Aalto-yliopisto
Perustieteiden
korkeakoulu

Sovellukset (osa 2) ja verkko-ohjelmointi

CSE-C2400 Tietokoneverkot

21.1.2014

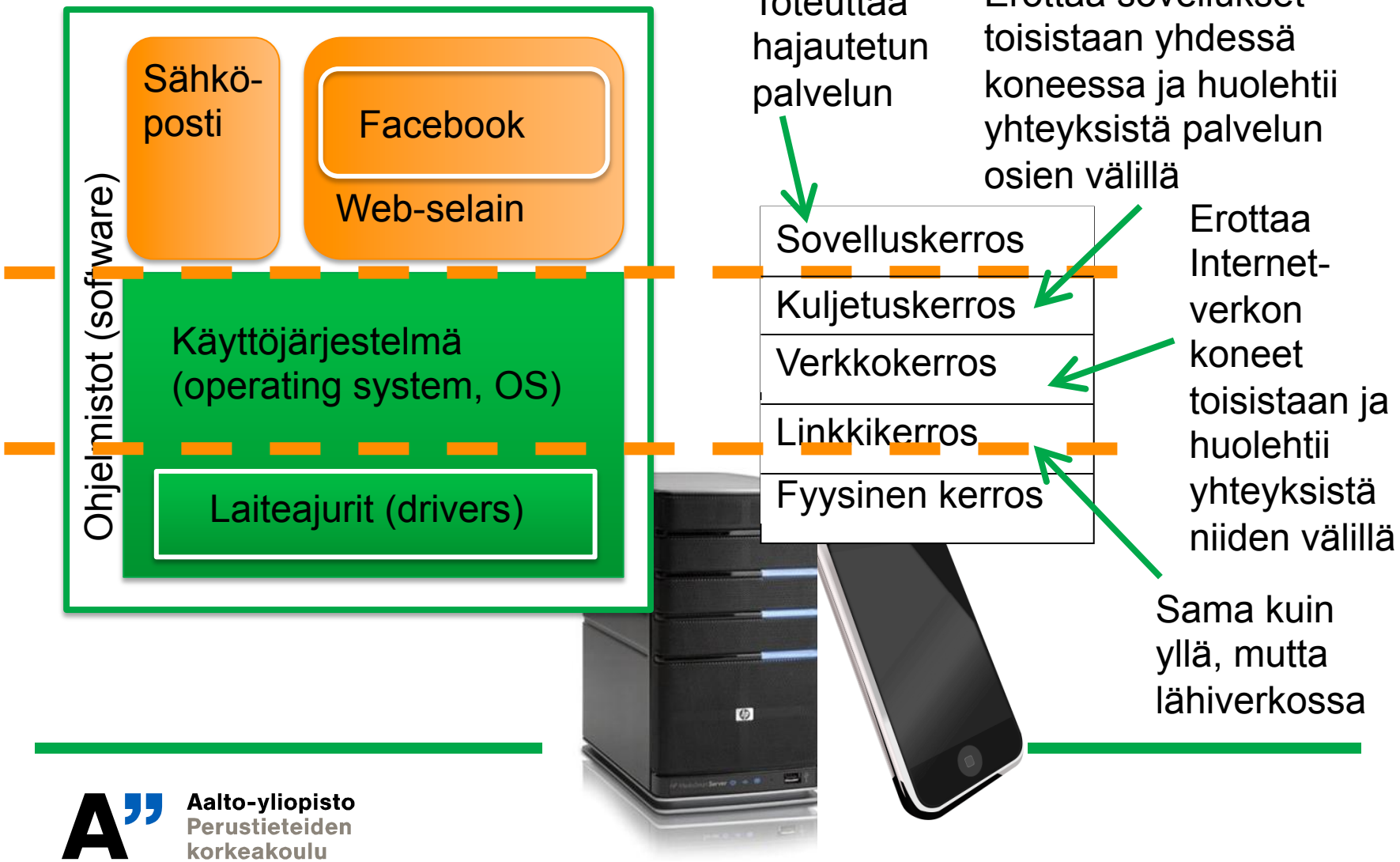
Matti Siekkinen

Sisältö adaptoitu seuraavista lähteistä: J.F. Kurose and K.W. Ross: Computer Networking: A Top-Down Approach 6th ed. -kirjan lisämateriaali, T. Lindholm, S. Tarkoma: Johdatus tietotekniikkaan-kurssin Sovelluskerros-luennon materiaali, S. Sorvakko: Tietokoneverkot-kurssin Network Programming-luennon materiaali.

Lyhenteitä ja terminologiaa

- P2P = peer-to-peer (vertaisverkko)
- DHT = Distributed Hash Table (Hajautettu tiivistetaulu)
- VoIP = Voice over IP (Internet-puhelu)
- Churn = vertaisverkossa päätelaitteiden poistuminen ja uusien saapuminen
- Chunk = BitTorrentissa tiedoston palanen
- BitTorrent = vertaisverkkoa käyttävä tiedostonjakelusovellus
- Tracker = BitTorrentin seurantapalvelin
- Torrent = joukko BitTorrentia ajavia päätelaitteita jotka jakavat samaa tiedostoa
- Overlay network = päätelaitteiden muodostama sovellustason looginen verkko (kuoriverkko)
- Topologia = verkon rakenne, miten laitteet ovat yhteydessä toisiinsa

Internet-protokollapino

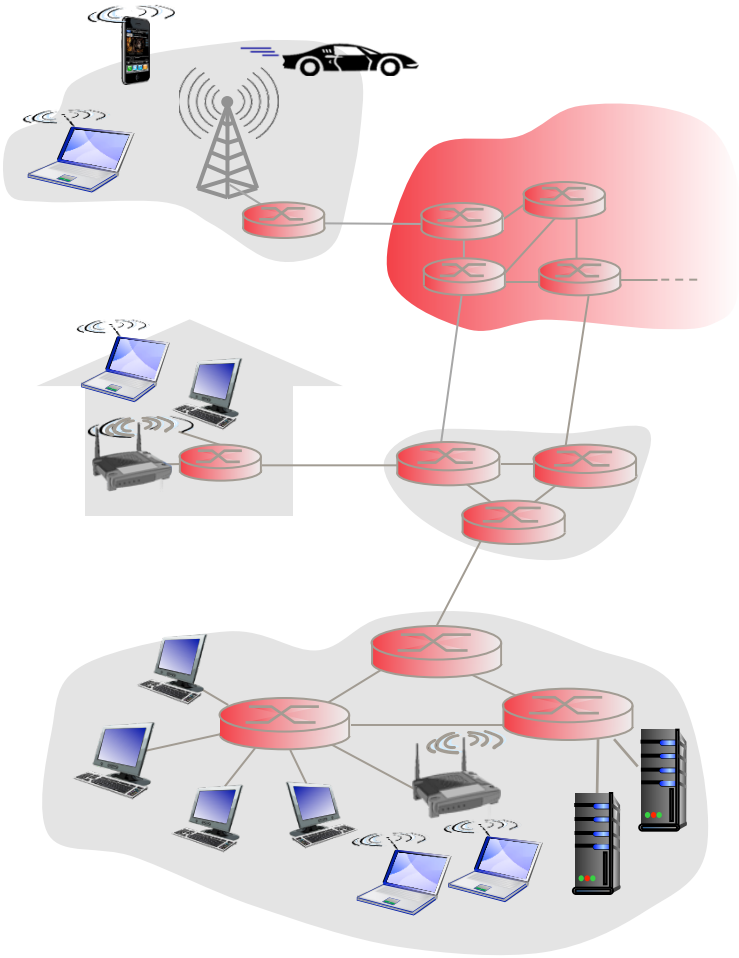


Sisältö

- Jatkoa sovelluksille: vertaisverkot (P2P)
  - Asiakas-palvelin vs. vertaisverkot
  - BitTorrent
  - Hajautetut tiivistetaulut (DHT)
- Verkko-ohjelmointi soketeilla
  - TCP ja UDP soketit ja käyttö
  - Blokkaava vs. blokkaamaton soketti
- Web-ohjelmointi ja WebSocket

Vertaisverkot (P2P)

- Toisenlainen sovellusarkkitehtuuri kuin asiakas-palvelin
- Ei 24/7 palvelimia
- Päätelaitteet viestivät suoraan toistensa kanssa
  - Sekä asiakas että palvelin
- Päätelaitteet (peer) eivät aina verkossa, IP osoitteet vaihtuvat
- Esimerkkejä sovelluksista:
  - tiedostonjakelu (BitTorrent)
  - median suoratoisto (KanKan)
  - VoIP eli IP-puhelut (Skype)

