

- 
- 
- 

## 4. Verkkokerros

- **sovelluskerros**
  - ‘asiakas’
- **kuljetuskerros**
  - ‘end-to-end’
- **verkkokerros**
  - ‘deliver packets given to it by its customers’
- **siirtoyhteyskerros**
- **peruskerros**

- 
- 
- 

# Verkkokerroksen palvelut

- **tavoitteet**

- palvelut riippumattomia aliverkkojen tekniikasta
- kuljetuskerros eristettävä aliverkkojen ominaisuuksista
  - lukumäärä
  - tyypit
  - topologia
- kuljetuskerroksen käyttämät **verkko-osoitteet globaaleja**

- 
- 
- 

## connection-oriented ~ connectionless

- **yhteydetön** (Internet, 30 vuoden kokemus)
  - aliverkot ovat luonnostaan epäluotettavia
    - tehtävä: bittien kuljetus
    - operaatiot: send packet, receive packet
    - virheen tarkistus, vuonvalvonta isäntäkoneille
- **yhteydellinen** (puhelin 100 vuoden kokemus)
  - muodostetaan yhteys, neuvotellaan parametrit ( palvelunlaatu (QOS), kustannus)
  - kaksisuuntainen kuljetus, paketit järjestyksessä
  - vuonvalvonta, virhevalvonta

