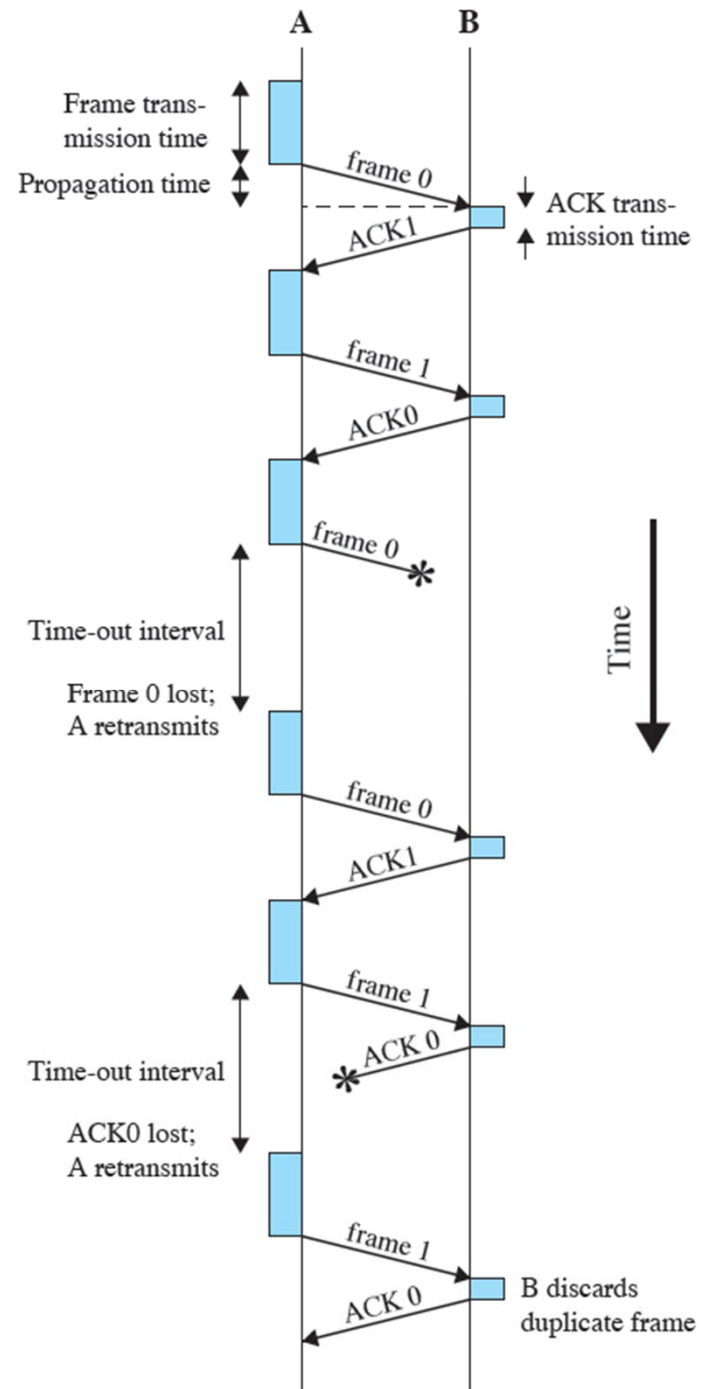


ETSF05:
Network models
Användarmodeller/Paradigmer
Länkprotokoll: Flödeskontroll vs
felhantering
Routingalgoritmer

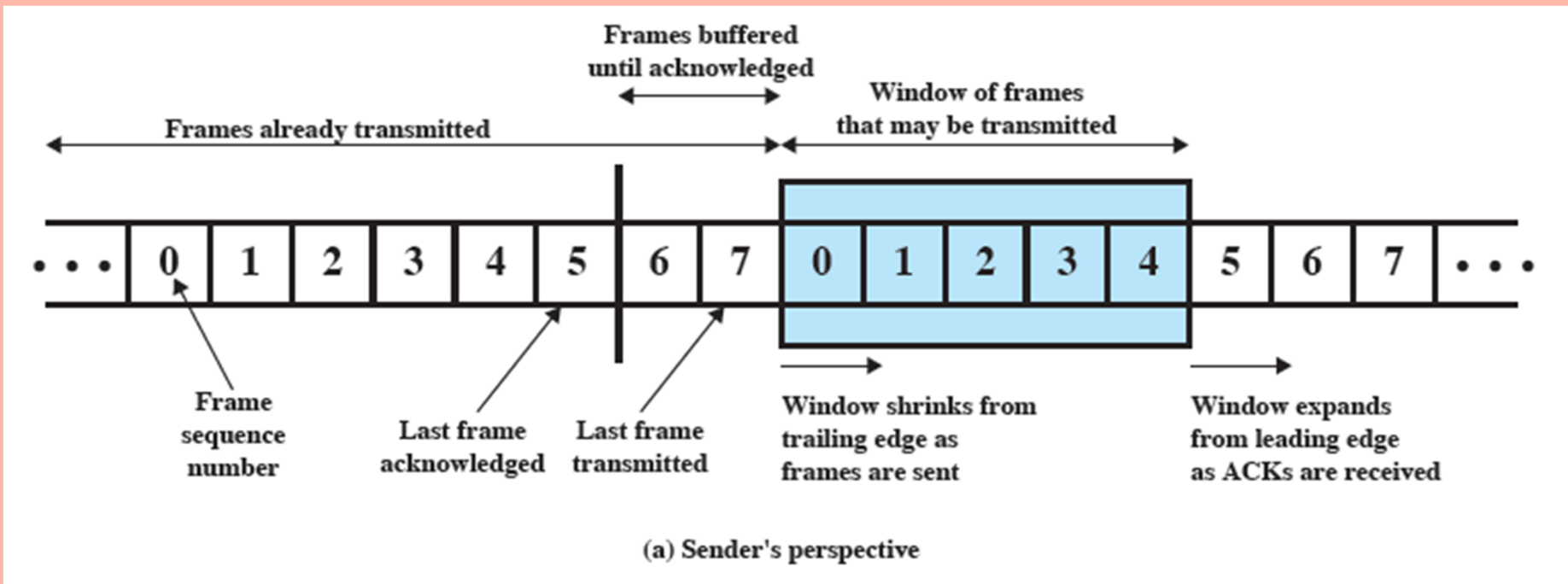
Jens A Andersson



Stop-and-wait ARQ flow diagram



Sliding window (sender)



Go-Back-N ARQ (the Stalling Way)

Most commonly used error control ACK (this or next)

Based on sliding-window

Use window size to control number of outstanding frames

While no errors occur, the destination will acknowledge incoming frames as usual

- ◆ **RR=receive ready, or piggybacked acknowledgment**

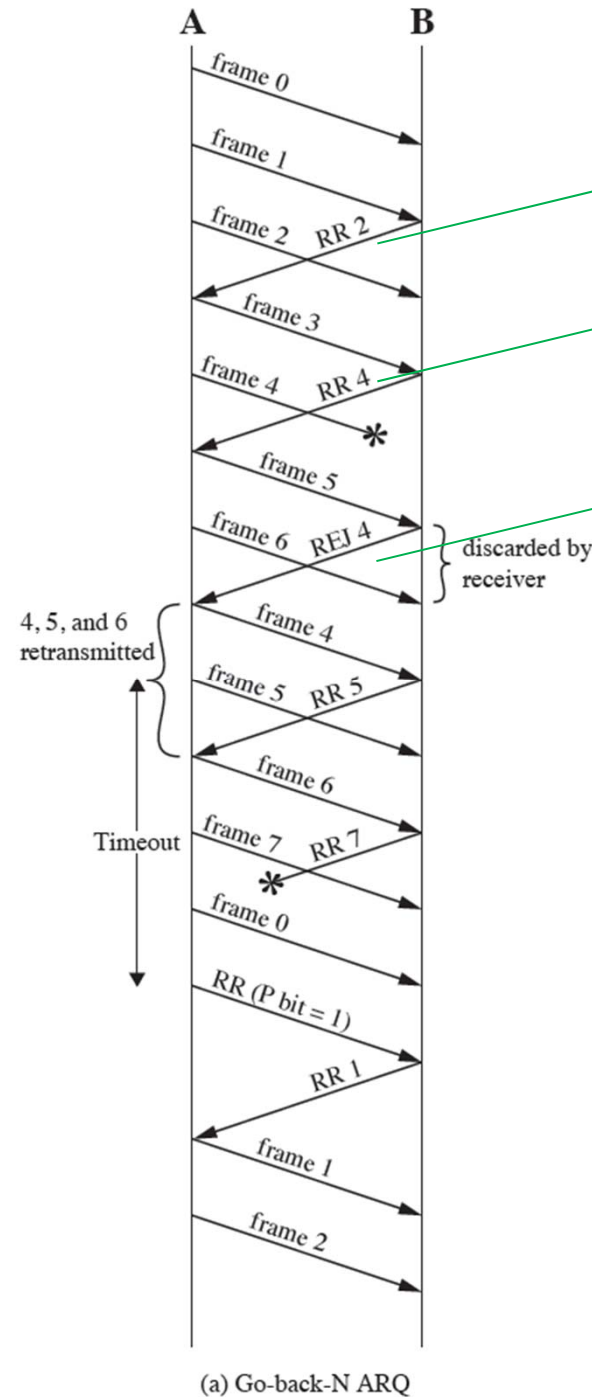
If the destination station detects an error in a frame, it may send a negative acknowledgment

ACK (previous or this)

- ◆ **REJ=reject**

- ◆ Destination will discard that frame and all future frames until the frame in error is received correctly
- ◆ Transmitter must go back and retransmit that frame and all subsequent frames

Go-Back-N



Vårt sätt

ACK2

ACK4

ACK4

(a) Go-back-N ARQ

Selective-Reject (ARQ) (Stalling)