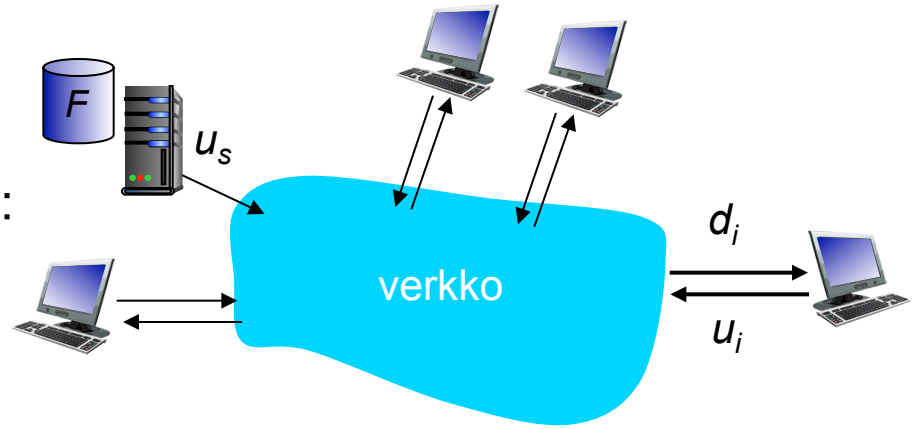


Tiedostonjakelu: asiakas-palvelin vs vertaisverkot

- Kauanko kestää jakaa tiedosto (*F* tavua) yhdeltä palvelimelta *N*:lle päätelaitteelle(PL)?
  - Päätelaitteiden ylä- ja alakaistat ovat rajallisen levyiset

Tiedoston jakoon kuluva aika: asiakas-palvelin

- *palvelin:* lähettää N kopiota tiedostosta peräkkäin:
  - yhteen lähetykseen kuluva aika: F/u_s
  - N lähet. kuluva aika: NF/u_s
- *PL (asiakas):* jokainen vastaanottaa yhden kopion
  - d_{min} = pienin asiakkaan latausnopeus
  - hitaimman asiakkaan latausaika: F/d_{min}



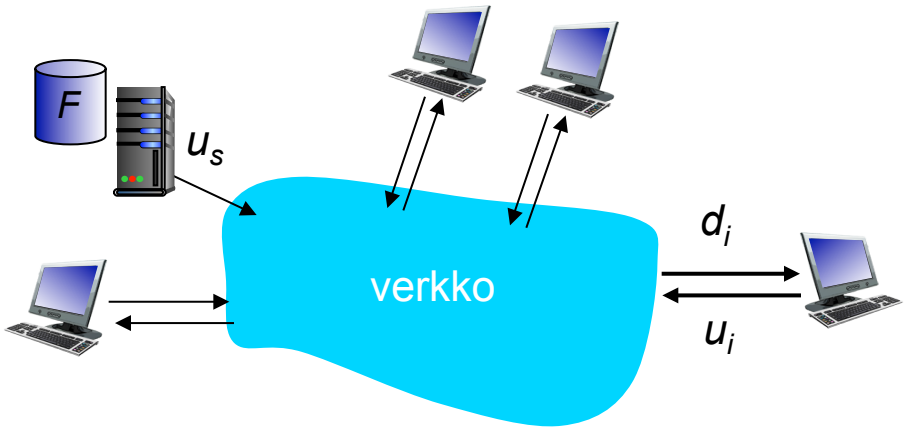
kasvaa lineaarisesti N:n funktiona

time to distribute F to N clients using client-server approach

$$D_{c\text{-}s} \geq max\{NF/u_s, F/d_{min}\}$$

Tiedoston jakoon kuluva aika: vertaisverkko

- *palvelin:* lähettää väh. yhden kopion
  - yhden kopion lähetysaika: F/u_s
- *PL:* jokainen lataa yhden kopion
  - min latausaika: F/d_{min}
- *PL:* yhteensä lataavat *NF* tavua
  - max lähetysnopeus (rajoittaa max latausnopeutta) on $u_s + \Sigma u_i$



F:n jakoon N asiakkaalle kuluva aika vertaisverkon avulla

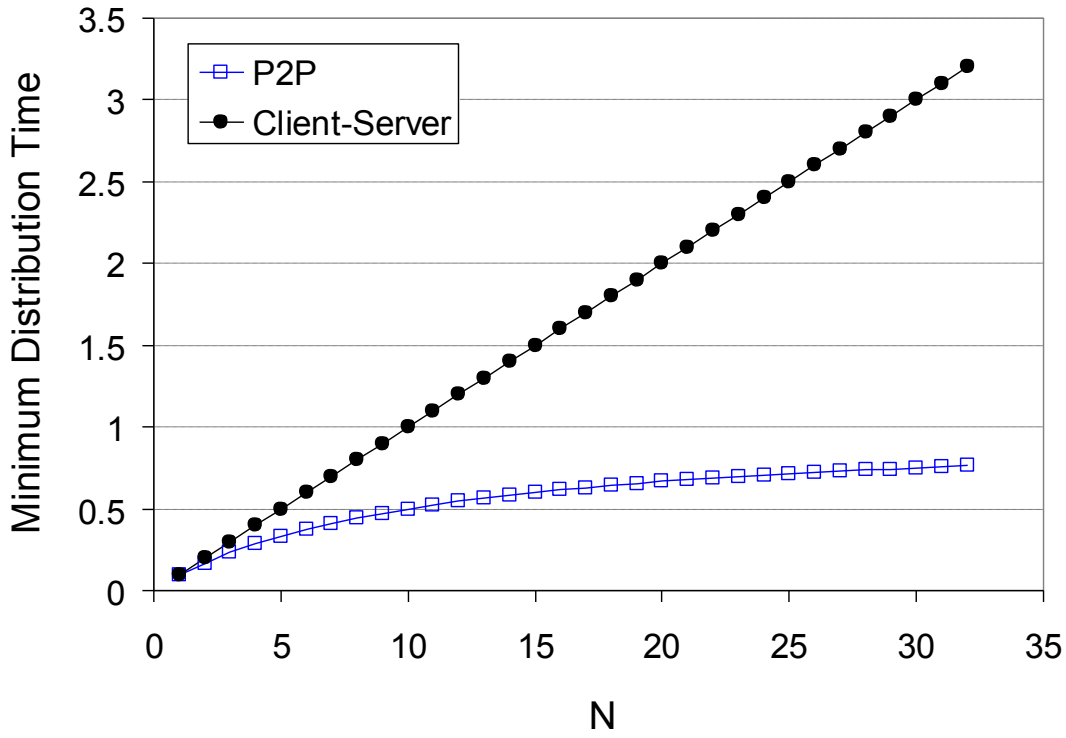
$$D_{P2P} \geq max\{F/u_s, F/d_{min}, NF/(u_s + \Sigma u_i)\}$$

kasvaa lineaarisesti *N:n* funktiona…

… mutta tämä samoin, koska jokainen asiakas lisää verkon kapasiteettia

Asiakas-palvelin vs. vertaisverkko: vertailua

asiakkaan lähetysnopeus = u, F/u = 1h, u_s = 10u, $d_{min} \geq u_s$



vertaisverkko on itseskaalautuva (self scaling)

Sisältö

- Jatkoa sovelluksille: vertaisverkot (P2P)
  - Asiakas-palvelin vs. vertaisverkot
  - BitTorrent
  - Hajautetut tiivistetaulut (DHT)
- Verkko-ohjelmointi soketeilla
  - TCP ja UDP soketit ja käyttö
  - Blokkaava vs. blokkaamaton soketti
- Web-ohjelmointi ja WebSocket