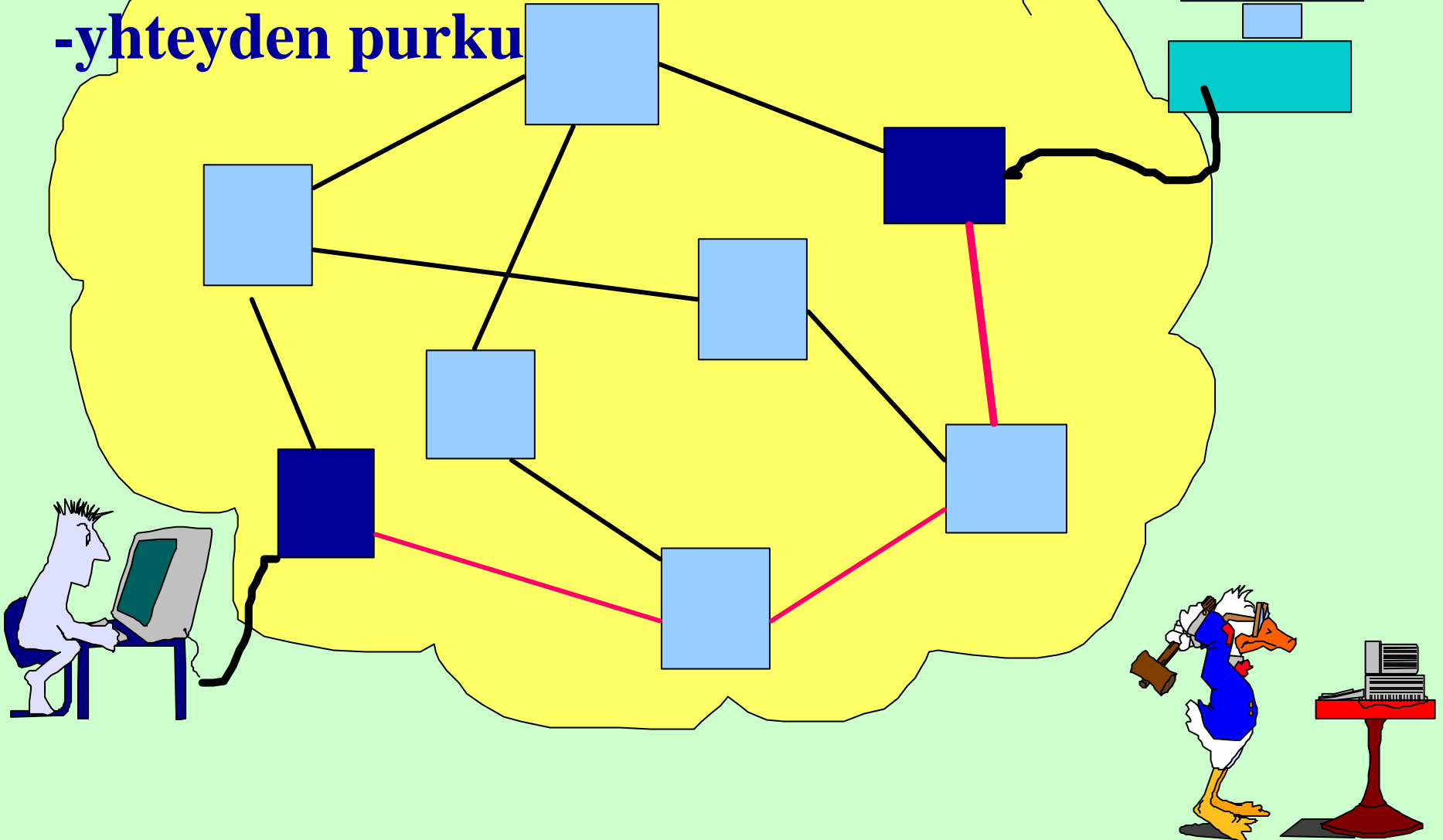
- 
- 
- 

# Virtuaalipiiri (virtual circuit)

- **Pakettikytkentäinen verkko voidaan toteuttaa kahdella tavalla**
  - datasähkeverkkona
    - jokainen paketti käsitellään ja reititetään erikseen
    - pakettien järjestys voi muuttua
  - virtuaalipiiriverkkona
    - ~ piirikytkentäinen verkko
      - ensin yhteyden (virtuaalipiirin) muodostus
      - sitten pakettien lähettäminen yhteyttä pitkin
    - ATM, X.25

# Piirikytkentäinen verkko

- ensin yhteyden muodostus
- sitten datan siirto yhteyttä pitkin
- yhteyden purku



- 
- 
- 

## 5.1. Verkkokerroksen tärkein tehtävä: reititys

- **(hajautettu) päätöksenteko reitistä**
  - yhteydellinen: alussa
  - yhteydetön: jatkuvasti
- **jatkuvaa muutosta verkossa**
  - rikkoutuvat komponentit, muuttuva topologia
- **ristiriitaisia vaatimuksia reititykselle**
  - optimaalisuus /reiluus (fairness)
- **reitityksen suorituskyky**
  - mean packet delay, network throughput

- 
- 
- 

# Reititysalgoritmi

- **Päättää, mikä reitti valitaan**
  - mihin paketti ohjataan seuraavaksi
- **dynaaminen verkkoympäristö => dynaaminen reititys**
  - jatkuvaan verkon tarkkailuun perustuva
    - Internetin reititys
  - muuttumaton ympäristö => käytetään kerran laskettuja reittejä tai sovittua lähetystapaa
    - tulvitus (flooding)
    - Dijkstran algoritmilla lasketut lyhyimmät reitit

- 
- 
- 

# Tulvitus

- jokainen saapunut paketti lähetetään kaikille muille ulosmenoille
  - => verkko täyttyy pian paketeista
- eri tapoja tulvituksen lopettamiseen
  - käsitellään harjoituksissa
- käyttö
  - tietyissä erityistilanteissa tilanteissa hyödyllinen
    - käsitellään harjoituksissa



- 
- 
- 

# Dijkstran algoritmi

- **'lyhyin' reitti yhdestä solmusta muihin**
  - $A \rightarrow \{\text{muut solmut}\}$
- **kaariin liittyy kustannus**
  - kapasiteetti (bps)
  - viive: hyppyjä, aikaa
  - raha
  - virhetodennäköisyys