Also called selective retransmission

Only rejected frames are retransmitted

Subsequent frames are accepted by the receiver and buffered

Minimizes retransmission

Receiver must maintain large enough buffer

More complex logic in transmitter

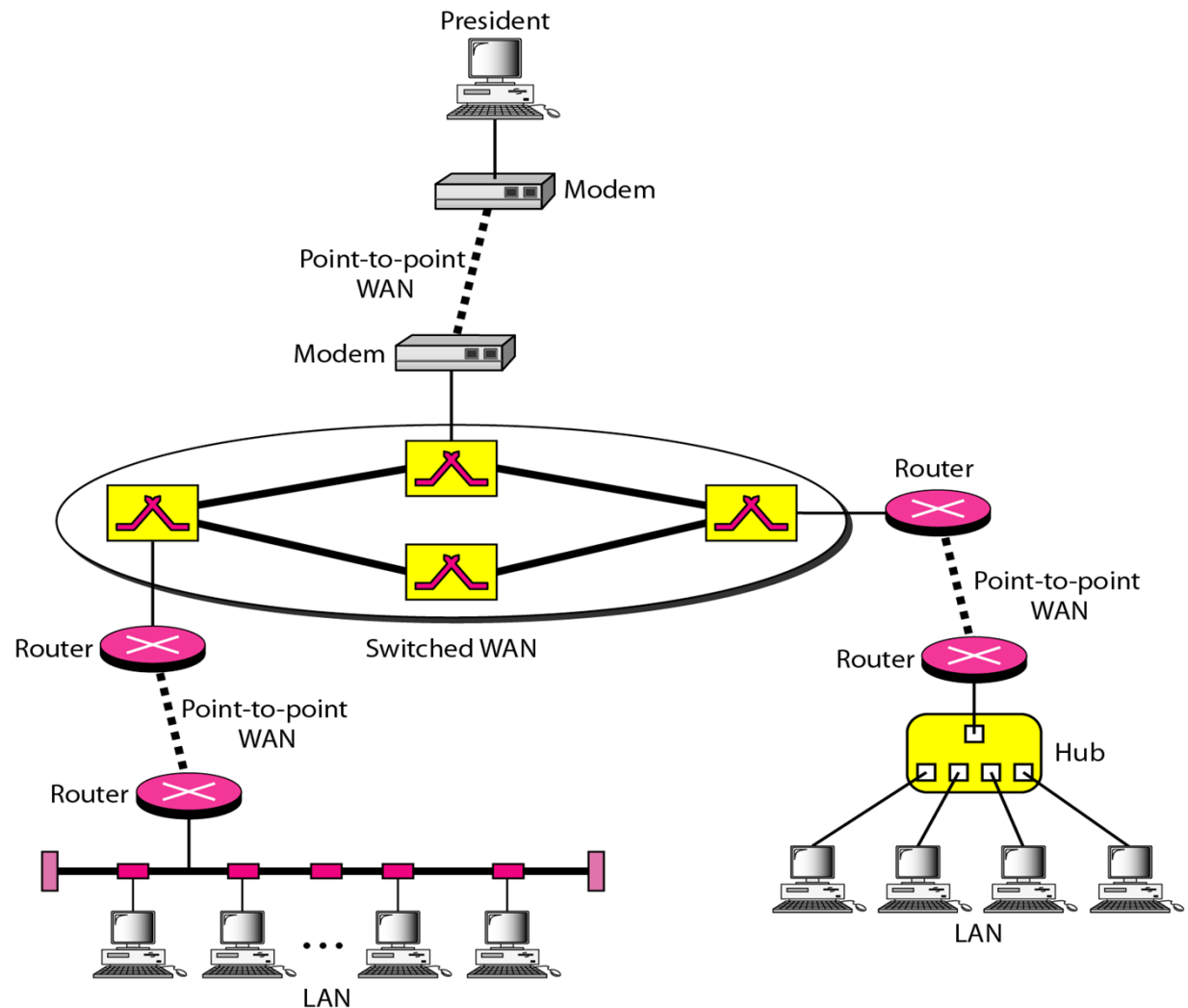
- ◆ Less widely used

Useful for satellite links with long propagation delays

Network engineering

High performance

- ◆ Reliability
- ◆ Throughput
- ◆ Speed
- ◆ Security



Network models

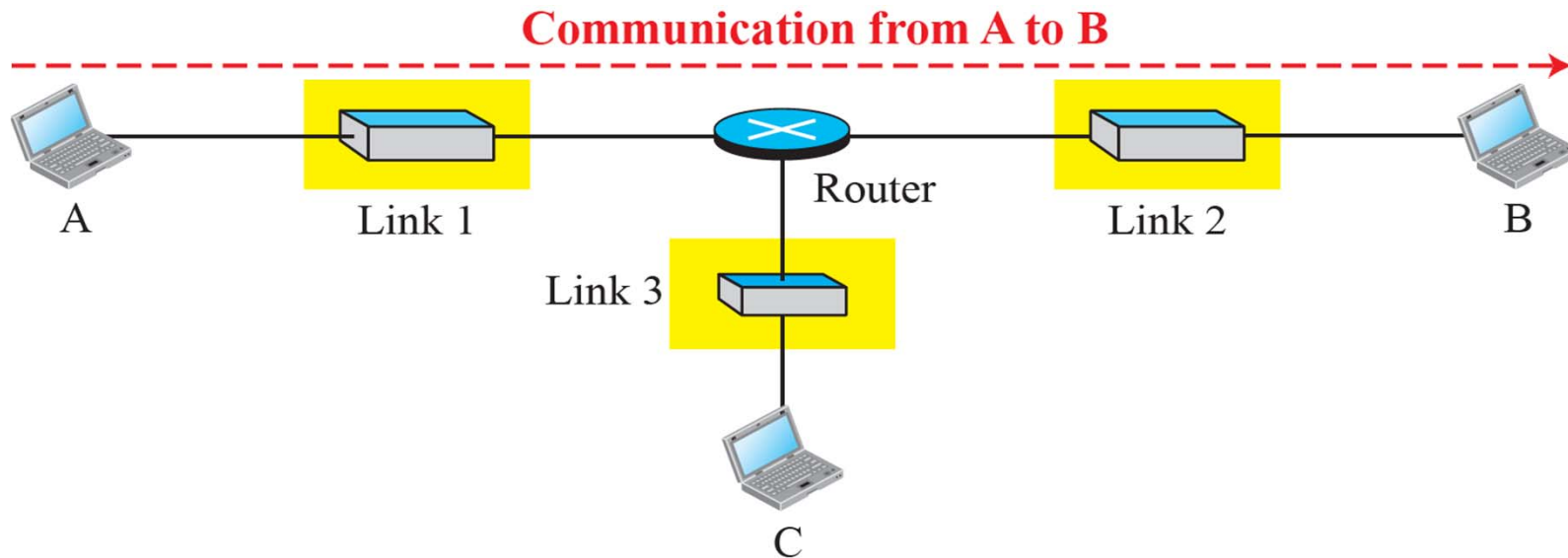
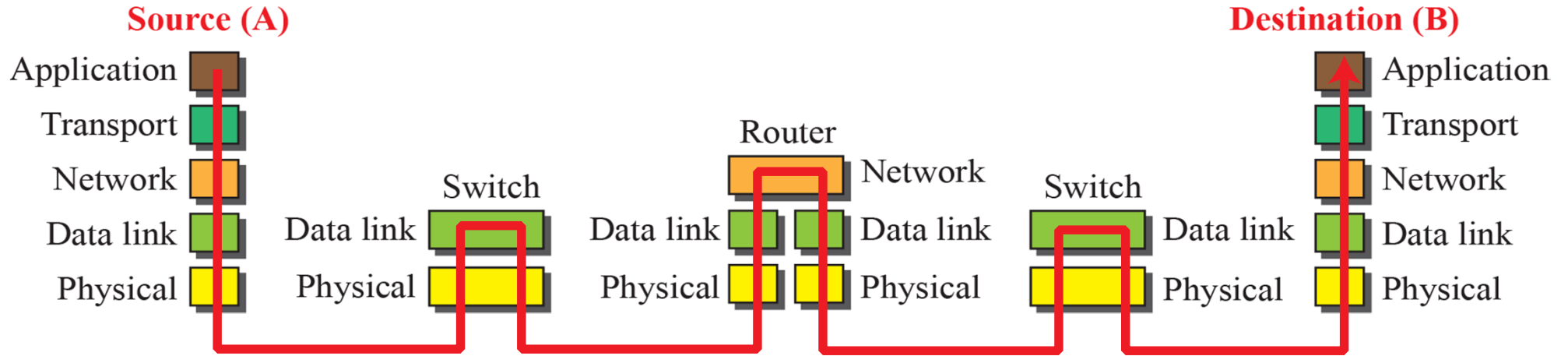
Too complicated

- ◆ Divide and conquer

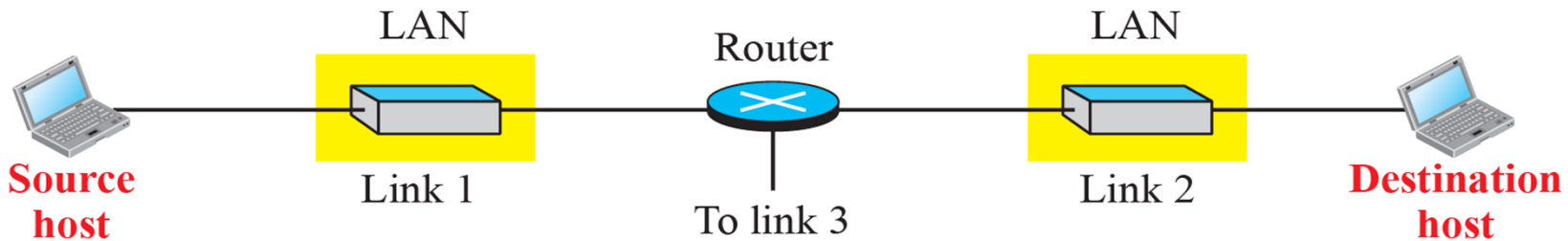
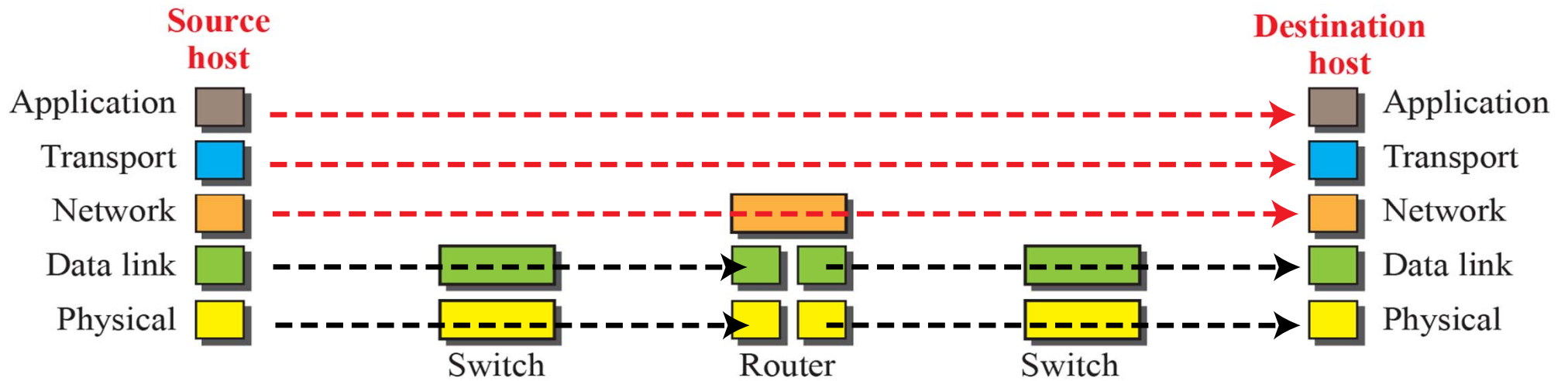
Layered architecture