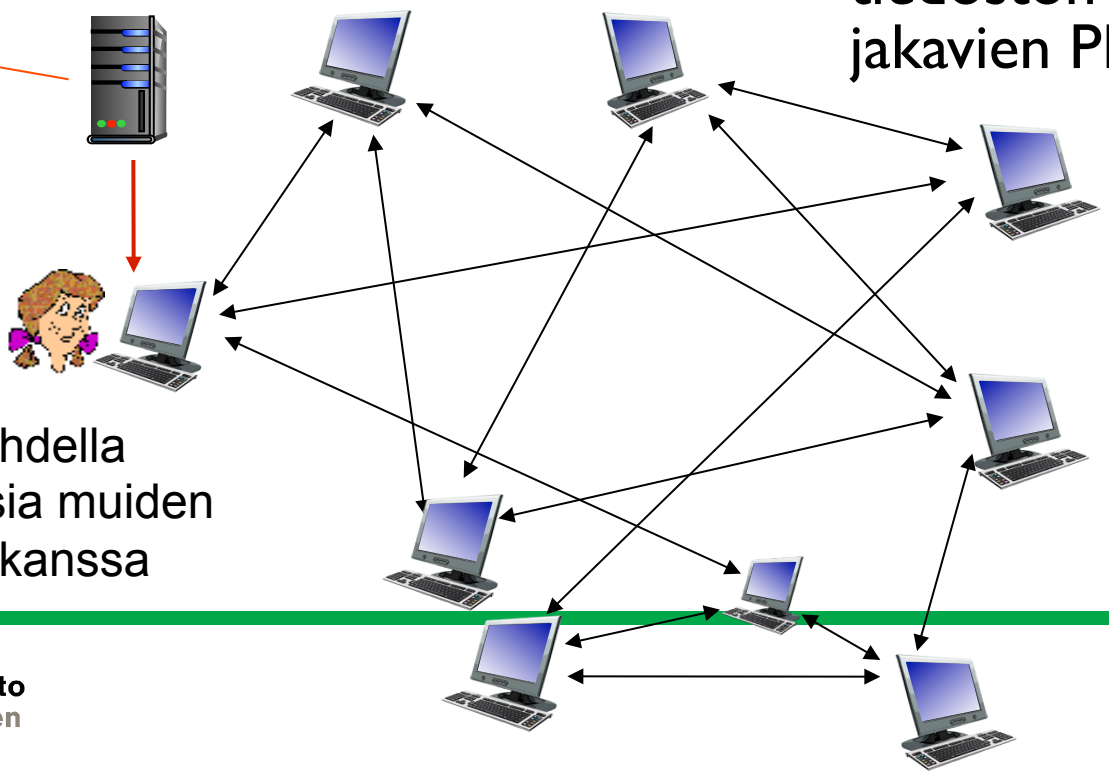
Tiedostonjakelu vertaisverkoissa: BitTorrent

- tiedosto jaettu 256Kb palasiin (chunks)
- PL:t (peers) torrentissa lähettävät ja vastaanottavat tiedostonpalasia

torrent: tietyn tiedoston palasia jakavien PL:n joukko

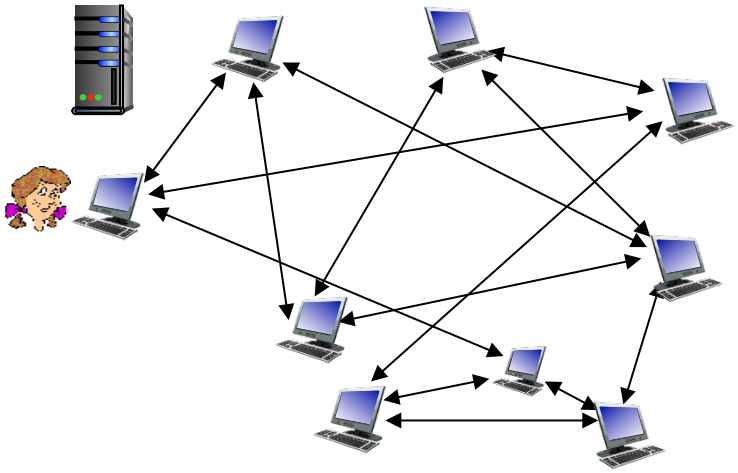
tracker: pitää kirjaa torrentiin osallistuvista PL:sta

Lissu liittyy …
… saa listan muista PL:sta trackerilta
… ja alkaa vaihdella tiedostonpalasia muiden torrentin PL:n kanssa



Tiedostonjakelu vertaisverkoissa: BitTorrent

- torrentiin liittyvä PL:
  - ei vielä tiedostonpalasia, kerryttää lataamalla muilta
  - rekisteröityy trackeriin saadakseen PL-listan, muodostaa yhteyden osaan PL:sta (“naapurit”)



- PL:t lähettävät toisilleen palasia samalla kun lataavat niitä
- PL voi vaihtaa naapureita joiden kanssa “käy vaihtokauppaa” palasista
- churn: PL:ita poistuu ja uusia liittyy
- kun koko tiedosto ladattu, PL voi joko poistua (itsekäs) tai jäädä torrentin osaksi auttamaan muita lataajia (altruistinen)

BitTorrent: tiedostonpalasten treidaus

tiedostonpalasten lataaminen:

- eri PL:lla eri tiedostonpalaset tietyllä ajanhetkellä
- Lissu pyytää aika ajoin jokaiselta naapuriltaan listan palasista jotka heillä on
- *rarest first:* Lissu pyytää seuraavaksi naapurien kesken harvinaisimmat palaset

palasten lähettäminen: tit-for-tat

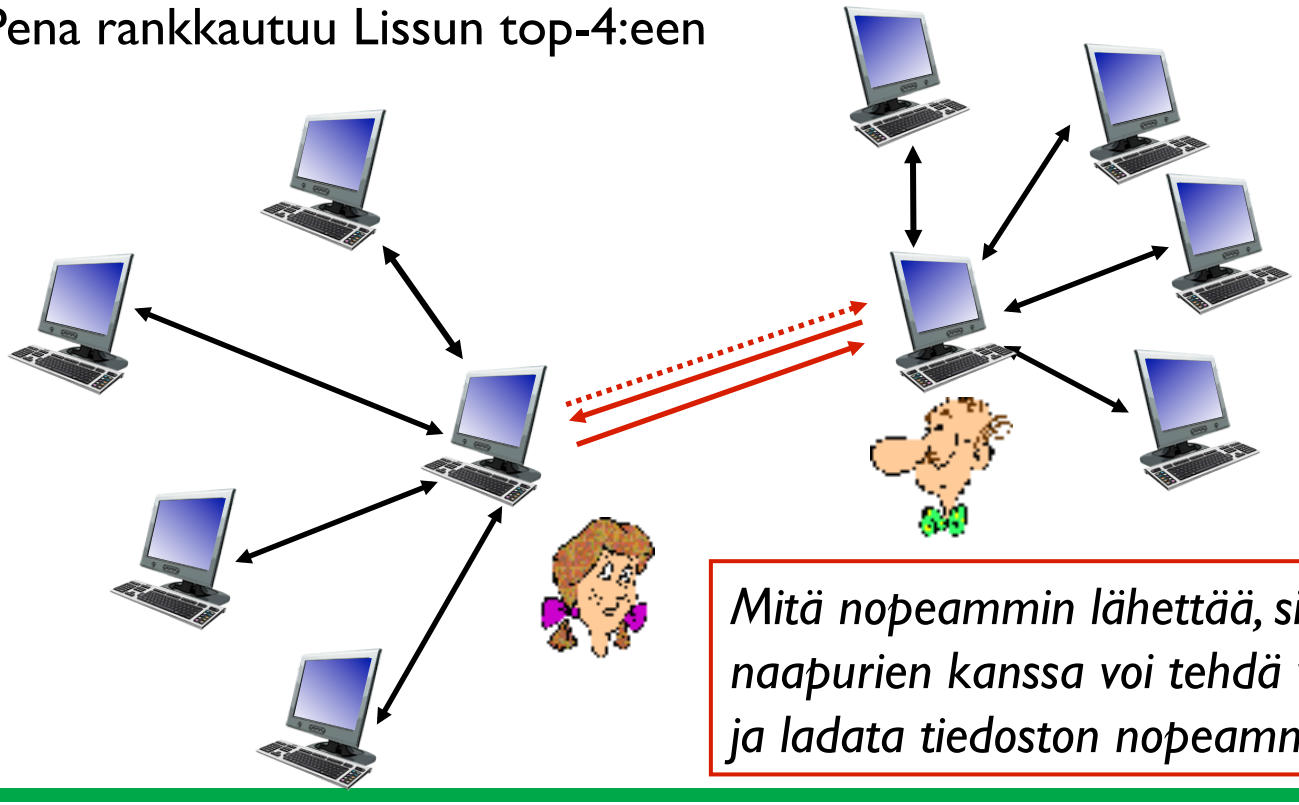
- Rankkaa naapurit niiden lähetysnopeuden mukaan
- Lähetä palasia top-4 naapurille
  - muut naapurit: ei latausta tai lähetystä (choked)
  - top-4 rankkaus 10s välein
- Valitse 30s välein satunnainen naapuri ja ala lähettää palasia
  - “optimistic unchoke”
  - uusi naapuri voi päästä top-4:ään

BitTorrent: tit-for-tat

(1) Lissu valkkaa satunnaisesti Penan (“optimistic unchoke”)

(2) Lissu rankkautuu Penan top-4:een; Pena alkaa myös lähettää

(3) Pena rankkautuu Lissun top-4:een



Mitä nopeammin lähettää, sitä parempien naapurien kanssa voi tehdä vaihtokauppaa ja ladata tiedoston nopeammin.