- 
- 
- 

## • Algoritmi

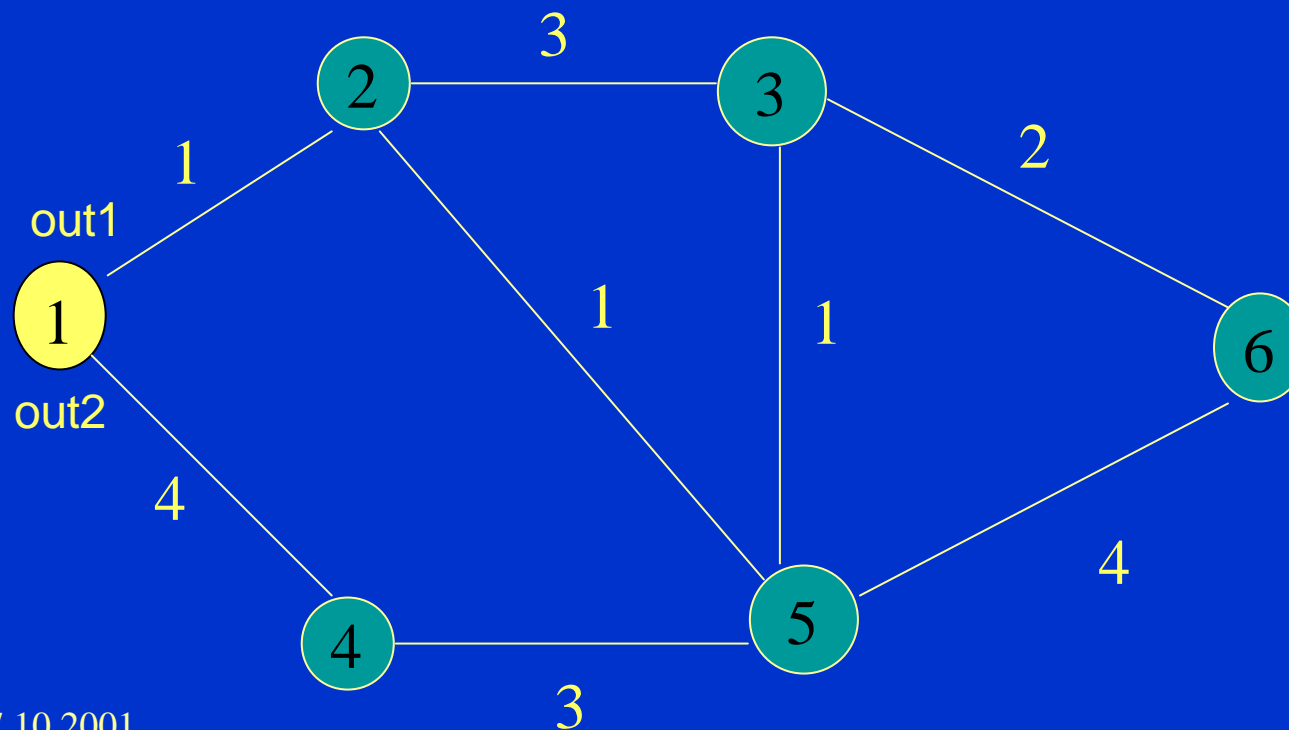
- merkitään  $D(v)$  on tähän asti tutkituista reiteistä lähtösolmusta  $A$  solmuun  $v$  halvin kustannus eli lyhyin pituus
- verkko  $G = (V, E)$ ,  $V$  on solmujen joukko,  $E$  kaarten joukko
- olkoon  $c(ij)$  on kaaren  $(i,j)$  kustannus ( $> 0$ ). Jos kaarta ei ole,  $d(ij)$  on ääretön
  - algoritmissa oletetaan, että kaikki kustannukset ovat ei-negatiivisiä

1.  $N := \{1\}; D(1) := 0; D(j) := d(j,1) \ (j \neq 1);$
2. while  $N \neq V$  do
3. etsi solmu  $w$ , joka ei vielä ole  
joukossa  $N$  ja jonka  $D(w)$  on pienin  
 $N$ :ään kuulumattomista solmuista
4.  $N := N \cup \{w\}$
5. kaikille muille  $P$ :hen kuulumattomille  
solmuille  $v \ D(v) := \min\{D(v), D(w) + c(w,v)\}$
6. end while
7. end

•  
•  
•

# Esimerkki

- Tarkastellaan esimerkkinä verkkoa



1.  $N = \{1\}$ ;  $D(1) := 0$ ;  $D(2) := 1$ ;  
 $D(3) := \text{ääretön}$ ,  $D(4) := 4$ ;  $D(5) := \text{ääretön}$ ,  
 $D(6) := \text{ääretön}$

3. pienin  $D(v)$  on solmulla 2 (=1)

4.  $N = \{1, 2\}$

5.  $D(3) := 1+3=4$ ,  $D(4) = 4$ ,  $D(5) := 1+1=2$ ,  
 $D(6) = \text{ääretön}$

3. pienin  $D(v)$  on nyt solmulla 5 (=2)