- ◆ Hierarchy
- ◆ Specialisation
- ◆ Simplification

Kommunikation på Internet

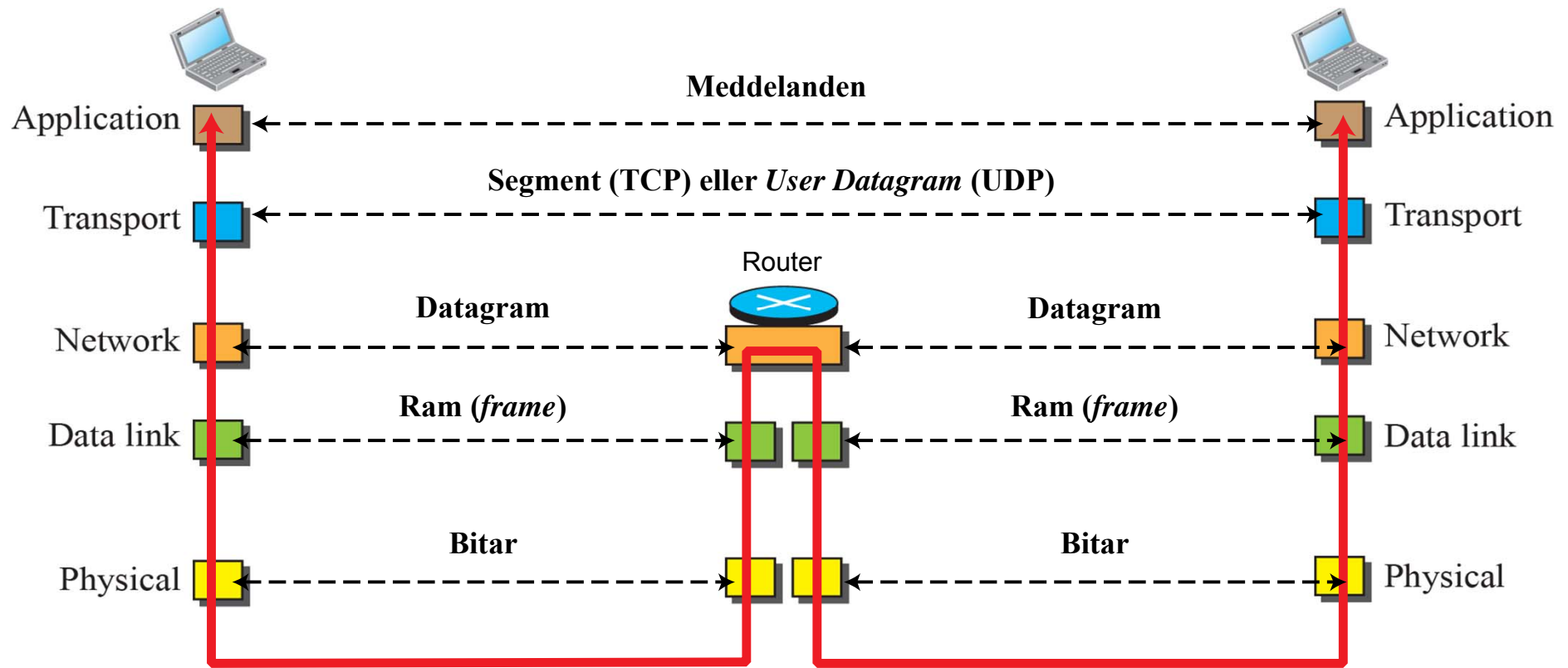


Logiska förbindelser enligt TCP/IP



---> end to end

Identiska objekt enligt TCP/IP

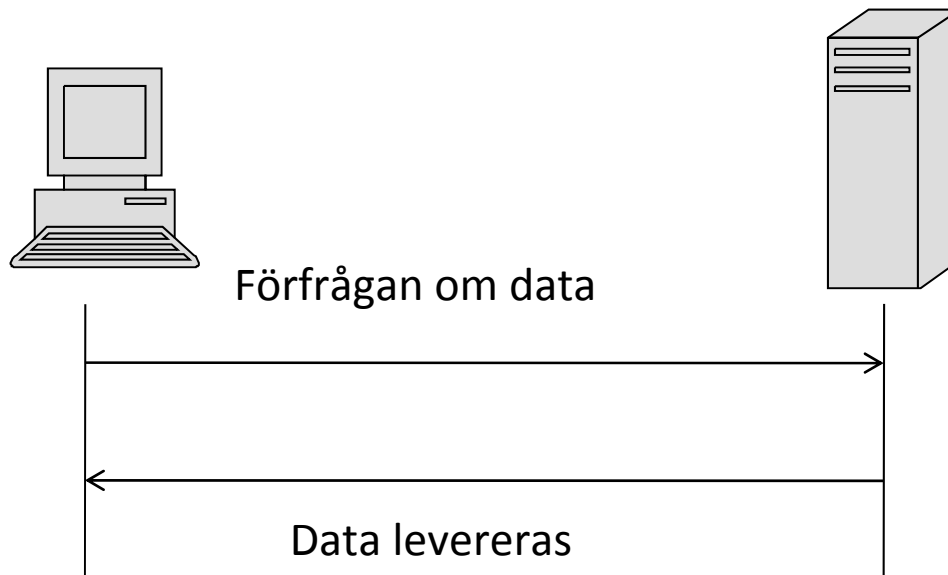


Switchar/Repeterare påverkar inte objekt!

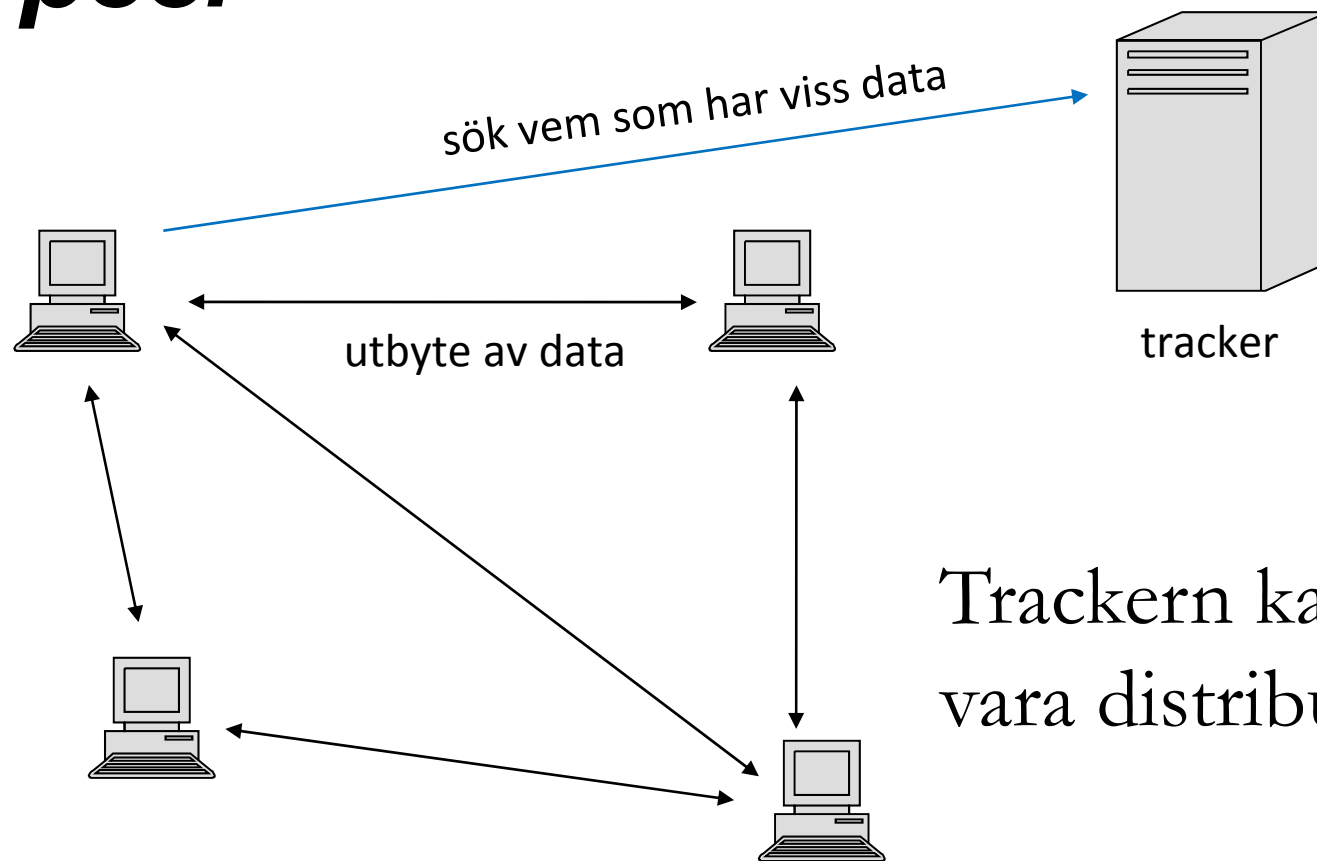
Användarmodeller

- Klient-server
 - *Client-Server Paradigm*
- ”Användare-användare”
 - *Peer-to-peer (P2P) Paradigm*