Sisältö

- Jatkoa sovelluksille: vertaisverkot (P2P)
  - Asiakas-palvelin vs. vertaisverkot
  - BitTorrent
  - Hajautetut tiivistetaulut (DHT)
- Verkko-ohjelmointi soketeilla
  - TCP ja UDP soketit ja käyttö
  - Blokkaava vs. blokkaamaton soketti
- Web-ohjelmointi ja WebSocket

Hajautetut tiivistetaulut (DHT)

- Tiivistetaulu
- Hajautetut tiivistetaulut
- Ympyrämalliset DHT:t ja kuoriverkot (overlay)
- Churn ja hajautetut tiivistetaulut

Yksinkertainen tietokanta

Tietokanta joka tallentaa (avain, arvo) pareja:

- avain: hlön nimi; arvo: sotu

Avain	Arvo
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
…….	……
Lisa Kobayashi	177-23-0199

- avain: elokuvan nimi; arvo: IP-osoite

Tiivistetaulu (Hash table)

- Numeerinen avain helpottaa käsittelyä
- avain = tiiviste(alkup. avain) (tiiviste=hash)

Alkuperäinen avain	Avain	Arvo
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
…….		……
Lisa Kobayashi	9290124	177-23-0199

Hajautettu tiivistetaulu

- Eli “Distributed Hash Table (DHT)”
- Hajautetaan (avain, arvo) parit miljooniin PL:iin
  - Parit jaetaan tasan PL:n kesken
- Ensimmäiset kehitetty 2001
  - Chord, CAN, Pastry, Tapestry: kaikki julkaistu samana vuonna
  - Peruskäsitteen alkuperä vanhempi (hajautetut tietokannat)
- Valtava määrä tutkimusta sittemmin

Hajautettu tiivistetaulu: ominaisuudet

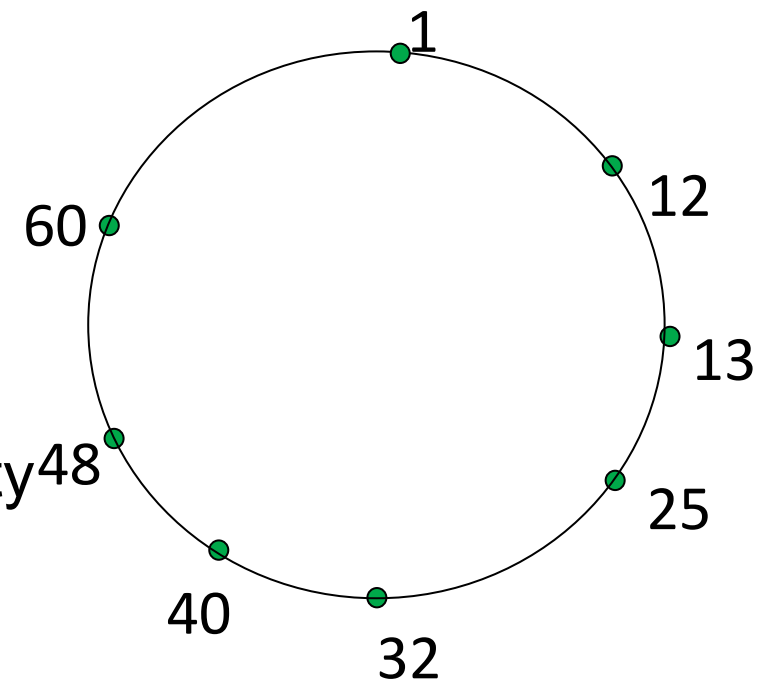
- Mikä tahansa PL:sta voi tehdä *kyselyn* tietokantaan avaimen avulla
  - Tietokanta palauttaa vastaavan arvon
  - Kyselyyn vastaaminen vaatii muutamien viestien lähetyksen PL:n kesken
- Jokainen PL pitää muistissa vain pienen osan toisista PL:sta
- Churn: tietokanta ei saa mennä rikki

Avain-arvo parien jakaminen

- Jokaisella PL:lle annetaan ID
  - Yleensä valitaan satunnaisesti
- yleisesti: avain-arvo pari annetaan sen PL:n hoidettavaksi jonka oma ID on lähinnä avainta
- useimmiten: lähin PL on se jonka ID tulee *ensimmäisenä avaimen jälkeen*
- esim. ID avaruus {0,1,2,3,…,63}
- 8 PL:a: 1,12,13,25,32,40,48,60
  - avain = 51 → PL 60 tallentaa
  - avain = 60 → PL 60 tallentaa
  - avain = 61 → PL 1 tallentaa

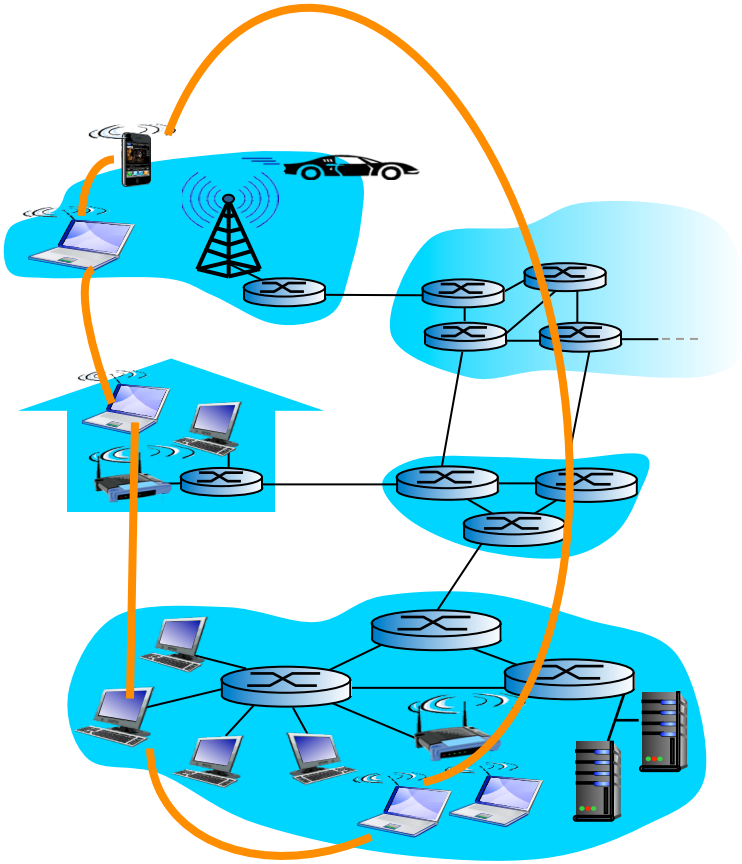
Hajautetun tiivistetaulun rakenne

- Rakenne/topologia määrittää sen miten kyselyt prosessoidaan
  - Viestien reititys
- Ympyrässä PL tietää vain heti seuraavan naapurin
- Chord noudattaa ympyrämallia
- Monia muita rakenteita on kehitetty