4.  $N = \{1, 2, 5\}$

5.  $D(3) := 1 + 2 = 3$ ,  $D(4) := 4$ ,  $D(6) := 4 + 2 = 6$

3. pienin  $D(v)$  solmulla 3 (=3)

4.  $N = \{1, 2, 3, 5\}$

5.  $D(4) := 4$ ,  $D(6) := 2 + 3 = 5$ ;

3. Pienin  $D(v)$  solmulla 4 (=4)

4.  $N = \{1, 2, 3, 4, 5\}$

5.  $D(6) := 5$

4.  $N = \{1, 2, 3, 4, 5, 6\}$

•  
•  
•

# Löydetyt reitit ja kustannukset

- 1-> 2 :1
- 1-> 2->5->3: 3
- 1-> 4: 4
- 1->2->5: 2
- 1->2->5->3->6: 5

Solmu	linkki	kustann.
2	1	1
3	1	3
4	2	4
5	1	2
6	1	5

Solmulle 1

# Reititystaulu

- **Kukin reititin pitää kirjaa reittitiedoista**
  - minne paketti seuraavaksi lähetetään

Kohde	minne lähetetään
Abc	reititin D, ulosmeno 2
...	.....
Xyz	reititin T, ulosmeno 3

- **reitittimien tietojen hankinta ja ylläpito?**
  - erityisen nopeasti muuttuvassa hyvin isossa verkossa



- 
- 
- 

# Reititystietojen keruu

- kukin reititin kerää ‘kustannustietoja’ omasta ympäristöstään
  - esim. viiveet naapurireitittimiin
- ja vaihtaa tietoja muiden reitittimien kanssa
  - tai lähettää tiedot reitittimelle, joka keskitetysti laskee parhaat reitit
- kukin laskee esim. Dijkstran algoritmilla parhaat reitit koko verkosta
  - tai saa tarvitsemansa reititystiedot ne laskeneelta

•  
•  
•

# Etäisyysvektoreireitys (distance vector)

- **Arpanetin alkuperäinen reitys**
  - vieläkin RIP jonkin verran käytössä Intenetissä
- **kullakin reitittimellä reititystaulu**
  - kullekin verkon reitittimelle
    - ulosmenolinja
    - aika/etäisyys kohteeseen
      - hyppyjen lkm
      - arvioitu viive
      - jononpituus
      - jokin mitattavissa oleva

- 
- 
- 

## reititystaulun ylläpito

- **tietojen vaihto naapurireitittimien kanssa**
  - tietyin aikavälein
  - tilan vaihtuessa
- **lasketaan uudet reittitaulut ('etäisyystaulut')**
  - 'kustannus' naapuriin +  
naapurin ilmoittama 'kustannus' kohteeseen
  - kullekin solmulle valitaan pienimmän 'kustannuksen'  
reitti

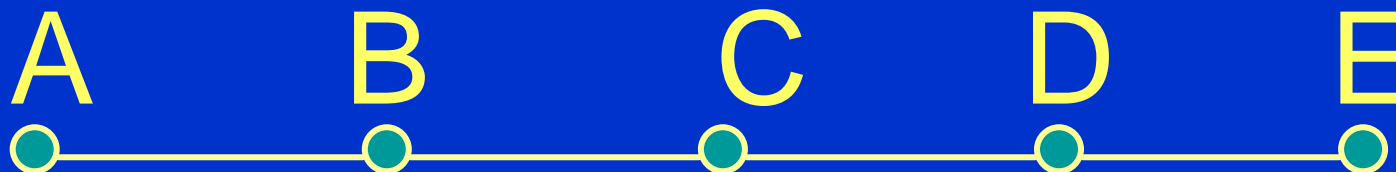
- 
- 
- 

## Ongelma: tietojen muuttumisnopeus

- **tietojen muuttamiseen kuluu aikaa**
- **reagoi melko nopeasti hyviin uutisiin**
  - uusi nopea reitti löytynyt/linkki jälleen pystyssä
  - tieto etenee joka vaihdossa yhden hypyn
- **reagoi hitaasti huonoihin uutisiin**
  - linkki nurin => etäisyys ääretön
  - joka vaihdossa ‘paras arvio’ huononee yhdellä
  - **count - to - infinity** -ongelma