Client-Server paradigm



Peer-to-peer



Trackern kan
vara distribuerad

Kontrollprotokoll på länk-nivån

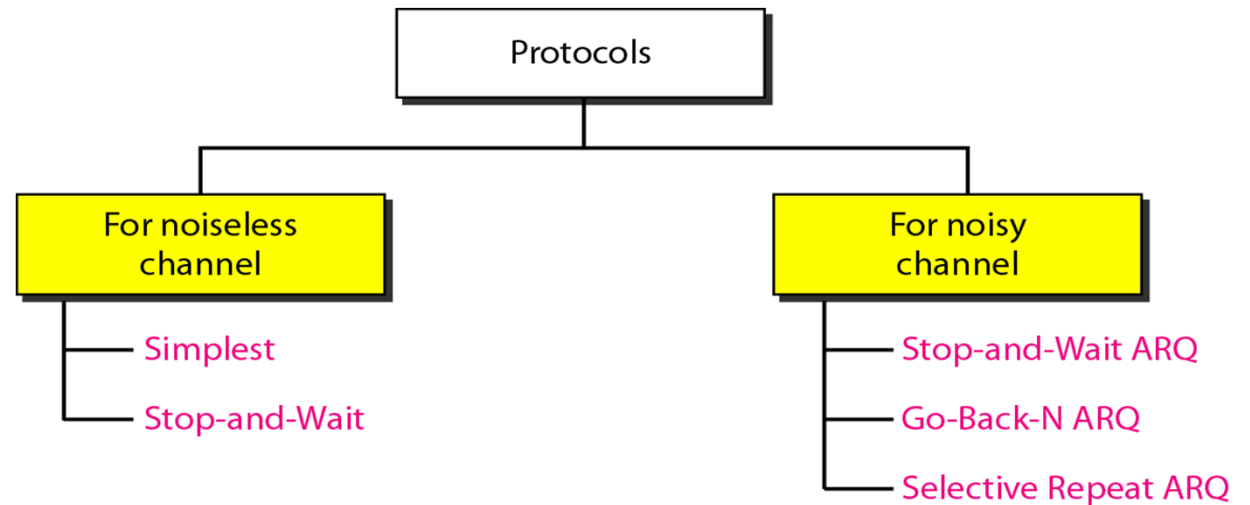
Framing

Flödeskontroll

- ◆ Sänd data
- ◆ Vänta på ACK

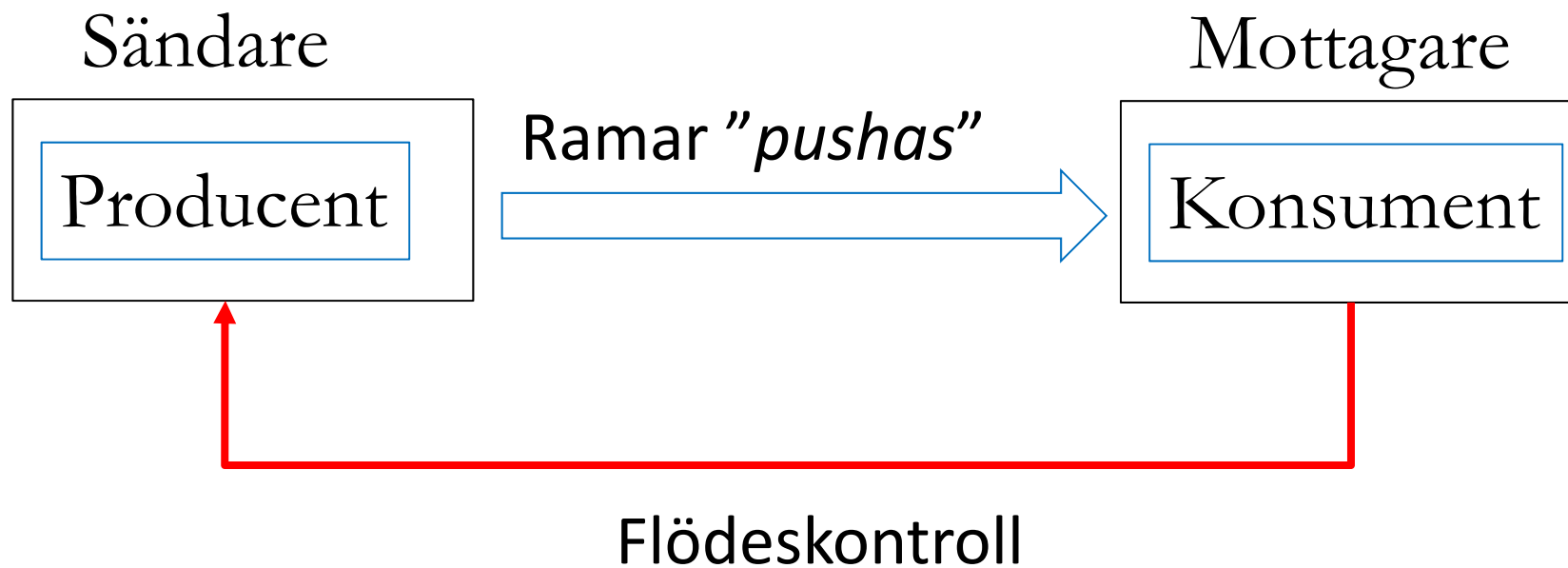
Felkontroll

- ◆ Upptäcka fel
- ◆ Omsändning



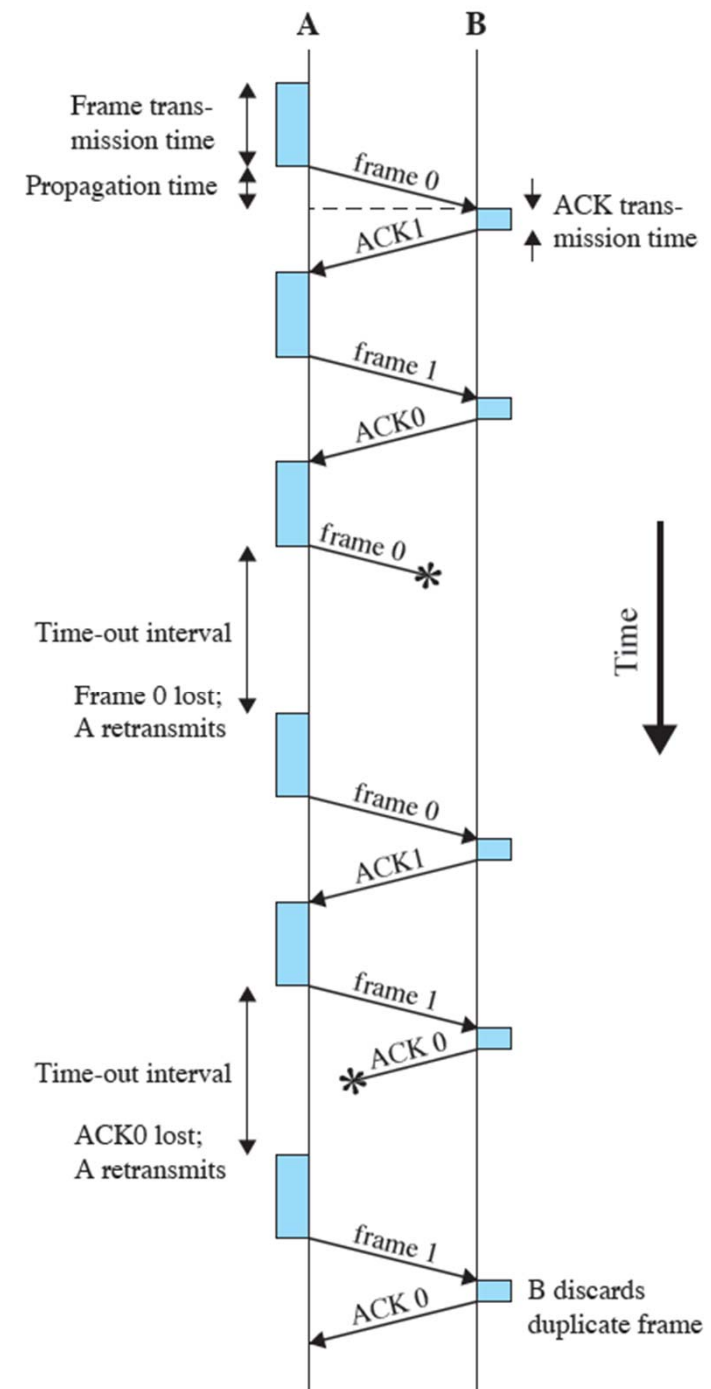
Inte alltid att flödeskontroll,
felhantering finns i
länklagret!

Flödeskontroll



Stop-and-wait ARQ flödesdiagram

- Ineffektivt
- Mycket väntan i onödan
 - Speciellt vid långa länkar



Go-Back-N?

- Viktigt inför nästa del och TCP (ETSF10)
- Föreläst i repetitionsföreläsningen
- Kommer på övning

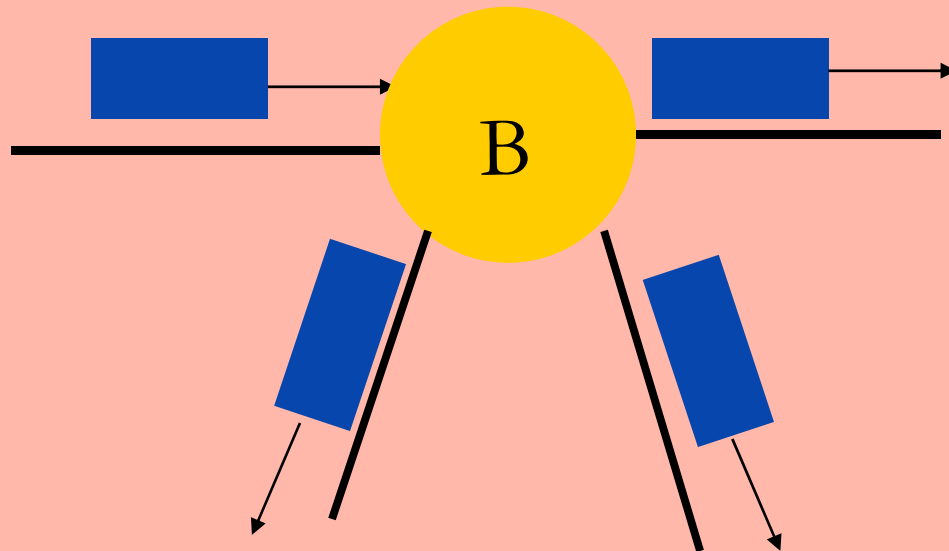
Routing

- Konsten att bygga least-cost trees
 - Från sändare till mottagare
 - Från varje nod till varje annan nod
- Tre principer
 - Distance Vector
 - Link State
 - Path Vector
 - Policy-based routing

Flooding

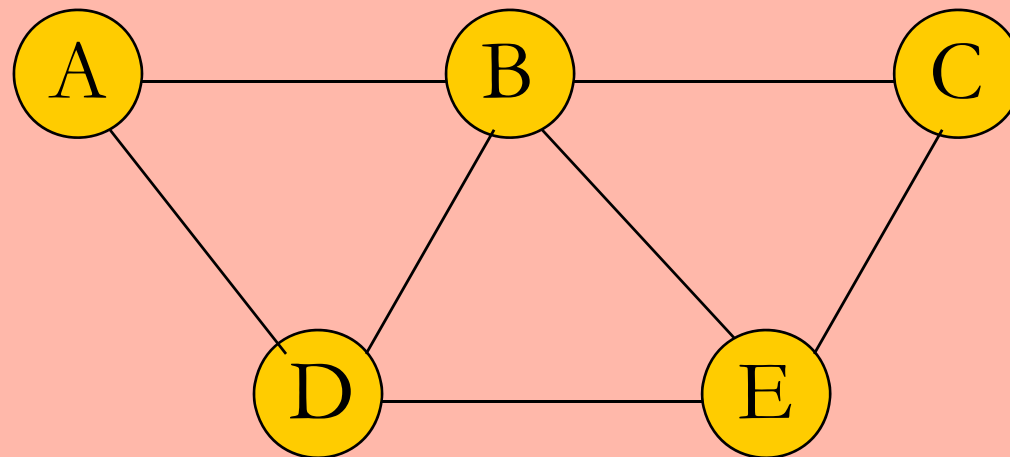
I Flooding skickas ett inkommande paket ut på samtliga länkar.