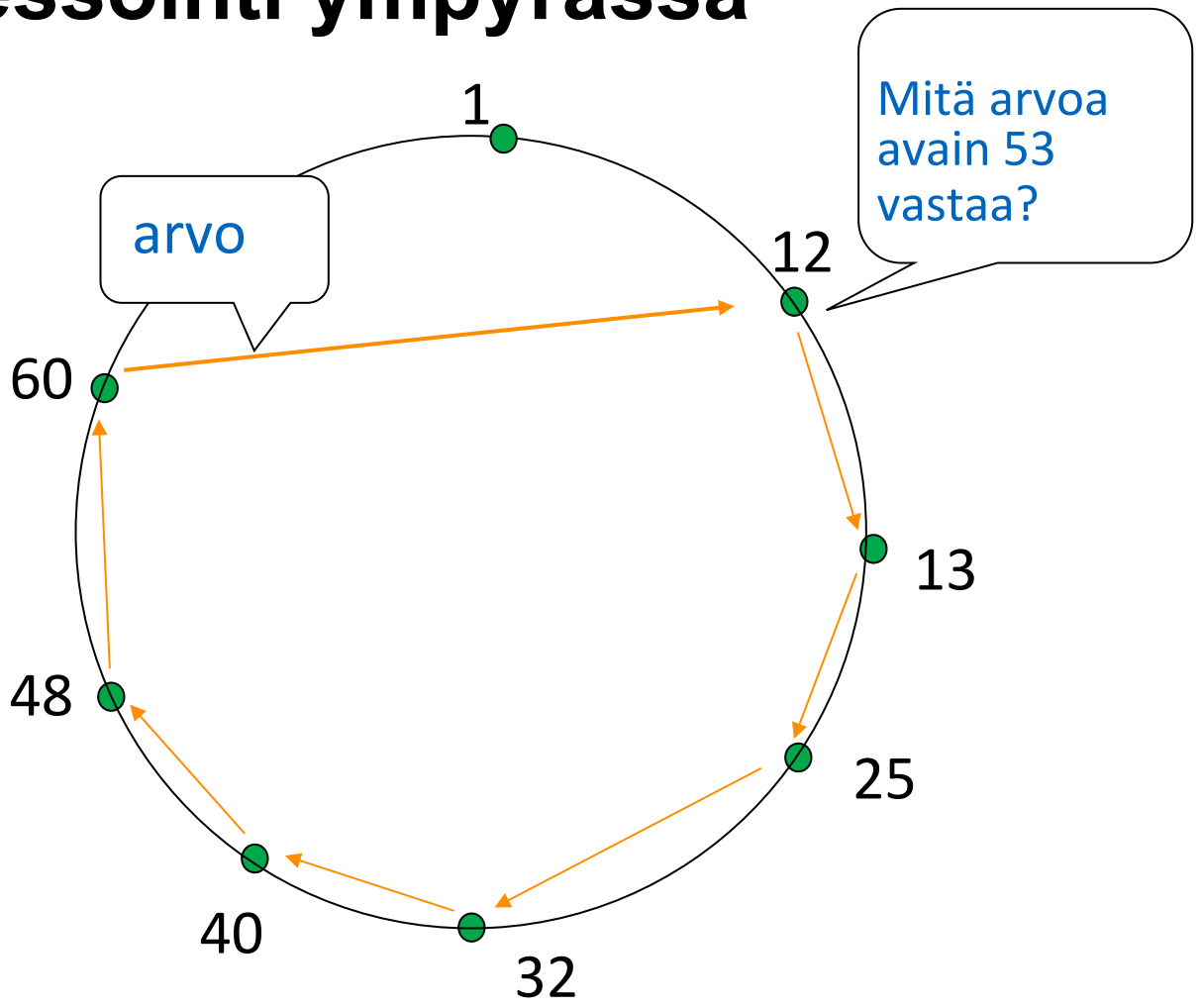
Hajautettu tiivistetaulu on kuoriverkko

- Kuoriverkko = overlay network
- Vain osajoukko fyysisen verkon PL:sta osallistuu
- Tieto siirtyy silti fyysisen verkon topologian mukaisesti
- Kuoriverkon topologiaa voidaan muokata helposti
  - Fyysistä topologiaa ei

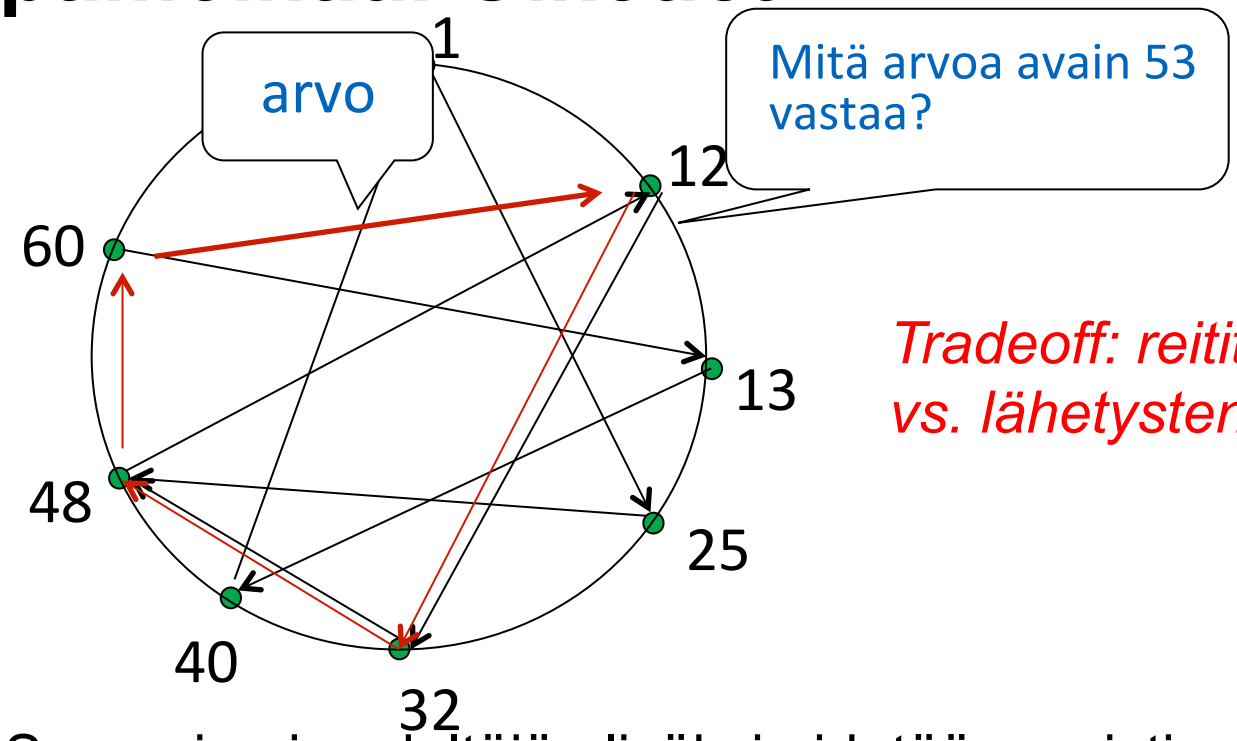


Kyselyn prosessointi ympyrässä

Tarvitaan keskimäärin *O(N)* viestiä prosessoimaan kysely kun DHT:n koko on *N*



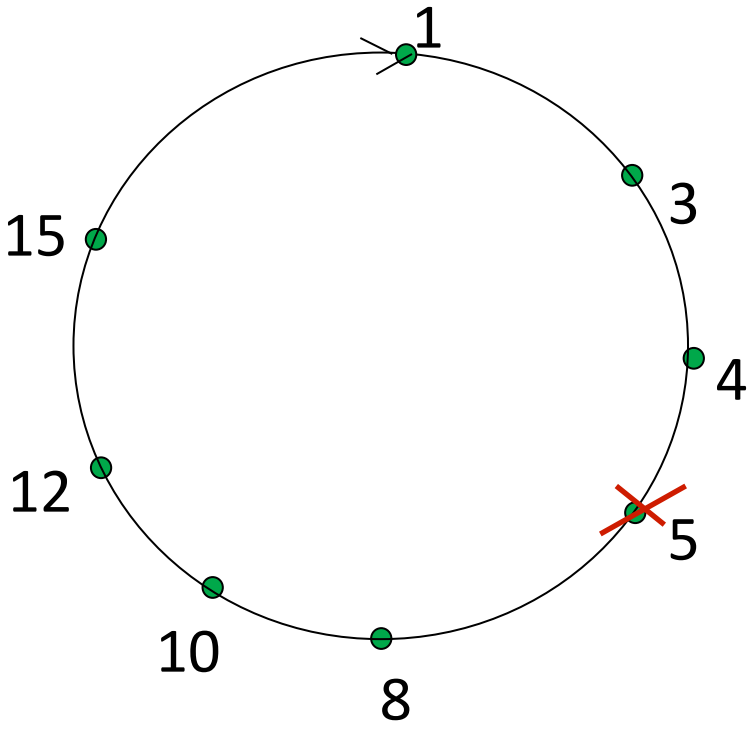
Optimointia: Oikotiet



Tradeoff: reititystaulun koko vs. lähetysten määrä!

- Seuraajan ja edeltäjän lisäksi pidetään muistissa oikoteitä (shortcuts)
- Kyselyn prosessointi lyhenee: 6→3 viestiä
- *O(log N)* naapuria muistissa ja *O(log N)* viestiä per kysely on mahdollista oikoteiden avulla

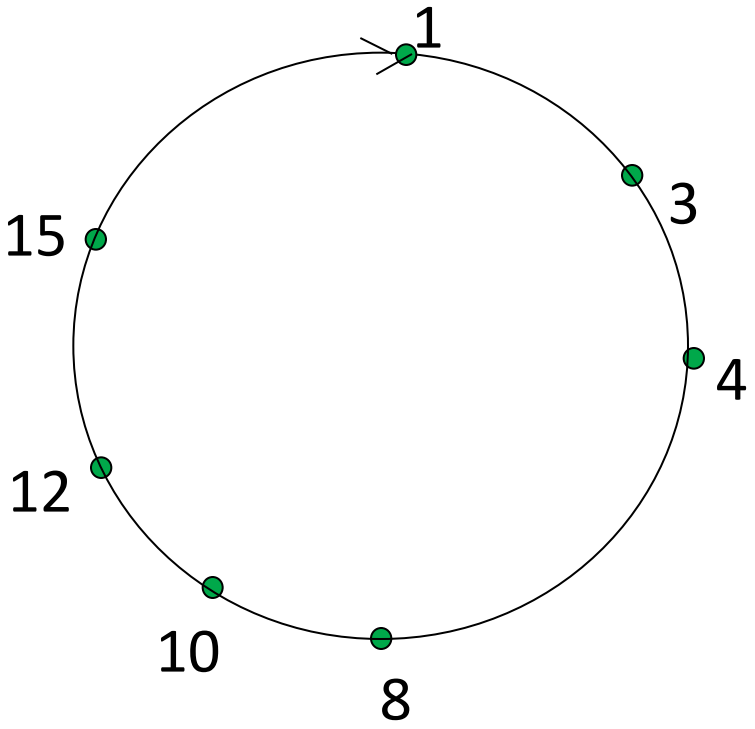
Churnin hallinta



esimerkki: PL 5 poistuu äkisti

- PL:t muistavat kaksi seuraavaa naapuria
- PL:t tarkistavat naapurien elonmerkit → ping viestin aika ajoin
- Seuraava naapuri poistuu → valitse sitä seuraava uudeksi seuraajaksi