- 
- 
- 

## Hyvät uutiset etenevät nopeasti:

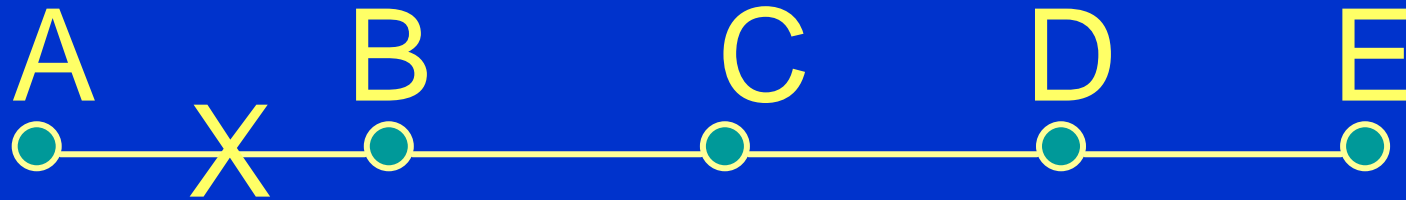


Aluksi yhteys A:han on poikki ja sitten linkki AB toimii taas:

	B	C	D	E
	ääretön	ääretön	ääretön	ääretön
1	ääretön	ääretön	ääretön	ääretön
1	2	ääretön	ääretön	ääretön
1	2	3	ääretön	ääretön
1	2	3	4	21

⋮

## Huonot uutiset etenevät hitaasti:



Toimiva linkki katkeaa välillä AB:

	B	C	D	E
1	1	2	3	4
3	3	2	3	4
3	3	4	3	4
5	5	4	5	4
5	5	6	5	6
7	7	6	7	6
7	7	8	7	8

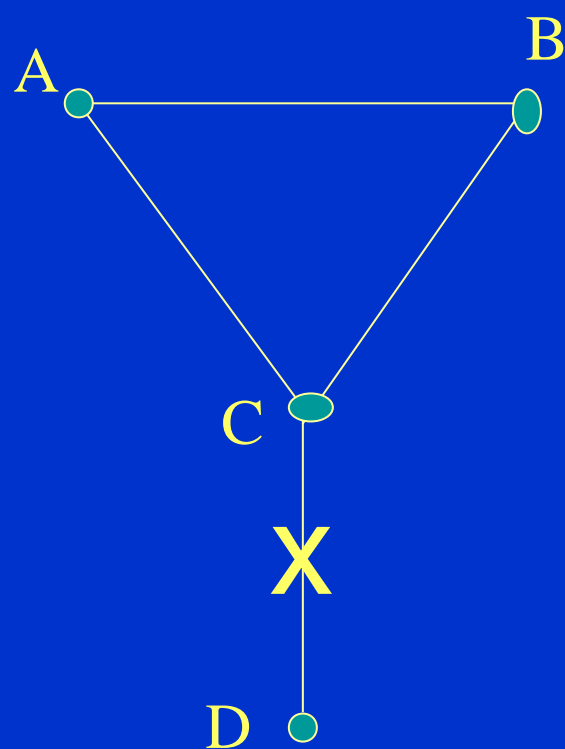
•  
•  
•

## Poisoned reverse (Split horizon)

- **raitkaisu ‘count -to-infinity’-ongelmaan**
  - reititystietoja vaihdettaessa
    - ilmoitetaan etäisyys reitittimeen X äärettömäksi sille naapurille, jonka kautta tämä reitti kulkee
    - muille kerrotaan oikea etäisyys
  - **tieto etenee yhden hypyn joka vaihdolla!**