En hop-count används för att inte skapa loopar.



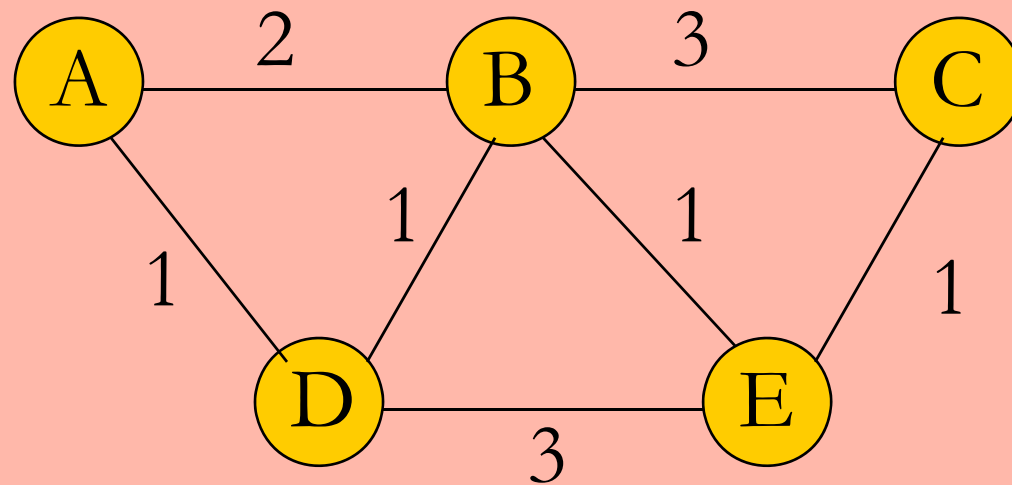
Nätgraf

I vägvalsalgoritmerna används en nätgraf som består av noder och länkar.



Nätgraf forts.

Varje länk i grafen har en kostnad som anger hur ”dyrt” det är att skicka ett paket över länken.



Länkkostnad

Länkkostnaden kan bero på flera saker:

⌘ Kapacitet

⌘ Belastning

⌘ Sträcka

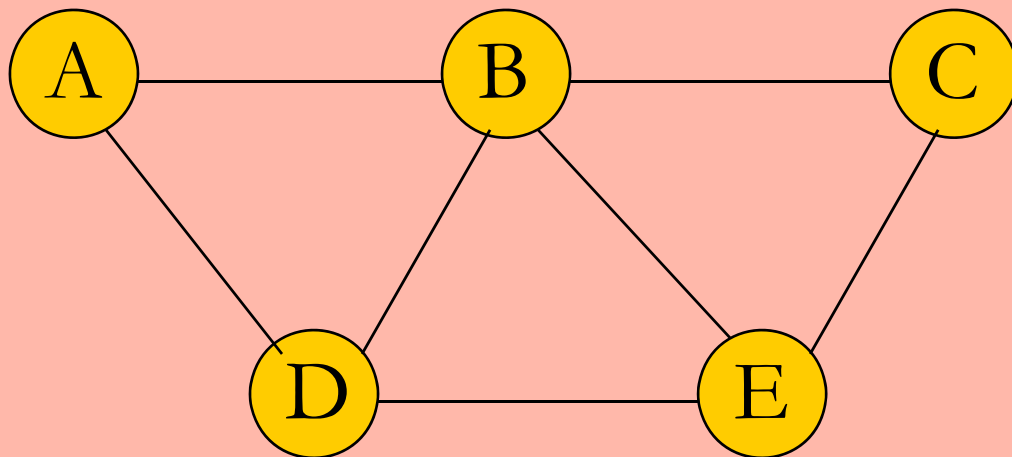
⌘ Utbredningsmedium

⌘ Osv...

Least-hop path

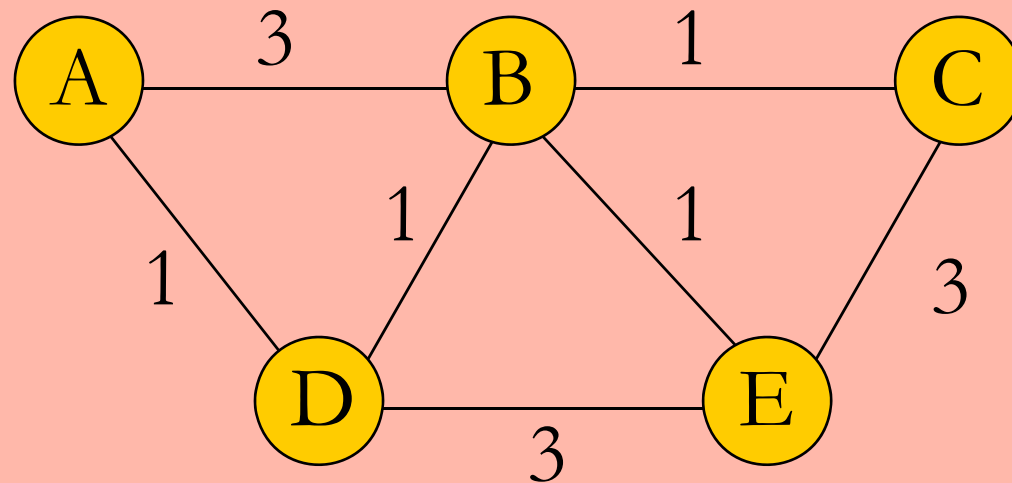
Least-hop path fungerar bäst om alla länkar har samma kostnad.

Den väg som innehåller minst antal steg är bäst.

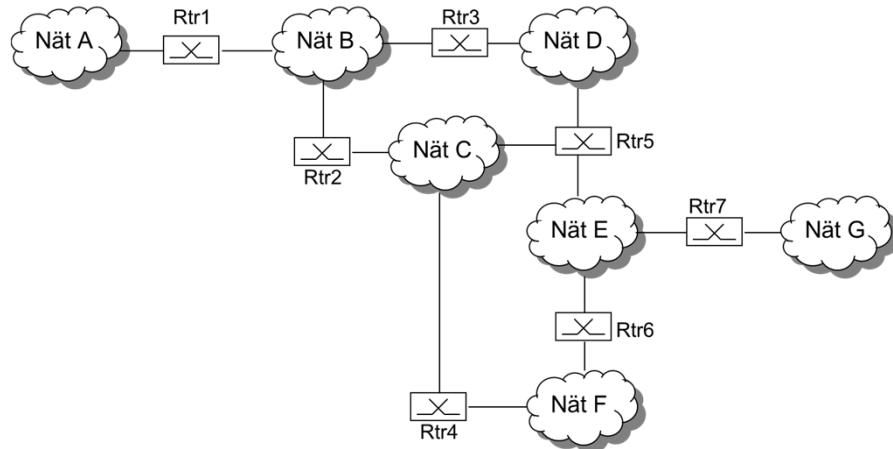


Least-cost path

I Least-cost path väljs de vägar ut som kostar minst.

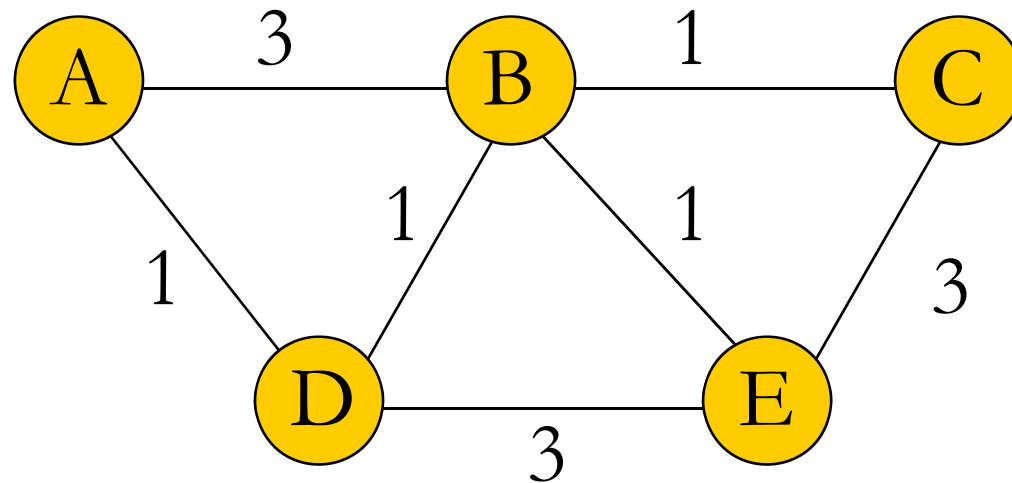


Ett Internet och dess grafiska representation



- Grafer består av bågar och noder
 - Bågar samman binder noder
 - Noder är ”destinationer”
- I routing finns två olika typer av noder:
 - Nät dvs destinationer
 - Routrar som sammanbinder nät
- Grafiska representationen kan bli något konstig ...

Nätgraf (exempel)



Distance vector routing

Alla kända bästa vägar **delas med grannar**

- ◆ Periodiskt
- ◆ (Vid varje förändring)