Churnin hallinta



- PL:t muistavat kaksi seuraavaa naapuria
- PL:t tarkistavat naapurien elonmerkit → ping viestin aika ajoin
- Seuraava naapuri poistuu → valitse sitä seuraava uudeksi seuraajaksi

esimerkki: PL 5 poistuu äkisti

• PL 4 huomaa PL 5:n poistuneen → PL 8 seuraajaksi

• PL 4 kysyy PL 8:lta kuka sen seuraaja on → PL 8:n seuraaja omaksi toiseksi seuraajaksi

Haasteet ja sovellutukset

Sovellutuksia

- BitTorrent käyttää hajautettua tiivistetaulua (Kademlia)
  - Tavallaan hajautettu trackeri: avain=torrent, arvo=IP-osoitteet
  - Käytetään keskitettyjen trackerien lisäksi
- Cassandra NoSQL tietokanta
  - Käyttäjiä mm. Facebook, Netflix, reddit, Twitter
- Amazonin Dynamo tietokanta
  - DynamoDB korvasi

Haasteita

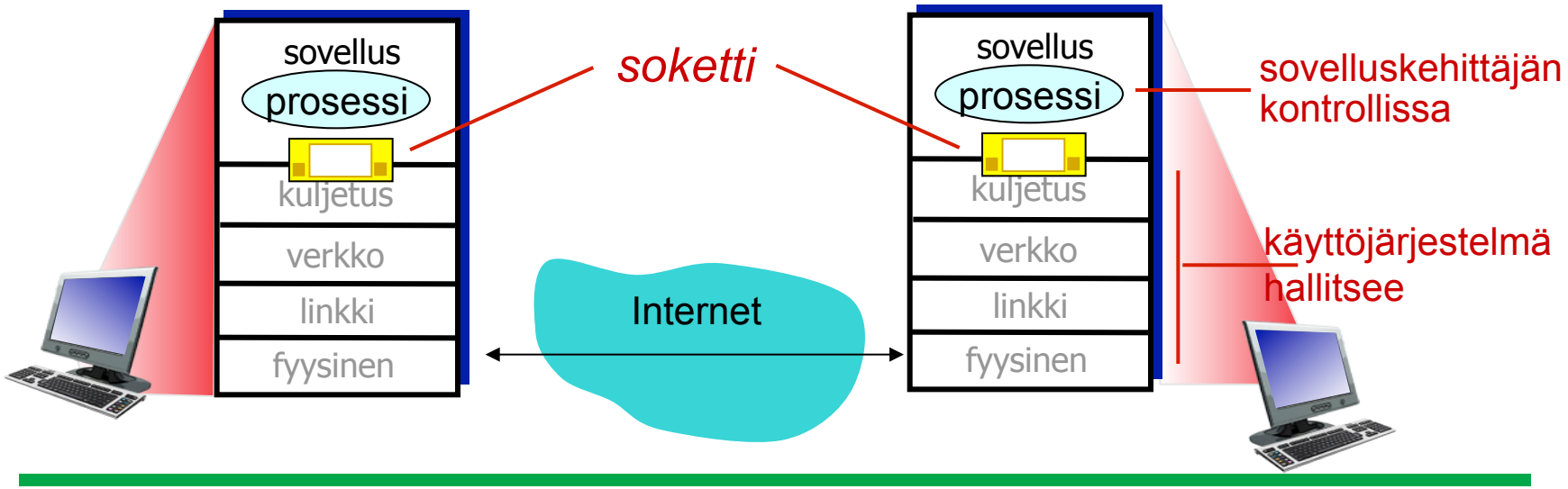
- Epätäsmälliset kyselyt
  - Range queries, esim. MUSE*
  - Tiivistefunktiot rikkovat (tarkoituksella) alkuperäisten avainten järjestyksen
- Kuormantasaus
  - Churn, heterogeeniset PL:t, epätasainen avainten suosio (kyselyt)
- Topologia: kuoriverkko vs. fyysinen verkko
  - Viestien reititys maapallon laidalta toiselle ja takaisin
- Mobiililaitteet ja jatkuva ylläpitoliikenne → akunkesto

Sisältö

- Jatkoa sovelluksille: vertaisverkot (P2P)
  - Asiakas-palvelin vs. vertaisverkot
  - BitTorrent
  - Hajautetut tiivistetaulut (DHT)
- Verkko-ohjelmointi soketeilla
  - TCP ja UDP soketit ja käyttö
  - Blokkaava vs. blokkaamaton soketti
- Web-ohjelmointi ja WebSocket

Verkko-ohjelmointi soketeilla

- *tavoite:* opitte ohjelmoimaan asiakas-palvelin sovelluksia jotka viestivät käyttäen soketteja
- *soketti/pistoke:* ”ovi” sovellusprosessin ja kuljetuskerroksen protokollan (TCP tai UDP) välissä



Sokettirajapinta (socket API)

- Käyttöjärjestelmä tarjoaa rajapintoja IP-verkkoon
  - Sokettirajapinta eli socket API (Application Programming Interface)
- Yleisin rajapinta nimeltä Berkeley sockets
  - Alkuperäinen versio BSD-Unixissa v 1983
  - Nykyään käytännössä joka käyttöjärjestelmässä
- Sisältää keskeiset funktiot joiden avulla sovelluskehittäjä voi toteuttaa verkon yli viestivän sovelluksen
  - Ohjelmien rajapinta TCP- ja UDP- pohjaiseen tiedonsiirtoon

Sokettiohjelmointi

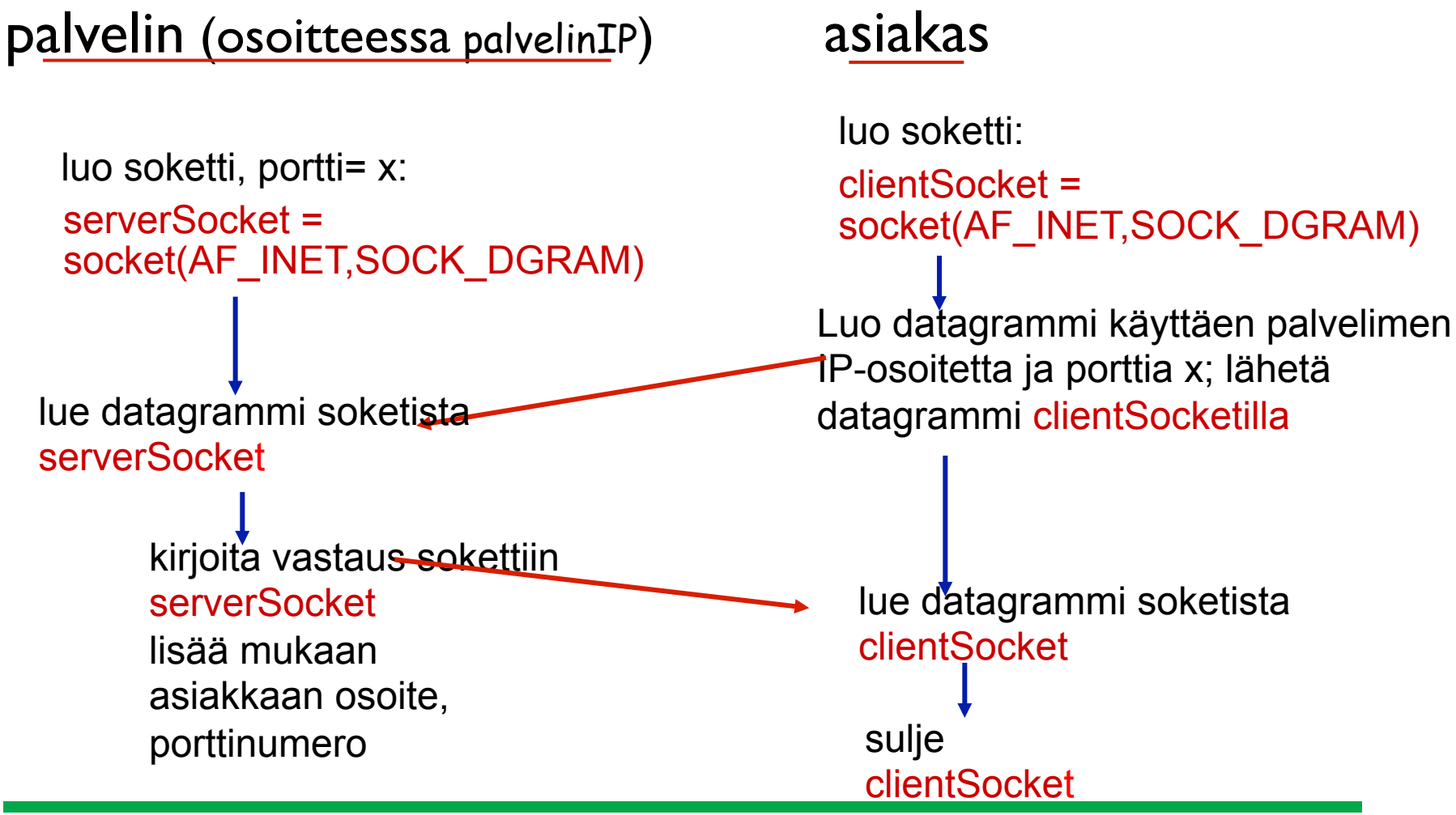
- Kaksi sokettityyppiä, erilaiset kuljetuspalvelut
  - *UDP:* epäluotettava datagrammipalvelu
  - *TCP:* luotettava bittivirtapalvelu
- Esimerkkisovellus:
  - Asiakas lukee rivin käyttäjän kirjoittamia merkkejä ja lähettää ne palvelimelle
  - Palvelin vastaanottaa datan ja muuntaa merkit isoiksi kirjaimiksi
  - Palvelin lähettää muunnetun tekstin asiakkaalle
  - Asiakas vastaanottaa muunnetun tekstin ja printtaa sen näytölle