- 
- 
- 

- **ratkaisu ei toimi aina**



Linkki CD katkeaa,  
A ja B ilmoittavat C:lle  
ettei D:hen pääse

C päättelee, että D:tä ei  
voi saavuttaa

Kuitenkin A kuulee B:ltä,  
että sillä on etäisyys 2  
D:hen => oma etäisyys 3



- 
- 
- 

## Linkkitilareititys (Link State Routing)

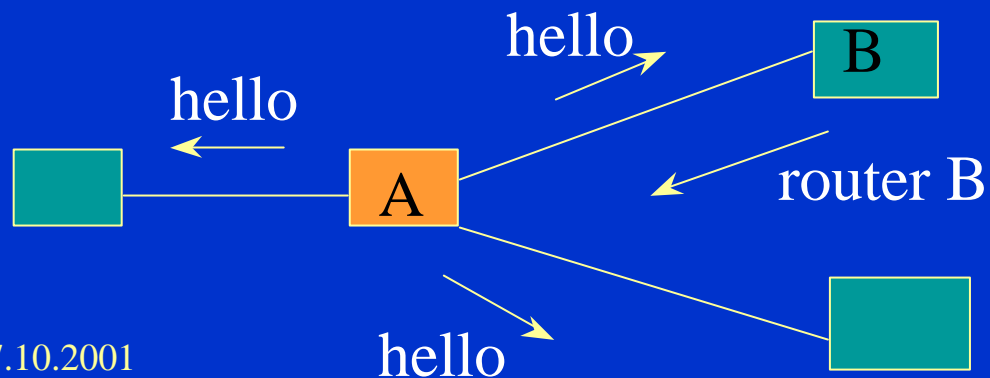
- **reitittimen tehtävät**

- selvitettävä naapurit ja niiden osoitteet
- mitattava etäisyys / kustannus naapureihin
- koottava tietopaketti ko. tiedoista
- lähetettävä tietopaketti kaikille reitittimille
- laskettava lyhin reitti kaikkiin muihin reitittimiin esim. Dijkstran algoritmilla

•  
•  
•

## Naapurien löytäminen

- reititin lähettää jokaiseen kaksipisteeyhteyteen **HELLO**-paketin
- linjan toisessa päässä oleva reititin vastaa ja lähettää nimensä
  - router ID
  - nimien oltava yksikäsitteisiä koko verkossa



- 
- 
- 

## Etäisyyden mittaaminen

- **kaikille naapureille ECHO-paketti**
  - vastaanottajan palautettava paketti välittömästi
- **=> kiertoviive (round-trip-time)**
  - dynaaminen etäisyysmitta
- **pitäisikö ottaa kuormitus huomioon?**
  - kello käynnistetään , kun paketti viedään jonoon
  - kello käynnistetään, kun paketti lähtee
  - kuormitus mukana kuvaa todellista tilannetta
  - jos kuormitus mukana => reititys muuttaa kuormitusta  
=> reititys suosii huonoa reittiä

- 
- 
- 

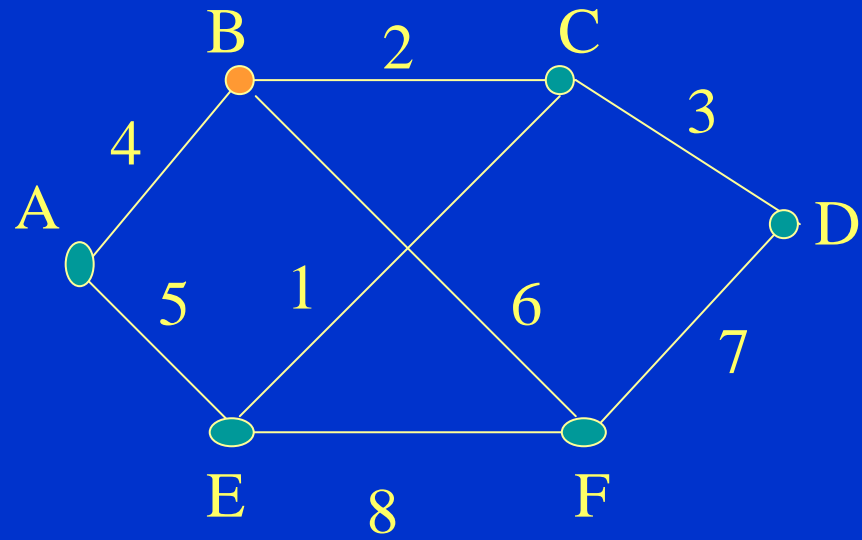
# Tietopakettien kokoaminen

- **muodostus**

- tietyin aikavälein
- kun muutoksia havaittu

- **sisältö**

- reitittimen tunnus
- paketin järjestysnumero
- paketin ikä
- ‘etäisyydet’ kuhunkin reitittimen naapuriin
  - Erilaisia etäisyysmittoja => eri reittejä eri liikenteelle