Routing-tabeller **uppdateras** med

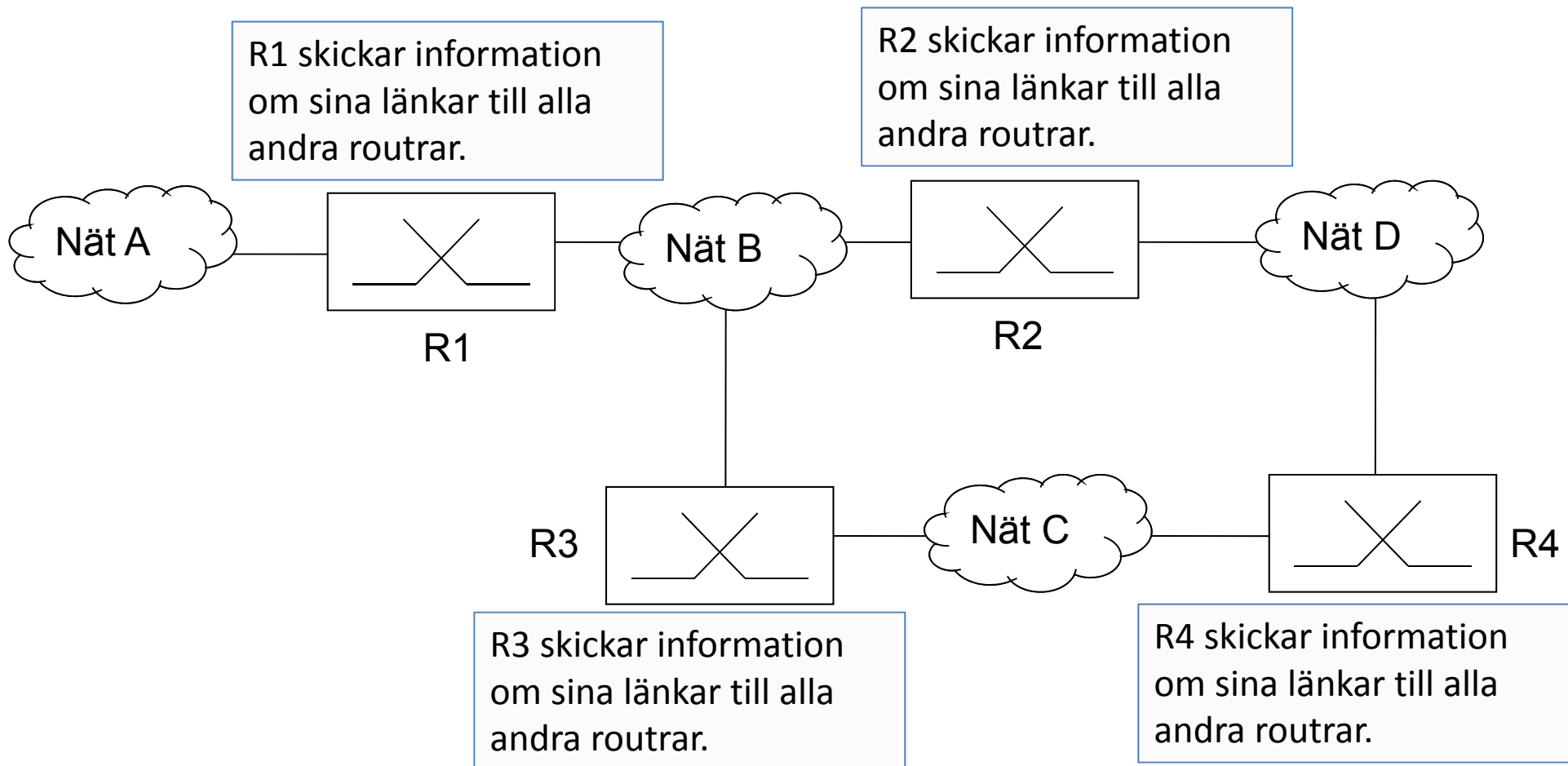
- ◆ Nya poster
- ◆ Ändring av kostnad

”Global kunskap sprids lokalt”

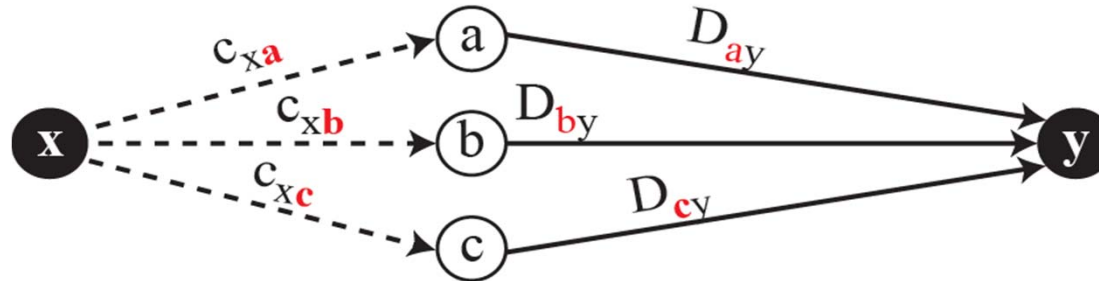
Kostnad

- Antal hopp (Bellman-Ford)
 - Kihl, Andersson kap 8
 - Stallings kap 19.4
 - RIP
- Uppskattad fördröjning
 - *Distributed Adaptive Routing* (Stallings kap 19.2)

Principen för *Distance Vector*

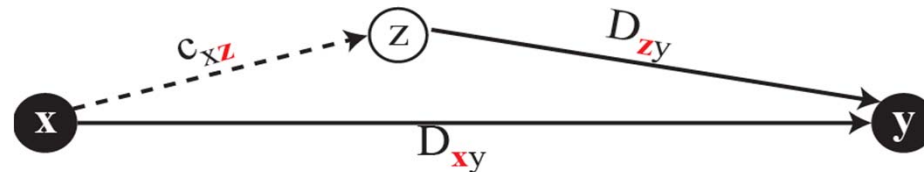


Bellman-Fords algoritm grafiskt



a. General case with three intermediate nodes

$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}) \dots\}$$



b. Updating a path with a new route

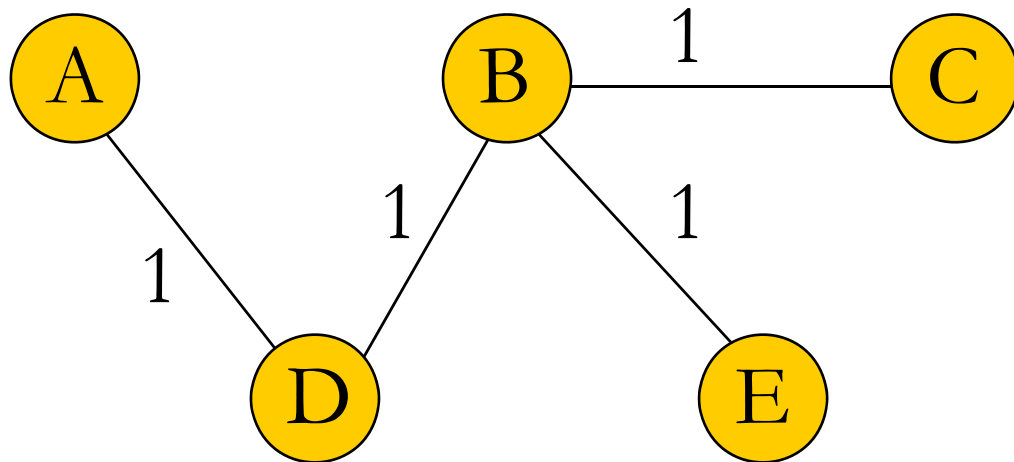
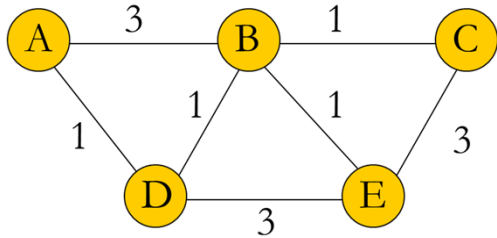
$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

Jmf Stallings kap 19.2

Bellman-Ford i pseudokod

```
if (advertised destination not in table)
{
  add new entry // rule #1
}
else if (adv. next hop = next hop in table)
{
  update cost // rule #2
}
else if (adv. cost < cost in table)
{
  replace old entry // rule #3
}
```

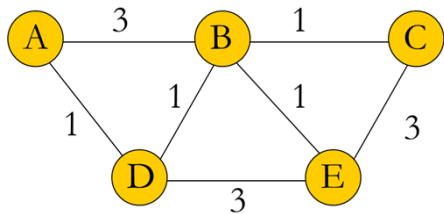

Nätgraf som träd



Distansvektor för A

Nod	Dist
A	0
B	2
C	3
D	1
E	3

Uppdateringar



$$A[] = \min(A[], 3 + B[])$$

A, uppdaterad

Nod	Dist
A	0
B	3
C	4
D	1
E	4

A, ursprunglig

Nod	Dist
A	0
B	3
D	1

B

Nod	Dist
A	3
B	0
C	1
D	1
E	1

Distance Vector, funderingar

- Periodiska uppdateringar!?
- Problem med länkar och noder som försvinner.
- Hur hitta grannar?
- Hur upptäcka att en granne försvinner?

Mer om distance vector

I ETSF10:

- Count to infinity
- Two, three node instability
- Split Horizon
- Poison Reverse
- Routingprotokoll RIP

Link state routing, princip

Lokal topologi-information flödas (*flooding*) globalt

- ◆ Vid varje förändring
- ◆ Periodiskt (i praktiken sällan)