Sokettiohjelmointi *UDP:llä*

- UDP ei luo yhteyttä asiakkaan ja palvelimen välille
  - ei kättelyä ennen tiedonsiirtoa
  - lähettäjän pitää määrittää vastaanottajan IP-osoite (tai nimi→DNS) ja porttinumero jokaisessa viestissä
  - vastaanottaja näkee lähettäjän IP osoitteen ja porttinumeron saapuneesta paketista

  → tilaton protokolla
- UDP on epäluotettava
  - Viestit ei välttämättä mene perille
  - Viestit voi saapua eri järjestyksessä kuin missä ne lähetettiin
- Sovelluksen näkökulmasta epäluotettava tiedonsiirto viestimuotoisena (datagrammi) asiakkaan ja palvelimen välillä

Asiakas/palvelin UDP-soketit



Esimerkkisovellus: UDP-asiakas

Python UDPClient

sisällytä Pythonin socket kirjasto →

```
from socket import *
serverName = ‘hostname’
serverPort = 12000
```

luo UDP-soketti →

```
clientSocket = socket(socket.AF_INET,
                      socket.SOCK_DGRAM)
```

tallenna input viesti →

```
message = raw_input(’Input lowercase sentence:’)
```

Liitä palvelimen osoite ja porttinro viestiin; lähetä sokettiin →

```
clientSocket.sendto(message,(serverName, serverPort))
```

lue vastaus soketista ja tallenna string muuttujaan →

```
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
```

printtaa vastaus ja sulje soketti →

```
print modifiedMessage
clientSocket.close()
```

Esimerkkisovellus: UDP-palvelin

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage, clientAddress)
```

luo UDP soketti

sido soketti paikalliseen porttiin 12000

ikuinen silmukka

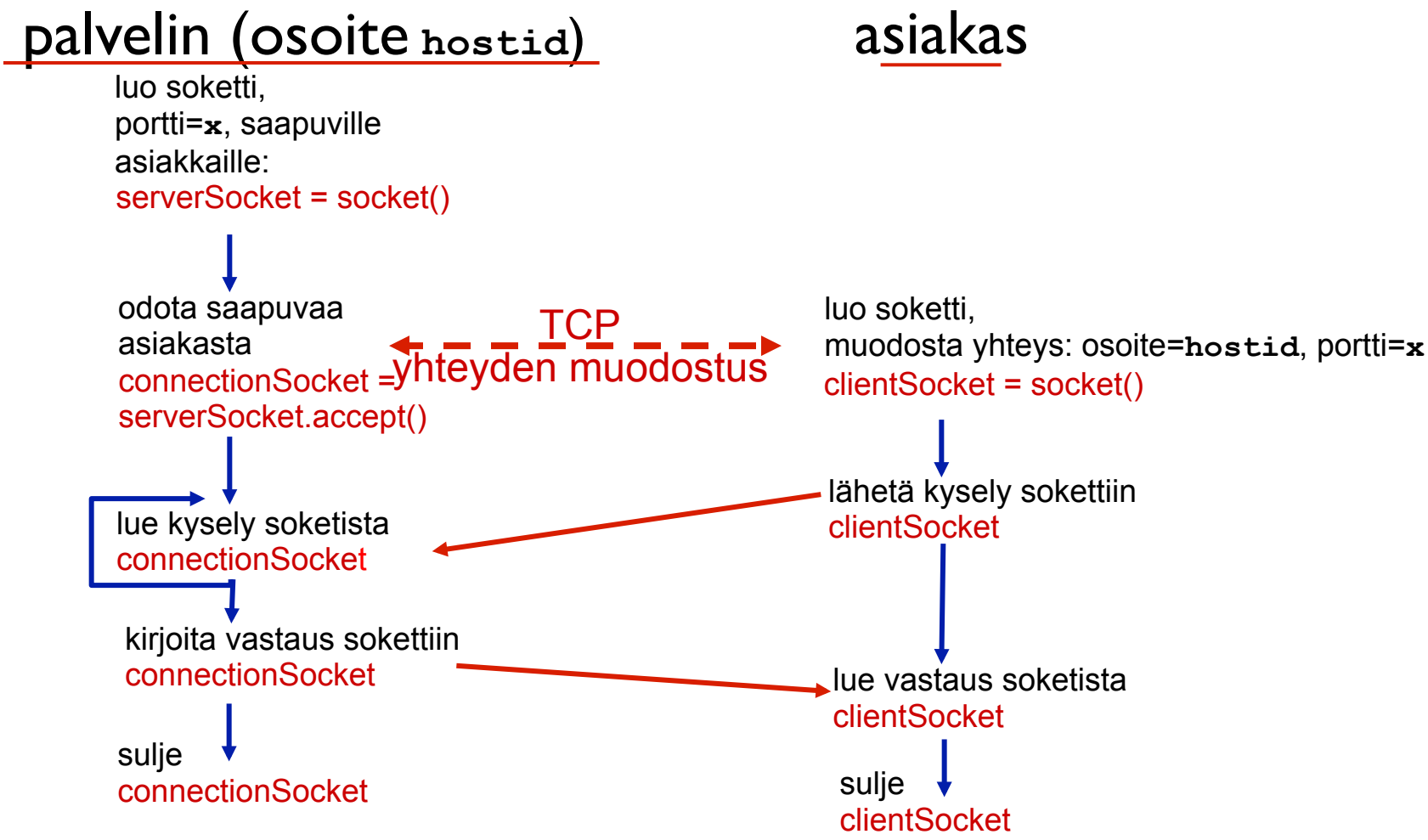
lue viesti ja asiakkaan IP osoite ja porttinro soketista

lähetä muokattu viesti takaisin asiakkaalle

Sokettiohjelmointi *TCP:llä*

- Asiakas ottaa yhteyttä palvelimeen
  - Palvelinprosessi pitää olla valmiiksi käynnissä
  - Palvelin on luonut soketin joka vastaanottaa uuden asiakkaan
- Asiakkaan soketti muodostaa TCP yhteyden
  - Sovelluskoodi spesifioi palvelimen IP-osoitteen ja TCP porttinron
- Palvelin luo yhteydenmuodostuksen aikana uuden soketin prosessille joka palvelee kyseistä asiakasta
  - Mahdollistaa usean samanaikaisen asiakkaan
  - Lähettäjän IP-osoite ja porttinro erottaa asiakkaat toisistaan
- Sovelluksen näkökulmasta TCP soketti on luotettava bittivirran siirtopalvelu

Asiakas/palvelin TCP-soketit



Esimerkkisovellus: TCP-asiakas

Python TCPClient

