RTT-M|
  - ajastimen arvo = RTT +4*D

- uudelleenlähetysten vaikutus ajastimeen
  - kumpaan segmenttiin kuittaus kohdistuu?
- Karnin algoritmi
  - ei oteta huomioon uudelleenlähettyjen segmenttien kuittauksia RTT:n laskemisessa

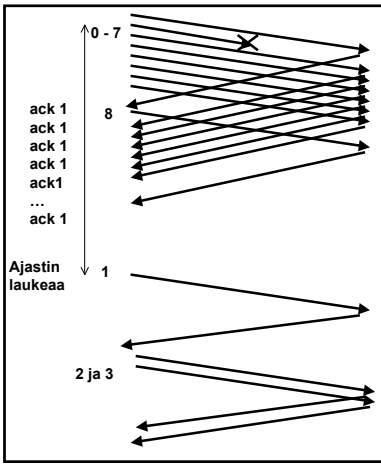


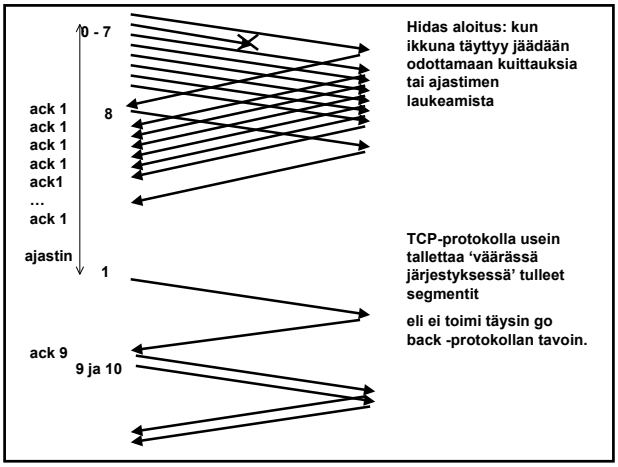


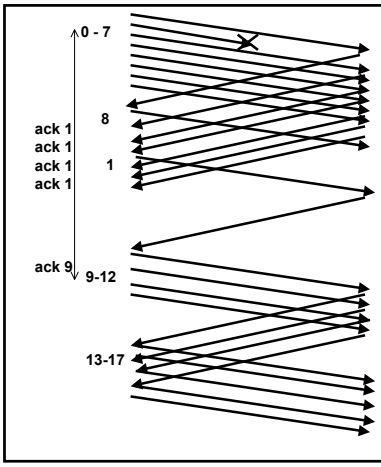
Parannuksia ruuhkanvalvontaan

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

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







- hidas aloitus ja ruuhkan valvonta ongelmallisia langattomassa yhteydessä
  - Miksi?
- Lisäparannuksia ruuhkanhallintaan
  - esim. Vegas
    - ruuhkan ennustaminen ennen ajastimen laukeamista
    - ruuhkaikkunaa ei kasvateta aina ruuhkaan asti
    - RED (random early detection)
  - entä UDP?

TCP langattomassa verkossa

- monet TCP-toteutukset optimoitu luotettaville lankaverkoille => suorituskyky langattomissa verkoissa erittäin huono
  - ruuhkanvalvonta-algoritmi olettaa ajastimen laukeamisen johtuvan ruuhkasta
    - lähettämistä hidastetaan, jotta verkon kuormitus pienenisi ja ruuhkaa ei syntyisi
  - langattomat yhteydet ovat epäluotettavia ja paketteja katoaa
    - kadonneet paketit syytä lähettää nopeasti uudelleen
    - lähetystä pitäisi päinvastoin nopeuttaa!

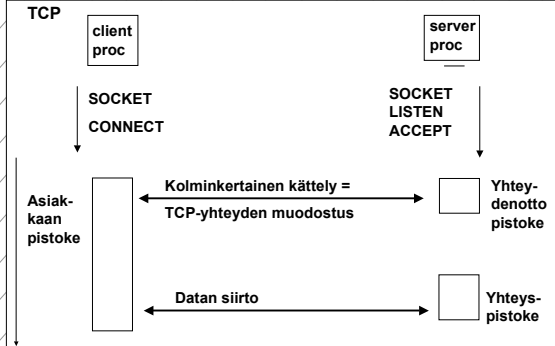
TCP-yhteyden hallinta

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

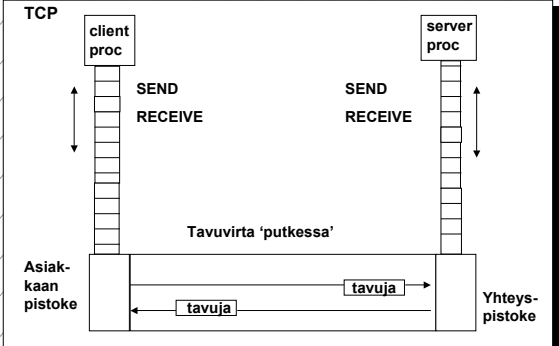
- CONNECT-primitiivi
  - parametreina
    - IP-osoite ja porttinumero
    - suurin hyväksyttävä segmentin koko
    - muuta tietoa, esim. salasana

↓

- TCP-segmentti, jossaSYN-segmentti
  - SYN = 1
  - ACK = 0



TCP-yhteyden muodostaminen



Pistoke + TCP = tavuputki prosessien välissä

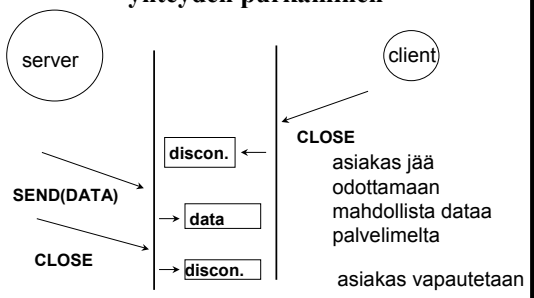
- **TCP-yhteys on tavuvirtaa, ei sanomavirtaa**
  - **lähetettäessä neljä 512 tavun pätkää vastaanottaja saa joko**
    - **neljä 512 tavun pätkää**
    - **kaksi 1024 tavun pätkää**
    - **yhden 2048 tavun pätkän**

Segmentit lähetetään neljänä eri IP-pakettina

Ne luovutetaan vastaanottajalle yhdellä READ-kutsulla

neljä 512 tavun segmenttiä **yksi 2048 tavun data**

yhteyden purkaminen



symmetrinen yhteyden purku

C-rutiineina

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

palvelin:

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

int listen(int socket, int backlog)

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

asiakas:

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

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

sanoman lähetys annetun pistokkeen kautta

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

sanoma vastaanotto annetusta pistokkeesta ilmoitettuun puskuriin

Pistokeohjelmointia Javalla

- **Socket clientSocket = new Socket("hostname", 6789);**
- **clientSocket.close();**
- **ServerSocket welcomeSocket = new Server Socket (6789);**
- **Socket connectionSocket = welcomeSocket;**
- **accept()**

- **(esimerkki kirjassa Kurose, Ross, Computer Networking, A Top-Down Approach Featuring the Interbet)**

Pistokeohjelmointi

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

Yhteenveto

- **Kuljetuskerroksen palvelut**
  - **UDP**
  - **TCP**
    - **luotettava tavuvirta**
      - **yhteyden muodostus ja purku**
      - **numerointi, tarkistussumma,**
      - **kuittaus, uudelleenlähetys, Go-back N**
      - **vuonvalvonta: vastaanottoikkuna (liukuva ikkuna)**
      - **ruuhkanhallinta: hidas aloitus**
  - **pistokeohjelmointi**