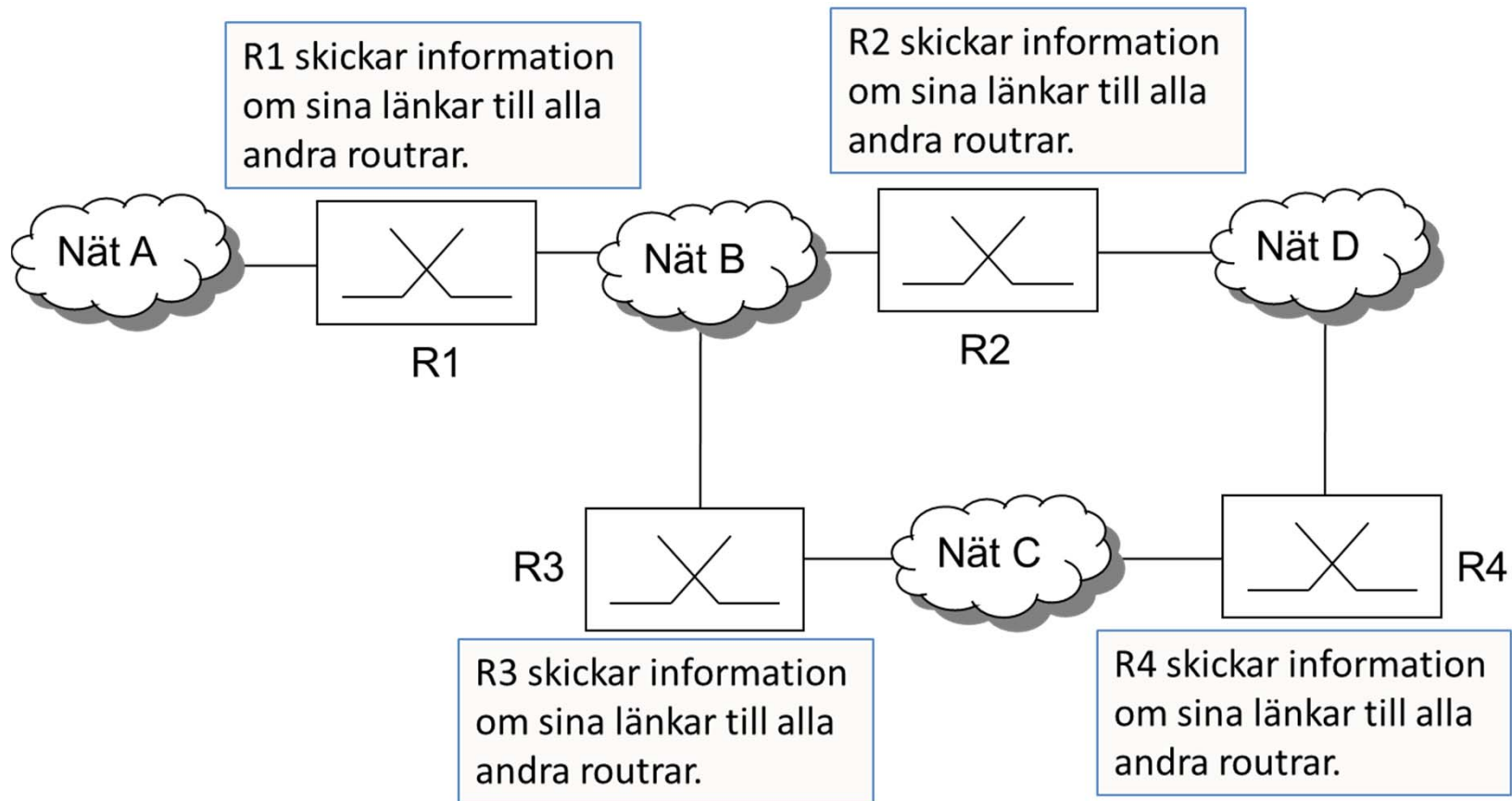
Varje nod bygger egen databas

Routing-tabellen uppdateras med

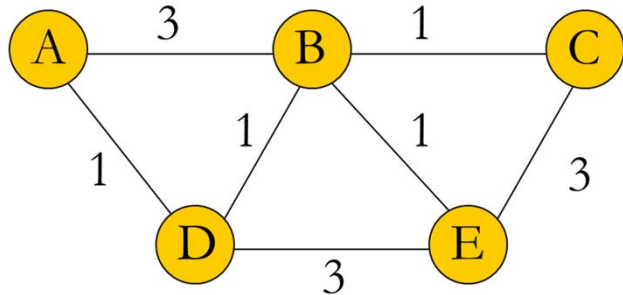
- ◆ Nya poster
- ◆ Förändring av kostnad

”Lokal kunskap sprids globalt”

Link state routing, princip

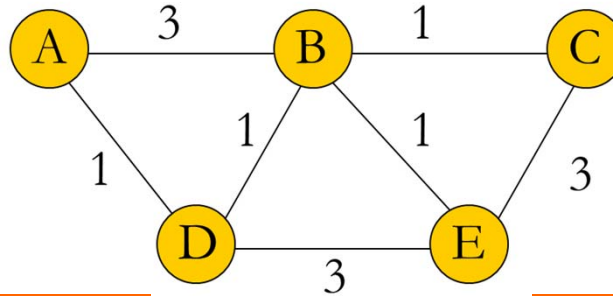


Exempel på *link state* databas



	A	B	C	D	E
A	0	3	∞	1	∞
B	3	0	1	1	1
C	∞	1	0	∞	3
D	1	1	∞	0	3
E	∞	1	3	3	0

Link State Advertisements



Nod	Kostn
1	1
B	1
E	3

Nod	Kostn
B	1
C	3
D	3

Uppdateras vid förändring!

Shortest Path First

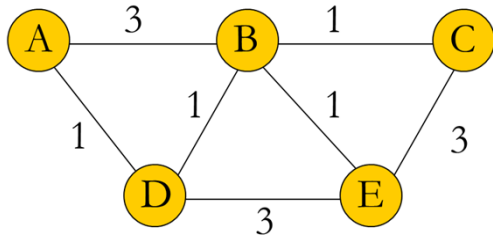
Tree generation algorithm (Dijkstra)

```
put yourself to tentative list
while tentative list not empty
{
    pick node which can be reached
        with least cumulative cost
    add it to your tree*
    put its neighbours to tentative list**
        with cumulative costs to reach them
}
```

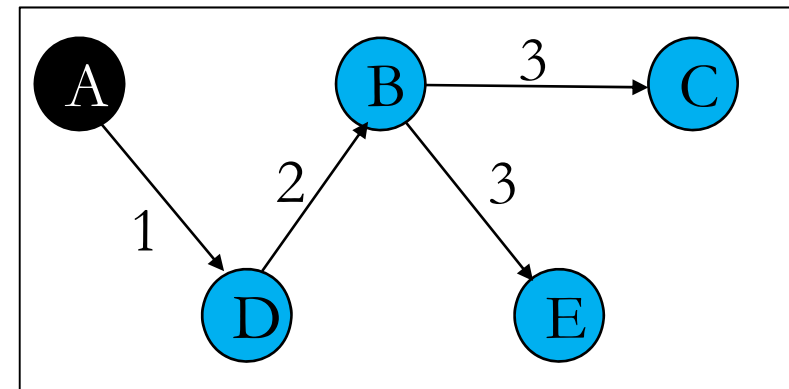
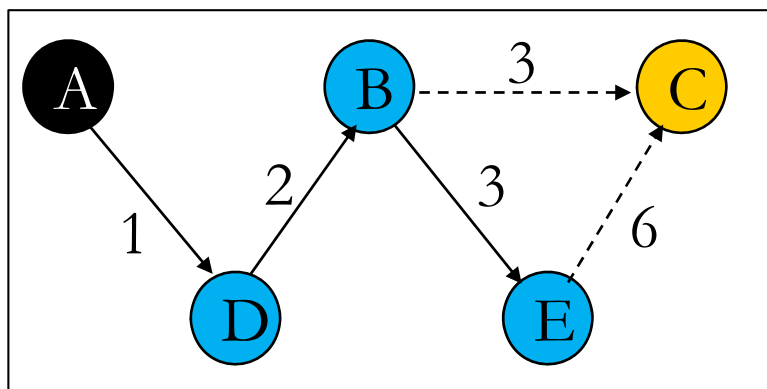
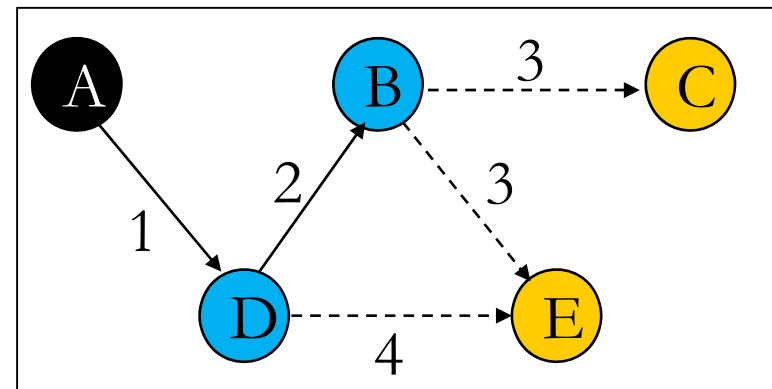
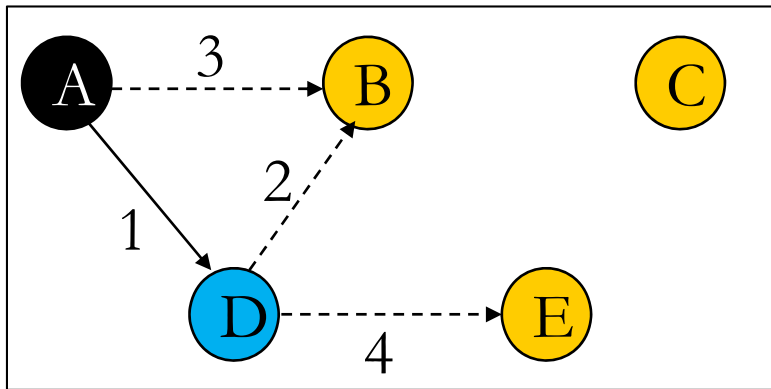
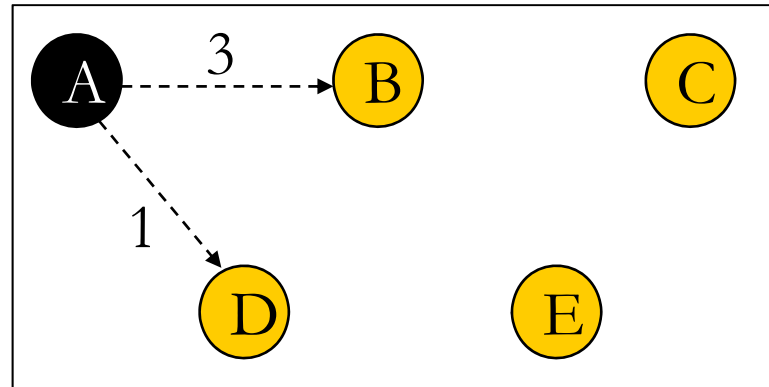
* (a. k. a. permanent list)

** (if not already there with lower cost)

SPF exempel



- Rotnod
- Permanent nod
- Preliminär nod
- > Potentiell väg
- > Permanent väg



Link State, funderingar

- Periodiska uppdateringar!?
- Problem med länkar och noder som försvinner.
- Hur hitta grannar?
- Hur upptäcka att en granne försvinner?

Mer om link state

I ETSF10:

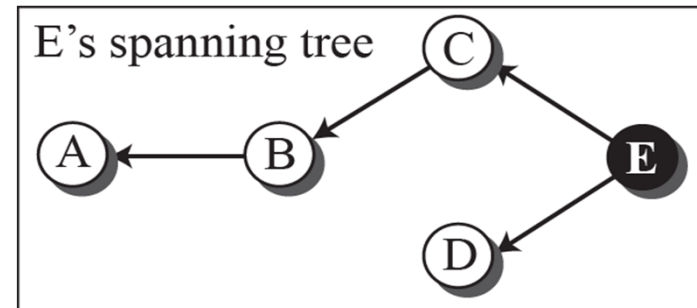
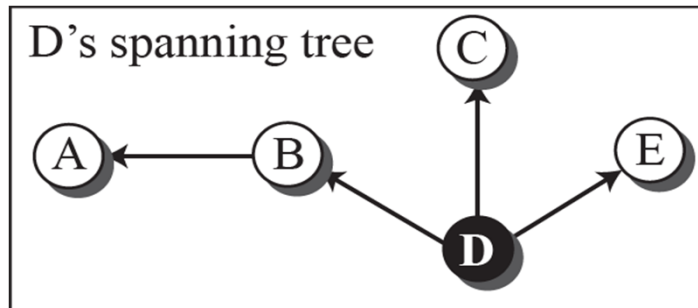
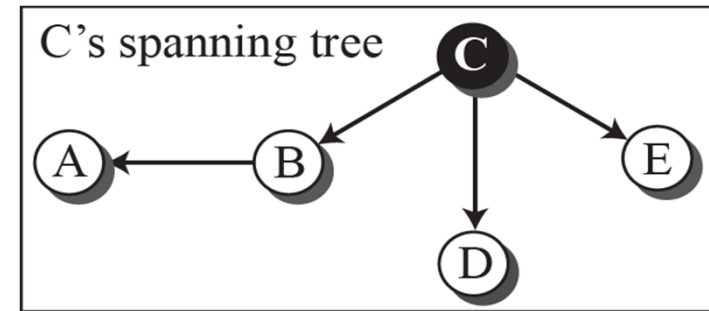
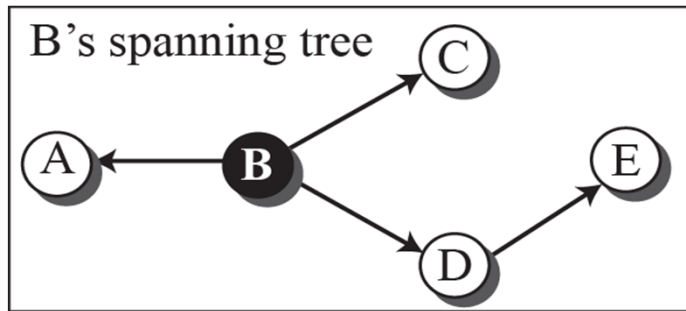
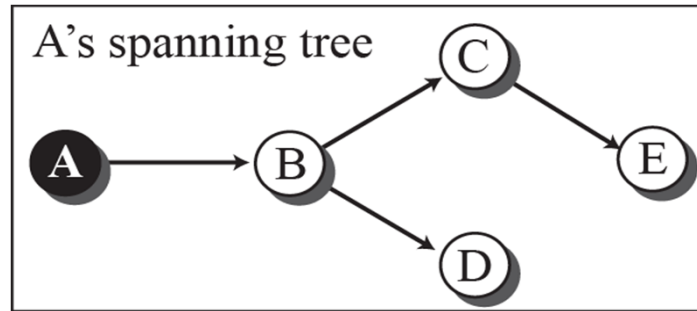
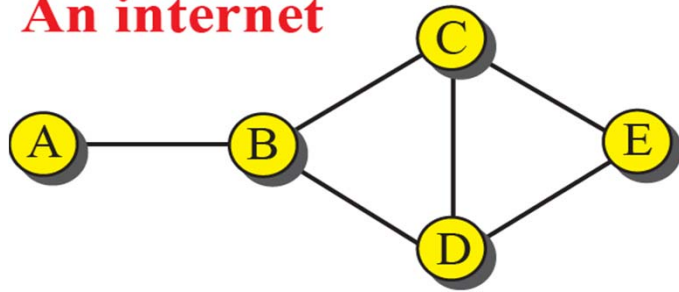
- Routingprotokoll OSPF
- Konceptet *Areor* eller hur man minskar flooding

Path Vector Routing

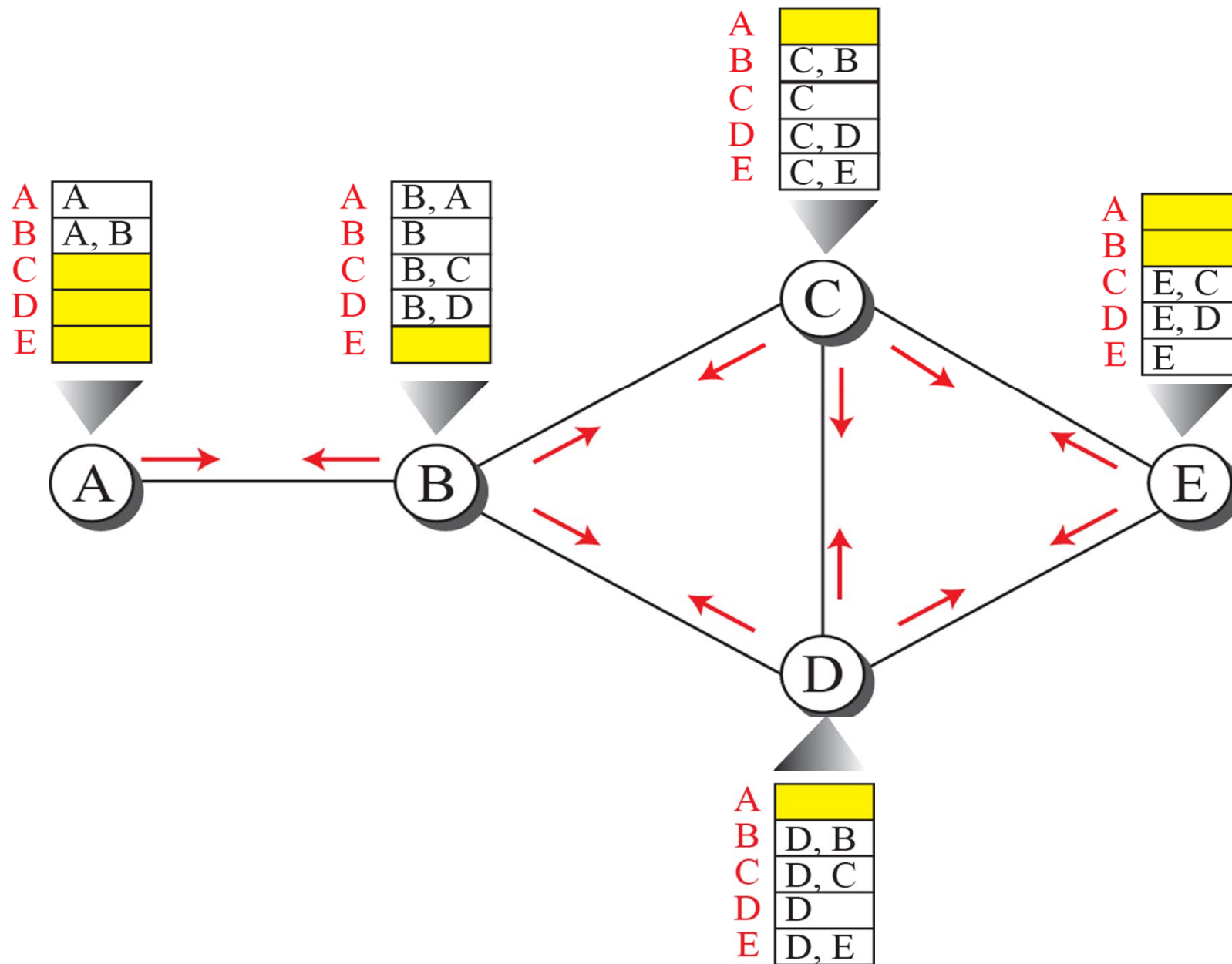
- Lägg till *path vector* för varje destination
- Påminner om *Distance Vector*
 - *Path Vector* innehåller info om hela vägen till destinationen
- Finn ***best path*** bland många tillgängliga vägar
- *Policy Based Routing*
 - Använd bara vägar som går igenom accepterade noder
 - Använd inte vägar där egna noden ingår (loop!)
 - Längd på *Path Vector* är ofta(?) viktigaste parameter

Spanning trees in path-vector routing

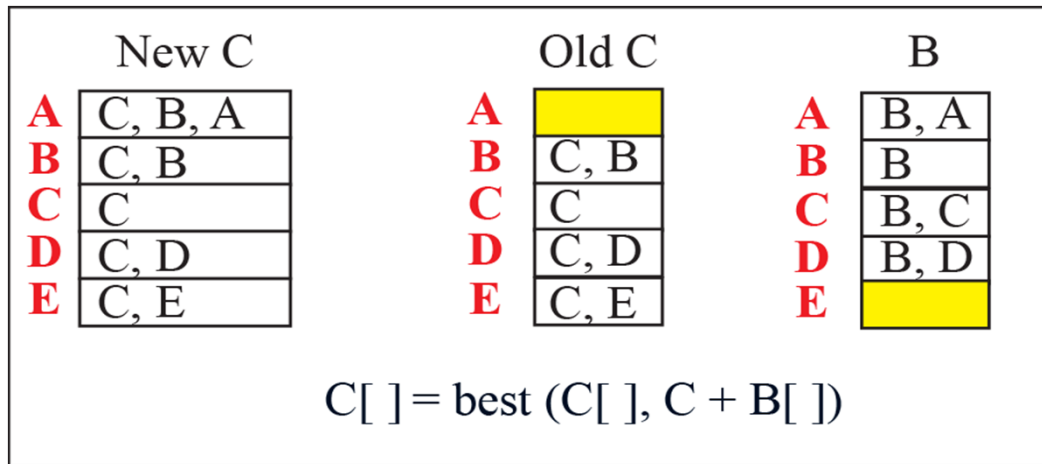
An internet



Path vectors made at booting time



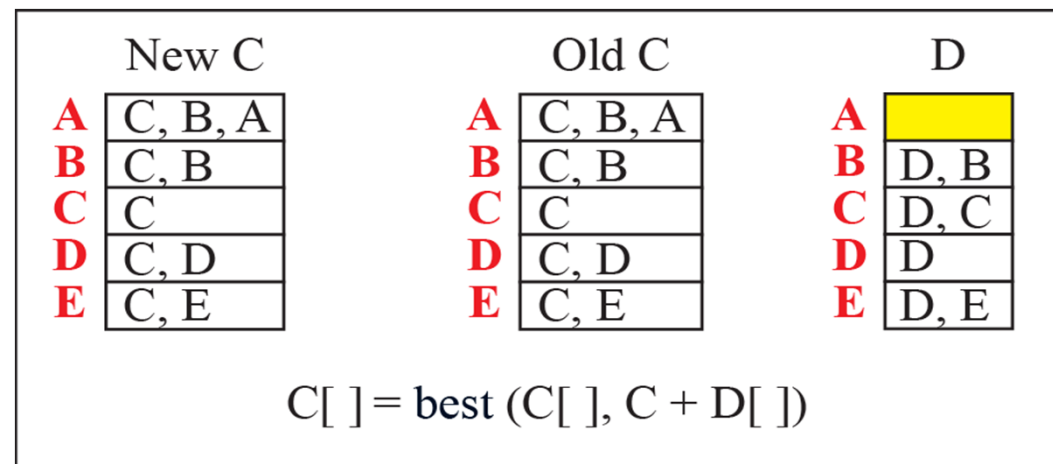
Updating path vectors



Note:
X []: vector X
Y: node Y

Event 1: C receives a copy of B's vector

”best” är
 implementering
 av policyn



Event 2: C receives a copy of D's vector

Mer om path vector

I ETSF10:

- Autonoma system, AS
- Routing mellan domäner
- Policy routing
- Routingprotokoll BGP

Comparison of LS and DV algorithms

Message complexity

LS: with n nodes, E links, $O(nE)$ msgs for full knowledge, changes sent to all nodes

DV: exchange between neighbours only

Speed of Convergence

LS: $O(n^2)$ algorithm requires $O(nE)$ msgs

- ◆ may have oscillations

DV: convergence time varies

- ◆ may be routing loops
- ◆ count-to-infinity problem

Robustness: what happens if router malfunctions?

LS:

- ◆ node can advertise incorrect *link* cost for its “own” links
- ◆ node can break broadcast path or change broadcasted info
- ◆ each node computes only its *own* table

DV:

- ◆ DV node can advertise incorrect *path* cost to any path
- ◆ each node’s table used by others
 - error propagate thru network