```
from socket import *
serverName = ’servername’
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input(‘Input lowercase sentence:’)
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print ‘From Server:’, modifiedSentence
clientSocket.close()
```

luo TCP soketti, palvelimen portti 12000

Ei tarvita palvelimen nimeä/osoitetta/porttia

Esimerkkisovellus: TCP-palvelin

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024)
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence)
    connectionSocket.close()
```

luo TCP soketti uuden asiakkaan vastaanottamiseksi

palvelin alkaa kuunnella saapuvia TCP asiakkaita

ikuinen luuppi

palvelin blokkaa accept() funktiossa kunnes uusi asiakas saapuu, luodaan uusi soketti palautusarvona

lue tavut soketista (muttei läh. osoitetta kuten UDP soketin kanssa)

sulje asiakkaan soketti (*ei* vastaanottosokettia)

Blokkaava soketti

- “Normaalisti” soketti on blokkaava
  - Esim. kaikki aiemmat esimerkit
- Ohjelman suoritus pysähtyy kunnes jotain tapahtuu, esim:
  - accept() kunnes uusi asiakas saapuu (yhteysavaus)
  - recv() kunnes uutta dataa saapuu
- Hyöty: helppo ohjelmoida
- Haitta: vain yksi aktiivinen soketti per säie (thread)
  - Monta sokettia vaatii yhtä monta threadia → overhead
  - Palvelin tai vertaisverkkoasiakas

Blokkaamaton soketti

- Soketti voidaan määritellä blokkaamattomaksi
  - Esim. `fcntl(sockfd, F_SETFL, O_NONBLOCK);`
  - Nyt esim. read() ei blokkaa
    - Jos ei dataa, palauttaa -1 ja asettaa errno == EAGAIN tai EWOULDBLOCK
- Etu: voidaan hoitaa monta sokettia per säie
- Haitta: Sokettien jatkuva pollaaminen kuormittaa CPU:ta
- Fiksumpikin tapa on olemassa
  - Tarvitaan uusi funktio: select()

Blokkaamaton soketti: select()

- select()-funktion avulla voidaan tarkistaa monta sokettia kerralla
- Blokkaa kunnes:
  - Jossain soketeista tapahtuu jotakin TAI
  - Ajastin kuluu loppuun
  - `O_NONBLOCK` ei vaikutusta
- Palauduttuaan on muokannut bittikartat
  - Esim. merkkaa `readfds` ne joista voi lukea

```
int select(int ndfs, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

bittikartta kirjoitettavista FD:sta (soketit)

suurin käytetty file descriptor (FD) nro

poikkeustilanteet

bittikartta luettavista FD:sta (soketit)

kauanko enintään odotetaan (NULL ikuisesti, 0 ei ollenkaan)

Vielä soketeista

- Asiakasohjelma on mahdollista tehdä ilman sokettiohjelmointiakin
  - Käytetään valmiita kirjastoja jotka piilottaa soketit
    - Kirjastokoodi hoitaa tarvittavat soketit automaattisesti
  - Esim. java.net API: URL luokan openStream() metodi
  - Kirjastokoodin pitää yleensä tukea asiakas-palvelin protokollaa
    - URL voidaan käyttää esim. HTTP-asiakkaan ohjelmoinnissa

Sisältö

- Jatkoa sovelluksille: vertaisverkot (P2P)
  - Asiakas-palvelin vs. vertaisverkot
  - BitTorrent
  - Hajautetut tiivistetaulut (DHT)
- Verkko-ohjelmointi soketeilla
  - TCP ja UDP soketit ja käyttö
  - Blokkaava vs. blokkaamaton soketti
- Web-ohjelmointi ja WebSocket

Web-ohjelmointi

- Isohko joukko paradigmoja, tekniikoita, teknologioita, protokollia
  - Ajax, REST, SOAP, XML, JSON, SPDY, HTML5, JavaScript CSS, WebRTC, Websocket…
  - Kurssi T-110.5140 Network Application Frameworks käsittelee tarkemmin
- *Websocket*
  - Kehitetty osana HTML5:ttä
  - Kätevämpi soketti Web-ohjelmointiin
    - Toteutetaan selaimissa ja Web-palvelimissa
  - Korkeamman tason abstraktio kuin TCP/UDP-soketti
  - Mahdollistaa kahdensuuntaisen viestinnän

Miksi WebSocket?

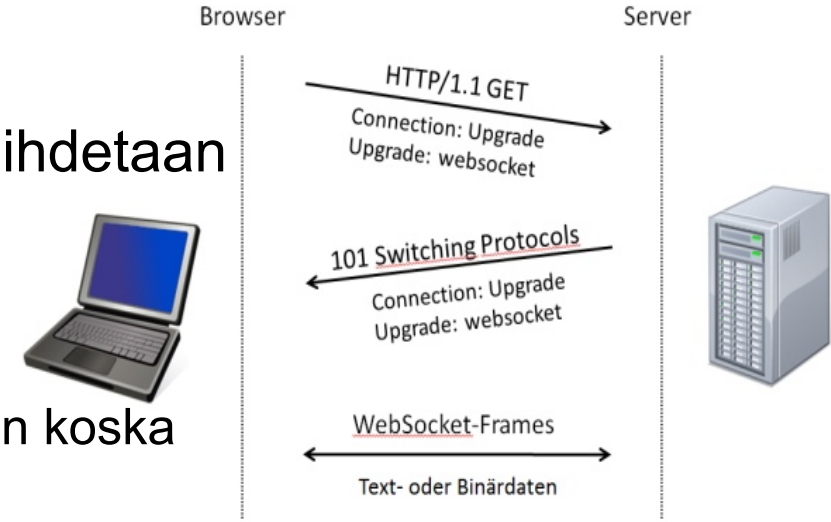
- HTTP on kysely/vastaus protokolla → soveltuu asiakaslähtöiseen tiedon hakemiseen
  - Web sivujen lataus: HTTP GET (+sivun URL), HTTP 200 OK (+sivun sisältö)
  - HTTP vaatii aina kyselyn ennen vastausta → palvelin ei voi lähettää mitään spontaanisti
- Monet modernit Web-sovellukset tarvitsevat palvelinlähtöisen “push”-mekanismin
  - Reaaliaikaiset päivitykset ja synkronointi
  - Pikaviestintä ja some, sähköposti, pelit, sensoridata,…

Palvelimen “push” HTTP:n avulla

- “Polling”: Asiakas tekee jatkuvasti kyselyjä
  - Turhaa kuormaa palvelimelle
  - Kauhea overhead, ei skaalaudu
- “Long polling”: Palvelin viivyttää vastausta jos uutta dataa ei ole
  - Vaatii silti kyselyn per vastaus → lisäliikenne ja mahd. viive
  - Vastauksessa aina HTTP otsakkeet → lisäliikenne
    - HTTP otsakkeet suhteellisen isot (voi olla 8KB) → hyvin merkittävä pienille viestille
  - Ei sovellu lyhyin väliajoin puskevilla sovelluksilla (esim. pelit)

WebSocket: aito push

- Aloitetaan HTTP protokollalla ja vaihdetaan WebSocketiin
  - HTTP Upgrade otsake
- Full-duplex viestintä
  - Kumpikin osapuoli voi lähettää viestin koska vaan Websocketin kautta
  - Lähetetään vain data, ei HTTP-headereita
    - 2-tavun framing
- Websocket toimii TCP:n päällä
  - Yksi TCP yhteys per Websocket
  - TCP-yhteyttä tarkkaillaan ja pidetään elossa Heartbeat-viesteillä (timeout)
  - Alla on siis käytössä TCP-soketti!



WebSocket: JavaScript esimerkki

```
// luodaan uusi WebSocket-objekti
var socket = new WebSocket("ws://echo.websocket.org?
encoding=text");

// Lähetetään viesti
socket.send('Hello, WebSocket');

// Callback-funktiot tapahtumille
socket.onopen = function(event) { socket.send('Hello,
WebSocket'); };
socket.onmessage = function(event) { alert(event.data); }
socket.onclose = function(event) { alert('closed'); }
```

Yhteenveto

- Sovellusarkkitehtuureita on erilaisia
  - Asiakas-palvelin
  - Vertaisverkot
  - Pilvipalvelut voitaisiin vielä erotella näistä
- Vertaisverkot
  - Self scaling ominaisuus
  - BitTorrentissa insentiivit jakaa rakennettu protokollaan
  - Hajautettu tiivistetaulu (DHT) on avain-arvo tietokanta ja muodostaa kuoriverkon
- Sokettiohjelmointi
  - Sokettirajapinta sovellusten ja kulj. kerroksen välissä
  - TCP ja UDP soketeilla ohjelmoidaan hieman eri tavalla
  - select() –funktion avulla voidaan käsitellä monia soketteja
  - WebSocket on korkeamman tason abstraktio
    - Toteutus käyttää TCP sokettia

Internetin protokollapino

Ensi viikon luento