B	
seq	
age	
A	4
C	2
F	6

- 
- 
- 

# Tietopakettien jakelu

- **käytetään tulvitusta (n. 10 minuutin välein)**
  - **pidetään kirjaa jo nähdyistä paketeista**
    - reititin A, paketti 145
  - => paketti lähetetään korkeintaan kerran
  - paketissa elinaikalaskuri (age, time-to-live)
    - väärät ja vanhentuneet tiedot katoavat aikanaan, vaikka reititin itse olisikin vikaantunut
- **tietopaketit kuitataan**
  - linjavirheiden takia
- **autentikointi paketteja vaihdettaessa**

- 
- 
- 

## Miksi elinaikalaskuri on tarpeen?

- **virheellinen järjestysnumero**

- kaatunut reititin aloittaa väärästä numerosta

- edennyt jo pakettiin 204 ja aloittaa uudestaan paketista 0 => kaikki seuraavat paketit hylätään duplikaatteina pakettiin 205 saakka

- virhe tietopaketin seq-kentässä

- 4 muuttuu virheellisesti 65540:ksi => seuraavat paketit hylätään pakettiin 65541 saakka

- 
- 
- 

## elinaikalaskuri (TTL-laskuri)

- **laskuri vähenee ajan kuluessa**
  - vähenee yhdellä sekunnin välein
- **paketti tuhotaan, kun laskuri = 0**
  - vanhentunut (virheellinen) tieto poistetaan
  - pitkäikäinen elinaika  $\gg$  päivitysten väli
    - tuhoaan vain jos reititin kaatunut
    - usea (6) paketti on jäänyt saapumatta reitittimeltä
- **käytössä myös tulvituksessa**
  - kukin reititin vähentää yhdellä



- 
- 
- 

## Lisäparannuksia

- **paketteja ei lähetetä välittömästi eteenpäin**
  - ne jätetään odottamaan
  - jos samalta reitittimeltä tulee muita paketteja, niistä valitaan vain yksi, tuorein edelleenlähetettäväksi

- 
- 
- 

## Reittitaulun laskeminen

- **kukin reititin laskee omat reittitaulunsa**
- **kaikki tarvittava tieto on saatu tietopakettien avulla**
  - kukin linkki molempiin suuntiin
- **laskeminen Dijkstran algoritmilla**
  - lyhyin reitti kuhunkin muuhun reitittimeen
  - isoissa verkoissa voi olla muisti- ja laskenta-aikaongelmia

- 
- 
- 

## ongelmia

- **väärin toimiva reititin**

- kertoo väriä tietoja
- ei välitä tietopaketteja
- väärentää tietopaketteja
- laskee reitit väärin

- **isossa verkossa aina joku toimii väärin**

– tavoitteena rajata ongelmat pienelle alueelle

- 
- 
- 

# Käyttö

- **paljon käytetty nykyisissä verkoissa**
  - Internetin OSPF-protokolla
  - ISO:n IS-IS -protokolla

•  
•  
•

# Hierarkkinen reititys

- **reitityksen skaalautuvuus**
  - isossa verkossa runsaasti reitittämiä (Internet: miljoonia)
    - reititystaulut suuria
    - reittien laskeminen raskasta
    - tietopaketit kuluttavat linjakapasiteettia
- **hallinta-autonomia => autonominen järjestelmä AS**
  - organisaatio päättää omista asioistaan
    - myös reitityksestä
      - oma sisäinen reititystapa

•  
•  
•

# Reitityshierarkia

- **Ylimmällä tasolla AS**
  - sama reititys AS:n sisällä
    - tehokkuus tärkeää
  - reititys AS:ien välillä
    - ‘poliittinen asia’
- **AS:n sisällä alueita**
  - jaetaan reitittimet ryhmiin (alueet, regions)
  - kukin reititin tuntee kaikki alueensa sisällä
  - tietää mikä reititin hoitaa liikenteen muihin alueisiin

- 
- 
- 

## Hierarkkisen reitityksen ongelmat

- **reitin pituus kasvaa**
  - aina ei voida käyttää optimaalista reittiä
  - yleensä siedettävä
- **hierarkiatasojen määrä**
  - suorituskyky
  - hallinto

- 
- 
- 

## 4.2. Reititin (Router)

